
wbia-vtool

Release latest

Dec 16, 2021

Contents:

1	vtool package	1
1.1	Submodules	1
1.2	vtool.__main__ module	1
1.3	vtool.grave module	1
1.4	vtool.old_matching module	2
1.5	vtool.pyflann_backend module	2
1.6	vtool.rhomb_dist module	2
1.7	vtool.blend module	2
1.8	vtool.chip module	6
1.9	vtool.clustering2 module	10
1.10	vtool.confusion module	19
1.11	vtool.coverage_grid module	27
1.12	vtool.coverage_kpts module	29
1.13	vtool.demodata module	33
1.14	vtool.deprecated module	35
1.15	vtool.distance module	36
1.16	vtool.ellipse module	42
1.17	vtool.exif module	43
1.18	vtool.features module	46
1.19	vtool.fondemo module	47
1.20	vtool.geometry module	48
1.21	vtool.histogram module	55
1.22	vtool.image module	64
1.23	vtool.image_filters module	88
1.24	vtool.image_shared module	89
1.25	vtool.inspect_matches module	89
1.26	vtool.keypoint module	91
1.27	vtool.linalg module	109
1.28	vtool.matching module	116
1.29	vtool.nearest_neighbors module	125
1.30	vtool.numpy_utils module	130
1.31	vtool.other module	135
1.32	vtool.patch module	160
1.33	vtool.quality_classifier module	172
1.34	vtool.score_normalization module	173
1.35	vtool.segmentation module	182

1.36	vtool.spatial_verification module	183
1.37	vtool.sver_c_wrapper module	192
1.38	vtool.symbolic module	192
1.39	vtool.trig module	193
1.40	vtool.util_math module	194
1.41	Module contents	201
2	Indices and tables	383
	Python Module Index	385
	Index	387

1.1 Submodules

1.2 vtool.__main__ module

```
vtool.__main__.main()
```

1.3 vtool._grave module

```
class vtool._grave.MultiMatchInspector
    Bases: object
    edge_doubleclick (qtindex)
    initialize (matches)
    populate_edge_model ()
class vtool._grave.ScoreNormalizerUnsupervised (X=None, **kwargs)
    Bases: object
    fit (X, y=None, verbose=True)
        Fits estimator to data.
    learn_probabilities (X, y=None, verbose=True)
    rrr (verbose=True, reload_module=True)
        special class reloading function This function is often injected as rrr of classes
    visualize ()
vtool._grave.bow_test ()
vtool._grave.match_inspect_graph ()
```

CommandLine: `python -m vtool.inspect_matches match_inspect_graph --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.inspect_matches import * # NOQA
>>> import wbia.guitool as gt
>>> import vtool as vt
>>> gt.ensure_qapp()
>>> ut.qtenure()
>>> self = match_inspect_graph()
>>> self.show()
>>> # xdoctest: +REQUIRES(--show)
>>> self.update()
>>> gt.qtapp_loop(qwin=self, freq=10)
```

1.4 vtool._old_matching module

1.5 vtool._pyflann_backend module

abstract which pyflann implementation is used

from vtool._pyflann_backend import pyflann

vtool._pyflann_backend.**FLANN_CLS**
alias of `pyflann.index.FLANN`

1.6 vtool._rhomb_dist module

vtool._rhomb_dist.**RhombicuboctahedronDistanceDemo()**

1.7 vtool.blend module

vtool.blend.**blend_images** (*img1*, *img2*, *mode*='average', ***kwargs*)

Parameters

- **img1** (*np.ndarray*) – first image
- **img2** (*np.ndarray*) – second image
- **mode** (*str*) – can be average, multiply, or overlay

vtool.blend.**blend_images_average** (*img1*, *img2*, *alpha*=0.5)

Parameters

- **img1** (*ndarray[uint8_t, ndim=2]*) – image data
- **img2** (*ndarray[uint8_t, ndim=2]*) – image data
- **alpha** (*float*) – (default = 0.5)

Returns *imgB*

Return type ndarray

References

https://en.wikipedia.org/wiki/Blend_modes

CommandLine: python -m vtool.blend blend_images_average:0 --show python -m vtool.blend blend_images_average:1 --show

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.blend import * # NOQA
>>> alpha = 0.8
>>> img1, img2 = testdata_blend()
>>> imgB = blend_images_average(img1, img2, alpha)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(imgB)
>>> ut.show_if_requested()
```

Ignore:

```
>>> # GRIDSEARCH
>>> from vtool.blend import * # NOQA
>>> test_func = blend_images_average
>>> args = testdata_blend()
>>> param_info = ut.ParamInfoList('blend_params', [
...     ut.ParamInfo('alpha', .8, 'alpha=',
...                   varyvals=np.linspace(0, 1.0, 25).tolist()),
... ])
>>> gridsearch_image_function(param_info, test_func, args)
>>> ut.show_if_requested()
```

vtool.blend.blend_images_average_stack(images, alpha=None)

Parameters

- **img1** (ndarray[uint8_t, ndim=2]) – image data
- **img2** (ndarray[uint8_t, ndim=2]) – image data
- **alpha** (float) – (default = 0.5)

Returns imgB

Return type ndarray

References

https://en.wikipedia.org/wiki/Blend_modes

CommandLine: python -m vtool.blend --test-blend_images_average:0 --show python -m vtool.blend --test-blend_images_average:1 --show

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.blend import * # NOQA
>>> alpha = 0.8
>>> img1, img2 = testdata_blend()
>>> imgB = blend_images_average(img1, img2, alpha)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(imgB)
>>> ut.show_if_requested()
```

`vtool.blend.blend_images_mult_average` (*img1*, *img2*, *alpha*=0.5)

Parameters

- **img1** (*ndarray*[*uint8_t*, *ndim*=2]) – image data
- **img2** (*ndarray*[*uint8_t*, *ndim*=2]) – image data
- **alpha** (*float*) – (default = 0.5)

Returns *imgB*

Return type *ndarray*

References

https://en.wikipedia.org/wiki/Blend_modes

CommandLine: `python -m vtool.blend --test-blend_images_mult_average:0 --show` `python -m vtool.blend --test-blend_images_mult_average:1 --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.blend import * # NOQA
>>> alpha = 0.8
>>> img1, img2 = testdata_blend()
>>> imgB = blend_images_mult_average(img1, img2, alpha)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(imgB)
>>> ut.show_if_requested()
```

Ignore:

```
>>> # GRIDSEARCH
>>> from vtool.blend import * # NOQA
>>> test_func = blend_images_mult_average
>>> args = testdata_blend()
>>> param_info = ut.ParamInfoList('blend_params', [
...     ut.ParamInfo('alpha', .8, 'alpha=',
...                   varyvals=np.linspace(0, 1.0, 9).tolist()),
... ])
>>> gridsearch_image_function(param_info, test_func, args)
>>> ut.show_if_requested()
```



```
vtool.blend.blend_images_multiply(img1, img2, alpha=0.5)
```

Parameters

- **img1** (`ndarray[uint8_t, ndim=2]`) – image data
- **img2** (`ndarray[uint8_t, ndim=2]`) – image data
- **alpha** (`float`) – (default = 0.5)

Returns imgB

Return type ndarray

References

https://en.wikipedia.org/wiki/Blend_modes

CommandLine: `python -m vtool.blend --test-blend_images_multiply:0 --show python -m vtool.blend --test-blend_images_multiply:1 --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.blend import * # NOQA
>>> alpha = 0.8
>>> img1, img2 = testdata_blend()
>>> imgB = blend_images_multiply(img1, img2, alpha)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(imgB)
>>> ut.show_if_requested()
```

Ignore:

```
>>> # GRIDSEARCH
>>> from vtool.blend import * # NOQA
>>> test_func = blend_images_multiply
>>> args = testdata_blend(scale=128)
>>> param_info = ut.ParamInfoList('blend_params', [
...     ut.ParamInfo('alpha', .8, 'alpha=',
...                   varyvals=np.linspace(0, 1.0, 9).tolist()),
... ])
>>> gridsearch_image_function(param_info, test_func, args)
>>> ut.show_if_requested()
```

```
vtool.blend.ensure_alpha_channel(img, alpha=1.0)
```

```
vtool.blend.ensure_grayscale(img, colorspace_hint='BGR')
```

```
vtool.blend.gamma_adjust(img, gamma=1.0)
```

CommandLine: `python -m vtool.blend --test-gamma_adjust:0 --show`

Ignore:

```
>>> # DISABLE_DOCTEST
>>> from vtool.blend import * # NOQA
>>> import vtool as vt
>>> test_func = gamma_adjust
>>> img_fpath = ut.grab_test_imgpath('carl.jpg')
>>> img = vt.rectify_to_float01(vt.imread(img_fpath))
>>> args = (img,)
>>> param_info = ut.ParamInfoList('blend_params', [
...     ut.ParamInfo('gamma', .8, 'gamma=',
...                   varyvals=np.linspace(.1, 2.5, 25).tolist()),
... ])
>>> gridsearch_image_function(param_info, test_func, args)
>>> ut.show_if_requested()
```

`vtool.blend.gridsearch_addWeighted()`

CommandLine: `xdoctest -m ~/code/vtool/vtool/blend.py gridsearch_addWeighted`

`vtool.blend.gridsearch_image_function(param_info, test_func, args=(), show_func=None)`
gridsearch for a function that produces a single image

`vtool.blend.overlay_alpha_images(img1, img2)`
places img1 on top of img2 respecting alpha channels

References

<http://stackoverflow.com/questions/25182421/overlay-numpy-alpha>

`vtool.blend.testdata_blend(scale=128)`

1.8 vtool.chip module

class `vtool.chip.ScaleStrat`

Bases: `object`

Scaling strategies

static area (*target, orig_wh, tol=0*)

The area becomes target

Parameters **target** (*int*) – target size

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> ut.assert_eq(ScaleStrat.area(800 ** 2, (190, 220)), (743, 861))
>>> ut.assert_eq(ScaleStrat.area(800 ** 2, (220, 190)), (861, 743))
```

static maxwh (*target, orig_wh, tol=0*)

The maximum dimension becomes target

Parameters **target** (*int*) – target size

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> ut.assert_eq(ScaleStrat.maxwh(800, (190, 220)), (691, 800))
>>> ut.assert_eq(ScaleStrat.maxwh(800, (220, 190)), (800, 691))
```

static width (*target, orig_wh, tol=0*)

The width becomes target

Parameters *target* (*int*) – target size

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> ut.assert_eq(ScaleStrat.width(800, (190, 220)), (800, 926))
>>> ut.assert_eq(ScaleStrat.width(800, (220, 190)), (800, 691))
```

vtool.chip.apply_filter_funcs (*chipBGR, filter_funcs*)

applies a list of preprocessing filters to a chip

DEPRICATE

vtool.chip.compute_chip (*gfpah, bbox, theta, new_size, filter_list=[], interpolation=4*)

Extracts a chip and applies filters

DEPRICATE

Parameters

- **gfpah** (*str*) – image file path string
- **bbox** (*tuple*) – bounding box in the format (x, y, w, h)
- **theta** (*float*) – angle in radians
- **new_size** (*tuple*) – must maintain the same aspect ratio or else you will get weirdness
- **filter_list** (*list*) –

Returns chipBGR - cropped image

Return type ndarray

CommandLine: python -m vtool.chip --test-compute_chip --show

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.chip import * # NOQA
>>> from vtool.util_math import TAU
>>> # build test data
>>> gfpah = ut.grab_test_imgpath('car1.jpg')
>>> bbox = (100, 3, 100, 100)
>>> theta = TAU / 8
>>> new_size = (32, 32)
>>> filter_list = []
```

(continues on next page)

(continued from previous page)

```

>>> # execute function
>>> chipBGR = compute_chip(gfpath, bbox, theta, new_size, filter_list)
>>> # verify results
>>> assert chipBGR.shape[0:2] == new_size[::-1], 'did not resize correctly'
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> import vtool as vt
>>> pt.imshow(vt.draw_verts(vt.imread(gfpath), vt.scaled_verts_from_bbox(bbox,
↳ theta, 1, 1)), pnum=(1, 2, 1))
>>> pt.imshow(chipBGR, pnum=(1, 2, 2))
>>> pt.show_if_requested()

```

`vtool.chip.extract_chip_from_gpath(gfpath, bbox, theta, new_size, interpolation=4)`

`vtool.chip.extract_chip_from_gpath_into_square(args)`

`vtool.chip.extract_chip_from_img(imgBGR, bbox, theta, new_size, interpolation=4)`

Crops chip from image ; Rotates and scales;

`ibs.show_annot_image(aid)[0].pt_save_and_view()`

Parameters

- `gfpath` (*str*) –
- `bbox` (*tuple*) – xywh
- `theta` (*float*) –
- `new_size` (*tuple*) – wy

Returns chipBGR

Return type ndarray

CommandLine: `python -m vtool.chip --test-extract_chip_from_img python -m vtool.chip --test-extract_chip_from_img --show`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.chip import * # NOQA
>>> # build test data
>>> imgBGR = gtool.imread(ut.grab_test_imgpath('car1.jpg'))
>>> bbox = (100, 3, 100, 100)
>>> theta = 0.0
>>> new_size = (58, 34)
>>> # execute function
>>> chipBGR = extract_chip_from_img(imgBGR, bbox, theta, new_size)
>>> # verify results
>>> assert chipBGR.shape[0:2] == new_size[::-1], 'did not resize correctly'
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(chipBGR)
>>> pt.show_if_requested()

```

`vtool.chip.extract_chip_into_square(imgBGR, bbox, theta, target_size)`

`vtool.chip.get_extramargin_measures(bbox_gs, new_size, halfoffset_ms=(64, 64))`

Computes a detection chip with a bit of spatial context so the detection algorithm doesn't clip boundaries

Returns

mbbox_gs, margin_size - margin bounding box in image size, size of entire margined chip,

CommandLine: `python -m vtool.chip --test-get_extramargin_measures --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.chip import * # NOQA
>>> gfp_path = ut.grab_test_imgpath('car1.jpg')
>>> bbox_gs = [40, 40, 150, 150]
>>> theta = .15 * (np.pi * 2)
>>> new_size = (150, 150)
>>> halfoffset_ms = (32, 32)
>>> mbbox_gs, margin_size = get_extramargin_measures(bbox_gs, new_size,
↳halfoffset_ms)
>>> # xdoctest: +REQUIRES(--show)
>>> testshow_extramargin_info(gfp_path, bbox_gs, theta, new_size, halfoffset_ms,
↳mbbox_gs, margin_size)
```

`vtool.chip.get_image_to_chip_transform(bbox, chipsize, theta)`

transforms image space into chip space

Parameters

- - bounding box of chip in image space (*bbox*) -
- - size of the chip (*chipsize*) -
- - rotation of the bounding box (*theta*) -

Ignore:

```
>>> # https://groups.google.com/forum/#!topic/sympy/k1HnZK_bNNA
>>> from vtool.patch import * # NOQA
>>> import sympy
>>> import sympy.abc
>>> theta = sympy.abc.theta
>>>
>>> x, y, w, h, target_area = sympy.symbols('x y w h, a')
>>> gx, gy = sympy.symbols('gx, gy')
>>>
>>> round = sympy.floor # hack
>>>
>>> ht = sympy.sqrt(target_area * h / w)
>>> wt = w * ht / h
>>> cw_, ch_ = round(wt), round(ht)
>>>
>>> from vtool import ltool
>>> T1 = ltool.translation_mat3x3(tx1, ty1, dtype=None)
>>> S = ltool.scale_mat3x3(sx, sy, dtype=None)
>>> R = ltool.rotation_mat3x3(-theta, sympy.sin, sympy.cos)
>>> T2 = ltool.translation_mat3x3(tx2, ty2, dtype=None)
>>>
```

(continues on next page)

(continued from previous page)

```

>>> def add_matmul_hold_prop(mat):
>>>     #import functools
>>>     mat = sympy.Matrix(mat)
>>>     def matmul_hold(other, hold=False):
>>>         new = sympy.MatMul(mat, other, hold=hold)
>>>         add_matmul_hold_prop(new)
>>>         return new
>>>     setattr(mat, 'matmul_hold', matmul_hold)
>>>     return mat
>>>
>>> T1 = add_matmul_hold_prop(T1)
>>> T2 = add_matmul_hold_prop(T2)
>>> R = add_matmul_hold_prop(R)
>>> S = add_matmul_hold_prop(S)
>>>
>>> C = T2.multiply(R.multiply(S.multiply(T1)))
>>> sympy.simplify(C)

```

`vtool.chip.get_scaled_size_with_dlen(target_dlen, w, h)`
 returns new_size which scales (w, h) as close to target_dlen as possible and maintains aspect ratio

`vtool.chip.gridsearch_chipextract()`

CommandLine: `xdoctest -m ~/code/vtool/vtool/chip.py gridsearch_chipextract --show`

Example

```

>>> # DISABLE_DOCTEST
>>> # GRIDSEARCH
>>> from vtool.chip import * # NOQA
>>> gridsearch_chipextract()
>>> ut.show_if_requested()

```

`vtool.chip.testshow_extramargin_info(gfpath, bbox_gs, theta, new_size, halfoffset_ms, mb-
 box_gs, margin_size)`

1.9 vtool.clustering2 module

Todo: Does HDBSCAN work on 128 dim vectors? <http://nbviewer.jupyter.org/github/lmcinnes/hdbscan/blob/master/notebooks/Comparing%20Clustering%20Algorithms.ipynb>

class `vtool.clustering2.AnnoyWrapper`

Bases: `object`

flann-like interface to annoy

build_annoy (*centroids, trees=3*)

nn (*data_vecs, query_vecs, num, trees=3, checks=-1*)

query_annoy (*query_vecs, num, checks=-1*)

`vtool.clustering2.apply_grouping(items, groupxs, axis=0)`
 applies grouping from group_indices apply_grouping

Parameters

- **items** (*ndarray*) –
- **groupxs** (*list of ndarrays*) –

Returns grouped items**Return type** list of ndarrays**SeeAlso:** group_indices invert_apply_grouping**CommandLine:** python -m vtool.clustering2 -test-apply_grouping**Example**

```
>>> # ENABLE_DOCTEST
>>> from vtool.clustering2 import * # NOQA
>>> idx2_groupid = np.array([2, 1, 2, 1, 2, 1, 2, 3, 3, 3])
>>> items = np.array([1, 8, 5, 5, 8, 6, 7, 5, 3, 0, 9])
>>> (keys, groupxs) = group_indices(idx2_groupid)
>>> grouped_items = apply_grouping(items, groupxs)
>>> result = str(grouped_items)
>>> print(result)
[array([8, 5, 6]), array([1, 5, 8, 7]), array([5, 3, 0, 9])]
```

vtool.clustering2.**apply_grouping_**(items, groupxs)

non-optimized version

vtool.clustering2.**apply_grouping_iter**(items, groupxs)vtool.clustering2.**apply_grouping_iter2**(items, groupxs)vtool.clustering2.**apply_jagged_grouping**(unflat_items, groupxs)

takes unflat_list and flat group indices. Returns the unflat grouping

vtool.clustering2.**example_binary**()vtool.clustering2.**find_duplicate_items**(item_arr)**Parameters** item_arr –**Returns** duplicate_items**Return type**

?

CommandLine: python -m vtool.clustering2 -test-find_duplicate_items**References**<http://stackoverflow.com/questions/21888406/getting-the-indexes-to-the-duplicate-columns-of-a-numpy-array>**Example**

```

>>> # DISABLE_DOCTEST
>>> from vtool.clustering2 import * # NOQA
>>> np.random.seed(0)
>>> item_arr = np.random.randint(100, size=30)
>>> duplicate_items = find_duplicate_items(item_arr)
>>> assert duplicate_items == list(six.iterkeys(ut.find_duplicate_items(item_
↪arr)))
>>> result = str(duplicate_items)
>>> print(result)
[9, 67, 87, 88]

```

`vtool.clustering2.group_indices` (*idx2_groupid*, *assume_sorted=False*)

Parameters `idx2_groupid` (*ndarray*) – numpy array of group ids (must be numeric)

Returns (keys, groupxs)

Return type `tuple` (*ndarray*, list of *ndarrays*)

CommandLine: `xdoctest -m ~/code/vtool/vtool/clustering2.py group_indices xdoctest -m ~/code/vtool/vtool/clustering2.py group_indices:0 xdoctest -m ~/code/vtool/vtool/clustering2.py group_indices:1`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.clustering2 import * # NOQA
>>> idx2_groupid = np.array([2, 1, 2, 1, 2, 1, 2, 3, 3, 3, 3])
>>> (keys, groupxs) = group_indices(idx2_groupid)
>>> result = ut.repr2((keys, groupxs), nl=2, nobr=True, with_dtype=True)
>>> print(result)
np.array([1, 2, 3], dtype=np.int64),
[
  np.array([1, 3, 5], dtype=np.int64),
  np.array([0, 2, 4, 6], dtype=np.int64),
  np.array([ 7,  8,  9, 10], dtype=np.int64),
],

```

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.clustering2 import * # NOQA
>>> idx2_groupid = np.array([[ 24], [129], [ 659], [ 659], [ 24],
... [ 659], [ 659], [ 822], [ 659], [ 659], [24]])
>>> # 2d arrays must be flattened before coming into this function so
>>> # information is on the last axis
>>> (keys, groupxs) = group_indices(idx2_groupid.T[0])
>>> result = ut.repr2((keys, groupxs), nl=2, nobr=True, with_dtype=True)
>>> print(result)
np.array([ 24, 129, 659, 822], dtype=np.int64),
[
  np.array([ 0,  4, 10], dtype=np.int64),
  np.array([1], dtype=np.int64),
  np.array([2, 3, 5, 6, 8, 9], dtype=np.int64),

```

(continues on next page)

(continued from previous page)

```
np.array([7], dtype=np.int64),
],
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.clustering2 import * # NOQA
>>> idx2_groupid = np.array([True, True, False, True, False, False, True])
>>> (keys, groupxs) = group_indices(idx2_groupid)
>>> result = ut.repr2((keys, groupxs), nl=2, nobr=True, with_dtype=True)
>>> print(result)
np.array([False,  True], dtype=np.bool),
[
  np.array([2, 4, 5], dtype=np.int64),
  np.array([0, 1, 3, 6], dtype=np.int64),
],
```

Time:

```
>>> # xdoctest: +SKIP
>>> import vtool as vt
>>> setup = ut.extract_timeit_setup(vt.group_indices, 2, 'groupxs =')
>>> print(setup)
>>> stmt_list = ut.codeblock(
    '''
        [sortx[lx:rx] for lx, rx in ut.itertwo(idxs)]
        [sortx[lx:rx] for lx, rx in zip(idxs, idxs[1:])]
        #[sortx[lx:rx] for lx, rx in ut.iter_window(idxs)]
        #[sortx[slice(*_)] for _ in ut.itertwo(idxs)]
        #[sortx[slice(lr, lx)] for lr, lx in ut.itertwo(idxs)]
        #np.split(sortx, idxs[1:-1])
        #np.hsplit(sortx, idxs[1:-1])
        np.array_split(sortx, idxs[1:-1])
    ''').split('\n')
>>> stmt_list = [x for x in stmt_list if not x.startswith('#')]
>>> passed, times, outputs = ut.timeit_compare(stmt_list, setup,
↳ iterations=10000)
```

```
>>> # xdoctest: +SKIP
>>> stmt_list = ut.codeblock(
    '''
        np.diff(groupids_sorted)
        np.ediff1d(groupids_sorted)
        np.subtract(groupids_sorted[1:], groupids_sorted[:-1])
    ''').split('\n')
>>> stmt_list = [x for x in stmt_list if not x.startswith('#')]
>>> passed, times, outputs = ut.timeit_compare(stmt_list, setup,
↳ iterations=10000)
```

Ignore: import numba group_indices_numba = numba.jit(group_indices)
group_indices_numba(idx2_groupid)

SeeAlso: apply_grouping

References

<http://stackoverflow.com/questions/4651683/numpy-grouping-using-itertools-groupby-performance>

Todo: Look into `np.split` <http://stackoverflow.com/questions/21888406/getting-the-indexes-to-the-duplicate-columns-of-a-numpy-array>

`vtool.clustering2.groupby(items, idx2_groupid)`

```
>>> items = np.array(np.arange(100))
>>> idx2_groupid = np.array(np.random.randint(0, 4, size=100))
>>> items = idx2_groupid
```

`vtool.clustering2.groupby_dict(items, idx2_groupid)`

`vtool.clustering2.groupby_gen(items, idx2_groupid)`

```
>>> items = np.array(np.arange(100))
>>> idx2_groupid = np.array(np.random.randint(0, 4, size=100))
```

`vtool.clustering2.groupedzip(id_list, datas_list)`

Function for grouping multiple lists of data (stored in `datas_list`) using `id_list`.

Parameters

- `id_list` (*list*) –
- `datas_list` (*list*) –

Returns `_iter`

Return type `iterator`

CommandLine: `python -m vtool.clustering2 --test-groupedzip`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.clustering2 import * # NOQA
>>> # build test data
>>> id_list = np.array([1, 2, 1, 2, 1, 2, 3])
>>> datas_list = [
...     ['a', 'b', 'c', 'd', 'e', 'f', 'g'],
...     ['A', 'B', 'C', 'D', 'E', 'F', 'G'],
... ]
>>> # execute function
>>> groupxs, grouped_iter = groupedzip(id_list, datas_list)
>>> grouped_tuples = list(grouped_iter)
>>> # verify results
>>> result = str(groupxs) + '\n'
>>> result += ub.repr2(grouped_tuples, nl=1)
>>> print(result)
[1 2 3]
[
```

(continues on next page)

(continued from previous page)

```
[(['a', 'c', 'e'], ['A', 'C', 'E']),
(['b', 'd', 'f'], ['B', 'D', 'F']),
(['g'], ['G']),
]
```

`vtool.clustering2.invert_apply_grouping(grouped_items, groupxs)`

Parameters

- **grouped_items** (*list*) – of lists
- **groupxs** (*list*) – of lists

Returns items

Return type *list*

CommandLine: `python -m vtool.clustering2 -test-invert_apply_grouping`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.clustering2 import * # NOQA
>>> grouped_items = [[8, 5, 6], [1, 5, 8, 7], [5, 3, 0, 9]]
>>> groupxs = [np.array([1, 3, 5]), np.array([0, 2, 4, 6]), np.array([ 7, 8, 9,
↪10])]
>>> items = invert_apply_grouping(grouped_items, groupxs)
>>> result = items
>>> print(result)
[1, 8, 5, 5, 8, 6, 7, 5, 3, 0, 9]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.clustering2 import * # NOQA
>>> grouped_items, groupxs = [], []
>>> result = invert_apply_grouping(grouped_items, groupxs)
>>> print(result)
[]
```

`vtool.clustering2.invert_apply_grouping2(grouped_items, groupxs, dtype=None)`

use only when ungrouping will be complete

`vtool.clustering2.invert_apply_grouping3(grouped_items, groupxs, maxval)`

`vtool.clustering2.jagged_group(groupids_list)`

flattens and returns group indexes into the flattened list

`vtool.clustering2.plot_centroids(data, centroids, num_pca_dims=3, whiten=False, labels='centroids', fnum=1, prefix='')`

Plots centroids and datapoints. Plots accurately up to 3 dimensions. If there are more than 3 dimensions, PCA is used to recude the dimensionality to the <num_pca_dims> principal components

`vtool.clustering2.sorted_indices_ranges(groupids_sorted)`

Like group sorted indices but returns a list of slices

`vtool.clustering2.tune_flann2(data)`

```
vtool.clustering2.uniform_sample_hypersphere(num, ndim=2, only_quadrent_1=False)
```

Not quite done yet

References

https://en.wikipedia.org/wiki/Regular_polytope https://en.wikipedia.org/wiki/Platonic_solid#Higher_dimensions <https://en.wikipedia.org/wiki/Cross-polytope>

Parameters

- **num** –
- **ndim** (*int*) – (default = 2)

CommandLine: `python -m vtool.clustering2 --test-uniform_sampe_hypersphere`

Ignore: `#pip install polytope sudo pip install cvxopt --no-deps`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.clustering2 import * # NOQA
>>> num = 100
>>> ndim = 3
>>> pts = uniform_sampe_hypersphere(num, ndim)
>>> print(pts)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> if ndim == 2:
>>>     pt.plot(pts.T[0], pts.T[1], 'gx')
>>> elif ndim == 3:
>>>     #pt.plot_surface3d(pts.T[0], pts.T[1], pts.T[2])
>>>     from mpl_toolkits.mplot3d import Axes3D # NOQA
>>>     fig = pt.figure(1, doclf=True, docla=True)
>>>     ax = fig.add_subplot(111, projection='3d')
>>>     ax.scatter(pts.T[0], pts.T[1], pts.T[2], s=20, marker='o', alpha=1)
>>>     ax.autoscale(enable=False)
>>>     ax.set_aspect('equal')
>>>     df2.dark_background(ax)
>>> pt.dark_background()
>>> ut.show_if_requested()
```

```
vtool.clustering2.unsupervised_multicut_labeling(cost_matrix, thresh=0)
```

Notes

requires CPLEX

CommandLine: `python -m vtool.clustering2 unsupervised_multicut_labeling --show`

Ignore:

```
>>> # synthetic data
>>> import vtool as vt
>>> size = 100
>>> thresh = 50
```

(continues on next page)

(continued from previous page)

```

>>> np.random.randint(0, 1)
>>> np.zeros((size, size))
>>> #np.random.rand(size, size)
>>> size = 45
>>> #size = 10
>>> size = 5
>>> aids = np.arange(size)
>>> rng = np.random.RandomState(443284320)
>>> encounter_lbls = rng.randint(0, size, size)
>>> separation = 5.0
>>> separation = 1.10
>>> grid1 = np.tile(encounter_lbls, (size, 1))
>>> is_match = grid1.T == grid1
>>> good_pos = np.where(is_match)
>>> bad_pos = np.where(~is_match)
>>> cost_matrix_ = np.zeros((size, size))
>>> cost_matrix_[good_pos] = rng.randn(len(good_pos[0])) + separation
>>> cost_matrix_[bad_pos] = rng.randn(len(bad_pos[0])) - separation
>>> false_val = min(cost_matrix_.min(), np.min(rng.randn(1000) -
↳separation))
>>> true_val = max(cost_matrix_.max(), np.max(rng.randn(500) +
↳separation))
>>> cost_matrix_[np.diag_indices_from(cost_matrix_)] = true_val
>>> #cost_matrix_[np.diag_indices_from(cost_matrix_)] = np.inf
>>> cost_matrix = (cost_matrix_ - false_val) / (true_val - false_val)
>>> cost_matrix = 2 * (cost_matrix - .5)
>>> thresh = 0
>>> labels = vt.unsupervised_multicut_labeling(cost_matrix, thresh)
>>> diff = ut.find_group_differences(
>>>     list(ut.group_items(aids, encounter_lbls).values()),
>>>     list(ut.group_items(aids, labels).values()))
>>> print('diff = %r' % (diff,))

```

```

#gm, = ut.exec_func_src(vt.unsupervised_multicut_labeling, #key_list=['gm'], sentinal='inf =
opengm') #parameter = opengm.InfParam() %%timeit opengm.inference.Multicut(gm, parame-
ter=parameter).infer()

```

Example

```

>>> # SCRIPT
>>> from vtool.clustering2 import * # NOQA
>>> import networkx as nx
>>> import wbia.plottool as pt
>>> rng = np.random.RandomState(443284320)
>>> pt.ensureqt()
>>> #
>>> def make_test_costmatrix(name_labels, view_labels, separation=2):
>>>     is_same = name_labels == name_labels[:, None]
>>>     is_comp = np.abs(view_labels - view_labels[:, None]) <= 1
>>>     good_pos = np.where(is_same)
>>>     bad_pos = np.where(~is_same)
>>>     cost_matrix_ = np.zeros((len(name_labels), len(name_labels)))
>>>     cost_matrix_[good_pos] = rng.randn(len(good_pos[0])) + separation
>>>     cost_matrix_[bad_pos] = rng.randn(len(bad_pos[0])) - separation
>>>     cost_matrix_ = (cost_matrix_.T + cost_matrix_) / 2

```

(continues on next page)

(continued from previous page)

```

>>> false_val = min(cost_matrix_.min(), np.min(rng.randn(1000) - separation))
>>> true_val = max(cost_matrix_.max(), np.max(rng.randn(500) + separation))
>>> cost_matrix_[np.diag_indices_from(cost_matrix_)] = true_val
>>> cost_matrix = (cost_matrix_ - false_val) / (true_val - false_val)
>>> cost_matrix = 2 * (cost_matrix - .5)
>>> cost_matrix[np.where(~is_comp)] = 0
>>> return cost_matrix
>>> #
>>> view_labels = np.array([0, 0, 2, 2, 1, 0, 0, 0])
>>> name_labels = np.array([0, 0, 0, 0, 0, 1, 1, 1])
>>> #cost_matrix = make_test_costmatrix(name_labels, view_labels, 2)
>>> cost_matrix = make_test_costmatrix(name_labels, view_labels, .9)
>>> #
>>> def multicut_value(cost_matrix, name_labels):
>>>     grid1 = np.tile(name_labels, (len(name_labels), 1))
>>>     isdiff = grid1.T != grid1
>>>     cut_value = cost_matrix[isdiff].sum()
>>>     return cut_value
>>> #
>>> aids = np.arange(len(name_labels))
>>> #
>>> graph = ut.nx_from_matrix(cost_matrix)
>>> weights = nx.get_edge_attributes(graph, 'weight')
>>> #
>>> floatfmt1 = ut.partial(ub.map_vals, lambda x: 'w=%.2f' % x)
>>> floatfmt2 = ut.partial(ub.map_vals, lambda x: 'l=%.2f' % x)
>>> #
>>> lens = ub.map_vals(lambda x: (1 - ((x + 1) / 2)) / 2, weights)
>>> labels = floatfmt1(weights)
>>> #labels = floatfmt2(lens)
>>> nx.set_edge_attributes(graph, name='label', values=labels)
>>> #nx.set_edge_attributes(graph, name='len', values=lens)
>>> nx.set_node_attributes(graph, name='shape', values='ellipse')
>>> encounter_lbls_str = [str(x) for x in name_labels]
>>> node_name_lbls = dict(zip(aids, encounter_lbls_str))
>>> import vtool as vt
>>> #
>>> mcut_labels = vt.unsupervised_multicut_labeling(cost_matrix, thresh=vt.eps)
>>> diff = ut.find_group_differences(
>>>     list(ut.group_items(aids, name_labels).values()),
>>>     list(ut.group_items(aids, mcut_labels).values()))
>>> print('diff = %r' % (diff,))
>>> #
>>> nx.set_node_attributes(graph, name='label', values=node_name_lbls)
>>> node_mcut_lbls = dict(zip(aids, mcut_labels))
>>> nx.set_node_attributes(graph, name='mcut_label', values=node_mcut_lbls)
>>> #
>>> print('mc_val(name) ' + str(multicut_value(cost_matrix, name_labels)))
>>> print('mc_val(mcut) ' + str(multicut_value(cost_matrix, mcut_labels)))
>>> #
>>> ut.color_nodes(graph, 'mcut_label')
>>> #
>>> # remove noncomparable edges
>>> is_comp = np.abs(view_labels - view_labels[:, None]) <= 1
>>> #
>>> noncomp_edges = list(zip(*np.where(~is_comp)))
>>> graph.remove_edges_from(noncomp_edges)

```

(continues on next page)

(continued from previous page)

```

>>> #
>>> layoutkw = {
>>>     'sep' : 5,
>>>     'prog': 'neato',
>>>     'overlap': 'false',
>>>     'splines': 'spline',
>>> }
>>> pt.show_nx(graph, layoutkw=layoutkw)
>>> ut.show_if_requested()

```

1.10 vtool.confusion module

Module for – Confusion matrix, contingency, error matrix,

References

http://en.wikipedia.org/wiki/Confusion_matrix

class vtool.confusion.**ConfusionMetrics**

Bases: ubelt.util_mixins.NiceRepr

Can compute average percision using the PASCAL definition

References

http://www.flinders.edu.au/science_engineering/fms/School-CSEM/publications/tech_reps-research_artfcts/TRRA_2007.pdf http://www.alta.asn.au/events/altss_w2003_proc/altss/courses/powers/Bookmaker-all/200302-ICCS-Bookmaker.pdfcs <http://www.cs.bris.ac.uk/Publications/Papers/1000704.pdf> http://en.wikipedia.org/wiki/Information_retrieval http://en.wikipedia.org/wiki/Precision_and_recall https://en.wikipedia.org/wiki/Confusion_matrix http://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve

SeeAlso: sklearn.metrics.ranking._binary_clf_curve

Notes

From oxford: Precision is defined as the ratio of retrieved positive images to the total number retrieved. Recall is defined as the ratio of the number of retrieved positive images to the total number of positive images in the corpus.

Ignore: varname_list = 'tp, fp, fn, tn, fpr, tpr, tpa'.split(', ') lines = ['self.{varname} = {varname}'.format(varname=varname) for varname in varname_list] print(ut.indent('n'.join(lines)))

CommandLine: python -m vtool.confusion ConfusionMetrics --show

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.confusion import * # NOQA
>>> scores, labels = testdata_scores_labels()
>>> c = self = confusions = ConfusionMetrics().fit(scores, labels)
>>> assert np.all(c.n_pos == c.n_tp + c.n_fn)
>>> assert np.all(c.n_neg == c.n_tn + c.n_fp)
>>> assert np.all(np.isclose(c.rp + c.rn, 1.0))
>>> assert np.all(np.isclose(c.pp + c.pn, 1.0))
>>> assert np.all(np.isclose(c.fpr, 1 - c.tnr))
>>> assert np.all(np.isclose(c.fnr, 1 - c.tpr))
>>> assert np.all(np.isclose(c.tpr, c.tp / c.rp))
>>> assert np.all(np.isclose(c.tpa, c.tp / c.pp))
>>> assert np.all(np.isclose(c.jacc, c.tp / (c.tp + c.fn + c.fp)))
>>> assert np.all(np.isclose(c.mcc, np.sqrt(c.mk * c.bm)))
>>> assert np.all(np.isclose(
>>>     c.acc, (c.tpr + c.c * (1 - c.fpr)) / (1 + c.c)))
>>> assert np.all(np.isclose(c.ppv, c.recall * c.prev / c.bias))
>>> assert np.all(np.isclose(
>>>     c.wracc, 4 * c.c * (c.tpr - c.fpr) / (1 + c.c) ** 2))
>>> # xdoctest: +REQUIRES(--show)
>>> confusions.draw_roc_curve()
>>> ut.show_if_requested()
```

acc

accuracy

aliases = {'acc': {'accuracy', 'rand_accuracy', 'tea', 'ter'}, 'bm': {'bookmaker_informedness'}}

auc

The AUC is a standard measure used to evaluate a binary classifier and represents the probability that a random correct case will receive a higher score than a random incorrect case.

References

https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve

auc_trap

bm

bookmaker informedness

c

cs

class ratio

cv

ratio of cost of making a mistake

draw_precision_recall_curve (*nSamples=11, **kwargs*)

draw_roc_curve (***kwargs*)

fit (*scores, labels, verbose=False*)

fn

false negative probability

fnr

miss rate, false negative rate

fp

false positive probability

fpr

fallout, false positive rate

classmethod from_tp_and_tn_scores (*tp_scores, tn_scores, verbose=False*)**get_ave_precision** ()**get_fpr_at_recall** (*target_recall*)**get_index_at_metric** (*at_metric, at_value, subindex=False, tiebreaker='maxthresh'*)

Finds the index that is closet to the metric at a given value

Parameters tiebreaker (*str*) – either ‘minimize’ or ‘maximize’ if ‘maximize’, then a larger threshold is considered better when resolving ambiguities. Otherwise a smaller thresh is better.

Doctest:

```
>>> from vtool.confusion import *
>>> pat1 = [0, 0, 0, 0]
>>> pat2 = [0, 0, 1, 1]
>>> pat3 = [0, 1, 1, 1]
>>> pat4 = [1, 1, 1, 1]
>>> pats = [pat1, pat2, pat3, pat4]
>>> n = 4
>>> import itertools as it
>>> s = it.count(0)
>>> # Create places of ambiguity and unambiguity
>>> x = list(ub.flatten([[next(s)] * len(pat) for pat in pats for _ in
↳ range(n)]))
>>> y = list(ub.flatten([pat for pat in pats for _ in range(n)]))
>>> self = ConfusionMetrics().fit(x, y)
>>> at_metric = 'n_false_pos'
>>> at_value = 0
>>> subindex = False
>>> idx1 = self.get_index_at_metric(at_metric, at_value, subindex=False,
↳ tiebreaker='minthresh')
>>> idx2 = self.get_index_at_metric(at_metric, at_value, subindex=False,
↳ tiebreaker='maxthresh')
>>> assert idx1 == 3
>>> assert idx2 == 0
```

get_metric_at_index (*metric, subindex*)

get_metric_at_metric (*get_metric, at_metric, at_value, subindex=False, tiebreaker='maxthresh'*)

Finds the corresponding value of *get_metric* at a specific value of *at_metric*.

get_metric = 'fpr' at_metric = 'tpr' at_value = .25 self.rrr()

self.get_metric_at_metric('fpr', 'tpr', .25) self.get_metric_at_metric('n_false_pos', 'tpr', .25)

self.get_metric_at_metric('n_true_pos', 'tpr', .25)

get_metric = 'n_true_pos' at_metric = 'n_false_pos' at_value = 0 subindex = False

get_metric_at_thresh (*metric, thresh*)**Parameters**

- **metric** (*str*) – name of a metric
- **thresh** (*float*) – desired threshold

Returns value - metric value

Return type float

CommandLine: python -m vtool.confusion --exec-get_metric_at_threshold

Ignore:

```
>>> self = cfms
>>> metric = 'fpr'
>>> thresh = 0
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.confusion import * # NOQA
>>> scores, labels = testdata_scores_labels()
>>> self = ConfusionMetrics().fit(scores, labels)
>>> metric = 'tpr'
>>> thresh = .8
>>> thresh = [0, .1, .9, 1.0]
>>> value = self.get_metric_at_thresh(metric, thresh)
>>> result = ('(None, None) = %s' % (str((None, None)),))
>>> print(result)
```

get_recall_at_fpr (*target_fpr*)

get_thresh_at_metric (*metric, value, maximize=None*)

Gets a threshold for a binary classifier using a target metric and value

Parameters

- **metric** (*str*) – name of metric like tpr or fpr
- **value** (*float*) – corresponding numeric value

Returns thresh

Return type float

CommandLine: python -m vtool.confusion get_thresh_at_metric python -m vtool.confusion --exec-interact_roc_factory --show

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.confusion import * # NOQA
>>> scores, labels = testdata_scores_labels()
>>> self = ConfusionMetrics().fit(scores, labels)
>>> metric = 'tpr'
>>> value = .85
>>> thresh = self.get_thresh_at_metric(metric, value)
>>> print('%s = %r' % (metric, value,))
>>> result = ('thresh = %s' % (str(thresh),))
```

(continues on next page)

(continued from previous page)

```
>>> print(result)
thresh = 22.5
```

Ignore: metric = 'fpr' value = 1e-4 self = cfms maximize = False

```
interpolate_replbounds(metric_values, self.thresholds, 0, maximize=maximize) inter-
polate_replbounds(metric_values, self.thresholds, 1e-4, maximize=maximize) interpo-
late_replbounds(metric_values, self.thresholds, 1e-3, maximize=maximize) interpo-
late_replbounds(metric_values, self.thresholds, 1e-2, maximize=maximize) interpo-
late_replbounds(metric_values, self.thresholds, 1e-2, maximize=maximize)
```

get_thresh_at_metric_max (*metric*)

metric = 'mcc' metric = 'fnr'

inv_aliases = {'acc': 'acc', 'accuracy': 'acc', 'bias': 'pp', 'bm': 'bm', 'bookmak

jacc

jaccard coefficient

mcc

matthews correlation coefficient

Also true that: $mcc == \text{np.sqrt}(self.bm * self.mk)$

minimizing_metrics = {'fn', 'fnr', 'fp', 'fpr'}

mk

markedness

paper_alias = [['dtp', 'determinant'], ['lr', 'likelihood-ratio'], ['nlr', 'negative-li

paper_relations = {'BMG': ['dtp / evenness_G'], 'BiasG2': ['bias * 1 - bias'], 'IBias

plot_metrics ()

plot_vs (*x_metric*, *y_metric*)

x_metric = 'thresholds' y_metric = 'fpr'

pn

predicted negative probability

pp

predicted positive probability

rn

real negative probability

rp

real positive probability

show_mcc ()

sqr_d_error

squared error

thresh

tn

true negative probability

tna

negative predictive value, inverse precision

tnr
true negative rate, inverse recall

tp
true positive probability

tpa
miss rate, false negative rate

tpr
sensitivity, recall, hit rate, tpr

wracc
weighted relative accuracy

```
vtool.confusion.draw_precision_recall_curve(recall_domain, p_interp, title_pref=None,
                                             fnum=1, pnum=None, color=None)
```

```
vtool.confusion.draw_roc_curve(fpr, tpr, fnum=None, pnum=None, marker='', target_tpr=None,
                               target_fpr=None, thresholds=None, color=None, name=None,
                               label=None, show_operating_point=False)
```

Parameters

- **fpr** –
- **tpr** –
- **fnum** (*int*) – figure number(default = None)
- **pnum** (*tuple*) – plot number(default = None)
- **marker** (*str*) – (default = 'x')
- **target_tpr** (*None*) – (default = None)
- **target_fpr** (*None*) – (default = None)
- **thresholds** (*None*) – (default = None)
- **color** (*None*) – (default = None)
- **show_operating_point** (*bool*) – (default = False)

CommandLine: `python -m vtool.confusion --exec-draw_roc_curve --show --lightbg`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.confusion import * # NOQA
>>> scores, labels = testdata_scores_labels()
>>> confusions = ConfusionMetrics().fit(scores, labels)
>>> fpr = confusions.fpr
>>> tpr = confusions.tpr
>>> thresholds = confusions.thresholds
>>> fnum = None
>>> pnum = None
>>> marker = 'x'
>>> target_tpr = .85
>>> target_fpr = None
>>> color = None
>>> show_operating_point = True
```

(continues on next page)

(continued from previous page)

```
>>> draw_roc_curve(fpr, tpr, fnum, pnum, marker, target_tpr, target_fpr,
>>>   thresholds, color, show_operating_point)
>>> ut.show_if_requested()
```

```
vtool.confusion.interact_roc_factory(confusions, target_tpr=None,
                                     show_operating_point=False)
```

Parameters `confusions` (*Confusions*) –

CommandLine: `python -m vtool.confusion --exec-interact_roc_factory --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.confusion import * # NOQA
>>> scores, labels = testdata_scores_labels()
>>> print('scores = %r' % (scores,))
>>> confusions = ConfusionMetrics().fit(scores, labels)
>>> print(ut.make_csv_table(
>>>   [confusions.fpr, confusions.tpr, confusions.thresholds],
>>>   ['fpr', 'tpr', 'thresh']))
>>> # xdoctest: +REQUIRES(--show)
>>> ROCInteraction = interact_roc_factory(confusions, target_tpr=.4, show_
↪operating_point=True)
>>> inter = ROCInteraction()
>>> inter.show_page()
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> ut.show_if_requested()
```

```
vtool.confusion.interpolate_precision_recall(precision, recall, nSamples=11)
```

Interpolates precision as a function of recall `p_{interp}(r)`

Reduce wiggles in average precision curve by taking interpolated values along a uniform sample.

References

http://en.wikipedia.org/wiki/Information_retrieval#Average_precision
http://en.wikipedia.org/wiki/Information_retrieval#Mean_Average_precision

http://en.wikipedia.org/wiki/Information_retrieval#Mean_Average_precision

CommandLine: `python -m vtool.confusion --test-interpolate_precision_recall --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.confusion import * # NOQA
>>> scores, labels = testdata_scores_labels()
>>> nSamples = 11
>>> confusions = ConfusionMetrics().fit(scores, labels)
>>> precision = confusions.precision
>>> recall = confusions.recall
>>> recall_domain, p_interp = interpolate_precision_recall(confusions.precision,
↪recall, nSamples=11)
```

(continues on next page)

(continued from previous page)

```

>>> result = ub.repr2(p_interp, precision=1, with_dtype=True)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> draw_precision_recall_curve(recall_domain, p_interp)
>>> ut.show_if_requested()
np.array([ 1. ,  1. ,  1. ,  1. ,  1. ,  1. ,  1. ,  0.9,  0.9,  0.8,  0.6],
      dtype=np.float64)

```

```

vtool.confusion.interpolate_replbounds(xdata, ydata, pt, maximize=True)
xdata = np.array([.1, .2, .3, .4, .5]) ydata = np.array([.1, .2, .3, .4, .5]) pt = .35

```

FIXME: if duplicate xdata is given bad things happen.

BUG: in `scipy.interpolate.interp1d` If there is a duplicate xdata, then `assume_sorted=False` will sort ydata by xdata, but xdata should retain its initial ordering in places of ambiguity. Currently it does not.

Parameters

- **xdata** (*ndarray*) –
- **ydata** (*ndarray*) –
- **pt** (*ndarray*) –

Returns *interp_vals*

Return type *float*

CommandLine: `python -m vtool.confusion --exec-interpolate_replbounds`

Example

```

>>> from vtool.confusion import * # NOQA
>>> scores, labels = testdata_scores_labels()
>>> self = ConfusionMetrics().fit(scores, labels)
>>> xdata = self.tpr
>>> ydata = self.thresholds
>>> pt = 1.0
>>> #xdata = self.fpr
>>> #ydata = self.thresholds
>>> #pt = 0.0
>>> thresh = interpolate_replbounds(xdata, ydata, pt, maximize=True)
>>> print('thresh = %r' % (thresh,))
>>> thresh = interpolate_replbounds(xdata, ydata, pt, maximize=False)
>>> print('thresh = %r' % (thresh,))

```

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.confusion import * # NOQA
>>> xdata = np.array([0.7, 0.8, 0.8, 0.9, 0.9, 0.9])
>>> ydata = np.array([34, 26, 23, 22, 19, 17])
>>> pt = np.array([.85, 1.0, -1.0])
>>> interp_vals = interpolate_replbounds(xdata, ydata, pt)
>>> result = ('interp_vals = %s' % (str(interp_vals),))

```

(continues on next page)

(continued from previous page)

```
>>> print(result)
interp_vals = [ 22.5  17.   34. ]
```

```
vtool.confusion.nan_to_num(arr, num)
```

```
vtool.confusion.testdata_scores_labels()
```

1.11 vtool.coverage_grid module

```
vtool.coverage_grid.compute_subbin_to_bins_dist(neighbor_bin_centers, subbin_xy_arr)
```

```
vtool.coverage_grid.get_coverage_grid_gridsearch_configs()
```

```
vtool.coverage_grid.get_subbin_xy_neighbors(subbin_index00, grid_steps, num_cols,
                                             num_rows)
```

Generate all neighbor of a bin subbin_index00 = left and up subbin index

```
vtool.coverage_grid.gridsearch_coverage_grid()
```

CommandLine: python -m vtool.coverage_grid -test-gridsearch_coverage_grid -show

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.coverage_grid import * # NOQA
>>> import wbia.plottool as pt
>>> gridsearch_coverage_grid()
>>> pt.show_if_requested()
```

```
vtool.coverage_grid.gridsearch_coverage_grid_mask()
```

CommandLine: python -m vtool.coverage_grid -test-gridsearch_coverage_grid_mask -show

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.coverage_grid import * # NOQA
>>> import wbia.plottool as pt
>>> gridsearch_coverage_grid_mask()
>>> pt.show_if_requested()
```

```
vtool.coverage_grid.make_grid_coverage_mask(kpts, chipsize, weights, pxl_per_bin=4,
                                             grid_steps=1, resize=False, out=None,
                                             grid_sigma=1.6)
```

Parameters

- **kpts** (`ndarray[float32_t, ndim=2]`) – keypoint
- **chipsize** (`tuple`) – width, height
- **weights** (`ndarray[float32_t, ndim=1]`) –
- **pxl_per_bin** (`float`) –
- **grid_steps** (`int`) –

Returns weightgrid

Return type ndarray

CommandLine: python -m vtool.coverage_grid -test-make_grid_coverage_mask -show

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.coverage_grid import * # NOQA
>>> import vtool as vt
>>> # build test data
>>> kpts, chipsize, weights = coverage_kpts.testdata_coverage('easy1.png')
>>> pxl_per_bin = 4
>>> grid_steps = 2
>>> # execute function
>>> weightgrid = make_grid_coverage_mask(kpts, chipsize, weights, pxl_per_bin,
↳ grid_steps)
>>> # verify result
>>> result = str(weightgrid)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(weightgrid)
>>> ut.show_if_requested()
```

`vtool.coverage_grid.show_coverage_grid(num_rows, num_cols, subbin_xy_arr, neighbor_bin_centers, neighbor_bin_weights, neighbor_bin_indices, fnum=None, pnum=None)`

visualizes the voting scheme on the grid. (not a mask, and no max)

`vtool.coverage_grid.sparse_grid_coverage(kpts, chipsize, weights, pxl_per_bin=0.3, grid_steps=1, grid_sigma=1.6)`

Parameters

- **kpts** (ndarray[float32_t, ndim=2]) – keypoint
- **chipsize** (tuple) –
- **weights** (ndarray) –

CommandLine: python -m vtool.coverage_grid -test-sparse_grid_coverage -show

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.coverage_grid import * # NOQA
>>> kpts, chipsize, weights = coverage_kpts.testdata_coverage()
>>> chipsize = (chipsize[0] + 50, chipsize[1])
>>> pxl_per_bin = 3
>>> grid_steps = 2
>>> grid_sigma = 1.6
>>> coverage_gridtuple = sparse_grid_coverage(kpts, chipsize, weights, pxl_per_bin,
↳ grid_steps, grid_sigma)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
```

(continues on next page)

(continued from previous page)

```
>>> show_coverage_grid(*coverage_gridtuple)
>>> pt.show_if_requested()
```

```
vtool.coverage_grid.weighted_gaussian_falloff(neighbor_subbin_sqrdist_arr, weights,
                                              grid_sigma)
```

1.12 vtool.coverage_kpts module

```
vtool.coverage_kpts.get_coverage_kpts_gridsearch_configs()
testing function
```

```
vtool.coverage_kpts.get_gaussian_weight_patch(gauss_shape=(19, 19),
                                              gauss_sigma_frac=0.3,
                                              gauss_norm_01=True)
```

2d gaussian image useful for plotting

Returns patch

Return type ndarray

CommandLine: python -m vtool.coverage_kpts --test-get_gaussian_weight_patch

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.coverage_kpts import * # NOQA
>>> patch = get_gaussian_weight_patch()
>>> result = str(patch)
>>> print(result)
```

```
vtool.coverage_kpts.gridsearch_kpts_coverage_mask()
testing function
```

CommandLine: python -m vtool.coverage_kpts --test-gridsearch_kpts_coverage_mask --show

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.coverage_kpts import * # NOQA
>>> import wbia.plottool as pt
>>> gridsearch_kpts_coverage_mask()
>>> pt.show_if_requested()
```

```
vtool.coverage_kpts.make_heatmask(mask, cmap='plasma')
```

```
vtool.coverage_kpts.make_kpts_coverage_mask(kpts, chipsize, weights=None, re-
                                             turn_patch=False, patch=None, re-
                                             size=False, out=None, cov_blur_on=True,
                                             cov_disk_hack=None, cov_blur_ksize=(17,
                                             17), cov_blur_sigma=5.0,
                                             cov_gauss_shape=(19, 19),
                                             cov_gauss_sigma_frac=0.3,
                                             cov_scale_factor=0.2,
                                             cov_agg_mode='max',
                                             cov_remove_shape=False,
                                             cov_remove_scale=False,
                                             cov_size_penalty_on=True,
                                             cov_size_penalty_power=0.5,
                                             cov_size_penalty_frac=0.1)
```

Returns a intensity image denoting which pixels are covered by the input keypoints

Parameters

- **kpts** (`ndarray[float32_t, ndim=2][ndims=2]`) – keypoints
- **chipsize** (`tuple`) – width height of the underlying image

Returns `dstimg, patch`

Return type `tuple` (`ndarray`, `ndarray`)

Example

```
>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> # xdoctest: +REQUIRES(module:wbia)
>>> from vtool.coverage_kpts import * # NOQA
>>> import vtool as vt
>>> import wbia.plottool as pt
>>> import pyhesaff
>>> img_fpath = ut.grab_test_imgpath('car1.jpg')
>>> (kpts, vecs) = pyhesaff.detect_feats(img_fpath)
>>> kpts = kpts[:10]
>>> chip = vt.imread(img_fpath)
>>> chipsize = chip.shape[0:2][::-1]
>>> # execute function
>>> dstimg, patch = make_kpts_coverage_mask(kpts, chipsize, resize=True, return_
↪ patch=True, cov_size_penalty_on=False, cov_blur_on=False)
>>> # show results
>>> # xdoctest: +REQUIRES(--show)
>>> mask = dstimg
>>> show_coverage_map(chip, mask, patch, kpts)
>>> pt.show_if_requested()
```

`vtool.coverage_kpts.make_kpts_heatmask(kpts, chipsize, cmap='plasma')`
makes a heatmap overlay for keypoints

CommandLine: `python -m vtool.coverage_kpts make_kpts_heatmask -show`

Example

```

>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> # xdoctest: +REQUIRES(module:wbia)
>>> from vtool.coverage_kpts import * # NOQA
>>> import vtool as vt
>>> import pyhesaff
>>> img_fpath = ut.grab_test_imgpath('car1.jpg')
>>> (kpts, vecs) = pyhesaff.detect_feats(img_fpath)
>>> chip = vt.imread(img_fpath)
>>> kpts = kpts[0:100]
>>> chipsize = chip.shape[0:2][::-1]
>>> heatmask = make_kpts_heatmask(kpts, chipsize)
>>> img1 = heatmask
>>> img2 = chip
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.qtenure()
>>> img3 = vt.overlay_alpha_images(heatmask, chip)
>>> pt.imshow(img3)
>>> #pt.imshow(heatmask)
>>> #pt.draw_kpts2(kpts)
>>> pt.show_if_requested()

```

`vtool.coverage_kpts.show_coverage_map` (*chip, mask, patch, kpts, fnum=None, ell_alpha=0.6, show_mask_kpts=False*)

testing function

`vtool.coverage_kpts.testdata_coverage` (*fname=None*)

testing function

`vtool.coverage_kpts.warp_patch_onto_kpts` (*kpts, patch, chipshape, weights=None, out=None, cov_scale_factor=0.2, cov_agg_mode='max', cov_remove_shape=False, cov_remove_scale=False, cov_size_penalty_on=True, cov_size_penalty_power=0.5, cov_size_penalty_frac=0.1*)

Overlays the source image onto a destination image in each keypoint location

Parameters

- **kpts** (*ndarray[float32_t, ndim=2]*) – keypoints
- **patch** (*ndarray*) – patch to warp (like gaussian)
- **chipshape** (*tuple*) –
- **weights** (*ndarray*) – score for every keypoint

Kwargs: `cov_scale_factor` (float):

Returns mask

Return type ndarray

CommandLine: `python -m vtool.coverage_kpts --test-warp_patch_onto_kpts python -m vtool.coverage_kpts --test-warp_patch_onto_kpts --show python -m vtool.coverage_kpts --test-warp_patch_onto_kpts --show --hole python -m vtool.coverage_kpts --test-warp_patch_onto_kpts --show --square python -m vtool.coverage_kpts --test-warp_patch_onto_kpts --show --square --hole`

Example

```

>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> # xdoctest: +REQUIRES(module:wbia)
>>> from vtool.coverage_kpts import * # NOQA
>>> import vtool as vt
>>> import pyhesaff
>>> img_fpath = ut.grab_test_imgpath('carl.jpg')
>>> (kpts, vecs) = pyhesaff.detect_feats(img_fpath)
>>> kpts = kpts[:15]
>>> chip = vt.imread(img_fpath)
>>> chipshape = chip.shape
>>> weights = np.ones(len(kpts))
>>> cov_scale_factor = 1.0
>>> srcshape = (19, 19)
>>> radius = srcshape[0] / 2.0
>>> sigma = 0.4 * radius
>>> SQUARE = ub.argflag('--square')
>>> HOLE = ub.argflag('--hole')
>>> if SQUARE:
>>>     patch = np.ones(srcshape)
>>> else:
>>>     patch = ptool.gaussian_patch(shape=srcshape, sigma=sigma) #, norm_
    ↪01=False)
>>>     patch = patch / patch.max()
>>> if HOLE:
>>>     patch[int(patch.shape[0] / 2), int(patch.shape[1] / 2)] = 0
>>> # execute function
>>> dstimg = warp_patch_onto_kpts(kpts, patch, chipshape, weights, cov_scale_
    ↪factor=cov_scale_factor)
>>> # verify results
>>> print('dstimg stats %r' % (ut.get_stats_str(dstimg, axis=None),))
>>> print('patch stats %r' % (ut.get_stats_str(patch, axis=None),))
>>> #print(patch.sum())
>>> assert np.all(ut.inbounds(dstimg, 0, 1, eq=True))
>>> # show results
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> mask = dstimg
>>> show_coverage_map(chip, mask, patch, kpts)
>>> pt.show_if_requested()

```

`vtool.coverage_kpts.warped_patch_generator(patch, dsize, affmat_list, weight_list, cov_size_penalty_on=True, cov_size_penalty_power=0.5, cov_size_penalty_frac=0.1)`

generator that warps the patches (like gaussian) onto an image with dsize using constant memory.

output must be used or copied on every iteration otherwise the next output will clobber the previous

References

http://docs.opencv.org/modules/imgproc/doc/geometric_transformations.html#warpaffine

1.13 vtool.demodata module

`vtool.demodata.dummy_img(w, h, intensity=200)`

Creates a demodata test image

`vtool.demodata.dummy_seed(seed=None)`

`vtool.demodata.force_kpts_feasibility(kpts, xys_nonneg=False)`

`vtool.demodata.get_dummy_dpts(num, dtype=<class 'numpy.uint8'>)`

Random SIFTish keypoints

`vtool.demodata.get_dummy_invV_mats(dtype=<class 'numpy.float32'>)`

`vtool.demodata.get_dummy_kpts(num=1, dtype=<class 'numpy.float32'>)`

Some testing data

Parameters

- **num** (*int*) – number of times to duplicate
- **dtype** (*type*) –

Returns kpts - keypoints

Return type ndarray[float32_t, ndim=2][ndims=2]

CommandLine: `xdoctest -m ~/code/vtool/vtool/demodata.py get_dummy_kpts`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.demodata import * # NOQA
>>> num = 1
>>> dtype = ktool.KPTS_DTYPE
>>> kpts = get_dummy_kpts(num, dtype)
>>> import ubelt as ub
>>> result = ub.repr2(kpts, precision=2, with_dtype=False)
```

`vtool.demodata.get_dummy_kpts_pair(wh_stride=(30, 30), wh_num=None)`

`vtool.demodata.get_dummy_matching_kpts(dtype=<class 'numpy.float32'>)`

`vtool.demodata.get_dummy_xy(seed=0)`

`vtool.demodata.get_kpts_dummy_img(kpts, sf=1.0, intensity=200)`

Parameters

- **kpts** (ndarray[float32_t, ndim=2]) – keypoints
- **sf** (*float*) –

Returns img

Return type tuple

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.demodata import * # NOQA
>>> kpts = get_dummy_kpts()
>>> sf = 1.0
>>> img = get_kpts_dummy_img(kpts, sf, 10)

```

`vtool.demodata.get_testdata_kpts` (*fname=None, with_vecs=False*)

`vtool.demodata.make_dummy_fm` (*nKpts*)

`vtool.demodata.perterb_kpts` (*kpts, xy_std=None, invV_std=None, ori_std=None, damping=None, seed=None, **kwargs*)

Adds normally distributed perturbations to keypoints

`vtool.demodata.perterbed_grid_kpts` (**args, **kwargs*)

`vtool.demodata.testdata_binary_scores` ()

`vtool.demodata.testdata_dummy_matches` ()

Returns `matches_testup`

Return type `tuple`

CommandLine: `python -m vtool.demodata -test-testdata_dummy_matches -show`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.demodata import * # NOQA
>>> matches_testup = testdata_dummy_matches()
>>> (kpts1, kpts2, fm, fs, rchip1, rchip2) = matches_testup
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.show_chipmatch2(rchip1, rchip2, kpts1, kpts2, fm, fs)
>>> pt.set_figtitle('Dummy matches')
>>> pt.show_if_requested()

```

`vtool.demodata.testdata_dummy_sift` (*nPts=10, asint=True, rng=None*)

Makes a demodata sift descriptor that has the uint8 * 512 hack like hesaff returns

Parameters `nPts` (*int*) – (default = 10)

CommandLine: `python -m vtool.demodata -test-testdata_dummy_sift`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.demodata import * # NOQA
>>> import vtool as vt
>>> nPts = 10
>>> rng = np.random.RandomState(0)
>>> sift = testdata_dummy_sift(nPts, rng)
>>> assert vt.check_sift_validity(sift), 'bad SIFT properties'
>>> #assert np.allclose((sift / 512) ** 2).sum(axis=1), 1, rtol=.01), 'bad SIFT_
↪property'
>>> #assert np.all(sift / 512 < .2), 'bad SIFT property'

```

```
vtool.demodata.testdata_nonmonotonic()
```

```
vtool.demodata.testdata_ratio_matches (fname1='easy1.png',          fname2='easy2.png',
                                         **kwargs)
```

Runs simple ratio-test matching between two images. Technically this is not demodata data.

Parameters

- **fname1** (*str*) –
- **fname2** (*str*) –

Returns matches_testtup

Return type tuple

CommandLine: python -m vtool.demodata -test-testdata_ratio_matches python -m vtool.demodata -test-testdata_ratio_matches -help python -m vtool.demodata -test-testdata_ratio_matches -show python -m vtool.demodata -test-testdata_ratio_matches -show -ratio_thresh=1.1 -rotation_invariance

python -m vtool.demodata -test-testdata_ratio_matches -show -ratio_thresh=.625 -rotation_invariance -fname1 easy1.png -fname2 easy3.png python -m vtool.demodata -test-testdata_ratio_matches -show -ratio_thresh=.625 -no-rotation_invariance -fname1 easy1.png -fname2 easy3.png

Example

```
>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.demodata import * # NOQA
>>> import vtool as vt
>>> fname1 = ut.get_argval('--fname1', type_=str, default='easy1.png')
>>> fname2 = ut.get_argval('--fname2', type_=str, default='easy2.png')
>>> default_dict = vt.get_extract_features_default_params()
>>> default_dict['ratio_thresh'] = .625
>>> kwargs = ut.parse_dict(default_dict)
>>> matches_testtup = testdata_ratio_matches(fname1, fname2, **kwargs)
>>> (kpts1, kpts2, fm_RAT, fs_RAT, rchip1, rchip2) = matches_testtup
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.show_chipmatch2(rchip1, rchip2, kpts1, kpts2, fm_RAT, fs_RAT, ori=True)
>>> num_matches = len(fm_RAT)
>>> score_sum = sum(fs_RAT)
>>> title = 'Simple matches using the Lowe\'s ratio test'
>>> title += '\n num_matches=%r, score_sum=%.2f' % (num_matches, score_sum)
>>> pt.set_figtitle(title)
>>> pt.show_if_requested()
```

1.14 vtool.deprecated module

```
class vtool.deprecated.ThumbnailCacheContext (uuid_list, asrgb=True, thumb_size=64,
                                              thumb_dpath=None, appname='vtool')
```

Bases: object

Lazy computation of images as thumbnails.

DEPRICATED

Just pass a list of uuids corresponding to the images. Then compute images flagged as dirty and give them back to the context. thumbs_list will be populated on context exit

filter_dirty_items (*list_*)

Returns only items marked by the context as dirty

save_dirty_thumbs_from_images (*img_list*)

Pass in any images marked by the context as dirty here

1.15 vtool.distance module

`vtool.distance.L1` (*hist1, hist2, dtype=<class 'numpy.float64'>*)

returns L1 (aka manhattan or grid) distance between two histograms

`vtool.distance.L2` (*hist1, hist2*)

returns L2 (aka euclidean or standard) distance between two histograms

`vtool.distance.L2_root_sift` (*hist1, hist2*)

Normalized Root-SIFT L2

Parameters

- **hist1** (*ndarray*) – Nx128 array of uint8 with pseudomax trick
- **hist2** (*ndarray*) – Nx128 array of uint8 with pseudomax trick

Returns euclidean distance between 0-1 normalized sift descriptors

Return type `ndarray`

`vtool.distance.L2_sift` (*hist1, hist2*)

Normalized SIFT L2

Parameters

- **hist1** (*ndarray*) – Nx128 array of uint8 with pseudomax trick
- **hist2** (*ndarray*) – Nx128 array of uint8 with pseudomax trick

Returns euclidean distance between 0-1 normalized sift descriptors

Return type `ndarray`

CommandLine: `python -m vtool.distance --test-L2_sift`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> hist1, hist2 = testdata_hist()
>>> sift1, sift2, sift3, sift4, sift5 = testdata_sift2()
>>> l2_dist = L2_sift(hist1, hist2)
>>> max_dist = L2_sift(sift4, sift5)
>>> assert np.isclose(max_dist, 1.0)
>>> result = ub.repr2(l2_dist, precision=2)
```

`vtool.distance.L2_sift_sqrd` (*hist1, hist2*)

Normalized SIFT L2**2

Parameters

- **hist1** (*ndarray*) – Nx128 array of uint8 with pseudomax trick

- **hist2** (*ndarray*) – Nx128 array of uint8 with pseudomax trick

Returns squared euclidean distance between 0-1 normalized sift descriptors

Return type *ndarray*

`vtool.distance.L2_sqrd(hist1, hist2, dtype=<class 'numpy.float64'>)`

returns the squared L2 distance

FIXME: if `hist1.shape = (0,)` and `hist.shape = (0,)` then `result=0.0`

SeeAlso: L2

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> import numpy
>>> ut.exec_func(L2_sqrd, globals())
>>> rng = np.random.RandomState(53)
>>> hist1 = rng.rand(5, 2)
>>> hist2 = rng.rand(5, 2)
>>> l2dist = L2_sqrd(hist1, hist2)
>>> result = ub.repr2(l2dist, precision=2, threshold=2)
```

`vtool.distance.bar_L2_sift(hist1, hist2)`

Normalized SIFT L2

Parameters

- **hist1** (*ndarray*) – Nx128 array of uint8 with pseudomax trick
- **hist2** (*ndarray*) – Nx128 array of uint8 with pseudomax trick

CommandLine: `python -m vtool.distance -test-bar_L2_sift`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> hist1, hist2 = testdata_hist()
>>> barl2_dist = bar_L2_sift(hist1, hist2)
>>> result = ub.repr2(barl2_dist, precision=2)
```

`vtool.distance.bar_cos_sift(hist1, hist2)`

1 - cos dist

`vtool.distance.closest_point(pt, pt_arr, distfunc=<function L2_sqrd>)`

finds the nearest point(s) in pts to (x, y) `pt = np.array([1])` `pt_arr = np.array([1.1, 2, .95, 20])[:, None]` `distfunc = vt.L2_sqrd`

`vtool.distance.compute_distances(hist1, hist2, dist_list=['L1', 'L2'])`

Parameters

- **hist1** (*ndarray*) –
- **hist2** (*ndarray*) –
- **dist_list** (*list*) – (default = ['L1', 'L2'])

Returns dist_dict

Return type dict

CommandLine: python -m vtool.distance --test-compute_distances

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> hist1 = np.array([[1, 2], [2, 1], [0, 0]])
>>> hist2 = np.array([[1, 2], [3, 1], [2, 2]])
>>> dist_list = ['L1', 'L2']
>>> dist_dict = compute_distances(hist1, hist2, dist_list)
>>> result = ub.repr2(dist_dict, precision=3)
>>> print(result)
```

vtool.distance.cos_sift(hist1, hist2)
cos dist

CommandLine: python -m vtool.distance --test-cos_sift

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> hist1, hist2 = testdata_hist()
>>> l2_dist = cos_sift(hist1, hist2)
```

vtool.distance.cosine_dist(hist1, hist2)

vtool.distance.cyclic_distance(arr1, arr2, modulo, out=None)
returns an unsigned distance

Parameters

- **arr1** (ndarray) –
- **arr2** (ndarray) –
- **modulo** (float or int) –
- **out** (ndarray) – (default = None)

Returns arr_dist

Return type ndarray

CommandLine: python -m vtool.distance cyclic_distance

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> out = None
>>> modulo = 8
```

(continues on next page)

(continued from previous page)

```
>>> offset = 0 # doesnt matter what offset is
>>> arr1 = np.hstack([np.arange(offset, modulo + offset), np.nan])
>>> arr2 = arr1[:, None]
>>> arr_dist = cyclic_distance(arr1, arr2, modulo, out)
>>> result = ('arr_dist =\n%s' % (ub.repr2(arr_dist),))
```

`vtool.distance.det_distance(det1, det2)`

Returns how far off determinants are from one another

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> rng = np.random.RandomState(53)
>>> det1 = rng.rand(5)
>>> det2 = rng.rand(5)
>>> scaledist = det_distance(det1, det2)
>>> result = ub.repr2(scaledist, precision=2, threshold=2)
```

`vtool.distance.emd(hist1, hist2, cost_matrix='sift')`

earth mover's distance by objects(lpSovle::lp.transport) require: lpsolve55-5.5.0.9.win32-py2.7.exe

CommandLine: python -m vtool.distance --test-emd

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> hist1, hist2 = testdata_hist()
>>> emd_dists = emd(hist1, hist2)
>>> result = ub.repr2(emd_dists, precision=2)
```

`vtool.distance.haversine(latlon1, latlon2)`

Calculate the great circle distance between two points on the earth (specified in decimal degrees)

Parameters

- **latlon1** (*ndarray*) –
- **latlon2** (*ndarray*) –

References

en.wikipedia.org/wiki/Haversine_formula gis.stackexchange.com/questions/81551/matching-gps-tracks
stackoverflow.com/questions/4913349/haversine-distance-gps-points

CommandLine: python -m vtool.distance --exec-haversine

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> import scipy.spatial.distance as spdist
>>> import vtool as vt
>>> import functools
>>> gpsarr_track_list_ = [
...     np.array([[ -80.21895315, -158.81099213],
...               [ -12.08338926,  67.50368014],
...               [ -11.08338926,  67.50368014],
...               [ -11.08338926,  67.50368014],]),
...     ],
...     np.array([[  9.77816711, -17.27471498],
...               [-51.67678814, -158.91065495],])
...     ]
>>> latlon1 = gpsarr_track_list_[0][0]
>>> latlon2 = gpsarr_track_list_[0][1]
>>> kilometers = vt.haversine(latlon1, latlon2)
>>> haversin_pdist = functools.partial(spdist.pdist, metric=vt.haversine)
>>> dist_vector_list = list(map(haversin_pdist, gpsarr_track_list_))
>>> dist_matrix_list = list(map(spdist.squareform, dist_vector_list))

```

`vtool.distance.hist_isect` (*hist1, hist2*)
returns histogram intersection distance between two histograms

`vtool.distance.nearest_point` (*x, y, pts, conflict_mode='next', __next_counter=[0]*)
finds the nearest point(s) in pts to (x, y)

TODO: deprecate

`vtool.distance.ori_distance` (*ori1, ori2, out=None*)
Returns the unsigned distance between two angles

References

<http://stackoverflow.com/questions/1878907/the-smallest-difference-between-2-angles>

CommandLine: `python -m vtool.distance --test-ori_distance`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> rng = np.random.RandomState(0)
>>> ori1 = (rng.rand(10) * TAU) - np.pi
>>> ori2 = (rng.rand(10) * TAU) - np.pi
>>> dist_ = ori_distance(ori1, ori2)
>>> result = ub.repr2(ori1, precision=1)
>>> result += '\n' + ub.repr2(ori2, precision=1)
>>> result += '\n' + ub.repr2(dist_, precision=1)
>>> print(result)

```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> ori1 = .3
>>> ori2 = 6.8
>>> dist_ = ori_distance(ori1, ori2)
>>> result = ub.repr2(dist_, precision=2)
>>> print(result)
```

`vtool.distance.pdist_argsort(x)`

Sorts 2d indices by their distance matrix output from `scipy.spatial.distance` `x = np.array([3.05555556e-03, 1.47619797e+04, 1.47619828e+04])`

Parameters `x` (`ndarray`) –

Returns `sortx_2d`

Return type `ndarray`

CommandLine: `python -m vtool.distance --test-pdist_argsort`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> x = np.array([ 21695.78, 10943.76, 10941.44, 25867.64, 10752.03,
>>>                10754.35, 4171.86, 2.32, 14923.89, 14926.2 ],
>>>                dtype=np.float64)
>>> sortx_2d = pdist_argsort(x)
>>> result = ('sortx_2d = %s' % (str(sortx_2d),))
>>> print(result)
sortx_2d = [(2, 3), (1, 4), (1, 2), (1, 3), (0, 3), (0, 2), (2, 4), (3, 4), (0, 1), (0, 4)]
```

`vtool.distance.pdist_indicies(num)`

`vtool.distance.safe_pdist(arr, *args, **kwargs)`

Kwargs: `metric = ut.absdiff`

SeeAlso: `scipy.spatial.distance.pdist`

`vtool.distance.signed_cyclic_distance(arr1, arr2, modulo, out=None)`

`vtool.distance.signed_ori_distance(ori1, ori2)`

Parameters

- `ori1` (`ndarray`) –
- `ori2` (`ndarray`) –

Returns `ori_dist`

Return type `ndarray`

CommandLine: `python -m vtool.distance --exec-signed_ori_distance`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> ori1 = np.array([0, 0, 3, 4, 0, 0])
>>> ori2 = np.array([3, 4, 0, 0, np.pi, np.pi - .1])
>>> ori_dist = signed_ori_distance(ori1, ori2)
>>> result = ('ori_dist = %s' % (ub.repr2(ori_dist, precision=3),))
```

`vtool.distance.testdata_hist()`

`vtool.distance.testdata_sift2()`

`vtool.distance.understanding_pseudomax_props(mode=2)`

Function showing some properties of distances between normalized pseudomax vectors

CommandLine: `python -m vtool.distance --test-understanding_pseudomax_props`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> for mode in [0, 1, 2, 3]:
...     print('+---')
...     print('mode = %r' % (mode,))
...     result = understanding_pseudomax_props(mode)
...     print('L___')
>>> print(result)
```

`vtool.distance.wrapped_distance(arr1, arr2, base, out=None)`

base = TAU corresponds to ori diff

1.16 vtool.ellipse module

OLD MODULE, needs reimplementatation of select features and deprication

This module should handle all things elliptical

`vtool.ellipse.adaptive_scale(img_fpath, kpts, nScales=4, low=-0.5, high=0.5, nSamples=16)`

`vtool.ellipse.check_kpts_in_bounds(kpts_, width, height)`

`vtool.ellipse.circular_distance(arr=None)`

`vtool.ellipse.expand_kpts(kpts, scales)`

`vtool.ellipse.expand_scales(kpts, nScales, low, high)`

`vtool.ellipse.expand_subscales(kpts, subscale_list)`

`vtool.ellipse.extrema_neighbors(extrema_list, nBins)`

`vtool.ellipse.find_maxima(y_list)`

`vtool.ellipse.find_maxima_with_neighbors(scalar_list)`

`vtool.ellipse.gradient_magnitude(img)`

`vtool.ellipse.homogenous_circle_pts(nSamples)`

Make a list of homogenous circle points

```

vtool.ellipse.interpolate_between(peak_list, nScales, high, low)
vtool.ellipse.interpolate_maxima(scalar_list)
vtool.ellipse.interpolate_peaks(x_data_list, y_data_list)
vtool.ellipse.interpolate_peaks2(x_data_list, y_data_list)
vtool.ellipse.kpts_matrices(kpts)
vtool.ellipse.sample_ell_border_pts(expanded_kpts, nSamples)
vtool.ellipse.sample_ell_border_vals(imgBGR, expanded_kpts, nKp, nScales, nSamples)
vtool.ellipse.sample_uniform(kpts, nSamples=128)

SeeAlso: python -m pyhesaff.tests.test_ellipse --test-in_depth_ellipse --show
vtool.ellipse.subscale_peaks(border_vals_sum, kpts, nScales, low, high)

```

1.17 vtool.exif module

References

<http://www.exiv2.org/tags.html>

Todo: <https://github.com/recurser/exif-orientation-examples>

```

vtool.exif.check_exif_keys(pil_img)
vtool.exif.convert_degrees(value)
    Helper function to convert the GPS coordinates stored in the EXIF to degrees in float format

```

References

http://en.wikipedia.org/wiki/Geographic_coordinate_conversion

```

vtool.exif.get_exif_dict(pil_img)
    Returns exif dictionary by TAGID
vtool.exif.get_exif_dict2(pil_img)
    Returns exif dictionary by TAG (less efficient)
vtool.exif.get_exif_tagids(tag_list)
vtool.exif.get_exist(data, key)
vtool.exif.get_lat_lon(exif_dict, default=(-1, -1))
    Returns the latitude and longitude, if available, from the provided exif_data2 (obtained through exif_data2
    above)

```

Notes

Might need to downgrade to Pillow 2.9.0 to solve a bug with getting GPS <https://github.com/python-pillow/Pillow/issues/1477>

```
python -c "from PIL import Image; print(Image.PILLOW_VERSION)"
```

pip uninstall Pillow pip install Pillow==2.9

CommandLine: python -m vtool.exif --test-get_lat_lon

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.exif import * # NOQA
>>> import numpy as np
>>> image_fpath = ut.grab_file_url('http://images.summitpost.org/original/769474.
↳JPG')
>>> pil_img = Image.open(image_fpath)
>>> exif_dict = get_exif_dict(pil_img)
>>> latlon = get_lat_lon(exif_dict)
>>> result = np.array_str(np.array(latlon), precision=3)
>>> print(result)
```

vtool.exif.get_orientation (exif_dict, default=0, on_error='warn')

Returns the image orientation, if available, from the provided exif_data2 (obtained through exif_data2 above)

CommandLine: python -m vtool.exif --test-get_orientation

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.exif import * # NOQA
>>> from os.path import join
>>> import numpy as np
>>> url = 'https://wildbookiarepository.azureedge.net/models/orientation.zip'
>>> images_path = ut.grab_zipped_url(url)
>>> result = []
>>> for index in range(3):
>>>     image_filename = 'orientation_%05d.JPG' % (index + 1, )
>>>     pil_img = Image.open(join(images_path, image_filename))
>>>     exif_dict = get_exif_dict(pil_img)
>>>     orient = get_orientation(exif_dict)
>>>     pil_img.close()
>>>     result.append(orient)
>>> print(result)
[1, 6, 8]
```

vtool.exif.get_orientation_str (exif_dict, **kwargs)

Returns the image orientation strings, if available, from the provided exif_data2 (obtained through exif_data2 above)

CommandLine: python -m vtool.exif --test-get_orientation_str

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.exif import * # NOQA
>>> from os.path import join
>>> import numpy as np
>>> url = 'https://wildbookiarepository.azureedge.net/models/orientation.zip'
```

(continues on next page)

(continued from previous page)

```

>>> images_path = ut.grab_zipped_url(url)
>>> result = []
>>> for index in range(3):
>>>     image_filename = 'orientation_%05d.JPG' % (index + 1, )
>>>     pil_img = Image.open(join(images_path, image_filename))
>>>     exif_dict = get_exif_dict(pil_img)
>>>     orient_str = get_orientation_str(exif_dict)
>>>     pil_img.close()
>>>     result.append(orient_str)
>>> print(result)
['Normal', '90 Clockwise', '90 Counter-Clockwise']

```

`vtool.exif.get_unixtime(exif_dict, default=-1)`
 TODO: Exif.Image.TimeZoneOffset

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.exif import * # NOQA
>>> image_fpath = ut.grab_file_url('http://images.summitpost.org/original/769474.
↳JPG')
>>> pil_img = Image.open(image_fpath)
>>> exif_dict = get_exif_dict(pil_img)
>>> pil_img.close()

```

`vtool.exif.get_unixtime_gps(exif_dict, default=-1)`

`vtool.exif.make_exif_dict_human_readable(exif_dict)`

`vtool.exif.parse_exif_unixtime(image_fpath)`

Parameters `image_fpath` (*str*) –

Returns unixtime

Return type `float`

CommandLine: `python -m vtool.exif --test-parse_exif_unixtime`

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.exif import * # NOQA
>>> image_fpath = ut.grab_file_url('http://images.summitpost.org/original/769474.
↳JPG')
>>> unixtime = parse_exif_unixtime(image_fpath)
>>> result = str(unixtime)
>>> print(result)

```

`vtool.exif.parse_exif_unixtime_gps(image_fpath)`

`vtool.exif.read_all_exif_tags(pil_img)`

`vtool.exif.read_exif(fpath, tag=None)`

`vtool.exif.read_exif_tags(pil_img, exif_tagid_list, default_list=None)`

```
vtool.exif.read_one_exif_tag(pil_img, tag)
```

1.18 vtool.features module

```
vtool.features.detect_opencv_keypoints()
```

```
vtool.features.extract_feature_from_patch(patch)
```

```
vtool.features.extract_features(img_or_fpath, feat_type='hesaff+sift', **kwargs)
```

calls pyhesaff's main driver function for detecting hessian affine keypoints. extra parameters can be passed to the hessian affine detector by using kwargs.

Parameters

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

Returns (kpts, vecs)

Return type tuple

CommandLine: `python -m vtool.features --test-extract_features` `python -m vtool.features --test-extract_features --show`

Example

```
>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.features import * # NOQA
>>> import vtool as vt
>>> # build test data
>>> img_fpath = ut.grab_test_imgpath(ut.get_argval('--fname', default='lena.png'))
>>> imgBGR = vt.imread(img_fpath)
>>> feat_type = ut.argval('--feat_type', default='hesaff+sift')
>>> import pyhesaff
>>> kwargs = ut.parse_dict_from_argv(pyhesaff.get_hesaff_default_params())
>>> # execute function
>>> #(kpts, vecs) = extract_features(img_fpath)
>>> (kpts, vecs) = extract_features(imgBGR, feat_type, **kwargs)
>>> # verify results
>>> result = str((kpts, vecs))
>>> print(result)
>>> # Show keypoints
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> #pt.figure(fnum=1, doclf=True, docla=True)
>>> #pt.imshow(imgBGR)
>>> #pt.draw_kpts2(kpts, ori=True)
>>> pt.interact_keypoints.imshow_keypoints(imgBGR, kpts, vecs, ori=True, ell_
↪alpha=.4, color='distinct')
>>> pt.show_if_requested()
```

```
vtool.features.get_extract_features_default_params()
```

Returns

Return type `dict`

CommandLine: `python -m vtool.features --test-get_extract_features_default_params`

Example

```
>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.features import * # NOQA
>>> # build test data
>>> # execute function
>>> param_dict = get_extract_features_default_params()
>>> result = ub.repr2(param_dict)
>>> # verify results
>>> print(result)
```

`vtool.features.test_mser()`

1.19 vtool.fontdemo module

References

<https://dbader.org/blog/monochrome-font-rendering-with-freetype-and-python-5488053#file-fontdemo-py-L244>

<https://gist.github.com/dbader/>

class `vtool.fontdemo.Bitmap` (*width, height, pixels=None*)

Bases: `object`

A 2D bitmap image represented as a list of byte values. Each byte indicates the state of a single pixel in the bitmap. A value of 0 indicates that the pixel is *off* and any other value indicates that it is *on*.

bitblt (*src, x, y*)

Copy all pixels from *src* into this bitmap

class `vtool.fontdemo.Font` (*filename, size*)

Bases: `object`

glyph_for_character (*char*)

kerning_offset (*previous_char, char*)

render_character (*char*)

render_text (*text, width=None, height=None, baseline=None*)

text_dimensions (*text*)

Return (width, height, baseline) of *text* rendered in the current font.

class `vtool.fontdemo.Glyph` (*pixels, width, height, top, advance_width*)

Bases: `object`

static from_glyphslot (*slot*)

Construct and return a Glyph object from a FreeType GlyphSlot.

height

static unpack_mono_bitmap (*bitmap*)

Unpack a freetype FT_LOAD_TARGET_MONO glyph bitmap into a bytearray where each pixel is represented by a single byte.

width

`vtool.fontdemo.font_demo()`

CommandLine: `python -m vtool.fontdemo font_demo --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.fontdemo import * # NOQA
>>> result = font_demo()
>>> import utool as ut
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> ut.show_if_requested()
```

`vtool.fontdemo.get_text_test_img(text)`

Parameters `text` (*str*) –

CommandLine: `python -m vtool.fontdemo get_text_test_img --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.fontdemo import * # NOQA
>>> import utool as ut
>>> text = 'A012'
>>> text_img = get_text_test_img(text)
>>> result = ('text_img = %s' % (ub.repr2(text_img),))
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(text_img)
>>> ut.show_if_requested()
```

1.20 vtool.geometry module

`vtool.geometry.bbox_center(bbox)`

`vtool.geometry.bbox_from_center_wh(center_xy, wh)`

`vtool.geometry.bbox_from_extent(extent)`

Parameters `extent` (*ndarray*) – tl_x, br_x, tl_y, br_y

Returns tl_x, tl_y, w, h

Return type `bbox` (*ndarray*)

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> import ubelt as ub
>>> extent = [0, 10, 0, 10]
>>> bbox = bbox_from_extent(extent)
>>> result = ('bbox = %s' % (ub.repr2(bbox, nl=0),))
>>> print(result)
bbox = [0, 0, 10, 10]
```

`vtool.geometry.bbox_from_verts` (*verts*, *castint=False*)

`vtool.geometry.bbox_from_xywh` (*xy*, *wh*, *xy_rel_pos*=[0, 0])
need to specify *xy_rel_pos* if *xy* is not in tl already

`vtool.geometry.bboxes_from_vert_list` (*verts_list*, *castint=False*)
Fit the bounding polygon inside a rectangle

`vtool.geometry.closest_point_on_bbox` (*p*, *bbox*)

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> p_list = np.array([[19, 7], [7, 14], [14, 11], [8, 7], [23, 21]], dtype=np.
↳float)
>>> bbox = np.array([10, 10, 10, 10], dtype=np.float)
>>> [closest_point_on_bbox(p, bbox) for p in p_list]
```

`vtool.geometry.closest_point_on_line` (*p*, *e1*, *e2*)

e1 and *e2* define two points on the line. Does not clip to the segment.

CommandLine: `python -m vtool.geometry closest_point_on_line --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> import vtool as vt
>>> verts = np.array([[ 21.83012702,  13.16987298],
>>>                    [ 16.83012702,  21.83012702],
>>>                    [  8.16987298,  16.83012702],
>>>                    [ 13.16987298,   8.16987298],
>>>                    [ 21.83012702,  13.16987298]])
>>> rng = np.random.RandomState(0)
>>> p_list = rng.rand(64, 2) * 20 + 5
>>> close_pts = []
>>> for p in p_list:
>>>     candidates = [closest_point_on_line(p, e1, e2) for e1, e2 in ut.
↳itertwo(verts)]
>>>     dists = np.array([vt.L2_sqrd(p, new_pt) for new_pt in candidates])
>>>     close_pts.append(candidates[dists.argmin()])
>>> close_pts = np.array(close_pts)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
```

(continues on next page)

(continued from previous page)

```
>>> pt.ensureqt()
>>> pt.plt.plot(p_list.T[0], p_list.T[1], 'ro', label='original point')
>>> pt.plt.plot(close_pts.T[0], close_pts.T[1], 'rx', label='closest point on_
↳shape')
>>> for x, y in list(zip(p_list, close_pts)):
>>>     z = np.array(list(zip(x, y)))
>>>     pt.plt.plot(z[0], z[1], 'r--')
>>> pt.plt.legend()
>>> pt.plt.plot(verts.T[0], verts.T[1], 'b-')
>>> pt.plt.xlim(0, 30)
>>> pt.plt.ylim(0, 30)
>>> pt.plt.axis('equal')
>>> ut.show_if_requested()
```

`vtool.geometry.closest_point_on_line_segment(p, e1, e2)`
 Finds the closet point from p on line segment (e1, e2)

Parameters

- **p** (*ndarray*) – and xy point
- **e1** (*ndarray*) – the first xy endpoint of the segment
- **e2** (*ndarray*) – the second xy endpoint of the segment

Returns `pt_on_seg` - the closest xy point on (e1, e2) from p

Return type `ndarray`

References

http://en.wikipedia.org/wiki/Distance_from_a_point_to_a_line <http://stackoverflow.com/questions/849211/shortest-distance-between-a-point-and-a-line-segment>

CommandLine: `python -m vtool.geometry --exec-closest_point_on_line_segment --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> import vtool as vt
>>> #bbox = np.array([10, 10, 10, 10], dtype=np.float)
>>> #verts_ = np.array(vt.verts_from_bbox(bbox, close=True))
>>> #R = vt.rotation_around_bbox_mat3x3(vt.TAU / 3, bbox)
>>> #verts = vt.transform_points_with_homography(R, verts_.T).T
>>> verts = np.array([[ 21.83012702,  13.16987298],
>>>                    [ 16.83012702,  21.83012702],
>>>                    [  8.16987298,  16.83012702],
>>>                    [ 13.16987298,  8.16987298],
>>>                    [ 21.83012702,  13.16987298]])
>>> rng = np.random.RandomState(0)
>>> p_list = rng.rand(64, 2) * 20 + 5
>>> close_pts = np.array([closest_point_on_vert_segments(p, verts) for p in p_
↳list])
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
```

(continues on next page)

(continued from previous page)

```

>>> pt.ensureqt()
>>> pt.plt.plot(p_list.T[0], p_list.T[1], 'ro', label='original point')
>>> pt.plt.plot(close_pts.T[0], close_pts.T[1], 'rx', label='closest point on_
↳shape')
>>> for x, y in list(zip(p_list, close_pts)):
>>>     z = np.array(list(zip(x, y)))
>>>     pt.plt.plot(z[0], z[1], 'r--')
>>> pt.plt.legend()
>>> pt.plt.plot(verts.T[0], verts.T[1], 'b-')
>>> pt.plt.xlim(0, 30)
>>> pt.plt.ylim(0, 30)
>>> pt.plt.axis('equal')
>>> ut.show_if_requested()

```

`vtool.geometry.closest_point_on_vert_segments(p, verts)`

`vtool.geometry.cvt_bbox_xywh_to_pt1pt2(xywh, sx=1.0, sy=1.0, round_=True)`

Converts bbox to thumb format with a scale factor

`vtool.geometry.distance_to_lineseg(p, e1, e2)`

`vtool.geometry.draw_border(img_in, color=(0, 128, 255), thickness=2, out=None)`

Parameters

- `img_in` (`ndarray[uint8_t, ndim=2]`) – image data
- `color` (`tuple`) – in bgr
- `thickness` (`int`) –
- `out` (`None`) –

CommandLine: `python -m vtool.geometry --test-draw_border --show`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> import vtool as vt
>>> img_in = vt.imread(ut.grab_test_imgpath('car1.jpg'))
>>> color = (0, 128, 255)
>>> thickness = 20
>>> out = None
>>> # xdoctest: +REQUIRES(module:plottool)
>>> img = draw_border(img_in, color, thickness, out)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(img)
>>> pt.show_if_requested()

```

`vtool.geometry.draw_verts(img_in, verts, color=(0, 128, 255), thickness=2, out=None)`

Parameters

- `img_in` –
- `verts` –
- `color` (`tuple`) –

- **thickness** (*int*) –

Returns img - image data

Return type ndarray[uint8_t, ndim=2]

CommandLine: python -m vtool.geometry --test-draw_verts --show python -m vtool.geometry --test-draw_verts:0 --show python -m vtool.geometry --test-draw_verts:1 --show

References

http://docs.opencv.org/modules/core/doc/drawing_functions.html#line

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> import vtool as vt
>>> # build test data
>>> img_in = vt.imread(ut.grab_test_imgpath('car1.jpg'))
>>> verts = ((10, 10), (10, 100), (100, 100), (100, 10))
>>> color = (0, 128, 255)
>>> thickness = 2
>>> # execute function
>>> out = None
>>> img = draw_verts(img_in, verts, color, thickness, out)
>>> assert img_in is not img
>>> assert out is not img
>>> assert out is not img_in
>>> # verify results
>>> # xdoctest: +REQUIRES(--show)
>>> pt.imshow(img)
>>> pt.show_if_requested()
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> import vtool as vt
>>> # build test data
>>> img_in = vt.imread(ut.grab_test_imgpath('car1.jpg'))
>>> verts = ((10, 10), (10, 100), (100, 100), (100, 10))
>>> color = (0, 128, 255)
>>> thickness = 2
>>> out = img_in
>>> # execute function
>>> img = draw_verts(img_in, verts, color, thickness, out)
>>> assert img_in is img, 'should be in place'
>>> assert out is img, 'should be in place'
>>> # verify results
```

(continues on next page)

(continued from previous page)

```
>>> # xdoctest: +REQUIRES (--show)
>>> pt.imshow(img)
>>> pt.show_if_requested()
```

```
out = img_in = np.zeros((500, 500, 3), dtype=np.uint8)
```

```
vtool.geometry.extent_from_bbox(bbox)
```

Parameters `bbox` (*ndarray*) – tl_x, tl_y, w, h

Returns tl_x, br_x, tl_y, br_y

Return type extent (*ndarray*)

CommandLine: xdoctest -m ~/code/vtool/vtool/geometry.py extent_from_bbox

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> import ubelt as ub
>>> bbox = [0, 0, 10, 10]
>>> extent = extent_from_bbox(bbox)
>>> result = ('extent = %s' % (ub.repr2(extent, nl=0),))
>>> print(result)
extent = [0, 10, 0, 10]
```

```
vtool.geometry.extent_from_verts(verts)
```

```
vtool.geometry.get_pointset_extent_wh(pts)
```

```
vtool.geometry.get_pointset_extents(pts)
```

```
vtool.geometry.point_inside_bbox(point, bbox)
```

Flags points that are strictly inside a bounding box. Points on the boundary are not considered inside.

Parameters

- **point** (*ndarray*) – one or more points to test (2xN)
- **bbox** (*tuple*) – a bounding box in (x, y, w, h) format

Returns True if the point is in the bbox

Return type *bool* or *ndarray*

CommandLine: python -m vtool.geometry point_inside_bbox --show

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> import ubelt as ub
>>> point = np.array([
>>>     [3, 2], [4, 1], [2, 3], [1, 1], [0, 0],
>>>     [4, 9.5], [9, 9.5], [7, 2], [7, 8], [9, 3]
>>> ]).T
```

(continues on next page)

(continued from previous page)

```

>>> bbox = (3, 2, 5, 7)
>>> flag = point_inside_bbox(point, bbox)
>>> flag = flag.astype(np.int)
>>> result = ('flag = %s' % (ub.repr2(flag),))
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> verts = np.array(verts_from_bbox(bbox, close=True))
>>> pt.plot(verts.T[0], verts.T[1], 'b-')
>>> pt.plot(point[0][flag], point[1][flag], 'go')
>>> pt.plot(point[0][~flag], point[1][~flag], 'rx')
>>> pt.plt.xlim(0, 10); pt.plt.ylim(0, 10)
>>> pt.show_if_requested()
flag = np.array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0])

```

`vtool.geometry.scale_bbox` (*bbox*, *sx*, *sy*=None)

`vtool.geometry.scale_extents` (*extents*, *sx*, *sy*=None)

Parameters *extent* (*ndarray*) – tl_x, br_x, tl_y, br_y

`vtool.geometry.scaled_verts_from_bbox` (*bbox*, *theta*, *sx*, *sy*)

Helps with drawing scaled bounding boxes on thumbnails

`vtool.geometry.scaled_verts_from_bbox_gen` (*bbox_list*, *theta_list*, *sx*=1, *sy*=1)

Helps with drawing scaled bounding boxes on thumbnails

Parameters

- **bbox_list** (*list*) – bboxes in x,y,w,h format
- **theta_list** (*list*) – rotation of bounding boxes
- **sx** (*float*) – x scale factor
- **sy** (*float*) – y scale factor

Yields: *new_verts* - vertices of scaled bounding box for every input

CommandLine: `python -m vtool.image --test-scaled_verts_from_bbox_gen`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> # build test data
>>> bbox_list = [(10, 10, 100, 100)]
>>> theta_list = [0]
>>> sx = .5
>>> sy = .5
>>> # execute function
>>> new_verts_list = list(scaled_verts_from_bbox_gen(bbox_list, theta_list, sx,
↪sy))
>>> result = str(new_verts_list)
>>> # verify results
>>> print(result)
[[[5, 5], [55, 5], [55, 55], [5, 55], [5, 5]]]

```

`vtool.geometry.union_extents` (*extents*)

`vtool.geometry.verts_from_bbox(bbox, close=False)`

Parameters

- **bbox** (*tuple*) – bounding box in the format (x, y, w, h)
- **close** (*bool*) – (default = False)

Returns `verts`

Return type `list`

CommandLine: `python -m vtool.geometry --test-verts_from_bbox`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> bbox = (10, 10, 50, 50)
>>> close = False
>>> verts = verts_from_bbox(bbox, close)
>>> result = ('verts = %s' % (str(verts),))
>>> print(result)
verts = ((10, 10), (60, 10), (60, 60), (10, 60))
```

`vtool.geometry.verts_list_from_bboxes_list(bboxes_list)`

Create a four-vertex polygon from the bounding rectangle

1.21 vtool.histogram module

`vtool.histogram.argsubextrema2(op, ydata, xdata=None, thresh_factor=None, normalize_x=True, flat=True)`

Determines approximate maxima values to subindex accuracy.

Parameters

- **ydata** (*ndarray*) – ydata, histogram frequencies
- **xdata** (*ndarray*) – xdata, histogram labels
- **thresh_factor** (*float*) – cutoff point for labeling a value as a maxima
- **flat** (*bool*) – if True allows for flat extrema to be found.

Returns (submaxima_x, submaxima_y)

Return type `tuple`

CommandLine: `python -m vtool.histogram argsubmaxima python -m vtool.histogram argsubmaxima --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> thresh_factor = .8
>>> ydata = np.array([6.73, 8.69, 0.00, 0.00, 34.62, 29.16, 0.00, 0.00, 6.73, 8.
↪ 69])
```

(continues on next page)

(continued from previous page)

```

>>> xdata = np.array([-0.39, 0.39, 1.18, 1.96, 2.75, 3.53, 4.32, 5.11, 5.89, 6.
↳68])
>>> op = 'min'
>>> (subextrema_x, subextrema_y) = argsubextrema2(op, ydata, xdata, thresh_factor)
>>> result = str((subextrema_x, subextrema_y))
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.draw_hist_subbin_maxima(ydata, xdata)
>>> pt.show_if_requested()

```

Doctest:

```

>>> from vtool.histogram import * # NOQA
>>> thresh_factor = .8
>>> ydata = np.array([1, 1, 1, 2, 1, 2, 3, 2, 4, 1.1, 5, 1.2, 1.1, 1.1, 1.2,
↳1.1])
>>> op = 'max'
>>> thresh_factor = .8
>>> (subextrema_x, subextrema_y) = argsubextrema2(op, ydata, thresh_
↳factor=thresh_factor)
>>> result = str((subextrema_x, subextrema_y))
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.qensure()
>>> xdata = np.arange(len(ydata))
>>> pt.figure(fnum=1, doclf=True)
>>> pt.plot(xdata, ydata)
>>> pt.plot(subextrema_x, subextrema_y, 'o')
>>> ut.show_if_requested()

```

`vtool.histogram.argsubmax(ydata, xdata=None)`

Finds a single submaximum value to subindex accuracy. If `xdata` is not specified, `submax_x` is a fractional index. Otherwise, `submax_x` is sub-`xdata` (essentially doing the index interpolation for you)

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> import ubelt as ub
>>> ydata = [ 0, 1, 2, 1.5, 0]
>>> xdata = [00, 10, 20, 30, 40]
>>> result1 = argsubmax(ydata, xdata=None)
>>> result2 = argsubmax(ydata, xdata=xdata)
>>> result = ub.repr2([result1, result2], precision=4, nl=1, nobr=True)
>>> print(result)
2.1667, 2.0208,
21.6667, 2.0208,

```

Example

```
>>> from vtool.histogram import * # NOQA
>>> hist_ = np.array([0, 1, 2, 3, 4])
>>> centers = None
>>> maxima_thresh=None
>>> argsubmax(hist_)
(4.0, 4.0)
```

`vtool.histogram.argsubmax2` (*ydata*, *xdata=None*)

Finds a single submaximum value to subindex accuracy. If *xdata* is not specified, *submax_x* is a fractional index. This version always normalizes x-coordinates.

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> import ubelt as ub
>>> ydata = [ 0, 1, 2, 1.5, 0]
>>> xdata = [00, 10, 20, 30, 40]
>>> result1 = argsubmax(ydata, xdata=None)
>>> result2 = argsubmax(ydata, xdata=xdata)
>>> result = ub.repr2([result1, result2], precision=4, nl=1, nobr=True)
>>> print(result)
2.1667, 2.0208,
21.6667, 2.0208,
```

Example

```
>>> from vtool.histogram import * # NOQA
>>> hist_ = np.array([0, 1, 2, 3, 4])
>>> centers = None
>>> thresh_factor = None
>>> argsubmax(hist_)
(4.0, 4.0)
```

`vtool.histogram.argsubmaxima` (*hist*, *centers=None*, *maxima_thresh=None*, *_debug=False*)

Determines approximate maxima values to subindex accuracy.

Parameters

- **hist_** (*ndarray*) – ydata, histogram frequencies
- **centers** (*ndarray*) – xdata, histogram labels
- **maxima_thresh** (*float*) – cutoff point for labeling a value as a maxima

Returns (submaxima_x, submaxima_y)

Return type `tuple`

CommandLine: `python -m vtool.histogram argsubmaxima python -m vtool.histogram argsubmaxima --show`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> maxima_thresh = .8
>>> hist = np.array([6.73, 8.69, 0.00, 0.00, 34.62, 29.16, 0.00, 0.00, 6.73, 8.
↪69])
>>> centers = np.array([-0.39, 0.39, 1.18, 1.96, 2.75, 3.53, 4.32, 5.11, 5.89,
↪6.68])
>>> (submaxima_x, submaxima_y) = argsubmaxima(hist, centers, maxima_thresh)
>>> result = str((submaxima_x, submaxima_y))
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.draw_hist_subbin_maxima(hist, centers)
>>> pt.show_if_requested()
(array([ 3.0318792]), array([ 37.19208239]))

```

`vtool.histogram.argsubmaxima2(ydata, xdata=None, thresh_factor=None, normalize_x=True)`

`vtool.histogram.argsubmin2(ydata, xdata=None)`

`vtool.histogram.argsubminima2(ydata, xdata=None, thresh_factor=None, normalize_x=True)`

`vtool.histogram.get_histinfo_str(hist, edges)`

`vtool.histogram.hist_argmaxima(hist, centers=None, maxima_thresh=None)`

must take positive only values

CommandLine: `python -m vtool.histogram hist_argmaxima`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> maxima_thresh = .8
>>> hist = np.array([ 6.73, 8.69, 0.00, 0.00, 34.62, 29.16, 0.00, 0.00, 6.73,
↪8.69])
>>> centers = np.array([-0.39, 0.39, 1.18, 1.96, 2.75, 3.53, 4.32, 5.11, 5.89,
↪6.68])
>>> maxima_x, maxima_y, argmaxima = hist_argmaxima(hist, centers)
>>> result = str((maxima_x, maxima_y, argmaxima))
>>> print(result)

```

`vtool.histogram.hist_argmaxima2(hist, maxima_thresh=0.8)`

must take positive only values

Setup:

```

>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA

```

GridSearch:

```

>>> hist1 = np.array([1, .9, .8, .99, .99, 1.1, .9, 1.0, 1.0])
>>> hist2 = np.array([1, .9, .8, .99, .99, 1.1, 1.0, 1.0])
>>> hist2 = np.array([1, .9, .8, .99, .99, 1.1, 1.0])
>>> hist2 = np.array([1, .9, .8, .99, .99, 1.1, 1.2])

```

(continues on next page)

(continued from previous page)

```
>>> hist2 = np.array([1, 1.2])
>>> hist2 = np.array([1, 1, 1.2])
>>> hist2 = np.array([1])
>>> hist2 = np.array([])
```

Example

```
>>> # ENABLE_DOCTEST
>>> maxima_thresh = .8
>>> hist = np.array([1, .9, .8, .99, .99, 1.1, .9, 1.0, 1.0])
>>> argmaxima = hist_argmaxima2(hist)
>>> print(argmaxima)
```

vtool.histogram.**hist_edges_to_centers**(edges)

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> edges = [-0.79, 0.00, 0.79, 1.57, 2.36, 3.14, 3.93, 4.71, 5.50, 6.28, 7.07]
>>> centers = hist_edges_to_centers(edges)
>>> result = str(centers)
>>> print(result)
[-0.395  0.395  1.18   1.965  2.75   3.535  4.32   5.105  5.89   6.675]
```

vtool.histogram.**interpolate_submaxima**(argmaxima, hist_, centers=None)

Parameters

- **argmaxima** (*ndarray*) – indices into ydata / centers that are argmaxima
- **hist_** (*ndarray*) – ydata, histogram frequencies
- **centers** (*ndarray*) – xdata, histogram labels

FIXME: what happens when `argmaxima[i] == len(hist_)`

CommandLine: `python -m vtool.histogram --test-interpolate_submaxima --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> import ubelt as ub
>>> argmaxima = np.array([1, 4, 7])
>>> hist_ = np.array([ 6.73, 8.69, 0.00, 0.00, 34.62, 29.16, 0.00, 0.00, 6.73,
↳ 8.69])
>>> centers = np.array([-0.39, 0.39, 1.18, 1.96, 2.75, 3.53, 4.32, 5.11, 5.89,
↳ 6.68])
>>> submaxima_x, submaxima_y = interpolate_submaxima(argmaxima, hist_, centers)
>>> locals_ = ut.exec_func_src(interpolate_submaxima,
>>>                             key_list=['x123', 'y123', 'coeff_list'])
>>> x123, y123, coeff_list = locals_
```

(continues on next page)

(continued from previous page)

```

>>> res = (submaxima_x, submaxima_y)
>>> result = ub.repr2(res, nl=1, nobr=True, precision=2, with_dtype=True)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.ensureqt()
>>> pt.figure(fnum=pt.ensure_fnum(None))
>>> pt.plot(centers, hist_, '-')
>>> pt.plot(centers[argmaxima], hist_[argmaxima], 'o', label='argmaxima')
>>> pt.plot(submaxima_x, submaxima_y, 'b*', markersize=20, label='interp maxima')
>>> # Extract parabola points
>>> pt.plt.plot(xl23, yl23, 'o', label='maxima neighbors')
>>> xpoints = [np.linspace(x1, x3, 50) for (x1, x2, x3) in xl23.T]
>>> ypoints = [np.polyval(coeff, x_pts) for x_pts, coeff in zip(xpoints, coeff_
    ↪list)]
>>> # Draw Submax Parabola
>>> for x_pts, y_pts in zip(xpoints, ypoints):
>>>     pt.plt.plot(x_pts, y_pts, 'g--', lw=2)
>>> pt.show_if_requested()
np.array([ 0.15,  3.03,  5.11], dtype=np.float64),
np.array([ 9.2 , 37.19,  0. ], dtype=np.float64),

```

Example

```

>>> hist_ = np.array([5])
>>> argmaxima = [0]

```

`vtool.histogram.interpolated_histogram`(data, weights, range_, bins, interpolation_wrap=True, _debug=False)

Follows `np.histogram`, but does interpolation

Parameters

- **data** (*ndarray*) –
- **weights** (*ndarray*) –
- **range** (*tuple*) – range from 0 to 1
- **bins** (*int*) –
- **interpolation_wrap** (*bool*) – (default = True)
- **_debug** (*bool*) – (default = False)

CommandLine: `python -m vtool.histogram --test-interpolated_histogram`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> data = np.array([ 0,  1,  2,  3.5,  3,  3,  4,  4])
>>> weights = np.array([1., 1., 1., 1., 1., 1., 1., 1.])
>>> range_ = (0, 4)
>>> bins = 5
>>> interpolation_wrap = False

```

(continues on next page)

(continued from previous page)

```

>>> hist, edges = interpolated_histogram(data, weights, range_, bins,
>>>                                     interpolation_wrap)
>>> assert np.abs(hist.sum() - weights.sum()) < 1E-9
>>> assert hist.size == bins
>>> assert edges.size == bins + 1
>>> result = get_histinfo_str(hist, edges)
>>> print(result)

```

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> data = np.array([ 0, 1, 2, 3.5, 3, 3, 4, 4])
>>> weights = np.array([4.5, 1., 1., 1., 1., 1., 1., 1.])
>>> range_ = (-.5, 4.5)
>>> bins = 5
>>> interpolation_wrap = True
>>> hist, edges = interpolated_histogram(data, weights, range_, bins,
>>>                                     interpolation_wrap)
>>> assert np.abs(hist.sum() - weights.sum()) < 1E-9
>>> assert hist.size == bins
>>> assert edges.size == bins + 1
>>> result = get_histinfo_str(hist, edges)
>>> print(result)

```

`vtool.histogram.linear_interpolation(arr, subindices)`
 Does linear interpolation to lookup subindex values

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> arr = np.array([0, 1, 2, 3])
>>> subindices = np.array([0, .1, 1, 1.8, 2, 2.5, 3] )
>>> subvalues = linear_interpolation(arr, subindices)
>>> assert np.allclose(subindices, subvalues)
>>> assert np.allclose(2.3, linear_interpolation(arr, 2.3))

```

`vtool.histogram.maxima_neighbors(argmaxima, hist_, centers=None)`

`vtool.histogram.maximum_parabola_point(A, B, C)`
 Maximum x point is where the derivative is 0

`vtool.histogram.show_hist_submaxima(hist_, edges=None, centers=None, maxima_thresh=0.8, pnum=(1, 1, 1))`

For C++ to show data

Parameters

- **hist** –
- **edges** (*None*) –
- **centers** (*None*) –

CommandLine: python -m vtool.histogram --test-show_hist_submaxima --show python -m pyhesaff._pyhesaff --test-test_rot_invar --show python -m vtool.histogram --test-show_hist_submaxima -dpath figures --save ~/latex/crall-candidacy-2015/figures/show_hist_submaxima.jpg

Example

```
>>> # xdoctest: +REQUIRES(module:wbia)
>>> import wbia.plottool as pt
>>> from vtool.histogram import * # NOQA
>>> hist_ = np.array(list(map(float, ut.get_argval('--hist', type_=list,
↳ default=[1, 4, 2, 5, 3, 3]))))
>>> edges = np.array(list(map(float, ut.get_argval('--edges', type_=list,
↳ default=[0, 1, 2, 3, 4, 5, 6]))))
>>> maxima_thresh = ut.get_argval('--maxima_thresh', type_=float, default=.8)
>>> centers = None
>>> show_hist_submaxima(hist_, edges, centers, maxima_thresh)
>>> pt.show_if_requested()
```

vtool.histogram.**show_ori_image**(gori, weights, patch, gradx=None, grady=None, gauss=None, fnum=None)

CommandLine: python -m pyhesaff._pyhesaff --test-test_rot_invar --show --nocpp

vtool.histogram.**show_ori_image_ondisk**()

CommandLine: python -m vtool.histogram --test-show_ori_image_ondisk --show

```
python -m vtool.histogram --test-show_ori_image_ondisk --show --patch_img_fpath
patches/KP_0_PATCH.png --ori_img_fpath patches/KP_0_orientations01.png --weights_img_fpath
patches/KP_0_WEIGHTS.png --grady_img_fpath patches/KP_0_ygradient.png --gradx_img_fpath
patches/KP_0_xgradient.png --title cpp_show_ori_ondisk
```

```
python -m pyhesaff._pyhesaff --test-test_rot_invar --show --rebuild-hesaff --no-rmbuild
```

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> import wbia.plottool as pt
>>> import vtool as vt
>>> result = show_ori_image_ondisk()
>>> pt.show_if_requested()
```

vtool.histogram.**subbin_bounds**(z, radius, low, high)

Gets quantized bounds of a sub-bin/pixel point and a radius. Useful for cropping using subpixel points

Parameters

- **z** (*float*) – center of a circle 1d pixel array
- **radius** (*float*) – radius of the circle
- **low** (*int*) – minimum index of 1d pixel array
- **high** (*int*) – maximum index of 1d pixel array

Returns

(iz1, iz2, z_offst) - quantized_bounds and subbin_offset iz1 - low radius endpoint iz2 - high radius endpoint z_offst - subpixel offset #Returns: quantized_bounds=(iz1, iz2), subbin_offset

Return type tuple

CommandLine: python -m vtool.histogram --test-subbin_bounds

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> z = 1.5
>>> radius = 5.666
>>> low = 0
>>> high = 7
>>> (iz1, iz2, z_offst) = subbin_bounds(z, radius, low, high)
>>> result = str((iz1, iz2, z_offst))
>>> print(result)
(0, 7, 1.5)
```

`vtool.histogram.wrap_histogram(hist_, edges_, _debug=False)`

Simulates the first and last histogram bin being adjacent to one another by replicating those bins at the last and first positions respectively.

Parameters

- **hist** (ndarray) –
- **edges** (ndarray) –

Returns (hist_wrap, edge_wrap)

Return type tuple

CommandLine: python -m vtool.histogram --test-wrap_histogram

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> import ubelt as ub
>>> hist_ = np.array([8., 0., 0., 34.32, 29.45, 0., 0., 6.73])
>>> edges_ = np.array([ 0., 0.78539816, 1.57079633,
...                    2.35619449, 3.14159265, 3.92699081,
...                    4.71238898, 5.49778714, 6.2831853 ])
>>> (hist_wrap, edge_wrap) = wrap_histogram(hist_, edges_)
>>> tup = (hist_wrap.tolist(), edge_wrap.tolist())
>>> result = ub.repr2(tup, nl=1, nobr=True, precision=2)
>>> print(result)
6.73, 8.00, 0.00, 0.00, 34.32, 29.45, 0.00, 0.00, 6.73, 8.00,
-0.79, 0.00, 0.79, 1.57, 2.36, 3.14, 3.93, 4.71, 5.50, 6.28, 7.07,
```

1.22 vtool.image module

`vtool.image.affine_warp_around_center` (*img*, *sx=1*, *sy=1*, *theta=0*, *shear=0*, *tx=0*, *ty=0*, *dsize=None*, *borderMode=0*, *flags=4*, *out=None*, ***kwargs*)

CommandLine: `python -m vtool.image -test-affine_warp_around_center -show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> img_fpath = ut.grab_test_imgpath('car1.jpg')
>>> img = vt.imread(img_fpath) / 255.0
>>> img = img.astype(np.float32)
>>> dsize = (1000, 1000)
>>> shear = .2
>>> theta = np.pi / 4
>>> tx = 0
>>> ty = 100
>>> sx = 1.5
>>> sy = 1.0
>>> borderMode = cv2.BORDER_CONSTANT
>>> flags = cv2.INTER_LANCZOS4
>>> img_warped = affine_warp_around_center(img, sx=sx, sy=sy,
...     theta=theta, shear=shear, tx=tx, ty=ty, dsize=dsize,
...     borderMode=borderMode, flags=flags, borderValue=(.5, .5, .5))
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow((img_warped * 255.0).astype(np.uint8))
>>> ut.show_if_requested()
```

`vtool.image.clipwhite` (*img*)
Strips white borders off an image

`vtool.image.clipwhite_ondisk` (*fpath_in*, *fpath_out=None*, *verbose=True*)
Strips white borders off an image on disk

Parameters

- **fpath_in** (*str*) –
- **fpath_out** (*None*) – (default = None)
- **verbose** (*bool*) – verbosity flag (default = True)

Returns *fpath_out*

Return type *str*

CommandLine: `python -m vtool.image clipwhite_ondisk`

`vtool.image.combine_offset_lists` (*offsets_list*, *sfs_list*, *offset_tups*, *sf_tups*)
Helper for stacking

`vtool.image.convert_colorspace` (*img*, *colorspace*, *src_colorspace='BGR'*)
Converts colorspace of *img*. Convenience function around `cv2.cvtColor`

Parameters

- **img** (`ndarray[uint8_t, ndim=2]`) – image data
- **colorspace** (`str`) – RGB, LAB, etc
- **src_colorspace** (`unicode`) – (default = u'BGR')

Returns img - image data

Return type `ndarray[uint8_t, ndim=2]`

CommandLine: `python -m vtool.image convert_colorspace --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> img_fpath = ut.grab_test_imgpath('zebra.png')
>>> img_fpath = ut.grab_file_url('http://itsnasb.com/wp-content/uploads/2013/03/
↳ lisa-frank-logol.jpg')
>>> img_fpath = ut.grab_test_imgpath('carl.jpg')
>>> img = vt.imread(img_fpath)
>>> img_float = vt.rectify_to_float01(img, np.float32)
>>> colorspace = 'LAB'
>>> src_colorspace = 'BGR'
>>> imgLAB = convert_colorspace(img, colorspace, src_colorspace)
>>> imgL = imgLAB[:, :, 0]
>>> fillL = imgL.mean()
>>> fillAB = 0 if ut.is_float(img) else 128
>>> imgAB_LAB = vt.embed_channels(imgLAB[:, :, 1:3], (1, 2), fill=fillL)
>>> imgA_LAB = vt.embed_channels(imgLAB[:, :, 1], (1,), fill=(fillL, fillAB))
>>> imgB_LAB = vt.embed_channels(imgLAB[:, :, 2], (2,), fill=(fillL, fillAB))
>>> imgAB_BGR = convert_colorspace(imgAB_LAB, src_colorspace, colorspace)
>>> imgA_BGR = convert_colorspace(imgA_LAB, src_colorspace, colorspace)
>>> imgB_BGR = convert_colorspace(imgB_LAB, src_colorspace, colorspace)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> #imgAB_HSV = convert_colorspace(convert_colorspace(imgAB_LAB, 'LAB', 'BGR'),
↳ 'BGR', 'HSV')
>>> imgAB_HSV = convert_colorspace(img, 'HSV', 'BGR')
>>> imgAB_HSV[:, :, 1:3] = .6 if ut.is_float(img) else 128
>>> imgCOLOR_BRG = convert_colorspace(imgAB_HSV, 'BGR', 'HSV')
>>> pt.imshow(img, pnum=(3, 4, 1), title='input')
>>> pt.imshow(imgL, pnum=(3, 4, 2), title='L (lightness)')
>>> pt.imshow((imgLAB[:, :, 1]), pnum=(3, 4, 3), title='A (grayscale)')
>>> pt.imshow((imgLAB[:, :, 2]), pnum=(3, 4, 4), title='B (grayscale)')
>>> pt.imshow(imgCOLOR_BRG, pnum=(3, 4, 5), title='Hue')
>>> pt.imshow(imgAB_BGR, pnum=(3, 4, 6), title='A+B (color overlay)')
>>> pt.imshow(imgA_BGR, pnum=(3, 4, 7), title='A (Red-Green)')
>>> pt.imshow(imgB_BGR, pnum=(3, 4, 8), title='B (Blue-Yellow)')
>>> rgblind_LAB = vt.embed_channels(imgLAB[:, :, (0, 2)], (0, 2), fill=fillAB)
>>> rgblind_BRG = convert_colorspace(rgblind_LAB, src_colorspace, colorspace)
>>> byblind_LAB = vt.embed_channels(imgLAB[:, :, (0, 1)], (0, 1), fill=fillAB)
>>> byblind_BGR = convert_colorspace(byblind_LAB, src_colorspace, colorspace)
>>> pt.imshow(byblind_BGR, title='colorblind B-Y', pnum=(3, 4, 11))
>>> pt.imshow(rgblind_BRG, title='colorblind R-G', pnum=(3, 4, 12))
>>> ut.show_if_requested()
```

`vtool.image.convert_image_list_colorspace` (*image_list*, *colorspace*,
src_colorspace='BGR')
 converts a list of images from <src_colorspace> to <colorspace>

vttool.image.crop_out_imgfill (*img, fillval=None, thresh=0, channel=None*)
Crops image to remove fillval

Parameters

- **img** (*ndarray*[*uint8_t*, *ndim*=2]) – image data
- **fillval** (*None*) – (default = None)
- **thresh** (*int*) – (default = 0)

Returns cropped_img

Return type ndarray

CommandLine: python -m vtool.image -exec-crop_out_imgfill

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> img = vt.get_stripe_patch()
>>> img = (img * 255).astype(np.uint8)
>>> print(img)
>>> img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
>>> fillval = np.array([25, 25, 25])
>>> thresh = 0
>>> cropped_img = crop_out_imgfill(img, fillval, thresh)
>>> cropped_img2 = cv2.cvtColor(cropped_img, cv2.COLOR_RGB2GRAY)
>>> result = ('cropped_img2 = \n%s' % (str(cropped_img2),))
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> img = vt.get_stripe_patch()
>>> img = (img * 255).astype(np.uint8)
>>> print(img)
>>> fillval = 25
>>> thresh = 0
>>> cropped_img = crop_out_imgfill(img, fillval, thresh)
>>> result = ('cropped_img = \n%s' % (str(cropped_img),))
>>> print(result)
```

```
vtool.image.cvt_BGR2L (imgBGR)
```

```
vtool.image.cvt_BGR2RGB (imgBGR)
```

```
vtool.image.draw_text(img, text, org, textcolor_rgb=[0, 0, 0], fontScale=1, thickness=2, fontFace=0,  
lineType=16, bottomLeftOrigin=False)
```

CommandLine: python -m vtool.image -test-draw_text:0 -show python -m vtool.image -test-draw_text:1 -show

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> font_names = sorted([key for key in cv2.__dict__.keys() if key.startswith(
↳ 'FONT_H')])
>>> text = 'opencv'
>>> img = np.zeros((400, 1024), dtype=np.uint8)
>>> thickness = 2
>>> fontScale = 1.0
>>> lineType = 4
>>> lineType = 8
>>> lineType = cv2.CV_AA
>>> for count, font_name in enumerate(font_names, start=1):
>>>     print(font_name)
>>>     fontFace = cv2.__dict__[font_name]
>>>     org = (10, count * 45)
>>>     text = 'opencv - ' + font_name
>>>     vt.draw_text(img, text, org,
...                 fontFace=fontFace, textcolor_rgb=[255, 255, 255],
...                 fontScale=fontScale, thickness=thickness)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(img)
>>> ut.show_if_requested()
```

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> font_names = sorted([key for key in cv2.__dict__.keys() if key.startswith(
↳ 'FONT_H')])
>>> text = 'opencv'
>>> img = np.zeros((400, 1024, 3), dtype=np.uint8)
>>> img[:200, :512, 0] = 255
>>> img[200:, 512:, 2] = 255
>>> thickness = 2
>>> fontScale = 1.0
>>> lineType = 4
>>> lineType = 8
>>> lineType = cv2.CV_AA
>>> for count, font_name in enumerate(font_names, start=1):
>>>     print(font_name)
>>>     fontFace = cv2.__dict__[font_name]
>>>     org = (10, count * 45)
>>>     text = 'opencv - ' + font_name
>>>     vt.draw_text(img, text, org,
...                 fontFace=fontFace, textcolor_rgb=[255, 255, 255],
...                 fontScale=fontScale, thickness=thickness)
>>> # xdoctest: +REQUIRES(--show)
```

(continues on next page)

(continued from previous page)

```
>>> import wbia.plottool as pt
>>> pt.imshow(img)
>>> ut.show_if_requested()
```

where each of the font IDs can be combined with FONT_ITALIC to get the slanted letters.

`vtool.image.embed_channels (img, input_channels=(0,), nchannels=3, fill=0)`

Parameters

- **img** (`ndarray[uint8_t, ndim=2]`) – image data
- **input_channels** (`tuple`) – (default = (0,))
- **nchannels** (`int`) – (default = 3)

CommandLine: `python -m vtool.image embed_channels --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> # Embed a (N,M,2) image into an (N,M,3) image
>>> img_fpath = ut.grab_test_imgpath('carl.jpg')
>>> img = vt.imread(img_fpath).T[1:3].T
>>> input_channels = (1, 2)
>>> nchannels = 3
>>> newimg = embed_channels(img, input_channels, nchannels)
>>> assert newimg.shape[-1] == 3
>>> assert np.all(newimg[:, :, input_channels] == img)
```

`vtool.image.embed_in_square_image (img, target_size, img_origin=(0.5, 0.5), target_origin=(0.5, 0.5))`

Embeds an image in the center of an empty image

Parameters

- **img** (`ndarray[uint8_t, ndim=2]`) – image data
- **target_size** (`tuple`) –
- **offset** (`tuple`) – position of

Returns `img_sqare`

Return type `ndarray`

CommandLine: `python -m vtool.image embed_in_square_image --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> img_fpath = ut.grab_test_imgpath('carl.jpg')
>>> img = vt.imread(img_fpath)
```

(continues on next page)

(continued from previous page)

```

>>> target_size = tuple(np.array(vt.get_size(img)) * 3)
>>> img_origin = (.5, .5)
>>> target_origin = (.5, .5)
>>> img_square = embed_in_square_image(img, target_size, img_origin, target_
->origin)
>>> assert img_square.sum() == img.sum()
>>> assert vt.get_size(img_square) == target_size
>>> img_origin = (0, 0)
>>> target_origin = (0, 0)
>>> img_square2 = embed_in_square_image(img, target_size, img_origin, target_
->origin)
>>> assert img_square.sum() == img.sum()
>>> assert vt.get_size(img_square) == target_size
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(img_square, pnum=(1, 2, 1))
>>> pt.imshow(img_square2, pnum=(1, 2, 2))
>>> ut.show_if_requested()

```

`vtool.image.ensure_3channel(patch)`
DEPRICATE IN FAVOR OF `atleast_3channels?`

Ensures that there are 3 channels in the image

Parameters `patch` (`ndarray[N, M, ...]`) – the image

Returns [N, M, 3]

Return type `ndarray`

CommandLine: `python -m vtool.image --exec-ensure_3channel --show`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> patch1 = vt.imread(ut.grab_test_imgpath('astro.png'))[0:512, 0:500, :]
>>> patch2 = vt.imread(ut.grab_test_imgpath('ada.jpg'))[:, :, 0:1]
>>> patch3 = vt.imread(ut.grab_test_imgpath('jeff.png'))[0:390, 0:400, 0]
>>> res1 = ensure_3channel(patch1)
>>> res2 = ensure_3channel(patch2)
>>> res3 = ensure_3channel(patch3)
>>> assert res1.shape[0:2] == patch1.shape[0:2], 'failed test1'
>>> assert res2.shape[0:2] == patch2.shape[0:2], 'failed test2'
>>> assert res3.shape[0:2] == patch3.shape[0:2], 'failed test3'
>>> assert res1.shape[-1] == 3
>>> assert res2.shape[-1] == 3
>>> assert res3.shape[-1] == 3

```

`vtool.image.ensure_4channel(img)`

`vtool.image.filterflags_valid_images(gpaths, valid_formats=None, invalid_formats=None, verbose=True)`

Flags images with a format that disagrees with its extension

Parameters

- **gpaths** (*list*) – list of image paths
- **valid_formats** (*None*) – (default = None)
- **invalid_formats** (*None*) – (default = None)
- **verbose** (*bool*) – verbosity flag (default = True)

Returns *isvalid_flags*

Return type *list*

CommandLine: `python -m vtool.image filterflags_valid_images --show`

Notes

An MPO (Multi Picture Object) file is a stereoscopic image and contains two JPG images side-by-side, and allows them to be viewed as a single 3D image.

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> gpaths = [ut.grab_test_imgpath('car1.jpg'),
>>>            ut.grab_test_imgpath('astro.png')]
>>> flags = filterflags_valid_images(gpaths)
>>> assert all(flags)
```

`vtool.image.find_pixel_value_index` (*img*, *pixel*)

Parameters

- **img** (*ndarray[uint8_t, ndim=2]*) – image data
- **pixel** (*ndarray or scalar*) –

CommandLine: `python -m vtool.util_math --test-find_pixel_value_index`

References

<http://stackoverflow.com/questions/21407815/get-column-row-index-from-numpy-array-that-meets-a-boolean-condition>

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> # build test data
>>> img = np.random.rand(10, 10, 3) + 1.0
>>> pixel = np.array([0, 0, 0])
>>> img[5, 5, :] = pixel
>>> img[2, 3, :] = pixel
>>> img[1, 1, :] = pixel
>>> img[0, 0, :] = pixel
>>> img[2, 0, :] = pixel
>>> # execute function
```

(continues on next page)

(continued from previous page)

```

>>> result = find_pixel_value_index(img, pixel)
>>> # verify results
>>> print(result)
[[0 0]
 [1 1]
 [2 0]
 [2 3]
 [5 5]]

```

`vtool.image.get_num_channels(img)`

Returns the number of color channels

`vtool.image.get_pixel_dist(img, pixel, channel=None)`

pixel = fillval isfill = mask2d

`vtool.image.get_round_scaled_dsize(dsize, scale)`

Returns an integer size and scale that best approximates the floating point scale on the original size

Parameters

- **dsize** (*tuple*) – original width height
- **scale** (*float or tuple*) – desired floating point scale factor

`vtool.image.get_scale_factor(src_img, dst_img)`

returns scale factor from one image to the next

`vtool.image.get_size(img)`

Returns the image size in (width, height)

`vtool.image.imread(img_fpath, grayscale=False, orient=False, flags=None, force_pil=None, delete_if_corrupted=False, **kwargs)`

Wrapper around the opencv imread function. Handles remote uris.

Parameters

- **img_fpath** (*str*) – file path string
- **grayscale** (*bool*) – (default = False)
- **orient** (*bool*) – (default = False)
- **flags** (*None*) – opencv flags (default = None)
- **force_pil** (*bool*) – (default = None)
- **delete_if_corrupted** (*bool*) – (default = False)

Returns imgBGR

Return type ndarray

CommandLine: `python -m vtool.image --test-imread python -m vtool.image --test-imread:1 python -m vtool.image --test-imread:2`

References

http://docs.opencv.org/modules/core/doc/utility_and_system_functions_and_macros.html#error
<http://stackoverflow.com/questions/23572241/cv2-threshold-error-210>

[http:](http://)

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> img_fpath = ut.grab_test_imgpath('car1.jpg')
>>> imgBGR1 = imread(img_fpath, grayscale=False)
>>> imgBGR2 = imread(img_fpath, grayscale=True)
>>> imgBGR3 = imread(img_fpath, orient=True)
>>> assert imgBGR1.shape == (250, 300, 3)
>>> assert imgBGR2.shape == (250, 300)
>>> # assert np.all(imgBGR1 == imgBGR3)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(imgBGR1, pnum=(2, 2, 1))
>>> pt.imshow(imgBGR2, pnum=(2, 2, 2))
>>> pt.imshow(imgBGR3, pnum=(2, 2, 3))
>>> ut.show_if_requested()
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> img_url = 'http://images.summitpost.org/original/769474.JPG'
>>> img_fpath = ut.grab_file_url(img_url)
>>> imgBGR1 = imread(img_url)
>>> imgBGR2 = imread(img_fpath)
>>> #imgBGR2 = imread(img_fpath, force_pil=False, flags=cv2.IMREAD_UNCHANGED)
>>> print('imgBGR1.shape = %r' % (imgBGR1.shape,))
>>> print('imgBGR2.shape = %r' % (imgBGR2.shape,))
>>> result = str(imgBGR1.shape)
>>> diff_pxls = imgBGR1 != imgBGR2
>>> num_diff_pxls = diff_pxls.sum()
>>> print(result)
>>> print('num_diff_pxls=%r/%r' % (num_diff_pxls, diff_pxls.size))
>>> assert num_diff_pxls == 0
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> diffMag = np.linalg.norm(imgBGR2 / 255. - imgBGR1 / 255., axis=2)
>>> pt.imshow(imgBGR1, pnum=(1, 3, 1))
>>> pt.imshow(diffMag / diffMag.max(), pnum=(1, 3, 2))
>>> pt.imshow(imgBGR2, pnum=(1, 3, 3))
>>> ut.show_if_requested()
(2736, 3648, 3)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> url = 'http://www.sherv.net/cm/emo/funny/2/big-dancing-banana-smiley-emoticon.
↳ gif'
>>> img_fpath = ut.grab_file_url(url)
>>> delete_if_corrupted = False
>>> grayscale = False
>>> imgBGR = imread(img_fpath, grayscale=grayscale)
```

(continues on next page)

(continued from previous page)

```
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(imgBGR)
>>> ut.show_if_requested()
```

```
vtool.image.imread_remote_s3(img_fpath, **kwargs)
vtool.image.imread_remote_url(img_url, **kwargs)
vtool.image.imwrite(img_fpath, imgBGR, fallback=False)
```

References

http://docs.opencv.org/modules/highgui/doc/reading_and_writing_images_and_video.html

Parameters

- **img_fpath** (*str*) – file path string
- **imgBGR** (*ndarray[uint8_t, ndim=2]*) – image data in opencv format (blue, green, red)
- **fallback** (*bool*) – (default = False)

CommandLine: python -m vtool.image --exec-implode

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> import utool as ut
>>> img_fpath1 = ut.grab_test_imgpath('zebra.png')
>>> imgBGR = vt.imread(img_fpath1)
>>> img_dpath = ut.ensure_app_cache_dir('vtool', 'testwrite')
>>> img_fpath2 = ut.unixjoin(img_dpath, 'zebra.png')
>>> fallback = False
>>> imwrite(img_fpath2, imgBGR, fallback=fallback)
>>> imgBGR2 = vt.imread(img_fpath2)
>>> assert np.all(imgBGR2 == imgBGR)
```

```
vtool.image.imwrite_fallback(img_fpath, imgBGR)
vtool.image.infer_vert(img1, img2, vert)
    which is the better stack dimension
vtool.image.make_channels_comparable(img1, img2)
    Broadcasts image arrays so they can have elementwise operations applied
CommandLine: python -m vtool.image make_channels_comparable
```

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> wh_basis = [(5, 5), (3, 5), (5, 3), (1, 1), (1, 3), (3, 1)]
>>> for w, h in wh_basis:
>>>     shape_basis = [(w, h), (w, h, 1), (w, h, 3)]
>>>     # Test all permutations of shap inputs
>>>     for shapel, shape2 in ut.product(shape_basis, shape_basis):
>>>         print('* input shapes: %r, %r' % (shapel, shape2))
>>>         img1 = np.empty(shapel)
>>>         img2 = np.empty(shape2)
>>>         img1, img2 = make_channels_comparable(img1, img2)
>>>         print('... output shapes: %r, %r' % (img1.shape, img2.shape))
>>>         elem = (img1 + img2)
>>>         print('... elem(+) shape: %r' % (elem.shape,))
>>>         assert elem.size == img1.size, 'outputs should have same size'
>>>         assert img1.size == img2.size, 'new imgs should have same size'
>>>         print('-----')
```

`vtool.image.make_white_transparent(imgBGR)`

Parameters `imgBGR` (`ndarray [uint8_t, ndim=2]`) – image data (blue, green, red)

Returns `imgBGRA`

Return type `ndarray`

CommandLine: `python -m vtool.image make_white_transparent --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> imgBGR = imread(ut.get_argval('--fpath', type=str))
>>> imgBGRA = make_white_transparent(imgBGR)
>>> result = ('imgBGRA = %s' % (ub.repr2(imgBGRA),))
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> ut.show_if_requested()
```

`vtool.image.montage(img_list, dsize, rng=<module 'numpy.random' from
'/home/docs/checkouts/readthedocs.org/user_builds/wbia-
vtool/envs/stable/lib/python3.7/site-packages/numpy/random/__init__.py'>,
method='random', return_debug=False)`

Creates a montage / collage from a set of images

CommandLine: `python -m vtool.image --exec-montage:0 --show python -m vtool.image --exec-montage:1`

Example

```
>>> # SLOW_DOCTEST
>>> # xdoctest: +SKIP
>>> from vtool.image import * # NOQA
>>> img_list0 = testdata_imglist()
>>> img_list1 = [resize_to_maxdims(img, (256, 256)) for img in img_list0]
```

(continues on next page)

(continued from previous page)

```

>>> num = 4
>>> img_list = list(ub.flatten([img_list1] * num))
>>> dsize = (700, 700)
>>> rng = np.random.RandomState(42)
>>> method = 'unused'
>>> #method = 'random'
>>> dst, debug_info = montage(img_list, dsize, rng, method=method,
>>>                             return_debug=True)
>>> place_img = debug_info.get('place_img_', np.ones((2, 2)))
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(dst, pnum=(1, 2, 1))
>>> pt.imshow(place_img / place_img.max(), pnum=(1, 2, 2))
>>> ut.show_if_requested()

```

Example

```

>>> # SLOW_DOCTEST
>>> # xdoctest: +SKIP
>>> import wbia
>>> import random
>>> from os.path import join, expanduser, abspath
>>> from vtool.image import * # NOQA
>>> ibs = wbia.opendb('GZC')
>>> gid_list0 = ibs.get_valid_gids()
>>> img_list = []
>>> for i in range(6000):
>>>     print(i)
>>>     try:
>>>         gid = random.choice(gid_list0)
>>>         image = ibs.get_images(gid)
>>>         image = resize_to_maxdims(image, (512, 512))
>>>         img_list.append(image)
>>>     except Exception:
>>>         pass
>>> dsize = (19200, 10800)
>>> rng = np.random.RandomState(42)
>>> dst = montage(img_list, dsize, rng)
>>> filepath = abspath(expanduser(join('~', 'Desktop', 'montage.jpg')))
>>> print('Writing to: %r' % (filepath, ))
>>> imwrite(filepath, dst)

```

`vtool.image.open_image_size(image_fpath)`

Gets image size from an image on disk

Parameters `image_fpath` (*str*) –

Returns size (width, height)

Return type `tuple`

CommandLine: `python -m vtool.image --test-open_image_size`

Doctest:

```
>>> from vtool.image import * # NOQA
>>> image_fpath = ut.grab_test_imgpath('patsy.jpg')
>>> size = open_image_size(image_fpath)
>>> result = ('size = %s' % (str(size),))
>>> print(result)
size = (800, 441)
```

Ignore:

```
>>> # Confirm that Image.open is a lazy load
>>> import vtool as vt
>>> import utool as ut
>>> import timeit
>>> setup = ut.codeblock(
>>>     '''
>>>     from PIL import Image
>>>     import utool as ut
>>>     import vtool as vt
>>>     image_fpath = ut.grab_test_imgpath('patsy.jpg')
>>>     '''
>>> )
>>> t1 = timeit.timeit('Image.open(image_fpath)', setup, number=100)
>>> t2 = timeit.timeit('Image.open(image_fpath).size', setup, number=100)
>>> t3 = timeit.timeit('vt.open_image_size(image_fpath)', setup, number=100)
>>> t4 = timeit.timeit('vt.imread(image_fpath).shape', setup, number=100)
>>> t5 = timeit.timeit('Image.open(image_fpath).getdata()', setup, number=100)
>>> print('t1 = %r' % (t1,))
>>> print('t2 = %r' % (t2,))
>>> print('t3 = %r' % (t3,))
>>> print('t4 = %r' % (t4,))
>>> print('t5 = %r' % (t5,))
>>> assert t2 < t5
>>> assert t3 < t4
```

`vtool.image.pad_image(imgBGR, pad_, value=0, borderType=0)`

`vtool.image.pad_image_ondisk(img_fpath, pad_, out_fpath=None, value=0, borderType=0, **kwargs)`

Returns out_fpath - file path string

Return type str

CommandLine: python -m vtool.image pad_image_ondisk

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> img_fpath = ut.get_argval('--fpath', type_=str)
>>> pad_ = '?'
>>> out_fpath = None
>>> value = 0
>>> borderType = 0
>>> out_fpath = pad_image_ondisk(img_fpath, pad_, out_fpath, value, borderType)
>>> result = ('out_fpath = %s' % (ub.repr2(out_fpath),))
>>> print(result)
```


`vtool.image.padded_resize` (*img*, *target_size*=(64, 64), *interpolation*=None)
 makes the image resize to the target size and pads the rest of the area with a fill value

Parameters

- **img** (*ndarray*[*uint8_t*, *ndim*=2]) – image data
- **target_size** (*tuple*) –

CommandLine: `python -m vtool.image --test-padded_resize --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> imgA = vt.imread(ut.grab_test_imgpath('carl.jpg'))
>>> imgB = vt.imread(ut.grab_test_imgpath('ada.jpg'))
>>> imgC = vt.imread(ut.grab_test_imgpath('carl.jpg'), grayscale=True)
>>> #target_size = (64, 64)
>>> target_size = (1024, 1024)
>>> img3_list = [padded_resize(img, target_size) for img in [imgA, imgB, imgC]]
>>> # verify results
>>> assert ut.allsame([vt.get_size(img3) for img3 in img3_list])
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pnum_ = pt.make_pnum_nextgen(1, 3)
>>> pt.imshow(img3_list[0], pnum=pnum_())
>>> pt.imshow(img3_list[1], pnum=pnum_())
>>> pt.imshow(img3_list[2], pnum=pnum_())
>>> ut.show_if_requested()
```

`vtool.image.perlin_noise` (*size*, *scale*=32.0, *rng*=<module 'numpy.random' from
 '/home/docs/checkouts/readthedocs.org/user_builds/wbia-
 vtool/envs/stable/lib/python3.7/site-packages/numpy/random/__init__.py'>)

References

<http://www.siafoo.net/snippet/229>

CommandLine: `python -m vtool.image perlin_noise --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> #size = (64, 64)
>>> size = (256, 256)
>>> #scale = 32.0
>>> scale = 64.0
>>> img = perlin_noise(size, scale)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
```

(continues on next page)

(continued from previous page)

```
>>> pt.imshow(img, pnum=(1, 1, 1))
>>> ut.show_if_requested()
```

```
vtool.image.rectify_to_float01 (img, dtype=<class 'numpy.float32'>)
```

Ensure that an image is encoded using a float properly

```
vtool.image.rectify_to_square (img, extreme='max')
```

```
vtool.image.rectify_to_uint8 (img)
```

Ensure that an image is encoded in uint8 properly

```
vtool.image.resize (img, dsize, interpolation=None)
```

```
vtool.image.resize_image_by_scale (img, scale, interpolation=None)
```

```
vtool.image.resize_mask (mask, chip, interpolation=None)
```

```
vtool.image.resize_thumb (img, max_dsize=(64, 64), interpolation=None)
```

Resize an image such that its max width or height is:

CommandLine: python -m vtool.image --test-resize_thumb --show

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> # build test data
>>> img_fpath = ut.grab_test_imgpath('car1.jpg')
>>> img = vt.imread(img_fpath)
>>> max_dsize = (64, 64)
>>> # execute function
>>> img2 = resize_thumb(img, max_dsize)
>>> print('img.shape = %r' % (img.shape,))
>>> print('img2.shape = %r' % (img2.shape,))
>>> # verify results
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(img2)
>>> ut.show_if_requested()
```

```
vtool.image.resize_to_maxdims (img, max_dsize=(64, 64), interpolation=None)
```

Parameters

- **img** (`ndarray[uint8_t, ndim=2]`) – image data
- **max_dsize** (`tuple`) –
- **interpolation** (`long`) –

CommandLine: python -m vtool.image --test-resize_to_maxdims --show

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> img_fpath = ut.grab_test_imgpath('carl.jpg')
>>> img = vt.imread(img_fpath)
>>> max_dsize = (1024, 1024)
>>> img2 = resize_to_maxdims(img, max_dsize)
>>> print('img.shape = %r' % (img.shape,))
>>> print('img2.shape = %r' % (img2.shape,))
>>> # verify results
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(img2)
>>> ut.show_if_requested()

```

`vtool.image.resize_to_maxdims_ondisk(img_fpath, max_dsize, out_fpath=None)`

Parameters

- **img_fpath** (*str*) – file path string
- **max_dsize** –
- **out_fpath** (*str*) – file path string (default = None)

CommandLine: python -m vtool.image resize_to_maxdims_ondisk -fpath ~/latex/crall-candidacy-2015/figures3/knormA.png -dsize=417, None python -m vtool.image resize_to_maxdims_ondisk -fpath ~/latex/crall-candidacy-2015/figures3/knormB.png -dsize=417, None python -m vtool.image resize_to_maxdims_ondisk -fpath ~/latex/crall-candidacy-2015/figures3/knormC.png -dsize=417, None python -m vtool.image resize_to_maxdims_ondisk -fpath ~/latex/crall-candidacy-2015/figures3/knormD.png -dsize=417, None python -m vtool.image resize_to_maxdims_ondisk -fpath ~/latex/crall-candidacy-2015/figures3/knormE.png -dsize=417, None python -m vtool.image resize_to_maxdims_ondisk -fpath ~/latex/crall-candidacy-2015/figures3/knormF.png -dsize=417, None python -m vtool.image resize_to_maxdims_ondisk -fpath ~/latex/crall-candidacy-2015/figures3/knormG.png -dsize=417, None python -m vtool.image resize_to_maxdims_ondisk -fpath ~/latex/crall-candidacy-2015/figures3/knormH.png -dsize=417, None python -m vtool.image resize_to_maxdims_ondisk -fpath ~/latex/crall-candidacy-2015/figures3/knormI.png -dsize=417, None python -m vtool.image resize_to_maxdims_ondisk -fpath ~/latex/crall-candidacy-2015/figures3/knormJ.png -dsize=417, None

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> img_fpath = ut.get_argval('--fpath')
>>> max_dsize = ut.get_argval('--dsize', type_=list)
>>> out_fpath = None
>>> resize_to_maxdims_ondisk(img_fpath, max_dsize, out_fpath)

```

`vtool.image.resized_clamped_thumb_dims(img_size, max_dsize)`

`vtool.image.resized_dims_and_ratio(img_size, max_dsize)`
returns resized dimensions to get `img_size` to fit into `max_dsize`

FIXME: Should specifying a None force the use of the original dim?

Parameters

- `img_size(tuple)` –
- `max_dsize(tuple)` –

Returns (dsize, ratio)

Return type tuple

CommandLine: python -m vtool.image resized_dims_and_ratio --show

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> img_size = (200, 100)
>>> max_dsize = (150, 150)
>>> (dsize, ratio) = resized_dims_and_ratio(img_size, max_dsize)
>>> result = ('(dsize, ratio) = %s' % (ub.repr2((dsize, ratio), nl=0),))
>>> print(result)
(dsize, ratio) = ((150, 75), 0.75)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> img_size = (200, 100)
>>> max_dsize = (5000, 1000)
>>> (dsize, ratio) = resized_dims_and_ratio(img_size, max_dsize)
>>> result = ('(dsize, ratio) = %s' % (ub.repr2((dsize, ratio), nl=0),))
>>> print(result)
(dsize, ratio) = ((2000, 1000), 10.0)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> img_size = (200, 100)
>>> max_dsize = (5000, None)
>>> (dsize, ratio) = resized_dims_and_ratio(img_size, max_dsize)
>>> result = ('(dsize, ratio) = %s' % (ub.repr2((dsize, ratio), nl=0),))
>>> print(result)
(dsize, ratio) = ((200, 100), 1.0)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> img_size = (200, 100)
>>> max_dsize = (None, None)
>>> (dsize, ratio) = resized_dims_and_ratio(img_size, max_dsize)
>>> result = ('(dsize, ratio) = %s' % (ub.repr2((dsize, ratio), nl=0),))
```

(continues on next page)

(continued from previous page)

```
>>> print(result)
(dsize, ratio) = ((200, 100), 1.0)
```

`vtool.image.rotate_image` (*img*, *theta*, *border_mode=None*, *interpolation=None*, *dsize=None*)
 Rotates an image around its center

Parameters

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

CommandLine: `python -m vtool.image --test-rotate_image`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> # build test data
>>> img = vt.get_test_patch('star2')
>>> theta = TAU / 16.0
>>> # execute function
>>> imgR = rotate_image(img, theta)
>>> if ut.get_argflag('--show') or ut.inIPython():
>>>     import wbia.plottool as pt
>>>     pt.imshow(img * 255, pnum=(1, 2, 1))
>>>     pt.imshow(imgR * 255, pnum=(1, 2, 2))
>>>     pt.show_if_requested()
```

`vtool.image.rotate_image_ondisk` (*img_fpath*, *theta*, *out_fpath=None*, ***kwargs*)
 Rotates an image on disk

Parameters

- **img_fpath** –
- **theta** –
- **out_fpath** (*None*) –

CommandLine: `python -m vtool.image --test-rotate_image_ondisk`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> # build test data
>>> img_fpath = ut.grab_test_imgpath('star.png')
>>> theta = TAU * 3 / 8
>>> # execute function
>>> out_fpath = None
>>> out_fpath_ = rotate_image_ondisk(img_fpath, theta, out_fpath)
>>> print(out_fpath_)
>>> if ut.get_argflag('--show') or ut.inIPython():
```

(continues on next page)

(continued from previous page)

```

>>> import wbia.plottool as pt
>>> pt.imshow(out_fpath_, pnum=(1, 1, 1))
>>> pt.show_if_requested()

```

`vtool.image.shear` (*img*, *x_shear*, *y_shear*, *dsize=None*, ***kwargs*)

Parameters

- **img** (`ndarray[uint8_t, ndim=2]`) – image data
- **x_shear** –
- **y_shear** –
- **dsize** (`tuple`) – width, height

CommandLine: `python -m vtool.image --test-shear --show`

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> img_fpath = ut.grab_test_imgpath('car1.jpg')
>>> img = vt.imread(img_fpath)
>>> x_shear = 0.05
>>> y_shear = -0.05
>>> dsize = None
>>> imgSh = shear(img, x_shear, y_shear, dsize)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(imgSh)
>>> ut.show_if_requested()

```

`vtool.image.stack_image_list` (*img_list*, *return_offset=False*, *return_sf=False*, *return_info=False*, ***kwargs*)

CommandLine: `python -m vtool.image --test-stack_image_list --show`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> # build test data
>>> img_list = testdata_imglist()
>>> vert = False
>>> return_offset = True
>>> modifysize = True
>>> return_sf=True
>>> kwargs = dict(modifysize=modifysize, vert=vert, use_larger=False)
>>> # execute function
>>> imgB, offset_list, sf_list = stack_image_list(img_list, return_offset=return_
↪ offset, return_sf=return_sf, **kwargs)
>>> # verify results
>>> result = ub.repr2(np.array(offset_list).T, precision=2, with_dtype=True)

```

(continues on next page)

(continued from previous page)

```

>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(imgB)
>>> wh_list = np.array([vt.get_size(img) for img in img_list])
>>> wh_list_ = wh_list * sf_list
>>> for offset, wh, color in zip(offset_list, wh_list_, pt.distinct_
↳ colors(len(offset_list))):
...     pt.draw_bbox((offset[0], offset[1], wh[0], wh[1]), bbox_color=color)
>>> pt.show_if_requested()
>>> #wh1 = img1.shape[0:2][::-1]
>>> #wh2 = img2.shape[0:2][::-1]
>>> #pt.draw_bbox((0, 0) + wh1, bbox_color=(1, 0, 0))
>>> #pt.draw_bbox((woff, hoff) + wh2, bbox_color=(0, 1, 0))
np.array([[ 0. , 76.96, 141.08, 181.87, 246. ],
          [ 0. , 0. , 0. , 0. , 0. ]], dtype=np.float64)

```

`vtool.image.stack_image_list_special` (*img1*, *img_list*, *num=1*, *vert=True*, *use_larger=True*,
initial_sf=None, *interpolation=None*)

TODO: add initial scale down factor?

CommandLine: `python -m vtool.image --test-stack_image_list_special --show`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> img_list_ = testdata_imglist()
>>> img1 = img_list_[0]
>>> img_list = img_list_[1:]
>>> vert = True
>>> return_offset = True
>>> use_larger = False
>>> num_bot = 1
>>> initial_sf = None
>>> initial_sf = .5
>>> imgB, offset_list, sf_list = stack_image_list_special(img1, img_list, num_bot,
↳ vert, use_larger, initial_sf)
>>> # xdoctest: +REQUIRES(--show)
>>> wh_list = np.array([vt.get_size(img1)] + list(map(vt.get_size, img_list)))
>>> wh_list_ = wh_list * sf_list
>>> import wbia.plottool as pt
>>> pt.imshow(imgB)
>>> print('imgB.shape = %r' % (imgB.shape,))
>>> for offset, wh, color in zip(offset_list, wh_list_, pt.distinct_
↳ colors(len(offset_list))):
...     pt.draw_bbox((offset[0], offset[1], wh[0], wh[1]), bbox_color=color)
>>> ut.show_if_requested()

```

`vtool.image.stack_image_recurse` (*img_list1*, *img_list2=None*, *vert=True*, *modifysize=False*, *re-*
turn_offsets=False, *interpolation=None*)

TODO: return offsets as well

Parameters

- `img_list1` (*list*) –

- `img_list2(list)` –
- `vert(bool)` –

Returns None

Return type ndarray

CommandLine: `python -m vtool.image --test-stack_image_recurse --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> # build test data
>>> img1 = vt.imread(ut.grab_test_imgpath('carl.jpg'))
>>> img2 = vt.imread(ut.grab_test_imgpath('astro.png'))
>>> img3 = vt.imread(ut.grab_test_imgpath('ada.jpg'))
>>> img4 = vt.imread(ut.grab_test_imgpath('jeff.png'))
>>> img5 = vt.imread(ut.grab_test_imgpath('star.png'))
>>> img_list1 = [img1, img2, img3, img4, img5]
>>> img_list2 = None
>>> vert = True
>>> # execute function
>>> imgB = stack_image_recurse(img_list1, img_list2, vert)
>>> # verify results
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> imshow(imgB)
>>> #wh1 = img1.shape[0:2][::-1]
>>> #wh2 = img2.shape[0:2][::-1]
>>> #pt.draw_bbox((0, 0) + wh1, bbox_color=(1, 0, 0))
>>> #pt.draw_bbox((woff, hoff) + wh2, bbox_color=(0, 1, 0))
>>> pt.show_if_requested()
```

`vtool.image.stack_images(img1, img2, vert=None, modifysize=False, return_sf=False, use_larger=True, interpolation=None, white_background=False, overlap=0)`

Parameters

- `img1(ndarray[uint8_t, ndim=2])` – image data
- `img2(ndarray[uint8_t, ndim=2])` – image data

CommandLine: `python -m vtool.image --test-stack_images --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> # build test data
>>> img1 = vt.imread(ut.grab_test_imgpath('carl.jpg'))
>>> img2 = vt.imread(ut.grab_test_imgpath('astro.png'))
>>> vert = True
```

(continues on next page)

(continued from previous page)

```

>>> modifysize = False
>>> # execute function
>>> return_sf = True
>>> #(imgB, woff, hoff) = stack_images(img1, img2, vert, modifysize, return_
↳sf=return_sf)
>>> overlap = 100
>>> imgB, offset2, sf_tup = stack_images(img1, img2, vert, modifysize,
>>>                                     return_sf=return_sf,
>>>                                     overlap=overlap)
>>> woff, hoff = offset2
>>> # verify results
>>> result = str((imgB.shape, woff, hoff))
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(imgB)
>>> wh1 = np.multiply(vt.get_size(img1), sf_tup[0])
>>> wh2 = np.multiply(vt.get_size(img2), sf_tup[1])
>>> pt.draw_bbox((0, 0, wh1[0], wh1[1]), bbox_color=(1, 0, 0))
>>> pt.draw_bbox((woff[1], hoff[1], wh2[0], wh2[0]), bbox_color=(0, 1, 0))
>>> pt.show_if_requested()
((662, 512, 3), (0.0, 0.0), (0, 150))

```

`vtool.image.stack_multi_images` (*img1*, *img2*, *offset_list1*, *sf_list1*, *offset_list2*, *sf_list2*, *vert=True*, *use_larger=False*, *modifysize=True*, *interpolation=None*)
 combines images that are already stacked

`vtool.image.stack_multi_images2` (*multiimg_list*, *offsets_list*, *sfs_list*, *vert=True*, *modifysize=True*)

Parameters

- **multiimg_list** (*list*) –
- **offset_lists** –
- **sfs_list** –
- **vert** (*bool*) –

Returns (*stacked_img*, *stacked_img*, *stacked_sfs*)

Return type `tuple`

CommandLine: `python -m vtool.image --test-stack_multi_images2 --show`

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> img_list = testdata_imglist()
>>> img_stack1, offset_list1, sf_list1 = stack_image_list(img_list[::-1],
↳vert=True, return_info=True, modifysize=True)
>>> img_stack2, offset_list2, sf_list2 = stack_image_list(img_list, vert=True,
↳return_info=True, modifysize=True)
>>> img_stack3, offset_list3, sf_list3 = stack_image_list(img_list, vert=True,
↳return_info=True, modifysize=False)

```

(continues on next page)

(continued from previous page)

```

>>> multiimg_list = [img_stack1, img_stack2, img_stack3]
>>> offsets_list = [offset_list1, offset_list2, offset_list3]
>>> sfs_list = [sf_list1, sf_list2, sf_list3]
>>> vert = False
>>> tup = stack_multi_images2(multiimg_list, offsets_list, sfs_list, vert)
>>> (stacked_img, stacked_offsets, stacked_sfs) = tup
>>> result = ut.remove_doublspaces(ub.repr2(np.array(stacked_offsets).T,
↳precision=2, with_dtype=True, linewidth=10000)).replace(' ', ', ')
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(stacked_img)
>>> wh_list = np.array([vt.get_size(img) for img in img_list[:-1] + img_list +
↳img_list])
>>> wh_list_ = wh_list * stacked_sfs
>>> for offset, wh, color in zip(stacked_offsets, wh_list_, pt.distinct_
↳colors(len(stacked_offsets))):
...     pt.draw_bbox((offset[0], offset[1], wh[0], wh[1]), bbox_color=color)
>>> ut.show_if_requested()
np.array([[ 0.,  0.,  0.,  0.,  0., 512., 512., 512., 512., 512., 1024., 1024., 1024.,
↳1024., 1024. ],
[ 0., 512.12, 1024.25, 1827., 2339., 0., 427., 939., 1742., 2254., 0., 373.18,
↳1137.45, 2073.38, 2670.47]], dtype=np.float64)

```

`vtool.image.stack_square_images (img_list, return_info=False, **kwargs)`

Parameters `img_list` (*list*) –

Returns

Return type ndarray

CommandLine: `python -m vtool.image --test-stack_square_images`

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> img_list = '?'
>>> result = stack_square_images(img_list)
>>> print(result)

```

`vtool.image.subpixel_values (img, pts)`

References

stackoverflow.com/questions/12729228/simple-efficient-bilinear-interpolation-of-images-in-numpy-and-python

SeeAlso: `cv2.getRectSubPix(image, patchSize, center[, patch[, patchType]])`

`vtool.image.testdata_imglist ()`

`vtool.image.warpAffine (img, Aff, dsize, assume_float01=True)`
 dsize = (width, height) of return image

Parameters

- **img** (*ndarray*[*uint8_t*, *ndim*=2]) – image data
- **Aff** (*ndarray*) – affine matrix
- **dsize** (*tuple*) – width, height

Returns *warped_img*

Return type *ndarray*

CommandLine: `python -m vtool.image --test-warpAffine --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> img_fpath = ut.grab_test_imgpath('car1.jpg')
>>> img = vt.imread(img_fpath)
>>> Aff = vt.rotation_mat3x3(TAU / 8)
>>> dsize = vt.get_size(img)
>>> warped_img = warpAffine(img, Aff, dsize)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(warped_img)
>>> ut.show_if_requested()
```

Ignore:

```
>>> import skimage.transform
>>> %timeit cv2.warpAffine(img, Aff[0:2], tuple(dsize), **CV2_WARP_KWARGS)
>>> 100 loops, best of 3: 7.95 ms per loop
>>> skimage.transform.AffineTransform
>>> tf = skimage.transform.AffineTransform(rotation=TAU / 8)
>>> Aff_ = tf.params
>>> out = skimage.transform._warps_cy._warp_fast(img[:, :, 0], Aff_, output_
↳ shape=dsize, mode='constant', order=1)
>>> %timeit skimage.transform._warps_cy._warp_fast(img[:, :, 0], Aff_, output_
↳ shape=dsize, mode='constant', order=1)
>>> 100 loops, best of 3: 5.74 ms per loop
>>> %timeit cv2.warpAffine(img[:, :, 0], Aff[0:2], tuple(dsize), **CV2_WARP_
↳ KWARGS)
>>> 100 loops, best of 3: 5.13 ms per loop
>>> CONCLUSION, cv2 transforms are better
```

`vtool.image.warpHomog` (*img*, *Homog*, *dsize*, *assume_float01=True*)
dsize = (width, height) of return image

Example

```
>>> img = np.random.rand(224, 224)
>>> Homog = np.random.rand(3, 3)
>>> dsize = (128, 128)
>>> warped_img = warpHomog(img, Homog, dsize)
```

1.23 vtool.image_filters module

class vtool.image_filters.IntensityPreproc

Bases: object

Preferred over old methods

CommandLine: python -m vtool.image_filters IntensityPreproc --show

Doctest:

```
>>> from vtool.image_filters import *
>>> import vtool as vt
>>> import utool as ut
>>> chipBGR = vt.imread(ut.grab_file_url('http://i.imgur.com/qVWQaex.jpg'))
>>> filter_list = [
>>>     ('medianblur', {}),
>>>     ('adapteq', {}),
>>> ]
>>> self = IntensityPreproc()
>>> chipBGR2 = self.preprocess(chipBGR, filter_list)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(chipBGR, pnum=(1, 2, 1), fnum=1)
>>> pt.imshow(chipBGR2, pnum=(1, 2, 2), fnum=1)
>>> ut.show_if_requested()
```

adapteq (*intensity*, *tileGridSize*=(8, 8), *clipLimit*=2.0)

histeq (*intensity*)

Histogram equalization of a grayscale image.

medianblur (*intensity*, *noise_thresh*=50, *ksize1*=3, *ksize2*=5)

preprocess (*chipBGR*, *filter_list*)

filter_list is a list of (name, config) tuples for performing filter ops

vtool.image_filters.**adapteq_fn** (*chipBGR*)

adaptive histogram equalization with CLAHE

Example

```
>>> from vtool.image_filters import *
>>> import vtool as vt
>>> import utool as ut
>>> chipBGR = vt.imread(ut.grab_file_url('http://i.imgur.com/qVWQaex.jpg'))
>>> chip2 = adapteq_fn(chipBGR)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(chipBGR, pnum=(1, 2, 1), fnum=1)
>>> pt.imshow(chip2, pnum=(1, 2, 2), fnum=1)
>>> ut.show_if_requested()
```

vtool.image_filters.**clean_mask** (*mask*, *num_dilate*=3, *num_erode*=3, *window_frac*=0.025)

Clean the mask (num_erode, num_dilate) = (1, 1) (w, h) = (10, 10)

vtool.image_filters.**grabcut_fn** (*chipBGR*)

naively segments a chip

```
vtool.image_filters.histeq_fn(chipBGR)
```

Histogram equalization of a grayscale image.

```
vtool.image_filters.manta_matcher_filters(chipBGR)
```

References

<http://onlinelibrary.wiley.com/doi/10.1002/ece3.587/full>

Ignore:

```
>>> from wbia.core_annots import * # NOQA
>>> import utool as ut
>>> import wbia
>>> ibs = wbia.opendb('Mantas')
>>> chipBGR = vt.imread(ut.grab_file_url('http://i.imgur.com/qVWQaex.jpg'))
```

```
vtool.image_filters.medianfilter_fn(chipBGR)
```

median filtering

Example

```
>>> from vtool.image_filters import *
>>> import vtool as vt
>>> import utool as ut
>>> chipBGR = vt.imread(ut.grab_file_url('http://i.imgur.com/qVWQaex.jpg'))
>>> chip2 = adapteq_fn(chipBGR)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(chipBGR, pnum=(1, 2, 1), fnum=1)
>>> pt.imshow(chip2, pnum=(1, 2, 2), fnum=1)
>>> ut.show_if_requested()
```

1.24 vtool.image_shared module

```
vtool.image_shared.open_pil_image(image_fpath)
```

```
vtool.image_shared.print_image_checks(img_fpath)
```

1.25 vtool.inspect_matches module

```
class vtool.inspect_matches.MatchInspector
```

Bases: `object`

A widget that contains

- (1) a viewport that displays an annotation pair with matches overlayed.
- (2) a control panel for tuning matching parameters
- (3) a text area displaying information about the match vector

CommandLine: python -m vtool.inspect_matches MatchInspector:0 --show python -m vtool.inspect_matches MatchInspector:1 --show

python -m vtool.inspect_matches MatchInspector:1 --db GZ_Master1 --aids=1041,1045 --show

Example

```
>>> # SCRIPT
>>> from vtool.inspect_matches import * # NOQA
>>> import vtool as vt
>>> gt.ensure_qapp()
>>> ut.qtensure()
>>> annot1 = lazy_test_annot('easy1.png')
>>> annot2 = lazy_test_annot('easy2.png')
>>> match = vt.PairwiseMatch(annot1, annot2)
>>> self = MatchInspector(match=match)
>>> self.show()
>>> # xdoctest: +REQUIRES(--show)
>>> #self.update()
>>> gt.qtapp_loop(qwin=self, freq=10)
```

Example

```
>>> # SCRIPT
>>> from vtool.inspect_matches import * # NOQA
>>> import vtool as vt
>>> import wbia
>>> gt.ensure_qapp()
>>> ut.qtensure()
>>> ibs = wbia.opendb(defaultdb='PZ_MTEST')
>>> aids = ub.argval('--aids', default=[1, 2])
>>> print('aids = %r' % (aids,))
>>> annots = ibs.annots(aids)
>>> annot1 = annots[0]._make_lazy_dict()
>>> annot2 = annots[1]._make_lazy_dict()
>>> cfgdict = MatchDisplayConfig().asdict()
>>> cfgdict = ut.argparse_dict(cfgdict)
>>> match = vt.PairwiseMatch(annot1, annot2)
>>> self = MatchInspector(match=match, cfgdict=cfgdict)
>>> self.show()
>>> # xdoctest: +REQUIRES(--show)
>>> #self.update()
>>> gt.qtapp_loop(qwin=self, freq=10)
```

```
closeEvent(event)

draw_pair()

draw_vsone()

embed()

execContextMenu(qpoint)

execute_vsone()

first_show(state=None)
```

```

initialize (match=None, on_context=None, autoupdate=True, info_text=None, cfgdict=None)
on_cfg_changed (*args)
on_chip_cfg_changed (*args)
on_feat_cfg_changed (*args)
screenshot ()
set_match (match=None, on_context=None, info_text=None)
showEvent (event)
update (state=None)

vtool.inspect_matches.lazy_test_annot (key)
vtool.inspect_matches.make_match_interaction (matches, metadata, type_='RAT+SV',
                                              **kwargs)
vtool.inspect_matches.show_matching_dict (matches, metadata, *args, **kwargs)

```

1.26 vtool.keypoint module

Keypoints are stored in the invA format by default. Unfortunately many places in the code reference this as A instead of invA because I was confused when I first started writing this.

to rectify this I am changing terminology.

Variables:

invV [maps from ucircle onto an ellipse (perdoch.invA)] **V** : maps from ellipse to ucircle (perdoch.A) **Z** : the conic matrix (perdoch.E)

Representation:

kpts (ndarray) [[x, y, iv11, iv21, iv22, ori]] a flat on disk representation of the keypoint

invV (ndarray): [(iv11, iv12, x), (iv21, iv22, y), (0, 0, 1),] a more conceptually useful representation mapping a unit circle onto an ellipse (without any rotation)

invVR (ndarray): [(iv11, iv12, x), (iv21, iv22, y), (0, 0, 1),].dot(R) same as invV but it is rotated before warping a unit circle into an ellipse.

Ignore:

```

>>> # DISABLE_DOCTEST
>>> # xdoctest: +SKIP
>>> # https://groups.google.com/forum/#!topic/sympy/k1HnZK_bNNA
>>> from vtool.patch import * # NOQA
>>> import sympy
>>> from sympy.abc import theta
>>> ori = theta
>>> x, y, iv11, iv21, iv22, patch_size = sympy.symbols('x y iv11 iv21 iv22 S')
>>> sx, sy, w1, w2, tx, ty = sympy.symbols('sx sy w1 w2 tx ty')
>>> kpts = np.array([[x, y, iv11, iv21, iv22, ori]])
>>> kp = ktool.get_invV_mats(kpts, with_trans=True)[0]
>>> invV = sympy.Matrix(kp)
>>> V = invV.inv()
>>> #
>>> print(ub.hzcat('invV = %s' % (repr(invV), )))

```

(continues on next page)

(continued from previous page)

```

>>> invV = sympy.Matrix([
>>>     [iv11, 0.0, x],
>>>     [iv21, iv22, y],
>>>     [0.0, 0.0, 1.0]])
>>> R = vt.sympy_mat(vt.rotation_mat3x3(theta, sin=sympy.sin, cos=sympy.cos))
>>> invVR = invV.multiply(R)
>>> trans = sympy.Matrix([
>>>     [1, 0.0, x],
>>>     [0, 1, y],
>>>     [0.0, 0.0, 1.0]])
>>> #
>>> Hypoth = sympy.Matrix([
>>>     [sx, w1, tx],
>>>     [w2, sy, ty],
>>>     [0, 0, 1],
>>>     ])
>>> #
>>> xyz = sympy.Matrix([x], [y], [1]))
>>> #
>>> invV_2x2 = invV[0:2, 0:2]
>>> Hypoth_2x2 = Hypoth[0:2, 0:2]
>>> #
>>> invV_t = sympy.simplify(Hypoth.multiply(invV))
>>> xyz_t = sympy.simplify(Hypoth.multiply(xyz))
>>> invV_2x2_t = Hypoth_2x2.multiply(invV_2x2)
>>> print('\n----')
>>> vt.evalprint('invV_t')
>>> vt.evalprint('xyz_t')
>>> vt.evalprint('invV_2x2_t')
>>> print('-----')
>>> #
>>> print('\n--- CHECKING 3x3 ---')
>>> vt.check_expr_eq(invV_t[:, 2], xyz_t)
>>> print('\n--- CHECKING 2x2 ---')
>>> vt.check_expr_eq(invV_t[0:2, 0:2], invV_2x2_t)
>>> #
>>> # Check with rotation component as well (probably ok)
>>> invVR_2x2 = invVR[0:2, 0:2]
>>> invVR_t = sympy.simplify(Hypoth.multiply(invVR))
>>> invVR_2x2_t = sympy.simplify(Hypoth_2x2.multiply(invVR_2x2))
>>> print('\n----')
>>> vt.evalprint('invVR_t')
>>> print('\n----')
>>> vt.evalprint('invVR_2x2_t')
>>> print('-----')
>>> #
>>> print('\n--- CHECKING ROTATION + TRANSLATION 3x3 ---')
>>> vt.check_expr_eq(invVR_t[:, 2], xyz_t)
>>> print('\n--- CHECKING ROTATION 2x2 ---')
>>> vt.check_expr_eq(invVR_t[0:2, 0:2], invVR_2x2_t)
>>> ####
>>> ####
>>> ####
>>> # Checking orientation property
>>> [[ivr11, ivr12, ivr13], [ivr21, ivr22, ivr23], [ivr31, ivr32, ivr33],] = 
    ↪ invVR.tolist()
>>> ori = sympy.atan2(ivr12, ivr11) # outputs from -TAU/2 to TAU/2

```

(continues on next page)

(continued from previous page)

```

>>> z = ori.subs(dict(iv11=1, theta=1))
>>> sympy.trigsimp(sympy.simplify(sympy.trigsimp(z)))
>>> #_oris = np.arctan2(_iv12s, _iv11s) # outputs from -TAU/2 to TAU/2
>>> # xdoctest: +SKIP
>>> # OLD STUFF
>>> #
>>> print(ub.hzcat('V = %s' % (repr(V), )))
V = Matrix([
  [ 1/iv11, 0, -1.0*x/iv11],
  [-iv21/(iv11*iv22), 1/iv22, -1.0*(y - iv21*x/iv11)/iv22],
  [ 0, 0, 1.0]])
>>> print(ub.hzcat('V = %s' % (repr(sympy.simplify(invV.inv())), )))
V = Matrix([
  [ 1/iv11, 0, -1.0*x/iv11],
  [-iv21/(iv11*iv22), 1/iv22, 1.0*(-iv11*y + iv21*x)/(iv11*iv22)],
  [ 0, 0, 1.0]])

```

Efficiency Notes: single index indexing is very fast

slicing seems to be very fast.

fancy indexing with `__getitem__` is very slow using `np.take` is a better idea, but its a bit harder to use with multidimensional arrays (nope use `axis=x`)

`vtool.keypoint.augment_2x2_with_translation(kpts, _mat2x2)`

helper function to augment shape matrix with a translation component.

`vtool.keypoint.cast_split(kpts, dtype=<class 'numpy.float32'>)`

breakup keypoints into location, shape, and orientation

`vtool.keypoint.convert_kptsZ_to_kpts(kpts_Z)`

Convert keypoints in Z format to invV format

`vtool.keypoint.decompose_Z_to_RV_mats2x2(Z_mats2x2)`

A, B, C = [0.016682, 0.001693, 0.014927] # A, B, C = [0.010141, -1.1e-05, 0.02863] Z = np.array([[A, B], [B, C]])

A, B, C = 0.010141, -1.1e-05, 0.02863

Ignore:

```

>>> # Working on figuring relationship between us and VGG
>>> A, B, _, C = Z_mats2x2[0].ravel()
>>> X, Y = 0, 0
>>> theta = np.linspace(0, np.pi * 2)
>>> circle_xy = np.vstack([np.cos(theta), np.sin(theta)])
>>> invV = invV_mats[0, 0:2, 0:2]
>>> x, y = invV.dot(circle_xy)
>>> V = np.linalg.inv(invV)
>>> E = V.T.dot(V)
>>> [[A, B], [_, C]] = E
>>> [[A_, B_], [_, C_]] = E
>>> print(A*(x-X) ** 2 + 2*B*(x-X)*(y-Y) + C*(y-Y) ** 2)
>>>
>>> Z_mats2x2 = np.array([
>>>     [[ .016682, .001693],
>>>      [ .001693, .014927]],
>>>     [[ .01662, .001693],
>>>      [ .001693, .014927]],

```

(continues on next page)

(continued from previous page)

```

>>> [[ .016682, .00193],
>>> [ .00193, .01492]],
>>> ])
>>>
>>> import scipy.linalg
>>> %timeit np.array([scipy.linalg.sqrtm(Z) for Z in Z_mats2x2])
>>> %timeit decompose_Z_to_VR_mats2x2(Z_mats2x2)

```

`vtool.keypoint.decompose_Z_to_V_2x2(Z_2x2)`

`vtool.keypoint.decompose_Z_to_invV_2x2(Z_2x2)`

`vtool.keypoint.decompose_Z_to_invV_mats2x2(Z_mats2x2)`

`vtool.keypoint.flatten_invV_mats_to_kpts(invV_mats)`

flattens invV matrices into kpts format

`vtool.keypoint.get_RV_mats2x2(kpts)`

Returns sequence of matrices that transform an ellipse to unit circle

Return type V_mats (ndarray)

`vtool.keypoint.get_RV_mats_3x3(kpts)`

preferred over get_invV_mats

Returns sequence of matrices that transform an ellipse to unit circle

Return type V_mats (ndarray)

`vtool.keypoint.get_V_mats(kpts, **kwargs)`

Returns sequence of matrices that transform an ellipse to unit circle

Return type V_mats (ndarray)

`vtool.keypoint.get_Z_mats(V_mats)`

transform into conic matrix Z $Z = (V.T).dot(V)$

Returns Z is a conic representation of an ellipse

Return type Z_mats (ndarray)

`vtool.keypoint.get_even_point_sample(kpts)`

gets even points sample along the boundary of the ellipse

SeeAlso: pyhesaff.tests.test_ellipse

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts()[0:2]
>>> ell_border_pts_list = get_even_point_sample(kpts)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.draw_line_segments(ell_border_pts_list)
>>> pt.set_title('even sample points')
>>> pt.show_if_requested()

```

```
vtool.keypoint.get_grid_kpts(wh=(300, 300), wh_stride=None, scale=20, wh_num=None,
                             dtype=<class 'numpy.float32'>, **kwargs)
```

Returns a regular grid of keypoints

Parameters

- **wh** (*tuple*) – (default = (300, 300))
- **wh_stride** (*tuple*) – stride of keypoints (defaults to (50, 50))
- **scale** (*int*) – (default = 20)
- **wh_num** (*tuple*) – desired number of keypoints in x and y direction. (incompatible with stride).
- **dtype** (*type*) – (default = <type 'numpy.float32'>)

Returns kpts - keypoints

Return type ndarray[float32_t, ndim=2]

CommandLine: python -m vtool.keypoint get_grid_kpts --show

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> wh = (300, 300)
>>> wh_stride = None
>>> scale = 20
>>> wh_num = (3, 3)
>>> dtype = np.float32
>>> kpts = get_grid_kpts(wh, wh_num=wh_num, dtype=dtype)
>>> assert len(kpts) == np.prod(wh_num)
>>> result = ('kpts = %s' % (ub.repr2(kpts.shape),))
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.show_kpts(kpts)
>>> pt.dark_background()
>>> ut.show_if_requested()
```

```
vtool.keypoint.get_invVR_mats2x2(kpts)
```

Returns the keypoint shape+rotation matrix (from unit circle to ellipse) Ignores translation component

Parameters **kpts** (ndarray[float32_t, ndim=2][ndims=2]) – keypoints

Returns invVR_mats

Return type ndarray

CommandLine: python -m vtool.keypoint --test-get_invVR_mats2x2

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> kpts = np.array([
```

(continues on next page)

(continued from previous page)

```

...     [0, 0, 1, 2, 3, 0],
...     [0, 0, 1, 2, 3, TAU / 4.0],
... ])
>>> invVR_mats2x2 = get_invVR_mats2x2(kpts)
>>> result = kpts_repr(invVR_mats2x2)
>>> print(result)

```

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> kpts = np.empty((0, 6))
>>> invVR_mats2x2 = get_invVR_mats2x2(kpts)
>>> assert invVR_mats2x2.shape == (0, 2, 2)

```

`vtool.keypoint.get_invVR_mats3x3(kpts)`
NEWER FUNCTION

Returns full keypoint transform matrices from a unit circle to an ellipse that has been rotated, scaled, skewed, and translated. Into the image keypoint position.

Parameters `kpts` (`ndarray[float32_t, ndim=2]`) – keypoints

Returns `invVR_mats`

Return type `ndarray[float32_t, ndim=3]`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> kpts = np.array([
...     [10, 20, 1, 2, 3, 0],
...     [30, 40, 1, 2, 3, TAU / 4.0],
... ])
>>> invVR_mats3x3 = get_invVR_mats3x3(kpts)
>>> # verify results
>>> result = kpts_repr(invVR_mats3x3)
>>> print(result)
array([[ [ 1.,  0., 10.],
        [ 2.,  3., 20.],
        [ 0.,  0.,  1.]],
       [[ 0., -1., 30.],
        [ 3., -2., 40.],
        [ 0.,  0.,  1.]])

```

`vtool.keypoint.get_invVR_mats_oris(invVR_mats)`

extracts orientation from matrix encoding, this is a bit trickier can use `-arctan2` or `(0, 0)` and `(0, 1)`, but then have to normalize

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> np.random.seed(0)
>>> invVR_mats = np.random.rand(7, 2, 2).astype(np.float64)
>>> output = get_invVR_mats_oris(invVR_mats)
>>> result = ub.repr2(output, precision=2, with_dtype=True)
```

`vtool.keypoint.get_invVR_mats_shape(invVR_mats)`
Extracts keypoint shape components

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> np.random.seed(0)
>>> invVR_mats = np.random.rand(1000, 3, 3).astype(np.float64)
>>> output = get_invVR_mats_shape(invVR_mats)
>>> result = ut.hash_data(output)
>>> print(result)
pibujdiaimwcnmomserkcytyikahjmp
```

References

TODO (a.ravel()[cols + (rows * a.shape[1]).reshape((-1,1)).ravel()]).reshape(rows.size, cols.size) <http://stackoverflow.com/questions/14386822/fast-numpy-fancy-indexing> # So, this doesn't work # Try this instead <http://docs.cython.org/src/userguide/memoryviews.html#memoryviews>

`vtool.keypoint.get_invVR_mats_sqrd_scale(invVR_mats)`
Returns the squared scale of the invVR keypoints

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> np.random.seed(0)
>>> invVR_mats = np.random.rand(7, 3, 3).astype(np.float64)
>>> det_arr = get_invVR_mats_sqrd_scale(invVR_mats)
>>> result = ub.repr2(det_arr, precision=2, with_dtype=True)
>>> print(result)
np.array([-0.16, -0.09, -0.34, 0.59, -0.2 , 0.18, 0.06], dtype=np.float64)
```

`vtool.keypoint.get_invVR_mats_xys(invVR_mats)`
extracts locations extracts xys from matrix encoding, Its just the (0, 2), and (1, 2) components

Parameters `invVR_mats` (*ndarray*) – list of matrices mapping uicircles to ellipses

Returns the xy location

Return type *ndarray*

Ignore:

```

>>> # DISABLE_DOCTEST
>>> import utool as ut
>>> setup = ut.codeblock(
...     '''
...     import numpy as np
...     np.random.seed(0)
...     invVR_mats = np.random.rand(1000, 3, 3).astype(np.float64)
...     '''
>>> stmt_list = ut.codeblock(
...     '''
...     invVR_mats[:, 0:2, 2].T
...     invVR_mats.T[2, 0:2]
...     invVR_mats.T.take(2, axis=0).take([0, 1], axis=0)
...     invVR_mats.T.take(2, axis=0)[0:2]
...     '''
... ).split('\n')
>>> ut.util_dev.timeit_compare(stmt_list, setup, int(1E5))

```

Example

```

>>> from vtool.keypoint import * # NOQA
>>> np.random.seed(0)
>>> invVR_mats = np.random.rand(1000, 3, 3).astype(np.float64)
>>> invVR_mats.T[2, 0:2]

```

`vtool.keypoint.get_invV_mats(kpts, with_trans=False, with_ori=False, ashomog=False, ascontiguous=False)`

TODO: DEPRICATE. too many conditionals

packs keypoint shapes into affine invV matrixes (default is just the 2x2 shape. But translation, orientation, homogenous, and contiguous flags can be set.)

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> kpts = np.array([[10, 20, 1, 2, 3, 0]])
>>> with_trans=True
>>> with_ori=True
>>> ashomog=True
>>> ascontiguous=False
>>> innVR_mats = get_invV_mats(kpts, with_trans, with_ori, ashomog, ascontiguous)
>>> result = kpts_repr(innVR_mats)
>>> print(result)
array([[ 1.,  0., 10.],
       [ 2.,  3., 20.],
       [ 0.,  0.,  1.]])

```

`vtool.keypoint.get_invV_mats2x2(kpts)`

Returns the keypoint shape (from unit circle to ellipse) Ignores translation and rotation component

Parameters `kpts` (`ndarray[float32_t, ndim=2]`) – keypoints

Returns `invV_mats`

Return type `ndarray[float32_t, ndim=3]`

CommandLine: `python -m vtool.keypoint --test-get_invV_mats2x2`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> kpts = np.array([
...     [0, 0, 1, 2, 3, 0],
...     [0, 0, 1, 2, 3, TAU / 4.0],
... ])
>>> invV_mats2x2 = get_invV_mats2x2(kpts)
>>> # verify results
>>> result = kpts_repr(invV_mats2x2)
```

`vtool.keypoint.get_invV_mats3x3(kpts)`
NEWER FUNCTION

Returns full keypoint transform matrices from a unit circle to an ellipse that has been scaled, skewed, and translated. Into the image keypoint position.

DOES NOT INCLUDE ROTATION

Parameters `kpts` (`ndarray[float32_t, ndim=2]`) – keypoints

Returns `invVR_mats` - keypoint shape and rotations (possibly translation)

Return type `ndarray[float32_t, ndim=3]`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> kpts = np.array([
...     [0, 0, 1, 2, 3, 0],
...     [0, 0, 1, 2, 3, TAU / 4.0],
... ])
>>> invV_arrs3x3 = get_invV_mats3x3(kpts)
>>> # verify results
>>> result = kpts_repr(invV_arrs3x3)
```

`vtool.keypoint.get_invVs(kpts)`
Keypoint shapes (oriented with the gravity vector)

`vtool.keypoint.get_kpts_dlen_sqrd(kpts, outer=False)`
returns diagonal length squared of keypoint extent

Parameters

- `kpts` (`ndarray[float32_t, ndim=2]`) – keypoints
- `outer` (`bool`) – loose if False tight if True

Returns `dlen_sqrd`

Return type `float`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts()
>>> dlen_sqrd = get_kpts_dlen_sqrd(kpts)
>>> result = '%.2f' % dlen_sqrd
>>> print(result)
3735.01
```

`vtool.keypoint.get_kpts_eccentricity(kpts)`

SeeAlso: `pyhesaff.tests.test_ellipse`

References

[https://en.wikipedia.org/wiki/Eccentricity_\(mathematics\)](https://en.wikipedia.org/wiki/Eccentricity_(mathematics))

Notes

For an ellipse/hyperbola the eccentricity is $\sqrt{1 - (b^2 / a^2)}$

Eccentricity is undefined for parabolas

where a is the length of the semi-major axis and b is the length of the semi minor axis. The length of the semi-major axis is 2 times the largest eigenvalue. And the length of the semi-minor axis is 2 times the smallest eigenvalue.

Parameters

- **kpts** (`ndarray[float32_t, ndim=2]`) – keypoints
- **offset** (`tuple`) – (default = (0.0, 0.0))
- **scale_factor** (`float`) – (default = 1.0)

CommandLine: `python -m vtool.keypoint --exec-get_kpts_eccentricity --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts_ = vt.demodata.get_dummy_kpts()
>>> kpts = np.append(kpts_, [[10, 10, 5, 0, 5, 0]], axis=0)
>>> ecc = get_kpts_eccentricity(kpts)
>>> result = 'ecc = %s' % (ub.repr2(ecc, precision=2))
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> colors = pt.scores_to_color(ecc)
>>> pt.draw_kpts2(kpts, color=colors, ell_linewidth=6)
>>> extent = vt.get_kpts_image_extent(kpts)
>>> ax = pt.gca()
>>> pt.set_axis_extent(extent, ax)
```

(continues on next page)

(continued from previous page)

```
>>> pt.dark_background()
>>> pt.colorbar(ecc, colors)
>>> ut.show_if_requested()
ecc = np.array([ 0.96, 0.99, 0.87, 0.91, 0.55, 0.  ])
```

`vtool.keypoint.get_kpts_image_extent(kpts, outer=False, only_xy=False)`

returns the width and height of keypoint bounding box This combines xy and shape information Does not take into account if keypoint extent goes under (0, 0)

Parameters

- **kpts** (`ndarray[float32_t, ndim=2][ndims=2]`) – keypoints
- **outer** – uses outer rectangle if True. Set to false for a tighter extent.

Returns (minx, maxx, miny, maxy)

Return type `tuple`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts()
>>> extent = get_kpts_image_extent(kpts, outer=False)
>>> result = ub.repr2(np.array(extent), precision=2)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.draw_kpts2(kpts, bbox=True)
>>> ax = pt.gca()
>>> pt.set_axis_extent(extent, ax)
>>> ut.show_if_requested()
np.array([ 14.78, 48.05, 0.32, 51.58])
```

`vtool.keypoint.get_kpts_strs(kpts)`

`vtool.keypoint.get_kpts_wh(kpts, outer=True)`

Gets the width / height diameter of a keypoint ie the diameter of the xaxis and yaxis of the keypoint.

Parameters

- **kpts** (`ndarray[float32_t, ndim=2][ndims=2]`) – keypoints
- **outer** (`bool`) – if True returns wh of bounding box. This is useful because extracting a patch needs a rectangle. If false it returns the otherwise gets the extent of the ellipse.

Returns (2xN) column1 is X extent and column2 is Y extent

Return type `ndarray`

Ignore:

```
>>> # Determine formula for min/maxing x and y
>>> import sympy
>>> x, y = sympy.symbols('x, y', real=True)
>>> a, d = sympy.symbols('a, d', real=True, positive=True)
```

(continues on next page)

(continued from previous page)

```

>>> c = sympy.symbols('c', real=True)
>>> theta = sympy.symbols('theta', real=True, nonnegative=True)
>>> xeqn = sympy.Eq(x, a * sympy.cos(theta))
>>> yeqn = sympy.Eq(y, c * sympy.sin(theta) + v * d)
>>> dxdt = sympy.solve(sympy.diff(xeqn, theta), 0)
>>> dydt = sympy.solve(sympy.diff(yeqn, theta), 0)
>>>
>>> # Ugg, cant get sympy to do trig derivative, do it manually
>>> dxdt = -a * sin(theta)
>>> dydt = d * cos(theta) - c * sin(theta)
>>> critical_thetas = solve(Eq(dxdt, 0), theta)
>>> critical_thetas += solve(Eq(dydt, 0), theta)
>>> [a, _, c, d] = invV.ravel()
>>> critical_thetas = [
>>>     0, np.pi,
>>>     -2 * np.arctan((c + np.sqrt(c ** 2 + d ** 2)) / d),
>>>     -2 * np.arctan((c - np.sqrt(c ** 2 + d ** 2)) / d),
>>> ]
>>> critical_uvs = np.vstack([np.cos(critical_thetas),
>>>                             np.sin(critical_thetas)])
>>> critical_xys = invV.dot(critical_uvs)

```

SeeAlso: get_kpts_major_minor

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts()[0:5]
>>> kpts[:, 0] += np.arange(len(kpts)) * 30
>>> kpts[:, 1] += np.arange(len(kpts)) * 30
>>> xyexnts = get_kpts_wh(kpts)
>>> result = ub.repr2(xyexnts)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.cla()
>>> pt.draw_kpts2(kpts, color='red', ell_linewidth=6, rect=True)
>>> ax = pt.gca()
>>> extent = np.array(get_kpts_image_extent(kpts))
>>> extent = vt.scale_extents(extent, 1.1)
>>> pt.set_axis_extent(extent, ax)
>>> xs, ys = vt.get_xys(kpts)
>>> radii = xyexnts / 2
>>> horiz_pts1 = np.array([(xs - radii.T[0]), ys]).T
>>> horiz_pts2 = np.array([(xs + radii.T[0]), ys]).T
>>> vert_pts1 = np.array([xs, (ys - radii.T[1])]).T
>>> vert_pts2 = np.array([xs, (ys + radii.T[1])]).T
>>> pt.draw_line_segments2(horiz_pts1, horiz_pts2, color='g')
>>> pt.draw_line_segments2(vert_pts1, vert_pts2, color='b')
>>> ut.show_if_requested()
np.array([[10.43315411, 58.5216589 ],
          [ 4.71017647, 58.5216589 ],
          [24.43314171, 45.09558868],

```

(continues on next page)

(continued from previous page)

```
[26.71114159, 63.47679138],
[32.10540009, 30.28536987]])
```

`vtool.keypoint.get_match_spatial_squared_error(kpts1, kpts2, H, fx2_to_fx1)`
transforms img2 to img2 and finds squared spatial error

Parameters

- **kpts1** (`ndarray[float32_t, ndim=2]`) – keypoints
- **kpts2** (`ndarray[float32_t, ndim=2]`) – keypoints
- **H** (`ndarray[float64_t, ndim=2]`) – homography/perspective matrix mapping image 1 to image 2 space
- **fx2_to_fx1** (`ndarray`) – has shape (nMatch, K)

Returns `fx2_to_xyerr_sqrd` has shape (nMatch, K)

Return type `ndarray`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> kpts1 = np.array([[ 129.83,  46.97,  15.84,   4.66,   7.24,   0. ],
...                  [ 137.88,  49.87,  20.09,   5.76,   6.2 ,   0. ],
...                  [ 115.95,  53.13,  12.96,   1.73,   8.77,   0. ],
...                  [ 324.88, 172.58, 127.69,  41.29,  50.5 ,   0. ],
...                  [ 285.44, 254.61, 136.06,  -4.77,  76.69,   0. ],
...                  [ 367.72, 140.81, 172.13,  12.99,  96.15,   0. ]],
dtype=np.float64)
>>> kpts2 = np.array([[ 318.93,  11.98,  12.11,   0.38,   8.04,   0. ],
...                  [ 509.47,  12.53,  22.4 ,   1.31,   5.04,   0. ],
...                  [ 514.03,  13.04,  19.25,   1.74,   4.72,   0. ],
...                  [ 490.19, 185.49,  95.67,  -4.84,  88.23,   0. ],
...                  [ 316.97, 206.07,  90.87,   0.07,  80.45,   0. ],
...                  [ 366.07, 140.05, 161.27, -47.01,  85.62,   0. ]],
dtype=np.float64)
>>> H = np.array([[ -0.70098,  0.12273,  5.18734],
...               [  0.12444, -0.63474, 14.13995],
...               [  0.00004,  0.00025, -0.64873]])
>>> fx2_to_fx1 = np.array([[5, 4, 1, 0],
...                        [0, 1, 5, 4],
...                        [0, 1, 5, 4],
...                        [2, 3, 1, 5],
...                        [5, 1, 0, 4],
...                        [3, 1, 5, 0]], dtype=np.int32)
>>> fx2_to_xyerr_sqrd = get_match_spatial_squared_error(kpts1, kpts2, H, fx2_to_
dtype=np.float64)
>>> fx2_to_xyerr = np.sqrt(fx2_to_xyerr_sqrd)
>>> result = ub.repr2(fx2_to_xyerr, precision=3)
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> kpts1 = np.array([[ 6.,  4., 15.84,  4.66,  7.24,  0. ],
...                  [ 9.,  3., 20.09,  5.76,  6.2 ,  0. ],
...                  [ 1.,  1., 12.96,  1.73,  8.77,  0. ],])
>>> kpts2 = np.array([[ 2.,  1., 12.11,  0.38,  8.04,  0. ],
...                  [ 5.,  1., 22.4 ,  1.31,  5.04,  0. ],
...                  [ 6.,  1., 19.25,  1.74,  4.72,  0. ],])
>>> H = np.array([[ 2, 0, 0],
...               [ 0, 1, 0],
...               [ 0, 0, 1]])
>>> fx2_to_fx1 = np.array([[2, 1, 0],
...                        [0, 1, 2],
...                        [2, 1, 0]], dtype=np.int32)
>>> fx2_to_xyerr_sqrd = get_match_spatial_squared_error(kpts1, kpts2, H, fx2_to_
↪fx1)
>>> fx2_to_xyerr = np.sqrt(fx2_to_xyerr_sqrd)
>>> result = ub.repr2(fx2_to_xyerr, precision=3)
>>> print(result)
```

`vtool.keypoint.get_ori_mats(kpts)`

Returns keypoint orientation matrixes

`vtool.keypoint.get_ori_strs(kpts)`

`vtool.keypoint.get_oris(kpts)`

Extracts keypoint orientations for kpts array

(in isotropic gaussian space relative to the gravity vector) (in simpler words: the orientation is taken from keypoints warped to the unit circle)

Parameters `kpts` (*ndarray*) – (N x 6) [x, y, a, c, d, theta]

Returns (*ndarray*) theta

`vtool.keypoint.get_scales(kpts)`

Gets average scale (does not take into account elliptical shape)

`vtool.keypoint.get_shape_strs(kpts)`

strings debugging and output

`vtool.keypoint.get_sqrd_scales(kpts)`

gets average squared scale (does not take into account elliptical shape)

Parameters `kpts` (*ndarray[`float32_t`, `ndim=2`]*) – keypoints

Returns `np.ndarray`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts()
>>> _scales_sqrd = get_sqrd_scales(kpts)
>>> result = (ub.repr2(_scales_sqrd, precision=2))
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> _scales_sqrd = get_sqrd_scales([])
>>> result = (ub.repr2(_scales_sqrd, precision=2))
>>> print(result)
```

`vtool.keypoint.get_transforms_from_patch_image_kpts(kpts, patch_shape, scale_factor=1.0)`

Given some patch (like a gaussian patch) transforms a patch to be overlayed on top of each keypoint in the image (adjusted for a scale factor)

Parameters

- **kpts** (`ndarray[float32_t, ndim=2]`) – keypoints
- **patch_shape** –
- **scale_factor** (`float`) –

Returns a list of 3x3 transformation matrices for each keypoint

Return type `M_list`

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts()
>>> patch_shape = (7, 7)
>>> scale_factor = 1.0
>>> M_list = get_transforms_from_patch_image_kpts(kpts, patch_shape, scale_
↳ factor)
>>> # verify results
>>> result = kpts_repr(M_list)
```

`vtool.keypoint.get_uneven_point_sample(kpts)`

for each keypoint returns an uneven sample of points along the elliptical boundaries.

Parameters **kpts** (`ndarray[float32_t, ndim=2]`) – keypoints

SeeAlso: `pyhesaff.tests.test_ellipse python -m pyhesaff.tests.test_ellipse --test-in_depth_ellipse --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts()[0:2]
>>> ellipse_pts1 = get_uneven_point_sample(kpts)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.draw_line_segments(ellipse_pts1)
>>> pt.set_title('uneven sample points')
>>> pt.show_if_requested()
```

`vtool.keypoint.get_xy_strs(kpts)`
strings debugging and output

`vtool.keypoint.get_xys(kpts)`
Keypoint locations in chip space

`vtool.keypoint.invert_invV_mats(invV_mats)`

Parameters `invV_mats` (`ndarray[float32_t, ndim=3]`) – keypoint shapes (possibly translation)

Returns `V_mats`

Return type `ndarray[float32_t, ndim=3]`

```
# Ignore: # >>> from vtool.keypoint import * # >>> invV_mats = np.array([[[ 18.00372824, 1.86434161, 32. ],
# >>> [ -0.61356842, 16.02202028, 27.2 ], # >>> [ 0. , 0. , 1. ]], # >>> # # >>> [[ 17.41989015, 2.51145917,
61. ], # >>> [ -2.94649591, 24.02540959, 22.9 ], # >>> [ 0. , 0. , 1. ]], # >>> # # >>> [[ 20.38098025,
0.88070646, 93.1 ], # >>> [ -0.93778675, 24.78261982, 23.6 ], # >>> [ 0. , 0. , 1. ]], # >>> # # >>> [[
16.25114793, -5.93213207, 120. ], # >>> [ 4.71295477, 21.80597527, 29.5 ], # >>> [ 0. , 0. , 1. ]], # >>> # #
>>> [[ 19.60863253, -11.43641248, 147. ], # >>> [ 8.45128003, 10.69925072, 42. ], # >>> [ 0. , 0. , 1. ]]])
# >>> ut.hash_data(invV_mats) # hcnoknyxgeecfyfrygblbvdeezmiulws # >>> V_mats = npl.inv(invV_mats) #
>>> ut.hash_data(V_mats) # yooneahjgcifojzpovddehyhtkkypldd
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts()
>>> invV_mats = vt.get_invVR_mats3x3(kpts)
>>> V_mats = invert_invV_mats(invV_mats)
>>> test = np.matmul(invV_mats, V_mats)
>>> # This should give us identity
>>> assert np.allclose(test, np.eye(3))
```

`vtool.keypoint.kp_cpp_infostr(kp)`
mirrors c++ debug code

`vtool.keypoint.kpts_docrepr(arr, name='arr', indent=True, *args, **kwargs)`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> np.random.seed(0)
>>> arr = np.random.rand(3, 3)
>>> args = tuple()
>>> kwargs = dict()
>>> result = kpts_docrepr(arr)
>>> # verify results
>>> print(result)
```

`vtool.keypoint.kpts_repr(arr, precision=2, suppress_small=True, linebreak=False)`

`vtool.keypoint.offset_kpts(kpts, offset=(0.0, 0.0), scale_factor=1.0)`
Transfoms keypoints by a scale factor and a translation

Parameters

- **kpts** (`ndarray[float32_t, ndim=2]`) – keypoints
- **offset** (`tuple`) –
- **scale_factor** (`float`) –

Returns kpts - keypoints

Return type `ndarray[float32_t, ndim=2]`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts().astype(np.float64)
>>> offset = (0.0, 0.0)
>>> scale_factor = (1.5, 0.5)
>>> kpts_ = offset_kpts(kpts, offset, scale_factor)
>>> # verify results (hack + 0. to fix negative 0)
>>> result = ut.repr3((kpts, kpts_ + 0.), precision=2, nobr=True, with_dtype=True)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.draw_kpts2(kpts, color=pt.ORANGE, ell_linewidth=6)
>>> pt.draw_kpts2(kpts_, color=pt.LIGHT_BLUE, ell_linewidth=4)
>>> extent1 = np.array(vt.get_kpts_image_extent(kpts))
>>> extent2 = np.array(vt.get_kpts_image_extent(kpts_))
>>> extent = vt.union_extents([extent1, extent2])
>>> ax = pt.gca()
>>> pt.set_axis_extent(extent)
>>> pt.dark_background()
>>> ut.show_if_requested()
np.array([[20. , 25. , 5.22, -5.11, 24.15, 0. ],
          [29. , 25. , 2.36, -5.11, 24.15, 0. ],
          [30. , 30. , 12.22, 12.02, 10.53, 0. ],
          [31. , 29. , 13.36, 17.63, 14.1 , 0. ],
          [32. , 31. , 16.05, 3.41, 11.74, 0. ]], dtype=np.float64),
np.array([[30. , 12.5 , 7.82, -2.56, 12.07, 0. ],
          [43.5 , 12.5 , 3.53, -2.56, 12.07, 0. ],
          [45. , 15. , 18.32, 6.01, 5.26, 0. ],
          [46.5 , 14.5 , 20.03, 8.82, 7.05, 0. ],
          [48. , 15.5 , 24.08, 1.7 , 5.87, 0. ]], dtype=np.float64),
```

`vtool.keypoint.rectify_invV_mats_are_up(invVR_mats)`

Useful if `invVR_mats` is no longer lower triangular rotates affine shape matrixes into downward (lower triangular) position

CommandLine: `python -m vtool.keypoint --exec-rectify_invV_mats_are_up --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
```

(continues on next page)

(continued from previous page)

```

>>> rng = np.random.RandomState(0)
>>> kpts = vt.demodata.get_dummy_kpts()[0:2]
>>> # Shrink x and y scales a bit
>>> kpts.T[2:4] /= 2
>>> kpts[1][3] *= 3 # increase skew
>>> # Set random orientation
>>> kpts.T[5] = TAU * np.array([.2, .6])
>>> invVR_mats = get_invVR_mats3x3(kpts)
>>> invVR_mats2, oris = rectify_invV_mats_are_up(invVR_mats)
>>> kpts2 = flatten_invV_mats_to_kpts(invVR_mats2)
>>> # Scale down in y a bit
>>> kpts2.T[1] += 100
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.show_kpts(np.vstack([kpts, kpts2]), ori=1, eig=True,
>>>               ori_color='green', rect=True)
>>> # Redraw oriented to show difference
>>> pt.draw_kpts2(kpts2, color='red', ell_linewidth=2, ori=1,
>>>               eig=True, ori_color='green', rect=True)
>>> ax = pt.gca()
>>> ax.set_aspect('auto')
>>> pt.dark_background()
>>> ut.show_if_requested()

```

```

pt.figure(docIf=True, fnum=pt.ensure_fnum(None)) ax = pt.gca() #ax.invert_yaxis() #pt.draw_kpts2(kpts,
color='blue', ell_linewidth=3, ori=1, eig=True, ori_color='green', rect=True) pt.draw_kpts2(kpts2,
color='red', ell_linewidth=2, ori=1, eig=True, ori_color='green', rect=True) extents =
np.array(vt.get_kpts_image_extent(np.vstack([kpts, kpts2]))) pt.set_axis_extent(extent, ax)
pt.dark_background() ut.show_if_requested()

```

Example

```

>>> from vtool.keypoint import * # NOQA
>>> rng = np.random.RandomState(0)
>>> invVR_mats = rng.rand(1000, 2, 2).astype(np.float64)
>>> output = rectify_invV_mats_are_up(invVR_mats)
>>> print(ut.hash_data(output))
oxvrkuiaffukpyalgxyhqikxgbuesutz

```

Ignore: `_invRs_2x2 = invVR_mats[:, 0:2, 0:2][0:1]` `A = _invRs_2x2[0]` `Q, R = np.linalg.qr(A)`

```

invVR_mats2, oris = rectify_invV_mats_are_up(_invRs_2x2[0:1]) L2, ori2 = invVR_mats2[0], oris[0] Q2
= vt.rotation_mat2x2(ori2)

```

```

np.linalg.det(Q)

```

```

vecs = np.random.rand(2, 4) Q2.dot(vecs) Q.dot(vecs)

```

```

np.linalg.cholesky(_invR_2x2)

```

`vtool.keypoint.transform_kpts(kpts, M)`

returns `M.dot(kpts_mat)` Currently, only works if `M` is affine.

Parameters

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

- **M** (*ndarray*) – affine transform matrix

Returns *ndarray*

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts()
>>> M = np.array([[10, 0, 0], [10, 10, 0], [0, 0, 1]], dtype=np.float64)
>>> kpts = transform_kpts(kpts, M)
>>> # verify results
>>> result = ub.repr2(kpts, precision=3, with_dtype=True).replace('-0. ', ' 0. ')
↪
```

`vtool.keypoint.transform_kpts_to_imgspace(kpts, bbox, bbox_theta, chipsz)`

Transforms keypoints so they are plotable in imagespace *kpts* - xyacdo keypoints *bbox* - chip bounding boxes in image space *theta* - chip rotations *invC* *chipsz* - chip extent (in keypoint / chip space)

`vtool.keypoint.transform_kpts_xys(H, kpts)`

Parameters

- **kpts** (*ndarray*[*float32_t*, *ndim*=2]) – keypoints
- **H** (*ndarray*[*float64_t*, *ndim*=2]) – homography/perspective matrix

Returns *xy_t*

Return type *ndarray*

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts()
>>> H = np.array([[ 3.,  3.,  5.],
...               [ 2.,  3.,  6.],
...               [ 1.,  1.,  2.]])
>>> xy_t = transform_kpts_xys(H, kpts)
>>> # verify results
```

1.27 vtool.linalg module

TODO: Look at this file <http://www.lfd.uci.edu/~gohlke/code/transformations.py.html>

```
# Ignore: # >>> # https://groups.google.com/forum/#!topic/sympy/k1HnZK_bNNA # >>> import vtool as vt # >>>
import sympy # >>> from sympy.abc import theta # >>> x, y, a, c, d, sx, sy = sympy.symbols('x y a c d, sx, sy') # >>>
R = vt.sympy_mat(vt.rotation_mat3x3(theta, sin=sympy.sin, cos=sympy.cos)) # >>> vt.evalprint('R') # >>> #eval-
print('R.inv()') # >>> vt.evalprint('sympy.simplify(R.inv())') # >>> #evalprint('sympy.simplify(R.inv().subs(theta,
4))') # >>> #print('——') # >>> #invR = sympy_mat(vt.rotation_mat3x3(-theta, sin=sympy.sin, cos=sympy.cos))
# >>> #evalprint('invR') # >>> #evalprint('invR.inv()') # >>> #evalprint('sympy.simplify(invR)') # >>> #eval-
print('sympy.simplify(invR.subs(theta, 4))') # >>> print('——') # >>> T = vt.sympy_mat(vt.translation_mat3x3(x,
```

```
y, None)) # >>> vt.evalprint('T') # >>> vt.evalprint('T.inv()') # >>> print('——') # >>> S =
vt.sympy_mat(vt.scale_mat3x3(sx, sy, dtype=None)) # >>> vt.evalprint('S') # >>> vt.evalprint('S.inv()') #
>>> print('——') # >>> print('LaTeX') # >>> print(ut.align('\n'.join(sympy.latex(R).split(r')).replace('{matrix}',
'{matrix}n'), '&'))
```

```
vtool.linalg.add_homogenous_coordinate(_xys)
```

CommandLine: python -m vtool.linalg -test-add_homogenous_coordinate

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.linalg import * # NOQA
>>> _xys = np.array([[ 2.,  0.,  0.,  2.],
...                 [ 2.,  2.,  0.,  0.]], dtype=np.float32)
>>> _xyzs = add_homogenous_coordinate(_xys)
>>> assert np.all(_xys == remove_homogenous_coordinate(_xyzs))
>>> result = ub.repr2(_xyzs, with_dtype=True)
>>> print(result)
```

```
vtool.linalg.affine_around_mat3x3(x, y, sx=1.0, sy=1.0, theta=0.0, shear=0.0, tx=0.0, ty=0.0,
                                x2=None, y2=None)
```

Executes an affine transform around center point (x, y). Equivalent to translation.dot(affine).dot(inv(translation))

Parameters

- **x** (*float*) – center x location in input space
- **y** (*float*) – center y location in input space
- **sx** (*float*) – x scale factor (default = 1)
- **sy** (*float*) – y scale factor (default = 1)
- **theta** (*float*) – counter-clockwise rotation angle in radians (default = 0)
- **shear** (*float*) – counter-clockwise shear angle in radians (default = 0)
- **tx** (*float*) – x-translation (default = 0)
- **ty** (*float*) – y-translation (default = 0)
- **x2** (*float, optional*) – center x location in output space (default = x)
- **y2** (*float, optional*) – center y location in output space (default = y)

CommandLine: python -m vtool.linalg affine_around_mat3x3 -show

CommandLine: xdoctest -m ~/code/vtool/vtool/linalg.py affine_around_mat3x3

Example

```
>>> from vtool.linalg import * # NOQA
>>> import vtool as vt
>>> orig_pts = np.array(vt.verts_from_bbox([10, 10, 20, 20]))
>>> x, y = vt.bbox_center(vt.bbox_from_verts(orig_pts))
>>> sx, sy = 0.5, 1.0
>>> theta = 1 * np.pi / 4
>>> shear = .1 * np.pi / 4
>>> tx, ty = 5, 0
```

(continues on next page)

(continued from previous page)

```

>>> x2, y2 = None, None
>>> Aff = affine_around_mat3x3(x, y, sx, sy, theta, shear,
>>>                             tx, ty, x2, y2)
>>> trans_pts = vt.transform_points_with_homography(Aff, orig_pts.T).T
>>> # xdoc: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.ensureqt()
>>> pt.plt.plot(x, y, 'bx', label='center')
>>> pt.plt.plot(orig_pts.T[0], orig_pts.T[1], 'b-', label='original')
>>> pt.plt.plot(trans_pts.T[0], trans_pts.T[1], 'r-', label='transformed')
>>> pt.plt.legend()
>>> pt.plt.title('Demo of affine_around_mat3x3')
>>> pt.plt.axis('equal')
>>> pt.plt.xlim(0, 40)
>>> pt.plt.ylim(0, 40)
>>> ut.show_if_requested()

```

Ignore:

```

>>> from vtool.linalg import * # NOQA
>>> x, y, sx, sy, theta, shear, tx, ty, x2, y2 = (
>>>     256.0, 256.0, 1.5, 1.0, 0.78, 0.2, 0, 100, 500.0, 500.0)
>>> for timer in ub.Timerit(1000, 'old'): # 19.0697 µs
>>>     with timer:
>>>         tr1_ = translation_mat3x3(-x, -y)
>>>         Aff_ = affine_mat3x3(sx, sy, theta, shear, tx, ty)
>>>         tr2_ = translation_mat3x3(x2, y2)
>>>         Aff1 = tr2_.dot(Aff_).dot(tr1_)
>>> for timer in ub.Timerit(1000, 'new'): # 11.0242 µs
>>>     with timer:
>>>         Aff2 = affine_around_mat3x3(x, y, sx, sy, theta, shear,
>>>                                     tx, ty, x2, y2)
>>> assert np.all(np.isclose(Aff2, Aff1))

```

Ignore:

```

>>> from vtool.linalg import * # NOQA
>>> import vtool as vt
>>> import sympy
>>> # Shows the symbolic construction of the code
>>> # https://groups.google.com/forum/#!topic/sympy/k1HnZK_bNNA
>>> from sympy.abc import theta
>>> x, y, sx, sy, theta, shear, tx, ty, x2, y2 = sympy.symbols(
>>>     'x, y, sx, sy, theta, shear, tx, ty, x2, y2')
>>> theta = sx = sy = tx = ty = 0
>>> # move to center xy, apply affine transform, move center xy2
>>> tr1_ = translation_mat3x3(-x, -y, dtype=None)
>>> Aff_ = affine_mat3x3(sx, sy, theta, shear, tx, ty, trig=sympy)
>>> tr2_ = translation_mat3x3(x2, y2, dtype=None)
>>> # combine transformations
>>> Aff = vt.sympy_mat(tr2_.dot(Aff_).dot(tr1_))
>>> vt.evalprint('Aff')
>>> print('-----')
>>> print('Numpy')
>>> vt.sympy_numpy_repr(Aff)

```

```
vtool.linalg.affine_mat3x3 (sx=1, sy=1, theta=0, shear=0, tx=0, ty=0, trig=<module 'numpy'
                             from '/home/docs/checkouts/readthedocs.org/user_builds/wbia-
                             vtool/envs/stable/lib/python3.7/site-packages/numpy/__init__.py'>)
```

Parameters

- **sx** (*float*) – x scale factor (default = 1)
- **sy** (*float*) – y scale factor (default = 1)
- **theta** (*float*) – rotation angle (radians) in counterclockwise direction
- **shear** (*float*) – shear angle (radians) in counterclockwise directions
- **tx** (*float*) – x-translation (default = 0)
- **ty** (*float*) – y-translation (default = 0)

References

https://github.com/scikit-image/scikit-image/blob/master/skimage/transform/_geometric.py

`vtool.linalg.det_ltri (ltri)`

Lower triangular determinant

`vtool.linalg.dot_ltri (ltri1, ltri2)`

Lower triangular dot product

`vtool.linalg.gauss2d_pdf (x_, y_, sigma=None, mu=None)`

Parameters

- **x** – x coordinate of a 2D Gaussian
- **y** – y coordinate of a 2D Gaussian
- **sigma** – covariance of vector
- **mu** – mean of vector

Returns float - The probability density at that point

`vtool.linalg.inv_ltri (ltri, det)`

Lower triangular inverse

`vtool.linalg.normalize (arr, ord=None, axis=None, out=None)`

Returns all row vectors normalized by their magnitude.

Parameters

- **arr** (*ndarray*) – row vectors to normalize
- **ord** (*int*) – type of norm to use (defaults to 2-norm) {non-zero int, inf, -inf}
- **axis** (*int*) – axis to normalize
- **out** (*ndarray*) – preallocated output

SeeAlso: `np.linalg.norm`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.linalg import * # NOQA
>>> arr = np.array([[1, 2, 3, 4, 5], [2, 2, 2, 2, 2]])
>>> arr_normed = normalize(arr, axis=1)
>>> result = ub.hzcat(['arr_normed = ', ub.repr2(arr_normed, precision=2, with_
↳dtype=True)])
>>> assert np.allclose((arr_normed ** 2).sum(axis=1), [1, 1])
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.linalg import * # NOQA
>>> arr = np.array([ 0.6, 0.1, -0.6])
>>> arr_normed = normalize(arr)
>>> result = ub.hzcat(['arr_normed = ', ub.repr2(arr_normed, precision=2)])
>>> assert np.allclose((arr_normed ** 2).sum(), [1])
>>> print(result)
```

Example

```
>>> from vtool.linalg import * # NOQA
>>> ord_list = [0, 1, 2, np.inf, -np.inf]
>>> arr = np.array([ 0.6, 0.1, -0.5])
>>> normed = [(ord, normalize(arr, ord=ord)) for ord in ord_list]
>>> result = ub.repr2(normed, precision=2, with_dtype=True)
>>> print(result)
```

`vtool.linalg.normalize_rows(arr, out=None)`
DEPRICATE

`vtool.linalg.random_affine_args(zoom_pdf=None, tx_pdf=None, ty_pdf=None, shear_pdf=None, theta_pdf=None, enable_flip=False, enable_stretch=False, default_fault_distribution='uniform', scalar_anchor='reflect', txy_pdf=None, rng=<module 'numpy.random' from '/home/docs/checkouts/readthedocs.org/user_builds/wbia-vtool/envs/stable/lib/python3.7/site-packages/numpy/random/__init__.py'>)`

TODO: allow for a pdf of ranges for each dimension

If pdfs are tuples it is interpreted as a default (uniform) distribution between the two points. A single scalar is a default distribution between -scalar and scalar.

Parameters

- **zoom_range** (*tuple*) – (default = (1.0, 1.0))
- **tx_range** (*tuple*) – (default = (0.0, 0.0))
- **ty_range** (*tuple*) – (default = (0.0, 0.0))
- **shear_range** (*tuple*) – (default = (0, 0))
- **theta_range** (*tuple*) – (default = (0, 0))
- **enable_flip** (*bool*) – (default = False)

- **enable_stretch** (*bool*) – (default = False)
- **rng** (*module*) – random number generator (default = numpy.random)

Returns affine_args

Return type tuple

CommandLine: xdoctest -m ~/code/vtool/vtool/linalg.py random_affine_args

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.linalg import * # NOQA
>>> import vtool as vt
>>> zoom_range = (0.9090909090909091, 1.1)
>>> tx_pdf = (0.0, 4.0)
>>> ty_pdf = (0.0, 4.0)
>>> shear_pdf = (0, 0)
>>> theta_pdf = (0, 0)
>>> enable_flip = False
>>> enable_stretch = False
>>> rng = np.random.RandomState(0)
>>> affine_args = random_affine_args(
>>>     zoom_range, tx_pdf, ty_pdf, shear_pdf, theta_pdf,
>>>     enable_flip, enable_stretch, rng=rng)
>>> print('affine_args = %s' % (ub.repr2(affine_args),))
>>> (sx, sy, theta, shear, tx, ty) = affine_args
>>> Aff = vt.affine_mat3x3(sx, sy, theta, shear, tx, ty)
>>> result = ub.repr2(Aff, precision=3, nl=1, with_dtype=0)
>>> print(result)
np.array([[ 1.009, -0.    ,  1.695],
          [ 0.    ,  1.042,  2.584],
          [ 0.    ,  0.    ,  1.    ]])
```

`vtool.linalg.random_affine_transform(*args, **kwargs)`

`vtool.linalg.remove_homogenous_coordinate(_xyzs)`

normalizes 3d homogenous coordinates into 2d coordinates

Parameters `_xyzs` (*ndarray*) – of shape (3, N)

Returns `_xys` of shape (2, N)

Return type ndarray

CommandLine: python -m vtool.linalg -test-remove_homogenous_coordinate

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.linalg import * # NOQA
>>> _xyzs = np.array([[ 2.,  0.,  0.,  2.],
...                  [ 2.,  2.,  0.,  0.],
...                  [ 1.2,  1.,  1.,  2.]], dtype=np.float32)
>>> _xys = remove_homogenous_coordinate(_xyzs)
>>> result = ub.repr2(_xys, precision=3, with_dtype=True)
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.linalg import * # NOQA
>>> _xyzs = np.array([[ 140.,  167.,  185.,  185.,  194.],
...                  [ 121.,  139.,  156.,  155.,  163.],
...                  [  47.,   56.,   62.,   62.,   65.]])
>>> _xys = remove_homogenous_coordinate(_xyzs)
>>> result = ub.repr2(_xys, precision=3)
>>> print(result)
```

```
vtool.linalg.rotation_around_bbox_mat3x3(theta, bbox0, bbox1=None)
vtool.linalg.rotation_around_mat3x3(theta, x0, y0, x1=None, y1=None)
vtool.linalg.rotation_mat2x2(theta)
vtool.linalg.rotation_mat3x3(radians, sin=<ufunc 'sin'>, cos=<ufunc 'cos'>)
```

References

https://en.wikipedia.org/wiki/Rotation_matrix

```
vtool.linalg.scale_around_mat3x3(sx, sy, x, y)
vtool.linalg.scale_mat3x3(sx, sy=None, dtype=<class 'numpy.float64'>)
vtool.linalg.shear_mat3x3(shear_x, shear_y, dtype=<class 'numpy.float64'>)
vtool.linalg.svd(M)
```

Parameters **M** (*ndarray*) – must be either float32 or float64

Returns (U, s, Vt)

Return type *tuple*

CommandLine: python -m vtool.linalg --test-svd

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.linalg import * # NOQA
>>> # build test data
>>> M = np.array([1, 2, 3], dtype=np.float32)
>>> M = np.array([[20.5812, 0], [3.615, 17.1295]], dtype=np.float64)
>>> # execute function
>>> (U, s, Vt) = svd(M)
```

Ignore: flags = cv2.SVD_FULL_UV %timeit cv2.SVDcomp(M, flags=flags) %timeit npl.svd(M)

```
vtool.linalg.transform_around(M, x, y)
    translates to origin, applies transform and then translates back
vtool.linalg.transform_points_with_homography(H, _xys)
```

Parameters

- **H** (*ndarray* [*float64_t*, *ndim*=2]) – homography/perspective matrix

- `_xys` (`ndarray [ndim=2]`) – (2 x N) array

`vtool.linalg.translation_mat3x3` (`x, y, dtype=<class 'numpy.float64'>`)

`vtool.linalg.whiten_xy_points` (`xy_m`)

whitens points to mean=0, stddev=1 and returns transformation

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.linalg import * # NOQA
>>> from vtool import demodata
>>> xy_m = demodata.get_dummy_xy()
>>> tup = whiten_xy_points(xy_m)
>>> xy_norm, T = tup
>>> result = (ub.hash_data(tup))
>>> print(result)
```

1.28 vtool.matching module

`vt python -m utool.util_inspect check_module_usage -pat="matching.py"`

class `vtool.matching.AnnotPairFeatInfo` (`columns=None, importances=None`)

Bases: `object`

Information class about feature dimensions of PairwiseMatch.

Notes

Can be used to compute marginal importances over groups of features used in the pairwise one-vs-one scoring algorithm

Can be used to construct an appropriate `cfgdict` for a new PairwiseMatch.

CommandLine: `python -m vtool.matching AnnotPairFeatInfo`

Example

```
>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.matching import * # NOQA
>>> import vtool as vt
>>> match = demodata_match({})
>>> match.add_global_measures(['time', 'gps'])
>>> index = pd.MultiIndex.from_tuples([(1, 2)], names=('aid1', 'aid2'))
>>> # Feat info without bins
>>> feat = match.make_feature_vector()
>>> X = pd.DataFrame(feat, index=index)
>>> print(X.keys())
>>> featinfo = AnnotPairFeatInfo(X)
>>> pairfeat_cfg, global_keys = featinfo.make_pairfeat_cfg()
>>> print('pairfeat_cfg = %r' % (pairfeat_cfg,))
>>> print('global_keys = %r' % (global_keys,))
>>> assert 'delta' not in global_keys
```

(continues on next page)

(continued from previous page)

```

>>> assert 'max' not in global_keys
>>> ut.cprint(featinfo.get_infostr(), 'blue')
>>> # Feat info with bins
>>> feat = match.make_feature_vector(indices=0, bins=[.7, .8], bin_key='ratio')
>>> X = pd.DataFrame(feat, index=index)
>>> print(X.keys())
>>> featinfo = AnnotPairFeatInfo(X)
>>> pairfeat_cfg, global_keys = featinfo.make_pairfeat_cfg()
>>> print('pairfeat_cfg = %s' % (ut.repr4(pairfeat_cfg),))
>>> print('global_keys = %r' % (global_keys,))
>>> ut.cprint(featinfo.get_infostr(), 'blue')

```

```
binsum_fmt = '{op}({measure}[{bin_key}<{binval}])'
```

```
dimkey_grammar()
```

CommandLine: python -m vtool.matching AnnotPairFeatInfo.dimkey_grammar

Example

```

>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.matching import * # NOQA
>>> import vtool as vt
>>> match = demodata_match({})
>>> match.add_global_measures(['view', 'qual', 'gps', 'time'])
>>> index = pd.MultiIndex.from_tuples([(1, 2)], names=('aid1', 'aid2'))
>>> # Feat info without bins
>>> feat = match.make_feature_vector()
>>> X = pd.DataFrame(feat, index=index)
>>> featinfo = AnnotPairFeatInfo(X)
>>> feat_grammar = featinfo.dimkey_grammar()
>>> for key in X.keys():
>>>     print(key)
>>>     print(feat_grammar.parseString(key))

```

feature (*key*)

find (*group_id*, *op*, *value*, *hack=False*)

groupid options: *measure_type* *global_measure*, *local_sorter*, *local_rank*, *local_measure*, *summary_measure*, *summary_op*, *summary_bin*, *summary_binval*, *summary_binkey*,

Ignore: *group_id* = 'summary_op' *op* = '==' *value* = 'len'

get_infostr ()

Summarizes the types (global, local, summary) of features in X based on standardized dimension names.

global_measure (*key*)

group_counts (*item*)

group_importance (*item*)

```
loc_fmt = 'loc[{sorter},{rank}]({measure})'
```

local_measure (*key*)

local_rank (*key*)

local_sorter (*key*)

```
make_pairfeat_cfg()
measure(key)
measure_type(key)
print_counts(group_id)
print_margins(group_id, ignore_trivial=True)
rrr(verbose=True, reload_module=True)
    special class reloading function This function is often injected as rrr of classes
select_columns(criteria, op='and')
```

Parameters **criteria** (*list*) – list of tokens denoting selection constraints can be one of:
measure_type global_measure, local_sorter, local_rank, local_measure, summary_measure,
summary_op, summary_bin, summary_binval, summary_binkey,

Ignore:

```
>>> featinfo.select_columns([
>>>     ('measure_type', '=', 'local'),
>>>     ('local_sorter', 'in', ['weighted_ratio_score', 'lnbnn_norm_dist
↪']),
>>> ], op='and')
```

```
sum_fmt = '{op}({measure})'
summary_binkey(key)
summary_binval(key)
summary_measure(key)
summary_op(key)

class vtool.matching.AssignTup(fm, match_dist, norm_fx1, norm_dist)
    Bases: tuple
    fm
        Alias for field number 0
    match_dist
        Alias for field number 1
    norm_dist
        Alias for field number 3
    norm_fx1
        Alias for field number 2

exception vtool.matching.MatchingError
    Bases: Exception

class vtool.matching.PairwiseMatch(annot1=None, annot2=None)
    Bases: ubelt.util_mixins.NiceRepr
    Newest (Sept-16-2016) object oriented one-vs-one matching interface
    Creates an object holding two annotations Then a pipeline of operations can be applied to generate score and
    refine the matches
```

Note: The annotation dictionaries are required to have certain attributes.

Required annotation attributes: (kpts, vecs) OR rchip OR rchip_fpath

Optional annotation attributes: aid, nid, flann, rchip, dlen_sqrd, weight

Ignore:

```
>>> from vtool.matching import * # NOQA
>>> import vtool as vt
>>> imgR = vt.imread(ut.grab_test_imgpath('easy1.png'))
>>> imgL = vt.imread(ut.grab_test_imgpath('easy2.png'))
>>> annot1 = {'rchip': imgR}
>>> annot2 = {'rchip': imgL}
>>> match = vt.PairwiseMatch(annot1, annot2)
>>> match.apply_all({'refine_method': 'affine', 'affine_invariance': False,
↳ 'rotation_invariance': False})
>>> dsize = imgR.shape[0:2][::-1]
>>> imgR_warp = vt.warpHomog(imgR, match.H_12, dsize)
>>> # xdoctest: +REQUIRES(--show)
>>> import kwplot
>>> kwplot.autompl()
>>> kwplot.imshow(imgL, pnum=(2, 1, 1))
>>> kwplot.imshow(imgR_warp, pnum=(2, 1, 2))
>>> kwplot.imshow(imgL, pnum=(2, 1, 1))
>>> kwplot.imshow(imgR_warp, pnum=(2, 1, 2))
>>> # xdoctest: +REQUIRES(--gui)
>>> import wbia.guitool as gt
>>> gt.ensure_qapp()
>>> match.ishow()
>>> from vtool.matching import * # NOQA
>>> import vtool as vt
>>> imgR = vt.imread(ut.grab_test_imgpath('easy1.png'))
>>> imgL = vt.imread(ut.grab_test_imgpath('easy2.png'))
>>> annot1 = {'rchip': imgR}
>>> annot2 = {'rchip': imgL}
>>> match = vt.PairwiseMatch(annot1, annot2)
>>> match.apply_all({'refine_method': 'affine', 'affine_invariance': False,
↳ 'rotation_invariance': False})
>>> dsize = imgR.shape[0:2][::-1]
>>> imgR_warp = vt.warpHomog(imgR, match.H_12, dsize)
>>> # xdoctest: +REQUIRES(--show)
>>> import kwplot
>>> kwplot.autompl()
>>> kwplot.imshow(imgL, pnum=(2, 1, 1))
>>> kwplot.imshow(imgR_warp, pnum=(2, 1, 2))
>>> kwplot.imshow(imgL, pnum=(2, 1, 1))
>>> kwplot.imshow(imgR_warp, pnum=(2, 1, 2))
>>> # xdoctest: +REQUIRES(--gui)
>>> import wbia.guitool as gt
>>> gt.ensure_qapp()
>>> match.ishow()
```

add_global_measures (global_keys)

add_local_measures (xy=True, scale=True)

apply_all (cfgdict)

apply_ratio_test (cfgdict={}, inplace=None)

apply_sver (cfgdict={}, inplace=None)

Ignore:

```
>>> from vtool.matching import * # NOQA
>>> cfgdict = {'symmetric': True, 'ratio_thresh': .8,
>>>             'thresh_bins': [.5, .6, .7, .8]}
>>> match = demodata_match(cfgdict, apply=False)
>>> match = match.assign(cfgbase)
>>> match.apply_ratio_test(cfgdict, inplace=True)
>>> flags1 = match.apply_sver(cfgdict)
```

assign (cfgdict={}, verbose=None)

Assign feature correspondences between annots

Example

```
>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.matching import * # NOQA
>>> cfgdict = {'symmetric': True}
>>> match = demodata_match({}, apply=False)
>>> m1 = match.copy().assign({'symmetric': False})
>>> m2 = match.copy().assign({'symmetric': True})
```

Example

```
>>> from vtool.matching import * # NOQA
>>> grid = {
>>>     'symmetric': [True, False],
>>> }
>>> for cfgdict in ut.all_dict_combinations(grid):
>>>     match = demodata_match(cfgdict, apply=False)
>>>     match.assign()
```

compress (flags, inplace=None)

copy ()

ishow ()

CommandLine: python -m vtool.matching ishow --show

Example

```
>>> # SCRIPT
>>> from vtool.matching import * # NOQA
>>> import vtool as vt
>>> import wbia.guitool as gt
>>> gt.ensure_qapp()
>>> match = demodata_match(use_cache=False)
>>> self = match.ishow()
>>> self.disp_config['show_homog'] = True
>>> self.update()
>>> # xdoctest: +REQUIRES(--show)
>>> gt.qtapp_loop(qwin=self, freq=10)
```

make_feature_vector (*local_keys=None, global_keys=None, summary_ops=None, sorters='ratio', indices=3, bin_key=None, bins=None*)

Constructs the pairwise feature vector that represents a match

Parameters

- **local_keys** (*None*) – (default = None)
- **global_keys** (*None*) – (default = None)
- **summary_ops** (*None*) – (default = None)
- **sorters** (*str*) – (default = 'ratio')
- **indices** (*int*) – (default = 3)

Returns feat

Return type *dict*

CommandLine: python -m vtool.matching make_feature_vector

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.matching import * # NOQA
>>> import vtool as vt
>>> match = demodata_match({})
>>> feat = match.make_feature_vector(indices=[0, 1])
>>> result = ('feat = %s' % (ub.repr2(feat, nl=2),))
>>> print(result)
```

matched_vecs2 ()

ratio_test_flags (cfgdict={})

show (ax=None, show_homog=False, show_ori=False, show_ell=True, show_pts=False, show_lines=True, show_rect=False, show_eig=False, show_all_kpts=False, mask_blend=0, ell_alpha=0.6, line_alpha=0.35, modifysize=False, vert=None, overlay=True, heatmask=False, line_lw=1.4)

sver_flags (cfgdict={}, return_extra=False)

Example

```
>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.matching import * # NOQA
>>> cfgdict = {'symmetric': True, 'newsym': True}
>>> match = demodata_match(cfgdict, apply=False)
>>> cfgbase = {'symmetric': True, 'ratio_thresh': .8}
>>> cfgdict = ut.dict_union(cfgbase, dict(thresh_bins=[.5, .6, .7, .8]))
>>> match = match.assign(cfgbase)
>>> match.apply_ratio_test(cfgdict, inplace=True)
>>> flags1 = match.sver_flags(cfgdict)
>>> flags2 = match.sver_flags(cfgbase)
```

take (indicies, inplace=None)

vtool.matching.assign_symmetric_matches (fx2_to_fx1, fx2_to_dist, fx1_to_fx2, fx1_to_dist, K, Knorm=None)

Ignore:

```
>>> import vtool as vt
>>> from vtool.matching import *
>>> K = 2
>>> Knorm = 1
>>> feat1 = np.random.rand(5, 3)
>>> feat2 = np.random.rand(7, 3)
>>>
>>> # Assign distances
>>> distmat = vt.L2(feat1[:, None], feat2[None, :])
>>>
>>> # Find nearest K
>>> fx1_to_fx2 = distmat.argsort()[:, 0:K + Knorm]
>>> fx2_to_fx1 = distmat.T.argsort()[:, 0:K + Knorm]
>>> # and order their distances
>>> fx1_to_dist = np.array([distmat[i].take(col) for i, col in enumerate(fx1_
↳to_fx2)])
>>> fx2_to_dist = np.array([distmat.T[j].take(row) for j, row in
↳enumerate(fx2_to_fx1)])
>>>
>>> # flat_matx1 = fx1_to_fx2 + np.arange(distmat.shape[0])[:, None] *
↳distmat.shape[1]
>>> # fx1_to_dist = distmat.take(flat_matx1).reshape(fx1_to_fx2.shape)
>>>
>>> fx21 = pd.DataFrame(fx2_to_fx1)
>>> fx21.columns.name = 'K'
>>> fx21.index.name = 'fx1'
>>>
>>> fx12 = pd.DataFrame(fx1_to_fx2)
>>> fx12.columns.name = 'K'
>>> fx12.index.name = 'fx2'
>>>
>>> fx12 = fx12.T[0:K].T.astype(np.float)
>>> fx21 = fx21.T[0:K].T.astype(np.float)
>>>
>>> fx12.values[~fx1_to_flags] = np.nan
>>> fx21.values[~fx2_to_flags] = np.nan
>>>
>>> print('fx12.values =\n%r' % (fx12,))
>>> print('fm_ =\n%r' % (fm_,))
>>>
>>> print('fx21.values =\n%r' % (fx21,))
>>> print('fm =\n%r' % (fm,))
>>>
>>> unflat_match_idx2 = -np.ones(fx2_to_fx1.shape)
>>> unflat_match_idx2.ravel()[flat_match_idx2] = flat_match_idx2
>>> inv_lookup21 = unflat_match_idx2.T[0:K].T
>>>
>>> for fx2 in zip(fx12.values[fx1_to_flags]:
>>>
>>> for fx1, fx2 in zip(match_fx1_, match_fx2_):
>>>     cx = np.where(fx2_to_fx1[fx2][0:K] == fx1)[0][0]
>>>     inv_idx = inv_lookup21[fx2][cx]
>>>     print('inv_idx = %r' % (inv_idx,))
```

`vtool.matching.assign_unconstrained_matches` (`fx2_to_fx1`, `fx2_to_dist`, `K`, `Knorm=None`,
`fx2_to_flags=None`)
 assigns vsone matches using results of nearest neighbors.

Ignore: `fx2_to_dist = np.arange(fx2_to_fx1.size).reshape(fx2_to_fx1.shape)`

CommandLine: `python -m vtool.matching -test-assign_unconstrained_matches -show python -m vtool.matching assign_unconstrained_matches:0 python -m vtool.matching assign_unconstrained_matches:1`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.matching import * # NOQA
>>> fx2_to_fx1, fx2_to_dist = empty_neighbors(0, 0)
>>> K = 1
>>> Knorm = 1
>>> fx2_to_flags = None
>>> assigtup = assign_unconstrained_matches(fx2_to_fx1, fx2_to_dist, K,
>>>                                       Knorm, fx2_to_flags)
>>> fm, match_dist, norm_fx1, norm_dist = assigtup
>>> result = ub.repr2(assigtup, precision=3, nobr=True, with_dtype=True)
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.matching import * # NOQA
>>> fx2_to_fx1 = np.array([[ 77,   971,  22],
>>>                        [116,   120,  34],
>>>                        [122,   128,  99],
>>>                        [1075,  692, 102],
>>>                        [ 530,   45, 120],
>>>                        [ 45,  530,  77]], dtype=np.int32)
>>> fx2_to_dist = np.array([[ 0.059,  0.238, .3],
>>>                        [ 0.021,  0.240, .4],
>>>                        [ 0.039,  0.247, .5],
>>>                        [ 0.149,  0.151, .6],
>>>                        [ 0.226,  0.244, .7],
>>>                        [ 0.215,  0.236, .8]], dtype=np.float32)
>>> K = 1
>>> Knorm = 1
>>> fx2_to_flags = np.array([[1, 1], [0, 1], [1, 1], [0, 1], [1, 1], [1, 1]])
>>> fx2_to_flags = fx2_to_flags[:, 0:K]
>>> assigtup = assign_unconstrained_matches(fx2_to_fx1, fx2_to_dist, K,
>>>                                       Knorm, fx2_to_flags)
>>> fm, match_dist, norm_fx1, norm_dist = assigtup
>>> result = ub.repr2(assigtup, precision=3, nobr=True, with_dtype=True)
>>> print(result)
>>> assert len(fm.shape) == 2 and fm.shape[1] == 2
>>> assert ub.allsame(list(map(len, assigtup)))
```

`vtool.matching.asymmetric_correspondence(annot1, annot2, K, Knorm, checks, allow_shrink=True)`

Find symmetric feature correspondences

`vtool.matching.csum(x)`

`vtool.matching.demodata_match(cfgdict={}, apply=True, use_cache=True, recompute=False)`

`vtool.matching.empty_assign()`

```
vtool.matching.empty_neighbors(num_vecs=0, K=0)
vtool.matching.ensure_metadata_dlen_sqrd(annot)
vtool.matching.ensure_metadata_feats(annot, cfgdict={})
    Adds feature evaluation keys to a lazy dictionary
```

Parameters

- **annot** (*utool.LazyDict*) –
- **suffix** (*str*) – (default = '')
- **cfgdict** (*dict*) – (default = {})

CommandLine: python -m vtool.matching --exec-ensure_metadata_feats

Example

```
>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.matching import * # NOQA
>>> rchip_fpath = ut.grab_test_imgpath('easy1.png')
>>> annot = ut.LazyDict({'rchip_fpath': rchip_fpath})
>>> cfgdict = {}
>>> ensure_metadata_feats(annot, cfgdict)
>>> assert len(annot._stored_results) == 1
>>> annot['kpts']
>>> assert len(annot._stored_results) >= 4
>>> annot['vecs']
>>> assert len(annot._stored_results) >= 5
```

```
vtool.matching.ensure_metadata_flann(annot, cfgdict)
    setup lazy flann evaluation
vtool.matching.ensure_metadata_normxy(annot, cfgdict={})
vtool.matching.ensure_metadata_vsone(annot1, annot2, cfgdict={})
vtool.matching.flag_sym_slow(fx1_to_fx2, fx2_to_fx1, K)
    Returns flags indicating if the matches in fx1_to_fx2 are reciprocal with the matches in fx2_to_fx1.
    Much slower version of flag_symmetric_matches, but more clear
vtool.matching.flag_symmetric_matches(fx2_to_fx1, fx1_to_fx2, K=2)
    Returns flags indicating if the matches in fx2_to_fx1 are reciprocal with the matches in fx1_to_fx2.
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.matching import * # NOQA
>>> K = 2
>>> fx2_to_fx1 = np.array([[ 0,  1], # 0
>>>                        [ 1,  4], # 1
>>>                        [ 3,  4], # 2
>>>                        [ 2,  3]], dtype=np.int32) # 3
>>> fx1_to_fx2 = np.array([[ 0, 1], # 0
>>>                        [ 2, 1], # 1
>>>                        [ 0, 1], # 2
```

(continues on next page)

(continued from previous page)

```

>>> [ 3, 1], # 3
>>> [ 0, 1]], dtype=np.int32) # 4
>>> fx2_to_flagsA = flag_symmetric_matches(fx2_to_fx1, fx1_to_fx2, K)
>>> fx2_to_flagsB = flag_sym_slow(fx2_to_fx1, fx1_to_fx2, K)
>>> assert np.all(fx2_to_flagsA == fx2_to_flagsB)
>>> result = ub.repr2(fx2_to_flagsB)
>>> print(result)

```

`vtool.matching.invsum(x)`

`vtool.matching.normalized_nearest_neighbors(flann1, vecs2, K, checks=800)`
 Computes matches from vecs2 to flann1.

uses flann index to return nearest neighbors with distances normalized between 0 and 1 using sifts uint8 trick

`vtool.matching.symmetric_correspondence(annot1, annot2, K, Knorm, checks, allow_shrink=True)`
 Find symmetric feature correspondences

`vtool.matching.testdata_annot_metadata(rchip_fpath, cfgdict={})`

1.29 vtool.nearest_neighbors module

Wrapper around flann (with caching)

python -c "import vtool, doctest; print(doctest.testmod(vtool.nearest_neighbors))"

class `vtool.nearest_neighbors.AnnoyWrapper`

Bases: `object`

Wrapper for annoy to use the FLANN api

build_index (*dvecs*, ***kwargs*)

nn_index (*qvecs*, *num_neighbs*, *checks=None*)

`vtool.nearest_neighbors.ann_flann_once(dpts, qpts, num_neighbors, flann_params={})`

Finds the approximate nearest neighbors of qpts in dpts

CommandLine: `xdoctest -m ~/code/vtool/vtool/nearest_neighbors.py ann_flann_once:0`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.nearest_neighbors import * # NOQA
>>> np.random.seed(1)
>>> dpts = np.random.randint(0, 255, (5, 128)).astype(np.uint8)
>>> qpts = np.random.randint(0, 255, (5, 128)).astype(np.uint8)
>>> qx2_dx, qx2_dist = ann_flann_once(dpts, qpts, 2)
>>> import ubelt as ub
>>> result = ub.repr2((qx2_dx.T, qx2_dist.T), precision=2, with_dtype=True, nl=2)
>>> print(result)
(
  np.array([[3, 3, 3, 3, 0],
            [2, 0, 1, 4, 4]], dtype=np.int32),
  np.array([[1037329., 1235876., 1168550., 1286435., 1075507.],

```

(continues on next page)

(continued from previous page)

```

[1038324., 1243690., 1304896., 1320598., 1369036.]], dtype=np.
↪float32),
)

```

Example

```

>>> # ENABLE_DOCTEST
>>> # Test upper bounds on sift descriptors
>>> # SeeAlso distance.understanding_pseudomax_props
>>> from vtool.nearest_neighbors import * # NOQA
>>> import vtool as vt
>>> import numpy as np
>>> np.random.seed(1)
>>> # get points on unit sphere
>>> nDpts = 5000 # 5
>>> nQpts = 10000 # 10
>>> dpts = vt.normalize_rows(np.random.rand(nDpts, 128))
>>> qpts = vt.normalize_rows(np.random.rand(nQpts, 128))
>>> qmag = np.sqrt(np.power(qpts, 2).sum(1))
>>> dmag = np.sqrt(np.power(dpts, 2).sum(1))
>>> assert np.all(np.allclose(qmag, 1)), 'not on unit sphere'
>>> assert np.all(np.allclose(dmag, 1)), 'not on unit sphere'
>>> # cast to uint8
>>> uint8_max = 512 # hack
>>> uint8_min = 0 # hack
>>> K = 100 # 2

```

```

>>> qpts8 = np.clip(np.round(qpts * uint8_max), uint8_min, uint8_max).astype(np.
↪uint8)
>>> dpts8 = np.clip(np.round(dpts * uint8_max), uint8_min, uint8_max).astype(np.
↪uint8)
>>> qmag8 = np.sqrt(np.power(qpts8.astype(np.float32), 2).sum(1))
>>> dmag8 = np.sqrt(np.power(dpts8.astype(np.float32), 2).sum(1))
>>> # test
>>> qx2_dx, qx2_dist = ann_flann_once(dpts8, qpts8, K)
>>> biggest_dist = np.sqrt(qx2_dist.max())
>>> print('biggest_dist = %r' % (biggest_dist))
>>> # Get actual distance by hand
>>> hand_dist = np.sum((qpts8 - dpts8[qx2_dx.T[0]]) ** 2, 0)
>>> # Seems like flann returns squared distance. makes sense
>>> result = ub.hash_data(repr((qx2_dx, qx2_dist)))
>>> print(result)

```

Example:

```

>>> # Build theoretically maximally distant vectors
>>> b = 512
>>> D = 128
>>> x = np.sqrt((float(b) ** 2) / float(D - 1))
>>> dpts = np.ones((2, 128)) * x
>>> qpts = np.zeros((2, 128))
>>> dpts[:, 0] = 0
>>> qpts[:, 0] = 512
>>> qpts[:, 0::2] = 1

```

(continues on next page)

(continued from previous page)

```

>>> dpts[:, 1::2] = 1
>>> qpts[:, 1::2] = 0
>>> dpts[:, 0::2] = 0
>>> qmag = np.sqrt(np.power(qpts.astype(np.float64), 2).sum(1))
>>> dmag = np.sqrt(np.power(dpts.astype(np.float64), 2).sum(1))
>>> # FIX TO ACTUALLY BE AT THE RIGHT NORM
>>> dpts = (dpts * (512 / np.linalg.norm(dpts, axis=1))[:, None]).
↳astype(np.float32)
>>> qpts = (qpts * (512 / np.linalg.norm(qpts, axis=1))[:, None]).
↳astype(np.float32)
>>> print(np.linalg.norm(dpts))
>>> print(np.linalg.norm(qpts))
>>> dist = np.sqrt(np.sum((qpts - dpts) ** 2, 1))
>>> # Because of norm condition another maximally disant pair of
↳vectors
>>> # is [1, 0, 0, ... 0] and [0, 1, .. 0, 0, 0]
>>> # verifythat this gives you same dist.
>>> dist2 = np.sqrt((512 ** 2 + 512 ** 2))
>>> print(dist2)
>>> print(dist)

```

```

vtool.nearest_neighbors.assign_to_centroids(dpts, qpts, num_neighbors=1,
                                              flann_params={})

```

Helper for akmeans

```

vtool.nearest_neighbors.flann_augment(dpts, new_dpts, cache_dir, cfgstr, new_cfgstr,
                                       flann_params, use_cache=True, save=True)

```

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.nearest_neighbors import * # NOQA
>>> import vtool.demodata as demodata # NOQA
>>> dpts = demodata.get_dummy_dpts(ut.get_nth_prime(10))
>>> new_dpts = demodata.get_dummy_dpts(ut.get_nth_prime(9))
>>> cache_dir = ut.get_app_resource_dir('vtool')
>>> cfgstr = '_testcfg'
>>> new_cfgstr = '_new_testcfg'
>>> flann_params = get_kdtree_flann_params()
>>> use_cache = False
>>> save = False

```

```

vtool.nearest_neighbors.flann_cache(dpts, cache_dir='default', cfgstr="", flann_params={},
                                     use_cache=True, save=True, use_params_hash=True,
                                     use_data_hash=True, appname='vtool', ver-
                                     bose=None)

```

Tries to load a cached flann index before doing anything from vtool.nn

```

vtool.nearest_neighbors.flann_index_time_experiment()

```

Shows a plot of how long it takes to build a flann index for a given number of KD-trees

CommandLine: python -m vtool.nearest_neighbors --test-flann_index_time_experiment

Example

```
>>> # SLOW_DOCTEST
>>> # xdoctest: +SKIP
>>> from vtool.nearest_neighbors import * # NOQA
>>> result = flann_index_time_experiment()
>>> print(result)
```

```
vtool.nearest_neighbors.get_flann_cfgstr(dpts, flann_params, cfgstr="",
                                         use_params_hash=True, use_data_hash=True)
```

CommandLine: python -m vtool.nearest_neighbors --test-get_flann_cfgstr

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.nearest_neighbors import * # NOQA
>>> rng = np.random.RandomState(1)
>>> dpts = rng.randint(0, 255, (10, 128)).astype(np.uint8)
>>> cache_dir = '.'
>>> cfgstr = '_FEAT(alg=hshes)'
>>> flann_params = get_kdtree_flann_params()
>>> result = get_flann_cfgstr(dpts, flann_params, cfgstr)
>>> print(result)
_FEAT(alg=hshes)_FLANN(4kdtree)_DPTS((10,128)xxaotseonmfjkzcr)
```

```
vtool.nearest_neighbors.get_flann_fpath(dpts, cache_dir='default', cfgstr="",
                                       flann_params={}, use_params_hash=True,
                                       use_data_hash=True, appname='vtool', verbose=True)
```

returns filepath for flann index

```
vtool.nearest_neighbors.get_flann_params(algorithm='kdtree', **kwargs)
Returns flann params that are relevant to the algorithm
```

References

http://www.cs.ubc.ca/research/flann/uploads/FLANN/flann_manual-1.8.4.pdf

Parameters `algorithm` (*str*) – (default = 'kdtree')

Returns flann_params

Return type dict

CommandLine: python -m vtool.nearest_neighbors --test-get_flann_params --algo=kdtree python -m vtool.nearest_neighbors --test-get_flann_params --algo=kmeans

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.nearest_neighbors import * # NOQA
>>> algorithm = ut.get_argval('--algo', default='kdtree')
>>> flann_params = get_flann_params(algorithm)
```

(continues on next page)

(continued from previous page)

```
>>> result = ('flann_params = %s' % (ub.repr2(flann_params),))
>>> print(result)
```

```
vtool.nearest_neighbors.get_flann_params_cfgstr(flann_params)
```

```
vtool.nearest_neighbors.get_kdtree_flann_params()
```

```
vtool.nearest_neighbors.invertible_stack(vecs_list, label_list)
```

Stacks descriptors into a flat structure and returns inverse mapping from flat database descriptor indexes (dx) to annotation ids (label) and feature indexes (fx). Feature indexes are w.r.t. annotation indexes.

Output: idx2_desc - flat descriptor stack idx2_label - inverted index into annotations idx2_fx - inverted index into features

Example with 2D Descriptors

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.nearest_neighbors import * # NOQA
>>> DESC_TYPE = np.uint8
>>> label_list = [1, 2, 3, 4, 5]
>>> vecs_list = [
...     np.array([[0, 0], [0, 1]], dtype=DESC_TYPE),
...     np.array([[5, 3], [2, 30], [1, 1]], dtype=DESC_TYPE),
...     np.empty((0, 2), dtype=DESC_TYPE),
...     np.array([[5, 3], [2, 30], [1, 1]], dtype=DESC_TYPE),
...     np.array([[3, 3], [42, 42], [2, 6]], dtype=DESC_TYPE),
...     ]
>>> idx2_vec, idx2_label, idx2_fx = invertible_stack(vecs_list, label_list)
>>> print(repr(idx2_vec.T))
array([[ 0,  0,  5,  2,  1,  5,  2,  1,  3, 42,  2],
       [ 0,  1,  3, 30,  1,  3, 30,  1,  3, 42,  6]], dtype=uint8)
>>> print(repr(idx2_label))
array([1, 1, 2, 2, 2, 4, 4, 4, 5, 5, 5])
>>> print(repr(idx2_fx))
array([0, 1, 0, 1, 2, 0, 1, 2, 0, 1, 2])
```

```
vtool.nearest_neighbors.test_annoy()
```

```
vtool.nearest_neighbors.test_cv2_flann()
```

Ignore: [name for name in dir(cv2) if 'create' in name.lower()] [name for name in dir(cv2) if 'stereo' in name.lower()]

ut.grab_zipped_url('https://priithon.googlecode.com/archive/a6117f5e81ec00abcfb037f0f9da2937bb2ea47f.tar.gz', download_dir='.')

```
vtool.nearest_neighbors.tune_flann(dpts, target_precision=0.9, build_weight=0.5, memory_weight=0.0, sample_fraction=0.01)
```

References

http://www.cs.ubc.ca/research/flann/uploads/FLANN/flann_pami2014.pdf http://www.cs.ubc.ca/research/flann/uploads/FLANN/flann_manual-1.8.4.pdf http://docs.opencv.org/trunk/modules/flann/doc/flann_fast_approximate_nearest_neighbor_search.html

Math: cost of an algorithm is:

LaTeX:

$$\text{cost} = \frac{\text{search} + \text{build_weight} * \text{build}}{\text{minoverparams}(\text{search} + \text{build_weight} * \text{build}) + \text{memory_weight} * \text{memory}}$$

Parameters

- **dpts** (*ndarray*) –
- **target_precision** (*float*) – number between 0 and 1 representing desired accuracy. Higher values are more accurate.
- **build_weight** (*float*) – importance weight given to minimizing build time relative to search time. This number can range from 0 to infinity. typically because building is a more complex computation you want to keep the number relatively low, (less than 1) otherwise you'll end up getting a linear search (no build time).
- **memory_weight** (*float*) – Importance of memory relative to total speed. A value less than 1 gives more importance to the time spent and a value greater than 1 gives more importance to the memory usage.
- **sample_fraction** (*float*) – number between 0 and 1 representing the fraction of the input data to use in the optimization. A higher number uses more data.

Returns tuned_params

Return type dict

CommandLine: python -m vtool.nearest_neighbors --test-tune_flann

1.30 vtool.numpy_utils module

These functions might be PR quality for numpy.

`vtool.numpy_utils.atleast_nd(arr, n, tofront=False)`

View inputs as arrays with at least n dimensions. TODO: Submit as a PR to numpy

Parameters

- **arr** (*array_like*) – One array-like object. Non-array inputs are converted to arrays. Arrays that already have n or more dimensions are preserved.
- **n** (*int*) – number of dimensions to ensure
- **tofront** (*bool*) – if True new dimensions are added to the front of the array. otherwise they are added to the back.

CommandLine: python -m vtool.numpy_utils atleast_nd

Returns An array with `a.ndim >= n`. Copies are avoided where possible, and views with three or more dimensions are returned. For example, a 1-D array of shape `(N,)` becomes a view of shape `(1, N, 1)`, and a 2-D array of shape `(M, N)` becomes a view of shape `(M, N, 1)`.

Return type ndarray

See also:

`ensure_shape`, `np.atleast_1d`, `np.atleast_2d`, `np.atleast_3d`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> import ubelt as ub
>>> n = 2
>>> arr = np.array([1, 1, 1])
>>> arr_ = atleast_nd(arr, n)
>>> result = ub.repr2(arr_.tolist())
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> import ubelt as ub
>>> n = 4
>>> arr1 = [1, 1, 1]
>>> arr2 = np.array(0)
>>> arr3 = np.array([[[[1]]]])
>>> arr1_ = atleast_nd(arr1, n)
>>> arr2_ = atleast_nd(arr2, n)
>>> arr3_ = atleast_nd(arr3, n)
>>> result1 = ub.repr2(arr1_.tolist())
>>> result2 = ub.repr2(arr2_.tolist())
>>> result3 = ub.repr2(arr3_.tolist())
>>> result = '\n'.join([result1, result2, result3])
>>> print(result)
```

`vtool.numpy_utils.ensure_shape(arr, dimshape)`

TODO: Submit as a PR to numpy?

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> ensure_shape(np.array([[1, 2]]), (None, 2))
>>> ensure_shape(np.array([]), (None, 2))
```

`vtool.numpy_utils.fromiter_nd(iter_, shape, dtype)`

Like `np.fromiter` but handles iterators that generated n-dimensional arrays. Slightly faster than `np.array`.

Note: `np.vstack(list_)` is still faster than `vt.fromiter_nd(ut.iflatten(list_))`

Parameters

- **iter_** (*iter*) – an iterable that generates homogenous ndarrays
- **shape** (*tuple*) – the expected output shape
- **dtype** (*dtype*) – the numpy datatype of the generated ndarrays

Note: The iterable must yeild a numpy array. It cannot yeild a Python list.

CommandLine: python -m vtool.numpy_utils fromiter_nd

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> dtype = np.float
>>> total = 11
>>> rng = np.random.RandomState(0)
>>> iter_ = (rng.rand(5, 7, 3) for _ in range(total))
>>> shape = (total, 5, 7, 3)
>>> result = fromiter_nd(iter_, shape, dtype)
>>> assert result.shape == shape
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> import utool as ut
>>> dtype = np.int
>>> qfxs = np.array([1, 2, 3])
>>> dfxs = np.array([4, 5, 6])
>>> iter_ = (np.array(x) for x in ut.product(qfxs, dfxs))
>>> total = len(qfxs) * len(dfxs)
>>> shape = (total, 2)
>>> result = fromiter_nd(iter_, shape, dtype)
>>> assert result.shape == shape
```

Ignore:

```
>>> dtype = np.uint8
>>> feat_dim = 128
>>> mu = 1000
>>> sigma = 500
>>> n_data = 1000
>>> rng = np.random.RandomState(42)
>>> n_feat_list = np.clip(rng.randn(n_data) * sigma + mu, 0, np.inf).
↳ astype(np.int)
>>> # Make a large list of vectors of various sizes
>>> print('Making random vectors')
>>> vecs_list = [(rng.rand(num, feat_dim) * 255).astype(dtype) for num in n_
↳ feat_list]
>>> mega_bytes = sum([x.nbytes for x in vecs_list]) / 2 ** 20
>>> print('mega_bytes = %r' % (mega_bytes,))
>>> import itertools as it
>>> import vtool as vt
>>> n_total = n_feat_list.sum()
>>> target1 = np.vstack(vecs_list)
>>> iter_ = it.chain.from_iterable(vecs_list)
>>> shape = (n_total, feat_dim)
```

(continues on next page)

(continued from previous page)

```

>>> target2 = vt.fromiter_nd(it.chain.from_iterable(vecs_list), shape, dtype=dtype)
>>> assert np.all(target1 == target2)
>>>
>>> %timeit np.vstack(vecs_list)
>>> 20.4ms
>>> %timeit vt.fromiter_nd(it.chain.from_iterable(vecs_list), shape, dtype)
>>> 102ms
>>>
>>> iter_ = it.chain.from_iterable(vecs_list)
>>> %time vt.fromiter_nd(iter_, shape, dtype)
>>> %time np.vstack(vecs_list)

```

`vtool.numpy_utils.index_to_boolmask` (*index_list*, *maxval=None*, *isflat=True*)
 transforms a list of indicies into a boolean mask

Parameters

- **index_list** (*ndarray*) –
- **maxval** (*None*) – (default = None)

Kwargs: maxval

Returns mask

Return type ndarray

CommandLine: python -m vtool.util_numpy index_to_boolmask

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.util_numpy import * # NOQA
>>> import vtool as vt
>>> index_list = np.array([(0, 0), (1, 1), (2, 1)])
>>> maxval = (3, 3)
>>> mask = vt.index_to_boolmask(index_list, maxval, isflat=False)
>>> result = ('mask =\n%s' % (str(mask.astype(np.uint8))),)
>>> print(result)
[[1 0 0]
 [0 1 0]
 [0 1 0]]

```

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.util_numpy import * # NOQA
>>> import vtool as vt
>>> index_list = np.array([0, 1, 4])
>>> maxval = 5
>>> mask = vt.index_to_boolmask(index_list, maxval, isflat=True)
>>> result = ('mask = %s' % (str(mask.astype(np.uint8))),)

```

(continues on next page)

(continued from previous page)

```
>>> print(result)
mask = [1 1 0 0 1]
```

`vtool.numpy_utils.iter_reduce_ufunc(ufunc, arr_iter, out=None)`

constant memory iteration and reduction

applies ufunc from left to right over the input arrays

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> arr_list = [
...     np.array([0, 1, 2, 3, 8, 9]),
...     np.array([4, 1, 2, 3, 4, 5]),
...     np.array([0, 5, 2, 3, 4, 5]),
...     np.array([1, 1, 6, 3, 4, 5]),
...     np.array([0, 1, 2, 7, 4, 5])
... ]
>>> memory = np.array([9, 9, 9, 9, 9, 9])
>>> gen_memory = memory.copy()
>>> def arr_gen(arr_list, gen_memory):
...     for arr in arr_list:
...         gen_memory[:] = arr
...         yield gen_memory
>>> print('memory = %r' % (memory,))
>>> print('gen_memory = %r' % (gen_memory,))
>>> ufunc = np.maximum
>>> res1 = iter_reduce_ufunc(ufunc, iter(arr_list), out=None)
>>> res2 = iter_reduce_ufunc(ufunc, iter(arr_list), out=memory)
>>> res3 = iter_reduce_ufunc(ufunc, arr_gen(arr_list, gen_memory), out=memory)
>>> print('res1      = %r' % (res1,))
>>> print('res2      = %r' % (res2,))
>>> print('res3      = %r' % (res3,))
>>> print('memory     = %r' % (memory,))
>>> print('gen_memory = %r' % (gen_memory,))
>>> assert np.all(res1 == res2)
>>> assert np.all(res2 == res3)
```

`vtool.numpy_utils.multiaxis_reduce(ufunc, arr, startaxis=0)`

used to get max/min over all axes after <startaxis>

CommandLine: `python -m vtool.numpy_utils -test-multiaxis_reduce`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> rng = np.random.RandomState(0)
>>> arr = (rng.rand(4, 3, 2, 1) * 255).astype(np.uint8)
>>> ufunc = np.amax
>>> startaxis = 1
>>> out_ = multiaxis_reduce(ufunc, arr, startaxis)
>>> result = out_
```

(continues on next page)

(continued from previous page)

```
>>> print(result)
[182 245 236 249]
```

`vtool.numpy_utils.unique_row_indexes(arr)`
 np.unique on rows

Parameters `arr` (*ndarray*) – 2d array

Returns `unique_rowx`

Return type *ndarray*

References

<http://stackoverflow.com/questions/16970982/find-unique-rows-in-numpy-array>

CommandLine: `python -m vtool.numpy_utils --test-unique_row_indexes`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> import ubelt as ub
>>> arr = np.array([[0, 0], [0, 1], [1, 0], [1, 1], [0, 0], [.534, .432], [.534, .
↪.432], [1, 0], [0, 1]])
>>> unique_rowx = unique_row_indexes(arr)
>>> result = ('unique_rowx = %s' % (ub.repr2(unique_rowx),))
>>> print(result)
unique_rowx = np.array([0, 1, 2, 3, 5], dtype=np.int64)
```

Ignore: `%timeit unique_row_indexes(arr)` `%timeit compute_unique_data_ids(arr)` `%timeit compute_unique_integer_data_ids(arr)`

1.31 vtool.other module

`vtool.other.and_lists(*args)`
 Like `np.logical_and`, but can take more than 2 arguments

CommandLine: `python -m vtool.other --test-and_lists`

SeeAlso: `or_lists`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> arg1 = np.array([1, 1, 1, 1,])
>>> arg2 = np.array([1, 1, 0, 1,])
>>> arg3 = np.array([0, 1, 0, 1,])
>>> args = (arg1, arg2, arg3)
>>> flags = and_lists(*args)
>>> result = str(flags)
```

(continues on next page)

(continued from previous page)

```
>>> print(result)
[False True False True]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> size = 10000
>>> rng = np.random.RandomState(0)
>>> arg1 = rng.randint(2, size=size)
>>> arg2 = rng.randint(2, size=size)
>>> arg3 = rng.randint(2, size=size)
>>> args = (arg1, arg2, arg3)
>>> flags = and_lists(*args)
>>> # ensure equal division
>>> segments = 5
>>> validx = np.where(flags)[0]
>>> endx = int(segments * (validx.size // (segments)))
>>> parts = np.split(validx[:endx], segments)
>>> result = str(list(map(np.sum, parts)))
>>> print(result)
[243734, 714397, 1204989, 1729375, 2235191]
```

%timeit reduce(np.logical_and, args) %timeit np.logical_and.reduce(args) # wins with more data

```
vtool.other.argsort_groups(scores_list, reverse=False, rng=<module 'numpy.random'
                        from '/home/docs/checkouts/readthedocs.org/user_builds/wbia-
                        vtool/envs/stable/lib/python3.7/site-packages/numpy/random/__init__.py'>,
                        randomize_levels=True)
```

Sorts each group normally, but randomizes order of level values.

TODO: move to vtool

Parameters

- **scores_list** (*list*) –
- **reverse** (*bool*) – (default = True)
- **rng** (*module*) – random number generator (default = numpy.random)

CommandLine: python -m wbia.init.filter_annots --exec-argsort_groups

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> scores_list = [
>>>     np.array([np.nan, np.nan], dtype=np.float32),
>>>     np.array([np.nan, 2], dtype=np.float32),
>>>     np.array([4, 1, 1], dtype=np.float32),
>>>     np.array([7, 3, 3, 0, 9, 7, 5, 8], dtype=np.float32),
>>>     np.array([2, 4], dtype=np.float32),
>>>     np.array([np.nan, 4, np.nan, 8, np.nan, 9], dtype=np.float32),
>>> ]
```

(continues on next page)

(continued from previous page)

```

>>> reverse = True
>>> rng = np.random.RandomState(0)
>>> idxs_list = argsort_groups(scores_list, reverse, rng)
>>> result = 'idxs_list = %s' % (ut.repr4(idxs_list, with_dtype=False),)
>>> print(result)

```

`vtool.other.argsort_records(arrays, reverse=False)`

Sorts arrays that form records. Same as `lexsort(arrays[::-1])` — ie. rows are reversed.

Parameters

- **arrays** (*ndarray*) – array of records
- **reverse** (*bool*) – (default = False)

Returns *sortx* - sorted indicies

Return type *ndarray*

CommandLine: `python -m vtool.other --exec-argsort_records`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> arrays = np.array([
>>>     [1, 1, 1, 2, 2, 2, 3, 4, 5],
>>>     [2, 0, 2, 6, 4, 3, 2, 5, 6],
>>>     [1, 1, 0, 2, 3, 4, 5, 6, 7],
>>> ],)
>>> reverse = False
>>> sortx = argsort_records(arrays, reverse)
>>> result = ('sortx = %s' % (str(sortx),))
>>> print('lxsrt = %s' % (np.lexsort(arrays[::-1]),))
>>> print(result)
sortx = [1 2 0 5 4 3 6 7 8]

```

`vtool.other.assert_zipcompress(arr_list, flags_list, axis=None)`

`vtool.other.assert_eq(output1, output2, thresh=1e-08, nestpath=None, level=0, lbl1=None, lbl2=None, output_lbl=None, verbose=True, iswarning=False)`
recursive equality checks

asserts that output1 and output2 are close to equal.

`vtool.other.atleast_3channels(arr, copy=True)`

Ensures that there are 3 channels in the image

Parameters

- **arr** (*ndarray* [N, M, ...]) – the image
- **copy** (*bool*) – Always copies if True, if False, then copies only when the size of the array must change.

Returns with shape (N, M, C), where C in {3, 4}

Return type *ndarray*

CommandLine: `python -m vtool.other atleast_3channels`

Doctest:

```
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> assert atleast_3channels(np.zeros((10, 10))).shape[-1] == 3
>>> assert atleast_3channels(np.zeros((10, 10, 1))).shape[-1] == 3
>>> assert atleast_3channels(np.zeros((10, 10, 3))).shape[-1] == 3
>>> assert atleast_3channels(np.zeros((10, 10, 4))).shape[-1] == 4
```

`vtool.other.atleast_nd(arr, n, tofront=False)`

View inputs as arrays with at least n dimensions. TODO: Commit to numpy

Parameters

- **arr** (*array_like*) – One array-like object. Non-array inputs are converted to arrays. Arrays that already have n or more dimensions are preserved.
- **n** (*int*) –
- **tofront** (*bool*) – if True new dims are added to the front of the array

CommandLine: `python -m vtool.other --exec-atleast_nd --show`

Returns An array with `a.ndim >= n`. Copies are avoided where possible, and views with three or more dimensions are returned. For example, a 1-D array of shape `(N,)` becomes a view of shape `(1, N, 1)`, and a 2-D array of shape `(M, N)` becomes a view of shape `(M, N, 1)`.

Return type ndarray

See also:

`atleast_1d`, `atleast_2d`, `atleast_3d`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> n = 2
>>> arr = np.array([1, 1, 1])
>>> arr_ = atleast_nd(arr, n)
>>> result = ub.repr2(arr_.tolist())
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> n = 4
>>> arr1 = [1, 1, 1]
>>> arr2 = np.array(0)
>>> arr3 = np.array([[[[1]]]])
>>> arr1_ = atleast_nd(arr1, n)
>>> arr2_ = atleast_nd(arr2, n)
>>> arr3_ = atleast_nd(arr3, n)
>>> result1 = ub.repr2(arr1_.tolist())
```

(continues on next page)

(continued from previous page)

```
>>> result2 = ub.repr2(arr2_.tolist())
>>> result3 = ub.repr2(arr3_.tolist())
>>> result = '\n'.join([result1, result2, result3])
>>> print(result)
```

`vtool.other.atleast_shape(arr, dimshape)`

Ensures that an array takes a certain shape. The total size of the array must not change.

Parameters

- **arr** (*ndarray*) – array to change the shape of
- **dimshape** (*tuple*) – desired shape (Nones can be used to broadcast dimensions)

Returns *ndarray* - the input array, which has been modified inplace.

CommandLine: `python -m vtool.other ensure_shape`

Doctest:

```
>>> from vtool.other import * # NOQA
>>> arr = np.zeros((7, 7))
>>> assert atleast_shape(arr, (1, 1, 3,)).shape == (7, 7, 3)
>>> assert atleast_shape(arr, (1, 1, 2, 4,)).shape == (7, 7, 2, 4)
>>> assert atleast_shape(arr, (1, 1,)).shape == (7, 7,)
>>> assert atleast_shape(arr, (1, 1, 1)).shape == (7, 7, 1)
>>> assert atleast_shape(np.zeros(()), (1,)).shape == (1,)
>>> assert atleast_shape(np.zeros(()), tuple()).shape == tuple()
>>> assert atleast_shape(np.zeros(()), (1, 2, 3,)).shape == (1, 2, 3)
>>> ut.assert_raises(ValueError, atleast_shape, arr, (2, 2))
>>> assert atleast_shape(np.zeros((7, 7, 3)), (1, 1, 3)).shape == (7, 7, 3)
>>> ut.assert_raises(ValueError, atleast_shape, np.zeros((7, 7, 3)), (1, 1, 4))
```

`vtool.other.calc_errorBarsFromSample(sample_size, num_positive, pop, conf_level=0.95)`

Determines a error bars of sample

References

<https://www.qualtrics.com/blog/determining-sample-size/> <http://www.surveysystem.com/sscalc.htm> https://en.wikipedia.org/wiki/Sample_size_determination <http://www.surveysystem.com/sample-size-formula.htm> http://courses.wcupa.edu/rbove/Berenson/10th%20ed%20CD-ROM%20topics/section8_7.pdf https://en.wikipedia.org/wiki/Standard_normal_table <https://www.unc.edu/~rls/s151-2010/class23.pdf>

`vtool.other.calc_sampleFromErrorBars(err_frac, pop, conf_level=0.95, prior=0.5)`

Determines a reasonable sample size to achieve desired error bars.

```
import sympy
p, n, N, z = sympy.symbols('prior, ss, pop, zval')
me = sympy.symbols('err_frac')
expr = (z * sympy.sqrt((p * (1 - p) / n) * ((N - n) / (N - 1))))
equation = sympy.Eq(me, expr)
nexpr = sympy.solve(equation, [n])[0]
nexpr = sympy.simplify(nexpr)
```

```
import autopep8
print(autopep8.fix_lines(['ss = ' + str(nexpr)], autopep8.get_options({}), False))
```

```
ss = -pop * prior * (zval**2) * (prior - 1) / ((err_frac**2) * pop - (err_frac**2) - prior * (zval**2) * (prior - 1))
ss = pop * prior * zval**2 * (prior - 1) / (-err_frac**2 * pop + err_frac**2 + prior * zval**2 * (prior - 1))
```

`vtool.other.check_sift_validity(sift_uint8, lbl=None, verbose=True)`

checks if a SIFT descriptor is valid

`vtool.other.clipnorm(arr, min_, max_, out=None)`
normalizes arr to the range 0 to 1 using min_ and max_ as clipping bounds

`vtool.other.colwise_operation(arr1, arr2, op)`

`vtool.other.compare_implementations(func1, func2, args, show_output=False, lbl1="", lbl2="", output_lbl=None)`
tests two different implementations of the same function

`vtool.other.compare_matrix_columns(matrix, columns, comp_op=<ufunc 'equal'>, logic_op=<ufunc 'logical_or'>)`

REPLACE WITH: `qfx2_invalid = logic_op.reduce([comp_op[:, None], qfx2_normnid] for col1 in qfx2_topnid.T])`

`vtool.other.compare_matrix_to_rows(row_matrix, row_list, comp_op=<ufunc 'equal'>, logic_op=<ufunc 'logical_or'>)`
Compares each row in row_list to each row in row matrix using comp_op Both must have the same number of columns. Performs logic_op on the results of each individual row

SeeAlso: `wbia.algo.hots.nn_weights.mark_name_valid_normalizers`

`compop = np.equal logic_op = np.logical_or`

`vtool.other.componentwise_dot(arr1, arr2)`
a dot product is a componentwise multiplication of two vector and then a sum.

Parameters

- **arr1** (ndarray) –
- **arr2** (ndarray) –

Returns cosangle**Return type** ndarray**Example**

```
>>> # DISABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> np.random.seed(0)
>>> arr1 = np.random.rand(3, 128)
>>> arr1 = arr1 / np.linalg.norm(arr1, axis=1)[:, None]
>>> arr2 = arr1
>>> cosangle = componentwise_dot(arr1, arr2)
>>> result = str(cosangle)
>>> print(result)
[ 1.  1.  1.]
```

`vtool.other.compress2(arr, flag_list, axis=None, out=None)`
Wrapper around numpy compress that makes the signature more similar to take

`vtool.other.compute_ndarray_unique_rowids_unsafe(arr)`
`arr = np.random.randint(2, size=(10000, 10)) vt.compute_unique_data_ids_(list(map(tuple, arr)))`
`len(vt.compute_unique_data_ids_(list(map(tuple, arr)))) len(np.unique(vt.compute_unique_data_ids_(list(map(tuple, arr))))))`

`%timeit vt.compute_unique_data_ids_(list(map(tuple, arr)))` `%timeit com-`
`pute_ndarray_unique_rowids_unsafe(arr)`

`vtool.other.compute_unique_arr_dataids(arr)`
specialized version for speed when arr is an ndarray

`vtool.other.compute_unique_data_ids(data)`
This is actually faster than `compute_unique_integer_data_ids` it seems

CommandLine: `python -m vtool.other --test-compute_unique_data_ids`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> data = np.array([[0, 0], [0, 1], [1, 0], [1, 1], [0, 0], [.534, .432], [.534,
↪.432], [1, 0], [0, 1]])
>>> dataid_list = compute_unique_data_ids(data)
>>> result = 'dataid_list = ' + ub.repr2(dataid_list, with_dtype=True)
>>> print(result)
dataid_list = np.array([0, 1, 2, 3, 0, 4, 4, 2, 1], dtype=np.int32)
```

`vtool.other.compute_unique_data_ids_(hashable_rows, iddict_=None)`

`vtool.other.compute_unique_integer_data_ids(data)`
This is actually slower than `compute_unique_data_ids` it seems

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> # build test data
>>> data = np.array([[0, 0], [0, 1], [1, 1], [0, 0], [0, 0], [0, 1], [1, 1], [0,
↪0], [9, 0]])
>>> data = np.random.randint(1000, size=(1000, 2))
>>> # execute function
>>> result1 = compute_unique_data_ids(data)
>>> result2 = compute_unique_integer_data_ids(data)
>>> # verify results
>>> print(result)
```

`%timeit compute_unique_data_ids(data) %timeit compute_unique_integer_data_ids(data)`

`vtool.other.ensure_rng(seed=None)`
Returns a numpy random number generator given a seed.

`vtool.other.ensure_shape(arr, dimshape)`
Ensures that an array takes a certain shape. The total size of the array must not change.

Parameters

- **arr** (*ndarray*) – array to change the shape of
- **dimshape** (*tuple*) – desired shape (Nones can be used to broadcast dimensions)

Returns ndarray - the input array, which has been modified inplace.

CommandLine: `python -m vtool.other ensure_shape`

Doctest:

```

>>> from vtool.other import * # NOQA
>>> arr = np.zeros((7, 7))
>>> dimshape = (None, None, 3)
>>> arr2 = ensure_shape(np.array([[1, 2]]), (None, 2))
>>> assert arr2.shape == (1, 2)
>>> arr3 = ensure_shape(np.array([]), (None, 2))
>>> assert arr3.shape == (0, 2)

```

`vtool.other.find_best_undirected_edge_indexes` (*directed_edges*, *score_arr=None*)

Parameters

- **directed_edges** (*ndarray*[*ndims=2*]) –
- **score_arr** (*ndarray*) –

Returns *unique_edge_xs*

Return type *list*

CommandLine: `python -m vtool.other -test-find_best_undirected_edge_indexes`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> directed_edges = np.array([[1, 2], [2, 1], [2, 3], [3, 1], [1, 1], [2, 3], [3,
↪ 2]])
>>> score_arr = np.array([1, 1, 1, 1, 1, 1, 2])
>>> unique_edge_xs = find_best_undirected_edge_indexes(directed_edges, score_arr)
>>> result = str(unique_edge_xs)
>>> print(result)
[0 3 4 6]

```

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> directed_edges = np.array([[1, 2], [2, 1], [2, 3], [3, 1], [1, 1], [2, 3], [3,
↪ 2]])
>>> score_arr = None
>>> unique_edge_xs = find_best_undirected_edge_indexes(directed_edges, score_arr)
>>> result = str(unique_edge_xs)
>>> print(result)
[0 2 3 4]

```

`vtool.other.find_elbow_point` (*curve*)

Finds the on the curve point furthest from the line defined by the endpoints of the curve.

Parameters **curve** (*ndarray*) – a monotonic curve

Returns *tradeoff_idx* - this is an elbow point in the curve

Return type *int*

References

<http://stackoverflow.com/questions/2018178/trade-off-point-on-curve>

CommandLine: `python -m vtool.other find_elbow_point --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> curve = np.exp(np.linspace(0, 10, 100))
>>> tradeoff_idx = find_elbow_point(curve)
>>> result = ('tradeoff_idx = %s' % (ub.repr2(tradeoff_idx),))
>>> print(result)
>>> assert tradeoff_idx == 76
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> import vtool as vt
>>> point = [tradeoff_idx, curve[tradeoff_idx]]
>>> segment = np.array([[0, len(curve) - 1], [curve[0], curve[-1]]])
>>> e1, e2 = segment.T
>>> dist_point = vt.closest_point_on_line_segment(point, e1, e2)
>>> dist_line = np.array([dist_point, point]).T
>>> pt.plot(curve, 'r', label='curve')
>>> pt.plot(point[0], point[1], 'go', markersize=10, label='tradeoff point')
>>> pt.plot(dist_line[0], dist_line[1], '-xb')
>>> pt.plot(segment[0], segment[1], '-xb')
>>> pt.legend()
>>> ut.show_if_requested()
```

`vtool.other.find_first_true_indices` (*flags_list*)

TODO: move to vtool

returns a list of indexes where the index is the first True position in the corresponding sublist or None if it does not exist

in other words: for each row finds the smallest True column number or None

Parameters `flags_list` (*list*) – list of lists of booleans

CommandLine: `python -m vtool.util_list --test-find_first_true_indices`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> # build test data
>>> flags_list = [[True, False, True],
...               [False, False, False],
...               [False, True, True],
...               [False, False, True]]
>>> # execute function
>>> index_list = find_first_true_indices(flags_list)
>>> # verify results
>>> result = str(index_list)
```

(continues on next page)

(continued from previous page)

```
>>> print(result)
[0, None, 1, 2]
```

`vtool.other.find_k_true_indicies(flags_list, k)`

Uses output of either this function or `find_first_true_indices` to find the next index of true flags

Parameters `flags_list` (*list*) – list of lists of booleans

CommandLine: `python -m utool.util_list --test-find_next_true_indices`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> flags_list = [[False, False, True],
...               [False, False, False],
...               [False, True, True],
...               [True, True, True]]
>>> k = 2
>>> indices = find_k_true_indicies(flags_list, k)
>>> result = str(indices)
>>> print(result)
[array([2]), None, array([1, 2]), array([0, 1])]
```

`vtool.other.find_next_true_indices(flags_list, offset_list)`

Uses output of either this function or `find_first_true_indices` to find the next index of true flags

Parameters `flags_list` (*list*) – list of lists of booleans

CommandLine: `python -m utool.util_list --test-find_next_true_indices`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> # build test data
>>> flags_list = [[True, False, True],
...               [False, False, False],
...               [False, True, True],
...               [False, False, True]]
>>> offset_list = find_first_true_indices(flags_list)
>>> # execute function
>>> index_list = find_next_true_indices(flags_list, offset_list)
>>> # verify results
>>> result = str(index_list)
>>> print(result)
[2, None, 2, None]
```

`vtool.other.flag_intersection(arr1, arr2)`

Flags the rows in `arr1` that contain items in `arr2`

Returns flags where `len(flags) == len(arr1)`

Return type ndarray

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> arr1 = np.array([0, 1, 2, 3, 4, 5])
>>> arr2 = np.array([2, 6, 4])
>>> flags = flag_intersection(arr1, arr2)
>>> assert len(flags) == len(arr1)
>>> result = ('flags = %s' % (ub.repr2(flags),))
>>> print(result)

```

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> import vtool as vt
>>> arr1 = np.array([[0, 0], [0, 1], [0, 2], [0, 3], [0, 4], [0, 5]])
>>> arr2 = np.array([[0, 2], [0, 6], [0, 4], [3, 0]])
>>> arr1, arr2 = vt.structure_rows(arr1, arr2)
>>> flags = flag_intersection(arr1, arr2)
>>> assert len(flags) == len(arr1)
>>> result = ('flags = %s' % (ub.repr2(flags),))
>>> print(result)

```

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> arr1 = np.array([0, 1, 2, 3, 4, 5])
>>> arr2 = np.array([])
>>> flags = flag_intersection(arr1, arr2)
>>> assert len(flags) == len(arr1)
>>> flags = flag_intersection(np.array([]), np.array([2, 6, 4]))
>>> assert len(flags) == 0

```

Ignore:

```

>>> setup = ut.codeblock(
>>>     r'''
        import vtool as vt
        import numpy as np
        rng = np.random.RandomState(0)
        arr1 = rng.randint(0, 100, 100000).reshape(-1, 2)
        arr2 = rng.randint(0, 100, 1000).reshape(-1, 2)
        arr1_, arr2_ = vt.structure_rows(arr1, arr2)
        '''
>>> stmt_list = ut.codeblock(
>>>     '''
        np.array([row in arr2_ for row in arr1_])
        np.logical_or.reduce([arr1_ == row_ for row_ in arr2_]).ravel()
        vt.iter_reduce_ufunc(np.logical_or, (arr1_ == row_ for row_ in arr2_
↵)).ravel()
        ''').split('\n')
>>> out = ut.timeit_compare(stmt_list, setup=setup, iterations=3)

```

`vtool.other.fromiter_nd(iter_, shape, dtype)`

Like `np.fromiter` but handles iterators that generated n-dimensional arrays. Slightly faster than `np.array`.

maybe commit to numpy?

Parameters

- **iter** (*iter*) – an iterable that generates homogenous ndarrays
- **shape** (*tuple*) – the expected output shape
- **dtype** (*dtype*) – the numpy datatype of the generated ndarrays

Note: The iterable must yeild a numpy array. It cannot yeild a Python list.

CommandLine: `python -m vtool.other fromiter_nd --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> dtype = np.float
>>> total = 11
>>> rng = np.random.RandomState(0)
>>> iter_ = (rng.rand(5, 7, 3) for _ in range(total))
>>> shape = (total, 5, 7, 3)
>>> result = fromiter_nd(iter_, shape, dtype)
>>> assert result.shape == shape
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> dtype = np.int
>>> qfxs = np.array([1, 2, 3])
>>> dfxs = np.array([4, 5, 6])
>>> iter_ = (np.array(x) for x in ut.product(qfxs, dfxs))
>>> total = len(qfxs) * len(dfxs)
>>> shape = (total, 2)
>>> result = fromiter_nd(iter_, shape, dtype)
>>> assert result.shape == shape
```

`vtool.other.get_covered_mask(covered_array, covering_array)`

`vtool.other.get_crop_slices(isfill)`

`vtool.other.get_uncovered_mask(covered_array, covering_array)`

Parameters

- **covered_array** (*ndarray*) –
- **covering_array** (*ndarray*) –

Returns flags

Return type ndarray

CommandLine: python -m vtool.other --test-get_uncovered_mask

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> covered_array = [1, 2, 3, 4, 5]
>>> covering_array = [2, 4, 5]
>>> flags = get_uncovered_mask(covered_array, covering_array)
>>> result = str(flags)
>>> print(result)
[ True False  True False False]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> covered_array = [1, 2, 3, 4, 5]
>>> covering_array = []
>>> flags = get_uncovered_mask(covered_array, covering_array)
>>> result = str(flags)
>>> print(result)
[ True  True  True  True  True]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> covered_array = np.array([
...     [1, 2, 3],
...     [4, 5, 6],
...     [7, 8, 9],
... ], dtype=np.int32)
>>> covering_array = [2, 4, 5]
>>> flags = get_uncovered_mask(covered_array, covering_array)
>>> result = ub.repr2(flags, with_dtype=True)
>>> print(result)
np.array([[ True, False,  True],
        [False, False,  True],
        [ True,  True,  True]], dtype=np.bool)
```

Ignore: covering_array = [1, 2, 3, 4, 5, 6, 7] %timeit get_uncovered_mask(covered_array, covering_array)
 100000 loops, best of 3: 18.6 µs per loop %timeit get_uncovered_mask2(covered_array, covering_array)
 100000 loops, best of 3: 16.9 µs per loop

vtool.other.get_undirected_edge_ids(*directed_edges*)

Parameters *directed_edges* (ndarray[ndims=2]) –

Returns edgeid_list

Return type list

CommandLine: python -m vtool.other --exec-get_undirected_edge_ids

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> directed_edges = np.array([[1, 2], [2, 1], [2, 3], [3, 1], [1, 1], [2, 3], [3,
↪ 2]])
>>> edgeid_list = get_undirected_edge_ids(directed_edges)
>>> result = ('edgeid_list = %s' % (ub.repr2(edgeid_list),))
>>> print(result)
edgeid_list = [0 0 1 2 3 1 1]
```

`vtool.other.grab_webcam_image()`

References

http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html

CommandLine: `python -m vtool.other -test-grab_webcam_image -show`

Example

```
>>> # SCRIPT
>>> from vtool.other import * # NOQA
>>> import vtool as vt
>>> img = grab_webcam_image()
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(img)
>>> vt.imwrite('webcap.jpg', img)
>>> ut.show_if_requested()
```

`vtool.other.greedy_setcover(universe, subsets, weights=None)`
Copied implementation of greedy set cover from stack overflow. Needs work.

References

<http://stackoverflow.com/questions/7942312/of-greedy-set-cover-faster>

Example

```
>>> # SLOW_DOCTEST
>>> # xdoctest: +SKIP
>>> from vtool.other import * # NOQA
>>> import vtool as vt
>>> universe = set([1,2,3,4])
>>> subsets = [set([1,2]), set([1]), set([1,2,3]), set([1]), set([3,4]),
>>>             set([4]), set([1,2]), set([3,4]), set([1,2,3,4])]
>>> weights = [1, 1, 2, 2, 2, 3, 3, 4, 4]
>>> chosen, costs = greedy_setcover(universe, subsets, weights)
>>> print('Cover: %r' % (chosen,))
>>> print('Total Cost: %r=sum(%r)' % (sum(costs), costs))
```


`vtool.other.inbounds (num, low, high, eq=False)`

Parameters

- **num** (*scalar or ndarray*) –
- **low** (*scalar or ndarray*) –
- **high** (*scalar or ndarray*) –
- **eq** (*bool*) –

Returns `is_inbounds`

Return type `scalar or ndarray`

CommandLine: `xdoctest -m ~/code/vtool/vtool/other.py inbounds`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> import utool as ut
>>> num = np.array([[ 0.    ,  0.431,  0.279],
...                 [ 0.204,  0.352,  0.08 ],
...                 [ 0.107,  0.325,  0.179]])
>>> low  = .1
>>> high = .4
>>> eq = False
>>> is_inbounds = inbounds(num, low, high, eq)
>>> result = ub.repr2(is_inbounds, with_dtype=True)
>>> print(result)
```

`vtool.other.index_partition (item_list, part1_items)`

returns two lists. The first are the indices of items in `item_list` that are in `part1_items`. the second is the indices in `item_list` that are not in `part1_items`. items in `part1_items` that are not in `item_list` are ignored

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> item_list = ['dist', 'fg', 'distinctiveness']
>>> part1_items = ['fg', 'distinctiveness']
>>> part1_indexes, part2_indexes = index_partition(item_list, part1_items)
>>> ut.assert_eq(part1_indexes.tolist(), [1, 2])
>>> ut.assert_eq(part2_indexes.tolist(), [0])
```

`vtool.other.intersect1d_reduce (arr_list, assume_unique=False)`

`vtool.other.intersect2d_flags (A, B)`

Checks intersection of rows of A against rows of B

Parameters

- **A** (*ndarray [ndims=2]*) –
- **B** (*ndarray [ndims=2]*) –

Returns (`flag_list1, flag_list2`)

Return type `tuple`

CommandLine: `python -m vtool.other --test-intersect2d_flags`

SeeAlso: `np.in1d` - the one dimensional version

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> A = np.array([[609, 307], [ 95, 344], [ 1, 690]])
>>> B = np.array([[ 422, 1148], [ 422, 968], [ 481, 1148], [ 750, 1132], [ 759,
↪159]])
>>> (flag_list1, flag_list2) = intersect2d_flags(A, B)
>>> result = str((flag_list1, flag_list2))
>>> print(result)
```

`vtool.other.intersect2d_indices(A, B)`

Parameters

- **A** (`ndarray[ndims=2]`) –
- **B** (`ndarray[ndims=2]`) –

Returns (`ax_list`, `bx_list`)

Return type `tuple`

CommandLine: `python -m vtool.other --test-intersect2d_indices`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> # build test data
>>> A = np.array([[ 158, 171], [ 542, 297], [ 955, 1113], [ 255, 1254], [ 976,
↪1255], [ 170, 1265]])
>>> B = np.array([[ 117, 211], [ 158, 171], [ 255, 1254], [ 309, 328], [ 447,
↪1148], [ 750, 357], [ 976, 1255]])
>>> # execute function
>>> (ax_list, bx_list) = intersect2d_indices(A, B)
>>> # verify results
>>> result = str((ax_list, bx_list))
>>> print(result)
```

`vtool.other.intersect2d_numpy(A, B, assume_unique=False, return_indices=False)`

References

<http://stackoverflow.com/questions/8317022/get-intersecting-rows-across-two-2d-numpy-arrays/8317155#8317155>

Parameters

- **A** (`ndarray[ndims=2]`) –

- `B(ndarray[ndims=2])` –
- `assume_unique(bool)` –

Returns C

Return type ndarray[ndims=2]

CommandLine: `python -m vtool.other --test-intersect2d_numpy`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> # build test data
>>> A = np.array([[ 0, 78, 85, 283, 396, 400, 403, 412, 535, 552],
...               [152, 98, 32, 260, 387, 285, 22, 103, 55, 261]]).T
>>> B = np.array([[403, 85, 412, 85, 815, 463, 613, 552],
...               [22, 32, 103, 116, 188, 199, 217, 254]]).T
>>> assume_unique = False
>>> # execute function
>>> C, Ax, Bx = intersect2d_numpy(A, B, return_indices=True)
>>> # verify results
>>> result = str((C.T, Ax, Bx))
>>> print(result)
(array([[ 85, 403, 412],
        [ 32, 22, 103]]), array([2, 6, 7]), array([0, 1, 2]))
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> A = np.array([[1, 2, 3], [1, 1, 1]])
>>> B = np.array([[1, 2, 3], [1, 2, 14]])
>>> C, Ax, Bx = intersect2d_numpy(A, B, return_indices=True)
>>> result = str((C, Ax, Bx))
>>> print(result)
(array([[1, 2, 3]]), array([0]), array([0]))
```

`vtool.other.intersect2d_structured_numpy(arr1, arr2, assume_unique=False)`

Parameters

- **arr1** – unstructured 2d array
- **arr2** – unstructured 2d array

Returns A_, B_, C_ - structured versions of arr1, and arr2, and their structured intersection

References

<http://stackoverflow.com/questions/16970982/find-unique-rows-in-numpy-array>

<http://stackoverflow.com/questions/8317022/get-intersecting-rows-across-two-2d-numpy-arrays>

`vtool.other.iter_reduce_ufunc(ufunc, arr_iter, out=None)`
constant memory iteration and reduction

applies ufunc from left to right over the input arrays

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> arr_list = [
...     np.array([0, 1, 2, 3, 8, 9]),
...     np.array([4, 1, 2, 3, 4, 5]),
...     np.array([0, 5, 2, 3, 4, 5]),
...     np.array([1, 1, 6, 3, 4, 5]),
...     np.array([0, 1, 2, 7, 4, 5])
... ]
>>> memory = np.array([9, 9, 9, 9, 9, 9])
>>> gen_memory = memory.copy()
>>> def arr_gen(arr_list, gen_memory):
...     for arr in arr_list:
...         gen_memory[:] = arr
...         yield gen_memory
>>> print('memory = %r' % (memory,))
>>> print('gen_memory = %r' % (gen_memory,))
>>> ufunc = np.maximum
>>> res1 = iter_reduce_ufunc(ufunc, iter(arr_list), out=None)
>>> res2 = iter_reduce_ufunc(ufunc, iter(arr_list), out=memory)
>>> res3 = iter_reduce_ufunc(ufunc, arr_gen(arr_list, gen_memory), out=memory)
>>> print('res1      = %r' % (res1,))
>>> print('res2      = %r' % (res2,))
>>> print('res3      = %r' % (res3,))
>>> print('memory    = %r' % (memory,))
>>> print('gen_memory = %r' % (gen_memory,))
>>> assert np.all(res1 == res2)
>>> assert np.all(res2 == res3)
```

`vtool.other.list_compress_(list_, flag_list)`

`vtool.other.list_take_(list_, index_list)`

`vtool.other.make_video(images, outvid=None, fps=5, size=None, is_color=True, format='XVID')`
Create a video from a list of images.

References

http://www.xavierdupre.fr/blog/2016-03-30_nojs.html http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html

@param outvid output video @param images list of images to use in the video @param fps frame per second
@param size size of each frame @param is_color color @param format see <http://www.fourcc.org/codecs.php>

The function relies on <http://opencv-python-tutroals.readthedocs.org/en/latest/>. By default, the video will have the size of the first image. It will resize every image to this size before adding them to the video.

`vtool.other.make_video2(images, outdir)`

`vtool.other.median_abs_dev(arr_list, **kwargs)`

References

https://en.wikipedia.org/wiki/Median_absolute_deviation

`vtool.other.mult_lists(*args)`

`vtool.other.multigroup_lookup(lazydict, keys_list, subkeys_list, custom_func)`

Efficiently calls `custom_func` for each item in `zip(keys_list, subkeys_list)` by grouping subkeys to minimize the number of calls to `custom_func`.

We are given multiple lists of keys, and subvals. The goal is to group the subvals by keys and apply the subval lookups (a call to a function) to the key only once and at the same time.

Parameters

- **lazydict** (*dict of vtool.LazyDict*) –
- **keys_list** (*list*) –
- **subkeys_list** (*list*) –
- **custom_func** (*func*) – must have signature `custom_func(lazydict, key, subkeys)`

SeeAlso: `vt.multigroup_lookup_naive` - unoptimized version, but simple to read

Example

```
>>> # SLOW_DOCTEST
>>> # xdoctest: +SKIP
>>> from vtool.other import * # NOQA
>>> import vtool as vt
>>> fpath_list = [ut.grab_test_imgpath(key) for key in ut.util_grabdata.get_valid_
↳test_imgkeys()]
>>> lazydict = {count: vt.testdata_annot_metadata(fpath) for count, fpath in_
↳enumerate(fpath_list)}
>>> aids_list = np.array([(3, 2), (0, 2), (1, 2), (2, 3)])
>>> fms = np.array([[2, 5], [2, 3], [2, 1], [3, 4]])
>>> keys_list = aids_list.T
>>> subkeys_list = fms.T
>>> def custom_func(lazydict, key, subkeys):
>>>     annot = lazydict[key]
>>>     kpts = annot['kpts']
>>>     rchip = annot['rchip']
>>>     kpts_m = kpts.take(subkeys, axis=0)
>>>     warped_patches = vt.get_warped_patches(rchip, kpts_m)[0]
>>>     return warped_patches
>>> data_lists1 = multigroup_lookup(lazydict, keys_list, subkeys_list, custom_
↳func)
>>> data_lists2 = multigroup_lookup_naive(lazydict, keys_list, subkeys_list,
↳custom_func)
>>> vt.sver_c_wrapper.assertreq(data_lists1, data_lists2)
```

Example

```
>>> keys_list = [np.array([]), np.array([]), np.array([])]
>>> subkeys_list = [np.array([]), np.array([]), np.array([])]
```

`vtool.other.multigroup_lookup_naive` (*lazydict, keys_list, subkeys_list, custom_func*)
Slow version of `multigroup_lookup`. Makes a call to `custom_func` for each item in `zip(keys_list, subkeys_list)`.

SeeAlso: `vt.multigroup_lookup`

`vtool.other.nearest_point` (*x, y, pts, mode='random'*)
finds the nearest point(s) in `pts` to (`x, y`)

`vtool.other.nonunique_row_flags` (*arr*)

`vtool.other.nonunique_row_indexes` (*arr*)
rows that are not unique (does not include the first instance of each pattern)

Parameters `arr` (*ndarray*) – 2d array

Returns `nonunique_rowx`

Return type `ndarray`

SeeAlso: `unique_row_indexes` `nonunique_row_flags`

CommandLine: `python -m vtool.other --test-unique_row_indexes`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> arr = np.array([[0, 0], [0, 1], [1, 0], [1, 1], [0, 0], [.534, .432], [.534, .
↪.432], [1, 0], [0, 1]])
>>> nonunique_rowx = unique_row_indexes(arr)
>>> result = ('nonunique_rowx = %s' % (ub.repr2(nonunique_rowx),))
>>> print(result)
nonunique_rowx = np.array([4, 6, 7, 8], dtype=np.int64)
```

`vtool.other.norm01` (*array, dim=None*)
normalizes a numpy array from 0 to 1 based in its extent

Parameters

- `array` (*ndarray*) –
- `dim` (*int*) –

Returns

Return type `ndarray`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> array = np.array([ 22, 1, 3, 2, 10, 42, ])
>>> dim = None
>>> array_norm = norm01(array, dim)
>>> result = ub.repr2(array_norm, precision=3)
>>> print(result)
```

`vtool.other.or_lists` (**args*)
Like `np.logical_and`, but can take more than 2 arguments

SeeAlso: `and_lists`

`vtool.other.pad_vstack(arrs, fill_value=0)`

Stacks values and pads arrays with different lengths with zeros

`vtool.other.rebuild_partition(part1_vals, part2_vals, part1_indexes, part2_indexes)`

Inverts work done by `index_partition`

Parameters

- `part1_vals` (*list*) –
- `part2_vals` (*list*) –
- `part1_indexes` (*dict*) –
- `part2_indexes` (*dict*) –

CommandLine: `python -m vtool.other --test-rebuild_partition`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> item_list = ['dist', 'fg', 'distinctiveness']
>>> part1_items = ['fg', 'distinctiveness']
>>> part1_indexes, part2_indexes = index_partition(item_list, part1_items)
>>> part1_vals = ut.take(item_list, part1_indexes)
>>> part2_vals = ut.take(item_list, part2_indexes)
>>> val_list = rebuild_partition(part1_vals, part2_vals, part1_indexes, part2_
↪indexes)
>>> assert val_list == item_list, 'incorrect inversin'
>>> print(val_list)
```

`vtool.other.rowwise_operation(arr1, arr2, op)`

DEPRICATE THIS IS POSSIBLE WITH STRICTLY BROADCASTING AND USING `np.newaxis`

DEPRICATE, numpy has better ways of doing this. Is the rowwise name correct? Should it be colwise?

performs an operation between an (N x A x B ... x Z) array with an (N x 1) array

`vtool.other.safe_argmax(arr, fill=nan, finite=False, nans=True)`

Doctest:

```
>>> from vtool.other import *
>>> assert safe_argmax([np.nan, np.nan], nans=False) == 0
>>> assert safe_argmax([-100, np.nan], nans=False) == 0
>>> assert safe_argmax([np.nan, -100], nans=False) == 1
>>> assert safe_argmax([-100, 0], nans=False) == 1
>>> assert np.isnan(safe_argmax([]))
```

`vtool.other.safe_cat(tup, axis=0, default_shape=(0,), default_dtype=<class 'numpy.float32'>)`

stacks a tuple even if it is empty Also deals with numpy bug where `cat` fails if an element in sequence is empty

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> import vtool as vt
>>> # test1
>>> tup = []
>>> ut.assert_eq(vt.safe_cat(tup, axis=0).shape, (0,))
>>> # test2
>>> tup = (np.array([[1, 2, 3]]), np.array([[ ]]))
>>> s = vt.safe_cat(tup, axis=0)
>>> print(ub.hzcat(['s = %s' % (ub.repr2(s), )]))
>>> ut.assert_eq(s.shape, (1, 3))
>>> # test3
>>> tup = (np.array([[1, 2, 3]]), np.array([[3, 4, 5]]))
>>> s = vt.safe_cat(tup, axis=1)
>>> print(ub.hzcat(['s = %s' % (ub.repr2(s), )]))
>>> ut.assert_eq(s.shape, (1, 6))
>>> # test3
>>> tup = (np.array(1), np.array(2), np.array(3))
>>> s = vt.safe_cat(tup, axis=1)
>>> print(ub.hzcat(['s = %s' % (ub.repr2(s), )]))
>>> ut.assert_eq(s.shape, (1, 6))

```

`vtool.other.safe_div(a, b)`

`vtool.other.safe_extreme(arr, op, fill=nan, finite=False, nans=True)`

Applies an extreme operation to an 1d array (typically max/min) but ensures a value is always returned even in operations without identities. The default identity must be specified using the *fill* argument.

Parameters

- **arr** (*ndarray*) – 1d array to take extreme of
- **op** (*func*) – vectorized operation like `np.max` to apply to array
- **fill** (*float*) – return type if arr has no elements (default = nan)
- **finite** (*bool*) – if True ignores non-finite values (default = False)
- **nans** (*bool*) – if False ignores nans (default = True)

`vtool.other.safe_max(arr, fill=nan, finite=False, nans=True)`

Parameters

- **arr** (*ndarray*) – 1d array to take max of
- **fill** (*float*) – return type if arr has no elements (default = nan)
- **finite** (*bool*) – if True ignores non-finite values (default = False)
- **nans** (*bool*) – if False ignores nans (default = True)

CommandLine: `python -m vtool.other safe_max -show`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> arrs = [[], [np.nan], [-np.inf, np.nan, np.inf], [np.inf], [np.inf, 1], [0,
↪ 1]]

```

(continues on next page)

(continued from previous page)

```

>>> arrs = [np.array(arr) for arr in arrs]
>>> fill = np.nan
>>> results1 = [safe_max(arr, fill, finite=False, nans=True) for arr in arrs]
>>> results2 = [safe_max(arr, fill, finite=True, nans=True) for arr in arrs]
>>> results3 = [safe_max(arr, fill, finite=True, nans=False) for arr in arrs]
>>> results4 = [safe_max(arr, fill, finite=False, nans=False) for arr in arrs]
>>> results = [results1, results2, results3, results4]
>>> result = ('results = %s' % (ub.repr2(results, nl=1),))
>>> print(result)
results = [
    [float('nan'), float('nan'), float('nan'), float('inf'), float('inf'), 1],
    [float('nan'), float('nan'), float('nan'), float('nan'), 1.0, 1],
    [float('nan'), float('nan'), float('nan'), float('nan'), 1.0, 1],
    [float('nan'), float('nan'), float('inf'), float('inf'), float('inf'), 1],
]

```

`vtool.other.safe_min(arr, fill=nan, finite=False, nans=True)`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> arrs = [[], [np.nan], [-np.inf, np.nan, np.inf], [np.inf], [np.inf, 1], [0, 1]]
>>> arrs = [np.array(arr) for arr in arrs]
>>> fill = np.nan
>>> results1 = [safe_min(arr, fill, finite=False, nans=True) for arr in arrs]
>>> results2 = [safe_min(arr, fill, finite=True, nans=True) for arr in arrs]
>>> results3 = [safe_min(arr, fill, finite=True, nans=False) for arr in arrs]
>>> results4 = [safe_min(arr, fill, finite=False, nans=False) for arr in arrs]
>>> results = [results1, results2, results3, results4]
>>> result = ('results = %s' % (ub.repr2(results, nl=1),))
>>> print(result)
results = [
    [float('nan'), float('nan'), float('nan'), float('inf'), 1.0, 0],
    [float('nan'), float('nan'), float('nan'), float('nan'), 1.0, 0],
    [float('nan'), float('nan'), float('nan'), float('nan'), 1.0, 0],
    [float('nan'), float('nan'), float('-inf'), float('inf'), 1.0, 0],
]

```

`vtool.other.safe_vstack(tup, default_shape=(0,), default_dtype=<class 'numpy.float32'>)`
stacks a tuple even if it is empty

`vtool.other.significant_shape(arr)`
find the shape without trailing 1's

`vtool.other.structure_rows(*arrs)`

CommandLine: `python -m vtool.other structure_rows`

SeeAlso: `unstructure_rows`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> arr1 = np.array([[609, 307], [ 95, 344], [ 1, 690]])
>>> arr2 = np.array([[ 422, 1148], [ 422, 968], [ 481, 1148], [ 750, 1132], [
↳759, 159]])
>>> arrs = (arr1, arr2)
>>> structured_arrs = structure_rows(*arrs)
>>> unstructured_arrs = unstructure_rows(*structured_arrs)
>>> assert np.all(unstructured_arrs[0] == arrs[0])
>>> assert np.all(unstructured_arrs[1] == arrs[1])
>>> union_ = np.union1d(*structured_arrs)
>>> union, = unstructure_rows(union_)
>>> assert len(union.shape) == 2
```

`vtool.other.take2(arr, index_list, axis=None, out=None)`

Wrapper around numpy compress that makes the signature more similar to take

`vtool.other.take_col_per_row(arr, colx_list)`

takes a column from each row

Ignore: num_rows = 1000 num_cols = 4

```
arr = np.arange(10 * 4).reshape(10, 4) colx_list = (np.random.rand(10) * 4).astype(np.int)
```

```
%timeit np.array([row[cx] for (row, cx) in zip(arr, colx_list)]) %timeit
arr.ravel().take(np.ravel_multi_index((np.arange(len(colx_list)), colx_list, arr.shape)) %timeit
arr.ravel().take(colx_list + np.arange(arr.shape[0]) * arr.shape[1])
```

`vtool.other.to_undirected_edges(directed_edges, upper=False)`

`vtool.other.trytake(list_, index_list)`

`vtool.other.unique_rows(arr, directed=True)`

Order or columns does not matter if directed = False

`vtool.other.unstructure_rows(*structured_arrs)`

SeeAlso: structure_rows

`vtool.other.weighted_average_scoring(fsv, weight_filtxs, nonweight_filtxs)`

does $\frac{\sum_i w^f_i * w^d_i * r_i}{\sum_i w^f_i * w^d_i}$ to get a weighed average of ratio scores

If we normalize the weight part to add to 1 then we can get per-feature scores.

References

http://en.wikipedia.org/wiki/Weighted_arithmetic_mean

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> fsv = np.array([
...     [ 0.82992172, 1.56136119, 0.66465378],
...     [ 0.8000412 , 2.14719748, 1.         ],
...     [ 0.80848503, 2.6816361 , 1.         ],
```

(continues on next page)

(continued from previous page)

```

...      [ 0.86761665,  2.70189977,  1.          ],
...      [ 0.8004055 ,  1.58753884,  0.92178345],])
>>> weight_filtxs = np.array([1, 2], dtype=np.int32)
>>> nonweight_filtxs = np.array([0], dtype=np.int32)
>>> new_fs = weighted_average_scoring(fsv, weight_filtxs, nonweight_filtxs)
>>> result = new_fs
>>> print(result)

```

`vtool.other.weighted_geometric_mean(data, weights)`

Parameters

- **data** (list of ndarrays)–
- **weights** (ndarray)–

Returns ndarray

CommandLine: `python -m vtool.other --test-weighted_geometric_mean`

References

https://en.wikipedia.org/wiki/Weighted_geometric_mean

SeeAlso: `scipy.stats.mstats.gmean`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> data = [.9, .5]
>>> weights = np.array([1.0, .5])
>>> gmean_ = weighted_geometric_mean(data, weights)
>>> result = ('gmean_ = %.3f' % (gmean_,))
>>> print(result)
gmean_ = 0.740

```

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> rng = np.random.RandomState(0)
>>> img1 = rng.rand(4, 4)
>>> img2 = rng.rand(4, 4)
>>> data = [img1, img2]
>>> weights = np.array([.5, .5])
>>> gmean_ = weighted_geometric_mean(data, weights)
>>> result = ub.hzcat(['gmean_ = %s' % (ub.repr2(gmean_, precision=2, with_
↳dtype=True), )])
>>> print(result)

```

Ignore: `res1 = ((img1 **.5 * img2 **.5)) **.5 res2 = np.sqrt(img1 * img2)`

`vtool.other.weighted_geometric_mean_unnormalized(data, weights)`

`vtool.other.zipcat (arr1_list, arr2_list, axis=None)`

Parameters

- **arr1_list** (*list*) –
- **arr2_list** (*list*) –
- **axis** (*None*) – (default = None)

Returns

Return type `list`

CommandLine: `python -m vtool.other --exec-zipcat --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> arr1_list = [np.array([0, 0, 0]), np.array([0, 0, 0, 0])]
>>> arr2_list = [np.array([1, 1, 1]), np.array([1, 1, 1, 1])]
>>> axis = None
>>> arr3_list = zipcat(arr1_list, arr2_list, axis)
>>> arr3_list0 = zipcat(arr1_list, arr2_list, axis=0)
>>> arr3_list1 = zipcat(arr1_list, arr2_list, axis=1)
>>> arr3_list2 = zipcat(arr1_list, arr2_list, axis=2)
>>> print('arr3_list = %s' % (ut.repr3(arr3_list),))
>>> print('arr3_list0 = %s' % (ut.repr3(arr3_list0),))
>>> print('arr3_list2 = %s' % (ut.repr3(arr3_list2),))
```

`vtool.other.zipcompress (arr_list, flags_list, axis=None)`

`vtool.other.zipcompress_safe (arr_list, flags_list, axis=None)`

`vtool.other.ziptake (arr_list, indices_list, axis=None)`

`vtool.other.zstar_value (conf_level=0.95)`

References

<http://stackoverflow.com/questions/28242593/correct-way-to-obtain-confidence-interval-with-sciPy>

1.32 vtool.patch module

`vtool.patch.GaussianBlurInplace (img, sigma, size=None)`

simulates code from helpers.cpp in hesaff

Parameters

- **img** (*ndarray*) –
- **sigma** (*float*) –

CommandLine: `python -m vtool.patch --test-GaussianBlurInplace:0 --show python -m vtool.patch --test-GaussianBlurInplace:1 --show`

References; http://www.cse.usf.edu/~r1k/MachineVisionBook/MachineVision.files/MachineVision_Chapter4.pdf http://en.wikipedia.org/wiki/Scale_space_implementation http://www.cse.psu.edu/~rtc12/CSE486/lecture10_6pp.pdf

Notes

The product of the convolution of two Gaussian functions with spread σ is a Gaussian function with spread $\sqrt{2} \cdot \sigma$ scaled by the area of the Gaussian filter

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> from mpl_toolkits.mplot3d import Axes3D # NOQA
>>> import wbia.plottool as pt
>>> img = get_test_patch('star2')
>>> img_orig = img.copy()
>>> sigma = .8
>>> GaussianBlurInplace(img, sigma)
>>> fig = pt.figure(fnum=1, pnum=(1, 3, 1))
>>> size = int((2.0 * 3.0 * sigma + 1.0))
>>> if not size & 1: # check if even
>>>     size += 1
>>> ksize = (size, size)
>>> fig.add_subplot(1, 3, 1, projection='3d')
>>> show_gaussian_patch(ksize, sigma, sigma)
>>> pt.imshow(img_orig * 255, fnum=1, pnum=(1, 3, 2))
>>> pt.imshow(img * 255, fnum=1, pnum=(1, 3, 3))
>>> pt.show_if_requested()
```

Example

```
>>> # DISABLE_DOCTEST
>>> # demonstrate cascading smoothing property
>>> # THIS ISNT WORKING WHY???
>>> from vtool.patch import * # NOQA
>>> from mpl_toolkits.mplot3d import Axes3D # NOQA
>>> import wbia.plottool as pt
>>> img = get_test_patch('star2')
>>> img1 = img.copy()
>>> img2 = img.copy()
>>> img3 = img.copy()
>>> img4 = img.copy()
>>> img_orig = img.copy()
>>> sigma1 = .6
>>> sigma2 = .9
>>> sigma3 = sigma1 + sigma2
>>> size = 7
>>> # components
>>> GaussianBlurInplace(img1, sigma1, size)
>>> GaussianBlurInplace(img2, sigma2, size)
>>> # all in one shot
>>> GaussianBlurInplace(img3, sigma3, size)
```

(continues on next page)

(continued from previous page)

```

>>> # additive
>>> GaussianBlurInplace(img4, sigma1, size)
>>> GaussianBlurInplace(img4, sigma2, size)
>>> print((img4 - img3).sum())
>>> # xdoctest: +REQUIRES(--show)
>>> fig = plt.figure(fnum=1, pnum=(2, 4, 1))
>>> ksize = (size, size)
>>> #fig.add_subplot(1, 3, 1, projection='3d')
>>> fig.add_subplot(2, 4, 1, projection='3d')
>>> show_gaussian_patch(ksize, sigma1, sigma1)
>>> fig.add_subplot(2, 4, 2, projection='3d')
>>> show_gaussian_patch(ksize, sigma2, sigma2)
>>> fig.add_subplot(2, 4, 3, projection='3d')
>>> show_gaussian_patch(ksize, sigma3, sigma3)
>>> plt.imshow(img_orig * 255, fnum=1, pnum=(2, 4, 4))
>>> plt.imshow(img1 * 255, fnum=1, pnum=(2, 4, 5), title='%r' % (sigma1))
>>> plt.imshow(img2 * 255, fnum=1, pnum=(2, 4, 6), title='%r' % (sigma2))
>>> plt.imshow(img3 * 255, fnum=1, pnum=(2, 4, 7), title='%r' % (sigma3))
>>> plt.imshow(img4 * 255, fnum=1, pnum=(2, 4, 8), title='%r + %r' % (sigma1,
↳ sigma2))
>>> plt.show_if_requested()

```

`vtool.patch.draw_kp_ori_steps()`

Shows steps in orientation estimation

CommandLine: `python -m vtool.patch -test-draw_kp_ori_steps -show -fname=zebra.png -fx=121`
`python -m vtool.patch -test-draw_kp_ori_steps -show -interact python -m vtool.patch -test-`
`draw_kp_ori_steps -save ~/latex/crall-candidacy-2015/figures/test_fint_kp_direction.jpg -dpath fig-`
`ures -caption=visualization of the steps in the computation of the dominant gradient orientations.'`
`-figsize=14,9 -dpi=160 -height=2.65 -left=.04 -right=.96 -top=.95 -bottom=.05 -wspace=.1 -hspace=.1`

`python -m vtool.patch -test-draw_kp_ori_steps -dpath ~/latex/crall-candidacy-2015/ -save fig-`
`ures/draw_kp_ori_steps.jpg -figsize=14,9 -dpi=180 -height=2.65 -left=.04 -right=.96 -top=.95 -bot-`
`tom=.05 -wspace=.1 -hspace=.1 -diskshow`

`python -m vtool.patch -test-draw_kp_ori_steps -dpath ~/latex/crall-candidacy-2015/ -save fig-`
`ures/draw_kp_ori_steps.jpg -figsize=14,9 -dpi=180 -djust=.04,.05,.1 -diskshow -fname=zebra.png`
`-fx=121`

Example

```

>>> # DISABLE_DOCTEST
>>> import wbia.plottool as pt
>>> from vtool.patch import * # NOQA
>>> draw_kp_ori_steps()
>>> pt.show_if_requested()

```

`vtool.patch.find_dominant_kp_orientations` (*imgBGR*, *kp*, *bins=36*, *maxima_thresh=0.8*, *DE-*
BUG_ROTINVAR=False)

References

http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf page 219

<http://www.cs.berkeley.edu/~malik/cs294/lowe-ijcv04.pdf> page 13.

Lowe uses a 36-bin histogram of edge orientations weighted by a gaussian distance to the center and gradient magnitude. He finds all peaks within 80% of the global maximum. Then he fine tunes the orientation using a 3-binned parabolic fit. Multiple orientations (and hence multiple keypoints) can be returned, but empirically only about 15% will have these and they do tend to be important.

Returns ori_offset - offset of current orientation to dominant orientation

Return type float

CommandLine: python -m vtool.patch --test-find_dominant_kp_orientations

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> import vtool as vt
>>> # build test data
>>> np.random.seed(0)
>>> #imgBGR = get_test_patch('cross', jitter=False)
>>> img_fpath = ut.grab_test_imgpath('star.png')
>>> imgBGR = vt.imread(img_fpath)
>>> kpts, vecs = vt.extract_features(img_fpath)
>>> assert len(kpts) == 1
>>> kp = kpts[0]
>>> print('kp = \n' + (vt.kp_cpp_infostr(kp)))
>>> bins = 36
>>> maxima_thresh = .8
>>> # execute function
>>> new_oris = find_dominant_kp_orientations(imgBGR, kp, bins,
>>>                                         maxima_thresh,
>>>                                         DEBUG_ROTINVAR=True)
>>> # verify results
>>> result = 'new_oris = %r' % (new_oris,)
```

`vtool.patch.find_kpts_direction (imgBGR, kpts, DEBUG_ROTINVAR=False)`

Parameters

- **imgBGR** (`ndarray [uint8_t, ndim=2]`) – image data in opencv format (blue, green, red)
- **kpts** (`ndarray [float32_t, ndim=2]`) – keypoints

Returns kpts - keypoints

Return type `ndarray [float32_t, ndim=2]`

CommandLine: python -m vtool.patch --test-find_kpts_direction

`vtool.patch.find_patch_dominant_orientations (patch, bins=36, maxima_thresh=0.8, DEBUG_ROTINVAR=False)`

helper

`vtool.patch.gaussian_average_patch (patch, sigma=None, copy=True)`

Parameters

- **patch** (`ndarray`) –
- **sigma** (`float`) –

CommandLine: `python -m vtool.patch --test-gaussian_average_patch`

References

<http://docs.opencv.org/modules/imgproc/doc/filtering.html#getgaussiankernel>

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> patch = get_star_patch()
>>> #sigma = 1.6
>>> sigma = None
>>> result = gaussian_average_patch(patch, sigma)
>>> print(result)
0.414210641527
```

Ignore: `import utool as ut import wbia.plottool as pt import vtool as vt import cv2`
`gauss_kernel_d0 = (cv2.getGaussianKernel(patch.shape[0], sigma)) gauss_kernel_d1 =`
`(cv2.getGaussianKernel(patch.shape[1], sigma)) weighted_patch = patch.copy() weighted_patch =`
`np.multiply(weighted_patch, gauss_kernel_d0) weighted_patch = np.multiply(weighted_patch.T,`
`gauss_kernel_d1).T gaussian_kern2 = gauss_kernel_d0.dot(gauss_kernel_d1.T) fig = pt.figure(fnum=1,`
`pnum=(1, 3, 1), doclf=True, docla=True) pt.imshow(patch * 255) fig = pt.figure(fnum=1, pnum=(1,`
`3, 2)) pt.imshow(ut.norm_zero_one(gaussian_kern2) * 255.0) fig = pt.figure(fnum=1, pnum=(1, 3, 3))`
`pt.imshow(ut.norm_zero_one(weighted_patch) * 255.0) pt.update()`

`vtool.patch.gaussian_patch(shape=(7, 7), sigma=1.0)`
 another version of the `gaussian_patch` function. hopefully better

References

<http://docs.opencv.org/modules/imgproc/doc/filtering.html#getgaussiankernel>

Parameters

- **shape** (*tuple*) – array dimensions
- **sigma** (*float*) –

CommandLine: `python -m vtool.patch --test-gaussian_patch --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> #shape = (7, 7)
>>> shape = (24, 24)
>>> sigma = None # 1.0
>>> gausspatch = gaussian_patch(shape, sigma)
>>> sum_ = gausspatch.sum()
>>> ut.assert_almost_eq(sum_, 1.0)
>>> # xdoctest: +REQUIRES(--show)
```

(continues on next page)

(continued from previous page)

```
>>> import wbia.plottool as pt
>>> pt.imshow(vt.norm01(gausspatch) * 255)
>>> ut.show_if_requested()
```

`vtool.patch.gaussian_weight_patch(patch, sigma=None)`

Applies two one dimensional gaussian operations to a patch which effectively weights it by a 2-dimensional gaussian. This is efficient because the actually 2-d gaussian never needs to be allocated.

`test_show_gaussian_patches`

`vtool.patch.generate_to_patch_transforms(kpts, patch_size=41)`

`vtool.patch.get_cross_patch(jitter=False)`
`test data patch`

`vtool.patch.get_no_symbol(variant='symbol', size=(100, 100))`

Returns errorimg

Return type ndarray

CommandLine: `python -m vtool.patch --test-get_no_symbol --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> errorimg = get_no_symbol()
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(errorimg)
>>> ut.show_if_requested()
```

`vtool.patch.get_orientation_histogram(gori, gori_weights, bins=36, DE-BUG_ROTINVAR=False)`

Parameters

- **gori** –
- **gori_weights** –
- **bins** (*int*) –

Returns (hist, centers)

Return type tuple

CommandLine: `python -m vtool.patch --test-get_orientation_histogram`

Ignore: `print(vt.kpts_docrepr(gori, 'gori = ')) print(vt.kpts_docrepr(gori_weights, 'gori_weights = '))`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> # build test data
>>> gori = np.array([[ 0. ,  0. ,  3.14,  3.14,  0. ],
```

(continues on next page)

(continued from previous page)

```

...         [ 4.71,  6.15,  3.13,  3.24,  4.71],
...         [ 4.71,  4.61,  0.5 ,  4.85,  4.71],
...         [ 1.57,  6.28,  3.14,  3.14,  1.57],
...         [ 0. ,  0. ,  3.14,  3.14,  0.  ]])
>>> gori_weights = np.array([[ 0. ,  0.11,  0.02,  0.13,  0.  ],
...                          [ 0.02,  0.19,  0.02,  0.21,  0.02],
...                          [ 0.11,  0.16,  0. ,  0.13,  0.11],
...                          [ 0. ,  0.17,  0.02,  0.19,  0.  ],
...                          [ 0. ,  0.11,  0.02,  0.13,  0.  ]])
>>> bins = 36
>>> # execute function
>>> (hist, centers) = get_orientation_histogram(gori, gori_weights, bins)
>>> # verify results
>>> result = str((hist, centers))
>>> print(result)

```

```
vtool.patch.get_star2_patch(jitter=False)
test data patch
```

```
vtool.patch.get_star_patch(jitter=False)
test data patch
```

```
vtool.patch.get_stripe_patch(jitter=False)
test data patch
```

```
vtool.patch.get_test_patch(key='star', jitter=False)
```

Parameters

- **key** (*str*) –
- **jitter** (*bool*) –

Returns patch

Return type ndarray

CommandLine: python -m vtool.patch -test-get_test_patch -show

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> import wbia.plottool as pt
>>> key = 'star2'
>>> jitter = False
>>> patch = get_test_patch(key, jitter)
>>> pt.imshow(255 * patch)
>>> pt.show_if_requested()

```

```
vtool.patch.get_unwarped_patch(imgBGR, kp, gray=False)
```

Returns unwarped warped patch around a keypoint

Parameters

- **img** (*ndarray*) – array representing an image
- **kpt** (*ndarray*) – keypoint ndarray in [x, y, a, c, d, theta] format

Returns (wpatch, wkp) the normalized 41x41 patches from the img corresponding to the keypoint

Return type `tuple`

`vtool.patch.get_unwarped_patches (img, kpts)`

Returns cropped unwarped (keypoint is still elliptical) patch around a keypoint

Parameters

- **img** (`ndarray`) – array representing an image
- **kpts** (`ndarrays`) – keypoint ndarrays in [x, y, a, c, d, theta] format

Returns

(**patches**, **subkpts**) - the unnormalized patches from the **img** corresponding to the keypoint

Return type `tuple`

`vtool.patch.get_warped_patch (imgBGR, kp, gray=False, flags=4, borderMode=1, patch_size=41)`

Returns warped (into a unit circle) patch around a keypoint

Parameters

- **img** (`ndarray`) – array representing an image
- **kpt** (`ndarray`) – keypoint ndarray in [x, y, a, c, d, theta] format

Returns

(**wpatch**, **wkp**) the normalized 41x41 patches from the **img** corresponding to the keypoint

Return type (`ndarray`, `ndarray`)

`vtool.patch.get_warped_patches (img, kpts, flags=4, borderMode=1, patch_size=41, use_cpp=False)`

Returns warped (into a unit circle) patch around a keypoint

FIXME: there is a slight translation difference in the way Python extracts patches and the way C++ extracts patches. C++ should be correct. TODO: have C++ able to extract color.

Parameters

- **img** (`ndarray [uint8_t, ndim=2]`) – array representing an image
- **kpts** (`ndarray [float32_t, ndim=2]`) – list of keypoint ndarrays in [[x, y, a, c, d, theta]] format
- **flags** (`long`) – cv2 interpolation flags
- **borderMode** (`long`) – cv2 border flags
- **patch_size** (`int`) – resolution of resulting image patch

Returns

(**warped_patches**, **warped_subkpts**) the normalized 41x41 patches from the **img** corresponding to the keypoint

Return type (`list`, `list`)

CommandLine: `python -m vtool.patch -test-get_warped_patches -show -use_cpp` `python -m vtool.patch -test-get_warped_patches -show -use_python`

Example

```
>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.patch import * # NOQA
>>> import vtool as vt
>>> import ubelt as ub
>>> # build test data
>>> img_fpath = ut.grab_test_imgpath('carl.jpg')
>>> img = vt.imread(img_fpath)
>>> use_cpp = ut.get_argflag('--use_cpp')
>>> kpts, desc = vt.extract_features(img_fpath)
>>> kpts = kpts[0:1]
>>> flags = cv2.INTER_LANCZOS4
>>> borderMode = cv2.BORDER_REPLICATE
>>> # execute function
>>> (warped_patches, warped_subkpts) = get_warped_patches(img, kpts, flags,
↳borderMode, use_cpp=use_cpp)
>>> # verify results
>>> print(np.array(warped_patches).shape)
>>> print(ub.repr2(np.array(warped_subkpts), precision=2))
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(warped_patches[0])
>>> #pt.draw_kpts2(warped_subkpts, pts=True, rect=True)
>>> pt.set_title('use_cpp = %r' % (use_cpp,))
>>> pt.show_if_requested()
```

`vtool.patch.gradient_fill(shape, theta=0, flip=False, vert=False, style='linear')`

FIXME: angle does not work properly

CommandLine: `python -m vtool.patch gradient_fill --show`

Example

```
>>> from vtool.patch import * # NOQA
>>> import vtool as vt
>>> shape = (9, 9)
>>> #style = 'linear'
>>> style = 'step'
>>> theta = np.pi / 4
>>> patch = vt.gradient_fill(shape, theta, style=style)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(vt.rectify_to_uint8(patch))
>>> ut.show_if_requested()
```

`vtool.patch.intern_warp_single_patch(img, x, y, ori, V, patch_size, flags=2, borderMode=1)`

Ignore:

```
>>> # https://groups.google.com/forum/#!topic/sympy/k1HnZK_bNNA
>>> from vtool.patch import * # NOQA
>>> import sympy
>>> from sympy.abc import theta
>>> ori = theta
>>> x, y, a, c, d, patch_size = sympy.symbols('x y a c d S')
```

(continues on next page)

(continued from previous page)

```

>>> half_patch_size = patch_size / 2
>>>
>>> def sympy_rotation_mat3x3(radians):
>>>     # TODO: handle array inputs
>>>     sin_ = sympy.sin(radians)
>>>     cos_ = sympy.cos(radians)
>>>     R = np.array(((cos_, -sin_, 0),
>>>                   (sin_, cos_, 0),
>>>                   (0, 0, 1)),)
>>>     return sympy.Matrix(R)
>>>
>>> kpts = np.array([[x, y, a, c, d, ori]])
>>> kp = ktool.get_invV_mats(kpts, with_trans=True)[0]
>>> invV = sympy.Matrix(kp)
>>> V = invV.inv()
>>> ss = sympy.sqrt(patch_size) * 3.0
>>> T = sympy.Matrix(ltool.translation_mat3x3(-x, -y, None)) # Center the_
↳patch
>>> R = sympy_rotation_mat3x3(-ori) # Rotate the centered unit circle patch
>>> S = sympy.Matrix(ltool.scale_mat3x3(ss, dtype=None)) # scale from unit_
↳circle to the patch size
>>> X = sympy.Matrix(ltool.translation_mat3x3(half_patch_size, half_patch_
↳size, None)) # Translate back to patch-image coordinates
>>>
>>> sympy.MatMul(X, S, hold=True)
>>>
>>> def add_matmul_hold_prop(mat):
>>>     #import functools
>>>     def matmul_hold(other, hold=True):
>>>         new = sympy.MatMul(mat, other, hold=hold)
>>>         add_matmul_hold_prop(new)
>>>         return new
>>>     #matmul_hold = functools.partial(sympy.MatMul, mat, hold=True)
>>>     setattr(mat, 'matmul_hold', matmul_hold)
>>> add_matmul_hold_prop(X)
>>> add_matmul_hold_prop(S)
>>> add_matmul_hold_prop(R)
>>> add_matmul_hold_prop(V)
>>> add_matmul_hold_prop(T)
>>>
>>> M = X.matmul_hold(S).matmul_hold(R).matmul_hold(V).matmul_hold(T)
>>> #M = X.multiply(S).multiply(R).multiply(V).multiply(T)
>>>
>>>
>>> V_full = R.multiply(V).multiply(T)
>>> sympy.latex(V_full)
>>> print(sympy.latex(R.multiply(V).multiply(T)))
>>> print(sympy.latex(X))
>>> print(sympy.latex(S))
>>> print(sympy.latex(R))
>>> print(sympy.latex(invV) + '^{-1}')
>>> print(sympy.latex(T))

```

vtool.patch.inverted_sift_patch(sift, dim=32)

Idea for inverted sift visualization

CommandLine: python -m vtool.patch test_sift_viz --show --name=star python -m vtool.patch test_sift_viz

```
-show -name=star2 python -m vtool.patch test_sift_viz -show -name=cross python -m vtool.patch
test_sift_viz -show -name=stripe
```

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> import vtool as vt
>>> patch = vt.get_test_patch(ut.get_argval('--name', default='star'))
>>> sift = vt.extract_feature_from_patch(patch)
>>> siftimg = test_sift_viz(sift)
>>> # Need to do some image blending
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.figure(fnum=1, pnum=(1, 2, 1))
>>> pt.mpl_sift.draw_sift_on_patch(siftimg, sift)
>>> pt.figure(fnum=1, pnum=(1, 2, 2))
>>> patch2 = patch
>>> patch2 = vt.rectify_to_uint8(patch2)
>>> patch2 = vt.rectify_to_square(patch2)
>>> pt.mpl_sift.draw_sift_on_patch(patch2, sift)
>>> ut.show_if_requested()
```

```
vtool.patch.make_test_image_keypoints (imgBGR, scale=1.0, skew=0, theta=0, shift=(0, 0))
```

```
vtool.patch.patch_gaussian_weighted_average_intensities (probchip, kpts_)
```

```
vtool.patch.patch_gradient (patch, ksize=1, gaussian_weighted=False)
```

```
vtool.patch.patch_mag (gradx, grady)
```

```
vtool.patch.patch_ori (gradx, grady)
```

returns patch orientation relative to the x-axis

```
vtool.patch.show_gaussian_patch (shape, sigma1, sigma2)
```

```
vtool.patch.show_patch_orientation_estimation (imgBGR, kpts, patch, gradx, grady,
                                                gmag, gori, hist, centers, gori_weights,
                                                fx=None)
```

```
vtool.patch.test_ondisk_find_patch_fpath_dominant_orientations (patch_fpath,
                                                                bins=36, max-
                                                                ima_thresh=0.8,
                                                                DE-
                                                                BUG_ROTINVAR=True)
```

Parameters

- `patch_fpath` –
- `bins` (*int*) –
- `maxima_thresh` (*float*) –

CommandLine: `python -m vtool.patch -test-test_ondisk_find_patch_fpath_dominant_orientations`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> import wbia.plottool as pt
>>> # build test data
>>> patch_fpath = ut.get_argval('--patch-fpath', type_=str, default=ut.grab_test_
↳imgpath('star.png'))
>>> bins = 36
>>> maxima_thresh = 0.8
>>> test_ondisk_find_patch_fpath_dominant_orientations(patch_fpath, bins, maxima_
↳thresh)
>>> pt.show_if_requested()
```

`vtool.patch.test_show_gaussian_patches(shape=(19, 19))`

CommandLine: python -m vtool.patch -test-test_show_gaussian_patches -show python -m vtool.patch -test-test_show_gaussian_patches -show -shape=7,7 python -m vtool.patch -test-test_show_gaussian_patches -show -shape=17,17 python -m vtool.patch -test-test_show_gaussian_patches -show -shape=41,41 python -m vtool.patch -test-test_show_gaussian_patches -show -shape=29,29 python -m vtool.patch -test-test_show_gaussian_patches -show -shape=41,7

References

http://matplotlib.org/examples/mplot3d/surface3d_demo.html

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> from mpl_toolkits.mplot3d import Axes3D # NOQA
>>> import wbia.plottool as pt
>>> shape = ut.get_argval('--shape',), type_=list, default=[19, 19])
>>> test_show_gaussian_patches(shape=shape)
>>> pt.show_if_requested()
```

`vtool.patch.test_show_gaussian_patches2(shape=(19, 19))`

CommandLine: python -m vtool.patch -test-test_show_gaussian_patches2 -show python -m vtool.patch -test-test_show_gaussian_patches2 -show -shape=7,7 python -m vtool.patch -test-test_show_gaussian_patches2 -show -shape=19,19 python -m vtool.patch -test-test_show_gaussian_patches2 -show -shape=41,41 python -m vtool.patch -test-test_show_gaussian_patches2 -show -shape=41,7

References

http://matplotlib.org/examples/mplot3d/surface3d_demo.html

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> from mpl_toolkits.mplot3d import Axes3D # NOQA
>>> import wbia.plottool as pt
>>> shape = ut.get_argval('--shape', type_=list, default=[19, 19])
>>> test_show_gaussian_patches2(shape=shape)
>>> pt.show_if_requested()
```

```
vtool.patch.testdata_patch()
```

1.33 vtool.quality_classifier module

References

% Single-image noise level estimation for blind denoising. % <http://www.ok.ctrl.titech.ac.jp/res/NLE/TIP2013-noise-level-estimation06607209.pdf>

```
vtool.quality_classifier.compute_average_contrast(img)
```

CommandLine: python -m vtool.quality_classifier --exec-compute_average_contrast --show

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.quality_classifier import * # NOQA
>>> import vtool as vt
>>> img_fpath = ut.grab_test_imgpath('car1.jpg')
>>> img = vt.imread(img_fpath, grayscale=True)
>>> average_contrast, gradmag_sqrd = compute_average_contrast(img)
>>> # xdoctest: +REQUIRES(module:plottool)
>>> import wbia.plottool as pt
>>> # xdoctest: +REQUIRES(--show)
>>> pt.figure(fnum=1)
>>> pt.plt.imshow(gradmag_sqrd)
>>> ut.show_if_requested()
```

```
vtool.quality_classifier.contrast_measures(img)
```

```
vtool.quality_classifier.fourier_devtest(img)
```

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

CommandLine: python -m vtool.quality_classifier --test-fourier_devtest --show

References

http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_imgproc/py_transforms/py_fourier_transform/py_fourier_transform.html <http://cns-alumni.bu.edu/~slehar/fourier/fourier.html>

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.quality_classifier import * # NOQA
>>> import vtool as vt
>>> img_fpath = ut.grab_test_imgpath('carl.jpg')
>>> img = vt.imread(img_fpath, grayscale=True)
>>> magnitude_spectrum = fourier_devtest(img)
```

```
vtool.quality_classifier.test_average_contrast()
```

1.34 vtool.score_normalization module

class vtool.score_normalization.ScoreNormVisualizeClass

Bases: object

HACK; eventually move all individual plots into a class structure

class vtool.score_normalization.ScoreNormalizer(**kwargs)

Bases: utool.util_cache.Cachable, vtool.score_normalization.ScoreNormVisualizeClass

Conforms to scikit-learn Estimator interface

CommandLine: python -m vtool.score_normalization -test-ScoreNormalizer -show -cmd

Kwargs: tpr (float): target true positive rate (default .90) fpr (float): target false positive rate (default None)
reverse (bool): True if lower scores are better, False if higher scores are better (default=None)

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.score_normalization import * # NOQA
>>> import vtool as vt
>>> encoder = ScoreNormalizer()
>>> X, y = vt.demodata.testdata_binary_scores()
>>> attrs = {'index': np.arange(len(y)) * ((2 * y) - 1)}
>>> encoder.fit(X, y, attrs)
>>> # xdoctest: +REQUIRES(--show)
>>> encoder.visualize()
>>> ut.show_if_requested()
```

fit (X, y, attrs=None, verbose=False, finite_only=True)

Fits estimator to data

Parameters

- **X** (ndarray) – one dimensional scores
- **y** (ndarray) – binary labels
- **attrs** (dict) – dictionary of data attributes

fit_partitioned (tp_scores, tn_scores, part_attrs=None, **kwargs)

convenience func to fit only scores that have been separated instead of labeled

get_accuracy (X, y)

get_confusion_indicies (X, y)
combination of get_correct_indices and get_error_indicies

get_correct_indices (X, y)

Parameters

- **X** (*ndarray*) – data
- **y** (*ndarray*) – labels

Returns (fp_indicies, fn_indicies)

Return type `tuple`

CommandLine: `python -m vtool.score_normalization --test-get_correct_indices`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.score_normalization import * # NOQA
>>> encoder, X, y = testdata_score_normalier()
>>> (tp_indicies, tn_indicies) = encoder.get_correct_indices(X, y)
>>> tp_X = X.take(tp_indicies)[0:3]
>>> tn_X = X.take(tn_indicies)[0:3]
>>> result = 'tp_X = ' + ub.repr2(tp_X)
>>> result += '\ntn_X = ' + ub.repr2(tn_X)
>>> print(result)
tp_X = np.array([ 8.883,  8.77 ,  8.759])
tn_X = np.array([ 0.727,  0.76 ,  0.841])
```

get_error_indicies (X, y)
Returns the indicies of the most difficult type I and type II errors.

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.score_normalization import * # NOQA
>>> encoder, X, y = testdata_score_normalier()
>>> (fp_indicies, fn_indicies) = encoder.get_error_indicies(X, y)
>>> fp_X = X.take(fp_indicies)[0:3]
>>> fn_X = X.take(fn_indicies)[0:3]
>>> result = 'fp_X = ' + ub.repr2(fp_X)
>>> result += '\nfn_X = ' + ub.repr2(fn_X)
>>> print(result)
fp_X = np.array([ 6.196,  5.912,  5.804])
fn_X = np.array([ 3.947,  4.277,  4.43 ])
```

get_partitioned_support ()
convinience get prepartitioned data

get_prefix ()

get_support (*finite_only=True*)
return X, y, and attrs

inverse_normalize (*probs*)

learn_probabilities (*verbose=False*)

Kernel density estimation

learn_threshold (*verbose=False, **thresh_kw*)

Learns cutoff threshold that achieves the target confusion metric Typically a desired false positive rate (recall) is specified

learn_threshold2 ()

Finds a cutoff where the probability of a truepos stats becoming greater than probability of trueneg

CommandLine: python -m vtool.score_normalization --exec-learn_threshold2 --show

Example

```
>>> from vtool.score_normalization import * # NOQA
>>> import vtool as vt
>>> #encoder, X, y = testdata_score_normalier([(3.5, 256), (9.5, 1024), (15.5,
↳ 2048)], [(6.5, 256), (12.5, 5064), (18.5, 128)], adjust=1, p_tp_method=
↳ 'ratio')
>>> encoder, X, y = testdata_score_normalier([(3.5, 64), (9.5, 1024), (15.5,
↳ 5064)], [(6.5, 256), (12.5, 2048), (18.5, 128)], adjust=1, p_tp_method=
↳ 'ratio')
>>> #encoder, X, y = testdata_score_normalier(adjust=1)
>>> #encoder, X, y = testdata_score_normalier([(3.5, 2048)], [(30.5, 128)],
↳ tn_scale=.1, adjust=1)
>>> #encoder, X, y = testdata_score_normalier([(0, 64)], [(-.1, 12)],
↳ adjust=8, min_clip=0)
>>> locals_ = ut.exec_func_src(encoder.learn_threshold2)
>>> exec(ut.execstr_dict(locals_))
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.ensureqt()
>>> #pt.plot(xdata[0:-2], np.diff(np.diff(closeness)))
>>> #maxima_x, maxima_y, argmaxima = vt.hist_argmaxima(closeness)
>>> fnum = 100
>>> pt.multi_plot(xdata, [tp_curve, tn_curve, closeness, ],
>>>                 label_list=['p(tp | s)', 'p(tn | s)', 'closeness', ],
↳ marker='t',
>>>                 linewidth_list=[4, 4, 1,], title='intersection points',
>>>                 pnum=(4, 1, 1), fnum=fnum, xmax=xdata.max(), xmin=0)
>>> pt.plot(xdata[argmaxima], closeness[argmaxima], 'rx', label='closeness_
↳ maxima')
>>> pt.plot(x_submax, y_submax, 'o', label='chosen')
>>> #pt.plot(xdata[argmaxima], curveness[argmaxima], 'rx', label='curveness_
↳ maxima')
>>> pt.legend()
>>> #pt.plot(x_submax, y_submax, 'o')
>>> pt.plot(xdata[argmaxima], tp_curve[argmaxima], 'rx')
>>> pt.plot(xdata[argmaxima], tn_curve[argmaxima], 'rx')
>>> pt.plot(xdata[argmaxima], tp_curve[argmaxima], 'rx')
>>> pt.plot(xdata[argmaxima], tn_curve[argmaxima], 'rx')
>>> #pt.plot(xdata[argmaxima], encoder.interp_fn(x_submax), 'rx')
>>> _mkinterp = ut.partial(
>>>     scipy.interpolate.interp1d, kind='linear', copy=False,
>>>     assume_sorted=False, bounds_error=False)
>>> _interp_sgtn = _mkinterp(xdata, tn_curve)
>>> _interp_sgtp = _mkinterp(xdata, tp_curve)
```

(continues on next page)

(continued from previous page)

```

>>> pt.plot(x_submax, _interp_sgtn(x_submax), 'go')
>>> pt.plot(x_submax, _interp_sgtp(x_submax), 'bx')
>>> #
>>> pt.multi_plot(xdata[argmaxima], [tp_area, fp_area, tn_area, fn_area],
↳title='intersection areas',
>>>             label_list=['tp_area', 'fp_area', 'tn_area', 'fn_area'],
↳markers=['o', 'd', 'o', '.'],
>>>             pnum=(4, 1, 2), fnum=fnum, xmax=xdata.max(), xmin=0)
>>> #
>>> pt.multi_plot(xdata[argmaxima], [lr_pos, lr_neg, acc], title=
↳'intersection quality (likelihood ratios)',
>>>             label_list=['lr_pos=tp/fp', 'lr_neg=fn/tn', 'acc'],
↳markers=['o', 'o', '*'],
>>>             pnum=(4, 1, 3), fnum=fnum, xmax=xdata.max(), xmin=0)
>>> #
>>> pnum_ = pt.make_pnum_nextgen(4, 3, start=9)
>>> encoder._plot_score_support_hist(fnum=fnum, pnum=pnum_())
>>> #encoder._plot_prebayes(fnum=fnum, pnum=pnum_())
>>> encoder._plot_postbayes(fnum=fnum, pnum=pnum_())
>>> encoder._plot_roc(fnum=fnum, pnum=pnum_())
>>> pt.adjust_subplots(hspace=.5, top=.95, bottom=.08)
>>> pt.show_if_requested()

```

normalize_scores (X)**predict** (X)

Predict true or false of X.

visualize (**kwargs)

shows details about the score normalizer

Kwargs: fnum figtitle with_hist interactive with_scores with_roc with_precision_recall**CommandLine:** python -m vtool.score_normalization --exec-ScoreNormalizer.visualize:0 --show python
-m vtool.score_normalization --exec-ScoreNormalizer.visualize:1 --show

Example

```

>>> # UNSTABLE_DOCTEST
>>> from vtool.score_normalization import * # NOQA
>>> import vtool as vt
>>> encoder = ScoreNormalizer()
>>> X, y = vt.demodata.testdata_binary_scores()
>>> encoder.fit(X, y)
>>> kwargs = dict(
>>>     with_pr=True, interactive=True, with_roc=True,
>>>     with_hist=True)
>>> encoder.visualize(**kwargs)
>>> ut.show_if_requested()

```

Example

```

>>> # UNSTABLE_DOCTEST
>>> from vtool.score_normalization import * # NOQA

```

(continues on next page)

(continued from previous page)

```

>>> import vtool as vt
>>> encoder = ScoreNormalizer()
>>> X, y = vt.demodata.testdata_binary_scores()
>>> encoder.fit(X, y)
>>> kwargs = dict(
>>>     with_pr=True, interactive=True, with_roc=True, with_hist=True,
>>>     with_scores=False, with_prebayes=False, with_postbayes=False)
>>> encoder.visualize(target_tpr=.95, **kwargs)
>>> ut.show_if_requested()

```

`vtool.score_normalization.check_unused_kwargs(kwargs, expected_keys)`

`vtool.score_normalization.estimate_pdf(data, gridsize=1024, adjust=1)`

References; <http://statsmodels.sourceforge.net/devel/generated/statsmodels.nonparametric.kde.KDEUnivariate.html> <https://jakevdp.github.io/blog/2013/12/01/kernel-density-estimation/>

Parameters

- **data** (*ndarray*) – 1 dimensional data of float64
- **gridsize** (*int*) – domain size
- **adjust** (*int*) – smoothing factor

Returns data_pdf

Return type ndarray

Example

```

>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.score_normalization import * # NOQA
>>> import vtool as vt
>>> rng = np.random.RandomState(0)
>>> data = rng.randn(1000)
>>> data_pdf = vt.estimate_pdf(data)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.plot(data_pdf.support[:-1], np.diff(data_pdf.cdf))
>>> ut.show_if_requested()

```

`vtool.score_normalization.find_clip_range(tp_support, tn_support, clip_factor=2.6180339887499997, reverse=None)`

TODO: generalize to arbitrary domains (not just 0->inf)

Finds score to clip true positives past. This is useful when the highest true positive scores can be much larger than the highest true negative score.

Parameters

- **tp_support** (*ndarray*) –
- **tn_support** (*ndarray*) –
- **clip_factor** (*float*) – factor of the true negative domain to search for true positives

Returns min_score, max_score

Return type tuple

CommandLine: `python -m vtool.score_normalization --test-find_clip_range`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.score_normalization import * # NOQA
>>> tp_support = np.array([100, 200, 50000])
>>> tn_support = np.array([10, 30, 110])
>>> clip_factor = ut.PHI + 1
>>> min_score, max_score = find_clip_range(tp_support, tn_support, clip_factor)
>>> result = '%.4f, %.4f' % (min_score, max_score)
>>> print(result)
10.0000, 287.9837
```

`vtool.score_normalization.flatten_scores(tp_scores, tn_scores, part_attrs=None)`
 convenience helper to translate partitioned to unpartitioned data

Parameters

- **tp_scores** (*ndarray*) –
- **tn_scores** (*ndarray*) –
- **part_attrs** (*dict*) – (default = None)

Returns (scores, labels, attrs)

Return type `tuple`

CommandLine: `python -m vtool.score_normalization --test-flatten_scores`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.score_normalization import * # NOQA
>>> tp_scores = np.array([5, 6, 6, 7])
>>> tn_scores = np.array([1, 2, 2])
>>> part_attrs = {
...     1: {'qaid': [21, 24, 25, 26]},
...     0: {'qaid': [11, 14, 15]},
... }
>>> tup = flatten_scores(
...     tp_scores, tn_scores, part_attrs)
>>> (X, y, attrs) = tup
>>> y = y.astype(np.int)
>>> resdict = ut.odict(zip(['X', 'y', 'attrs'], [X, y, attrs]))
>>> result = ub.repr2(resdict, nobraces=True, with_dtype=False,
>>>                    explicit=1, nl=1)
>>> print(result)
X=np.array([5, 6, 6, 7, 1, 2, 2]),
y=np.array([1, 1, 1, 1, 0, 0, 0]),
attrs='qaid': np.array([21, 24, 25, 26, 11, 14, 15]),
```

`vtool.score_normalization.get_left_area(ydata, xdata, index_list)`
 area to the left of each index point

`vtool.score_normalization.get_right_area(ydata, xdata, index_list)`
 area to the right of each index point

```
vtool.score_normalization.inspect_pdfs (tn_support,      tp_support,      score_domain,
                                         p_tp_given_score,      p_tn_given_score,
                                         p_score_given_tp,      p_score_given_tn,      p_score,
                                         prob_thresh=None,      score_thresh=None,
                                         with_scores=False,      with_roc=False,
                                         with_precision_recall=False,      with_hist=False,
                                         fnum=None,      figtitle=None,      interac-
                                         tive=None,      use_stems=None,      part_attrs=None,
                                         thresh_kw=None,      attr_callback=None,
                                         with_prebayes=True,      with_postbayes=True,
                                         score_range=None, **kwargs)
```

Shows plots of learned thresholds

CommandLine: python -m vtool.score_normalization --test-ScoreNormalizer --show python -m vtool.score_normalization --exec-ScoreNormalizer.visualize --show

```
vtool.score_normalization.learn_score_normalization (tp_support,      tn_support,
                                                       gridsize=1024,      ad-
                                                       just=8,      return_all=False,
                                                       monotonize=True,
                                                       clip_factor=2.6180339887499997,
                                                       verbose=False,      reverse=False,
                                                       p_tp_method='eq')
```

Takes collected data and applies parzen window density estimation and bayes rule.

#True positive scores must be larger than true negative scores. FIXME: might be an issue with pdfs summing to 1 here.

Parameters

- **tp_support** (*ndarray*) –
- **tn_support** (*ndarray*) –
- **gridsize** (*int*) – default 512
- **adjust** (*int*) – default 8
- **return_all** (*bool*) – default False
- **monotonize** (*bool*) – default True
- **clip_factor** (*float*) – default ϕ^2

Returns (score_domain, p_tp_given_score, p_tn_given_score, p_score_given_tp, p_score_given_tn, p_score)

Return type `tuple`

CommandLine: python -m vtool.score_normalization --test-learn_score_normalization

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.score_normalization import * # NOQA
>>> tp_support = np.linspace(100, 10000, 512)
>>> tn_support = np.linspace(0, 120, 512)
>>> gridsize = 1024
>>> adjust = 8
>>> return_all = False
```

(continues on next page)

(continued from previous page)

```

>>> monotonize = True
>>> clip_factor = 2.6180339887499997
>>> verbose = True
>>> reverse = False
>>> (score_domain, p_tp_given_score) = learn_score_normalization(tp_support, tn_
↳support)
>>> result = '%.2f' % (np.diff(p_tp_given_score).sum())
>>> print(result)
0.99

```

`vtool.score_normalization.normalize_scores(score_domain, p_tp_given_score, scores, interp_fn=None)`

Adjusts a raw scores to a probabilities based on a learned normalizer

Parameters

- **score_domain** (*ndarray*) – input score domain
- **p_tp_given_score** (*ndarray*) – learned probability mapping
- **scores** (*ndarray*) – raw scores

Returns probabilities

Return type *ndarray*

CommandLine: `python -m vtool.score_normalization --test-normalize_scores`

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.score_normalization import * # NOQA
>>> score_domain = np.linspace(0, 10, 10)
>>> p_tp_given_score = (score_domain ** 2) / (score_domain.max() ** 2)
>>> scores = np.array([-1, 0.0, 0.01, 2.3, 8.0, 9.99, 10.0, 10.1, 11.1])
>>> prob = normalize_scores(score_domain, p_tp_given_score, scores)
>>> #np.set_printoptions(suppress=True)
>>> result = ub.repr2(prob, precision=2, suppress_small=True)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.plot2(score_domain, p_tp_given_score, 'r-x', equal_aspect=False, label=
↳'learned probability')
>>> pt.plot2(scores, prob, 'yo', equal_aspect=False, title='Normalized scores',
↳pad=.2, label='query points')
>>> pt.legend('upper left')
>>> ut.show_if_requested()
np.array([ 0. ,  0. ,  0. ,  0.05,  0.64,  1. ,  1. ,  1. ,  1. ],
↳dtype=np.float64)

```

`vtool.score_normalization.partition_scores(X, y, attrs=None)`

convenience helper to translate partitioned to unpartitioned data

Parameters

- **tp_scores** (*ndarray*) –
- **tn_scores** (*ndarray*) –

- **attrs** (*dict*) – (default = None)

Returns (scores, labels, attrs)

Return type tuple

CommandLine: python -m vtool.score_normalization --test-partition_scores

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.score_normalization import * # NOQA
>>> X = np.array([5, 6, 6, 7, 1, 2, 2])
>>> attrs = {'qaid': np.array([21, 24, 25, 26, 11, 14, 15])}
>>> y = np.array([1, 1, 1, 1, 0, 0, 0], dtype=np.bool_)
>>> tup = partition_scores(X, y, attrs)
>>> resdict = ut.odict(zip(
>>>     ['tp_scores', 'tn_scores', 'part_attrs'], tup))
>>> result = ub.repr2(resdict, nobraces=True, with_dtype=False,
>>>                    explicit=1, nl=2)
>>> print(result)
tp_scores=np.array([5, 6, 6, 7]),
tn_scores=np.array([1, 2, 2]),
part_attrs=False: 'qaid': np.array([11, 14, 15]),
                  True: 'qaid': np.array([21, 24, 25, 26]),,
```

```
vtool.score_normalization.plot_postbayes_pdf(score_domain, p_tn_given_score,
                                             p_tp_given_score, score_thresh=None,
                                             prob_thresh=None, cfgstr="", fnum=None,
                                             pnum=(1, 1, 1))
```

```
vtool.score_normalization.plot_prebayes_pdf(score_domain, p_score_given_tn,
                                             p_score_given_tp, p_score, cfgstr="",
                                             fnum=None, pnum=(1, 1, 1), **kwargs)
```

```
vtool.score_normalization.test_score_normalization(tp_support, tn_support,
                                                    with_scores=True, ver-
                                                    bose=True, with_roc=True,
                                                    with_precision_recall=False,
                                                    figtitle=None,
                                                    normkw_varydict=None)
```

Gives an overview of how well threshold can be learned from raw scores.

DEPRICATE

CommandLine: python -m vtool.score_normalization --test-test_score_normalization --show

CommandLine: xdoctest -m ~/code/vtool/vtool/score_normalization.py test_score_normalization

Ignore:

```
>>> # GUI_DOCTEST
>>> # Shows how score normalization works with gaussian noise
>>> from vtool.score_normalization import * # NOQA
>>> verbose = True
>>> randstate = np.random.RandomState(seed=0)
>>> # Get a training sample
>>> tp_support = randstate.normal(loc=6.5, size=(256,))
>>> tn_support = randstate.normal(loc=3.5, size=(256,))
```

(continues on next page)

(continued from previous page)

```
>>> # xdoctest: +REQUIRES(module:plottool)
>>> test_score_normalization(tp_support, tn_support, verbose=verbose)
>>> ut.show_if_requested()
```

```
vtool.score_normalization.testdata_score_normalier(tp_bumps=[(6.5, 256)],
                                                    tn_bumps=[(3.5, 256)],
                                                    tp_scale=1.0, tn_scale=1.0,
                                                    min_clip=None, **kwargs)
```

1.35 vtool.segmentation module

```
vtool.segmentation.clean_mask(mask, num_dilate=3, num_erode=3, window_frac=0.025)
Clean the mask (num_erode, num_dilate) = (1, 1) (w, h) = (10, 10)
```

```
vtool.segmentation.demo_grabcut(bgr_img)
```

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

CommandLine: `python -m vtool.segmentation --test-demo_grabcut --show`

SeeAlso: `python -m wbia.algo.preproc.preproc_probchip --test-postprocess_dev`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.segmentation import * # NOQA
>>> # build test data
>>> import utool as ut
>>> import wbia.plottool as pt
>>> import vtool as vt
>>> img_fpath = ut.grab_test_imgpath('easy1.png')
>>> bgr_img = vt.imread(img_fpath)
>>> # execute function
>>> print(bgr_img.shape)
>>> result = demo_grabcut(bgr_img)
>>> # verify results
>>> print(result)
>>> ## xdoctest: +REQUIRES(--show)
>>> pt.show_if_requested()
```

```
vtool.segmentation.fill_holes(mask)
```

```
vtool.segmentation.grabcut(bgr_img, prior_mask, binary=True)
```

References: http://docs.opencv.org/trunk/doc/py_tutorials/py_imgproc/py_grabcut/py_grabcut.html

```
vtool.segmentation.grabcut2(rgb_chip)
```

```
vtool.segmentation.mask_colored_img(img_rgb, mask, encoding='bgr')
```

```
vtool.segmentation.printDBG(msg)
```

```
vtool.segmentation.resize_img_and_bbox(img_fpath, bbox_, new_size=None,
                                       sqrt_area=400.0)
```

```
vtool.segmentation.segment(img_fpath, bbox_, new_size=None)
```

1.36 vtool.spatial_verification module

Spatial verification of keypoint matches

Notation: 1_m = img1_matches; 2_m = img2_matches x and y are locations, invV is the elliptical shapes. fx are the original feature indexes (used for making sure 1 keypoint isn't assigned to 2)

Look Into: Standard `skimage.transform` <http://stackoverflow.com/questions/11462781/fast-2d-rigid-body-transformations-in-numpy-scipy> `skimage.transform.fast_homography(im, H)`

FIXME: is it scaled_thresh or scaled_thresh_sqrd

References

http://ags.cs.uni-kl.de/fileadmin/inf_ags/3dcv-ws11-12/3DCV_WS11-12_lec04.pdf http://www.imgfsr.com/CVPR2011/Tutorial6/RANSAC_CVPR2011.pdf http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf
Page 317

Notes

Invariants of affine transforms - parallel lines, ratios of parallel lengths, ratios of areas Invariants of homographies - cross-ratio of four points on a line (ratio of ratio)

`vtool.spatial_verification.build_affine_lstsqrs_Mx6(xy1_man, xy2_man)`
CURRENTLY NOT WORKING

CommandLine: `python -m vtool.spatial_verification --test-build_affine_lstsqrs_Mx6`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.demodata as demodata
>>> kpts1, kpts2 = demodata.get_dummy_kpts_pair()
>>> xy1_man = ktool.get_xys(kpts1).astype(np.float64)
>>> xy2_man = ktool.get_xys(kpts2).astype(np.float64)
>>> Mx6 = build_affine_lstsqrs_Mx6(xy1_man, xy2_man)
>>> import ubelt as ub
>>> print(ub.repr2(Mx6))
>>> result = ut.hashstr(Mx6)
>>> print(result)
```

Ignore:

```
>>> import sympy as sym
>>> x1, y1, x2, y2 = sym.symbols('x1, y1, x2, y2')
>>> A = sym.Matrix([
>>>     [x1, y1, 0, 0, 1, 0],
>>>     [0, 0, x1, y1, 0, 1],
>>> ])
>>> b = sym.Matrix([[x2], [y2]])
>>> x = (A.T.multiply(A)).inv().multiply(A.T.multiply(b))
>>> x = (A.T.multiply(A)).pinv().multiply(A.T.multiply(b))
```

References

<https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf> page 22

`vtool.spatial_verification.build_lstsqrs_Mx9(xy1_mn, xy2_mn)`

Builds the M x 9 least squares matrix

CommandLine: `python -m vtool.spatial_verification --test-build_lstsqrs_Mx9`

Example

```
>>> # DISABLE_DOCTEST
>>> # xdoctest: +SKIP
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.demodata as demodata
>>> kpts1, kpts2 = demodata.get_dummy_kpts_pair()
>>> xy1_mn = ktool.get_xys(kpts1).astype(np.float64)
>>> xy2_mn = ktool.get_xys(kpts2).astype(np.float64)
>>> Mx9 = build_lstsqrs_Mx9(xy1_mn, xy2_mn)
>>> import ubelt as ub
>>> result = (ub.repr2(Mx9[0:2], suppress_small=True, precision=2, with_
↳dtype=True))
>>> print(result)
np.array([[ 0.00e+00,  0.00e+00,  0.00e+00, -3.20e+01, -2.72e+01,
           -1.00e+00,  8.82e+02,  7.49e+02,  2.76e+01],
          [ 3.20e+01,  2.72e+01,  1.00e+00,  0.00e+00,  0.00e+00,
            0.00e+00, -1.09e+03, -9.28e+02, -3.42e+01]], dtype=np.float64)
```

References

http://dip.sun.ac.za/~stefan/TW793/attach/notes/homography_estimation.pdf http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf Page 317 <http://vision.ece.ucsb.edu/~zuliani/Research/RANSAC/docs/RANSAC4Dummies.pdf> page 53

`vtool.spatial_verification.compute_affine(xy1_man, xy2_man)`

Parameters

- **xy1_mn** (`ndarray[ndim=2]`) – xy points in image 1
- **xy2_mn** (`ndarray[ndim=2]`) – corresponding xy points in image 2

Returns A - affine matrix

Return type `ndarray[shape=(3,3)]`

CommandLine: `python -m vtool.spatial_verification --test-compute_affine:1 --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.demodata as demodata
>>> import vtool.keypoint as ktool
>>> kpts1, kpts2 = demodata.get_dummy_kpts_pair()
>>> xy1_mn = ktool.get_xys(kpts1)
```

(continues on next page)

(continued from previous page)

```
>>> xy2_mn = ktool.get_xys(kpts2)
>>> A = compute_affine(xy1_mn, xy1_mn)
>>> result =str(A)
>>> result = np.array_str(A, precision=2)
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.demodata as demodata
>>> import vtool.keypoint as ktool
>>> import wbia.plottool as pt
>>> xy1_man, xy2_man, rchip1, rchip2, T1, T2 = testdata_matching_affine_inliers_
↳normalized()
>>> A_prime = compute_affine(xy1_man, xy2_man)
>>> A = npl.solve(T2, A_prime).dot(T1)
>>> A /= A[2, 2]
>>> result = np.array_str(A, precision=2)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> rchip2_blendA = pt.draw_sv.get_blended_chip(rchip1, rchip2, A)
>>> pt.imshow(rchip2_blendA)
>>> ut.show_if_requested()
[[ 1.19e+00 -1.06e-02 -4.49e+01]
 [ -2.22e-01  1.12e+00 -2.78e+01]
 [ 0.00e+00  0.00e+00  1.00e+00]]
```

`vtool.spatial_verification.compute_homog(xy1_mn, xy2_mn)`

Generate 6 degrees of freedom homography transformation Computes homography from normalized (0 to 1) point correspondences from 2 → 1 (database→query)

Parameters

- **xy1_mn** (`ndarray[ndim=2]`) – xy points in image1
- **xy2_mn** (`ndarray[ndim=2]`) – corresponding xy points in image 2

Returns H - homography matrix

Return type `ndarray[shape=(3,3)]`

CommandLine: `python -m vtool.spatial_verification --test-compute_homog:1 --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.keypoint as ktool
>>> import vtool.demodata as demodata
>>> kpts1, kpts2 = demodata.get_dummy_kpts_pair()
>>> xy1_mn = ktool.get_xys(kpts1)
>>> xy2_mn = ktool.get_xys(kpts2)
>>> H = compute_homog(xy1_mn, xy2_mn)
>>> #result = ut.hashstr(H)
```

(continues on next page)

(continued from previous page)

```
>>> result = np.array_str(H, precision=2)
>>> print(result)
[[ 1.83e-03  2.85e-03 -7.11e-01]
 [ 2.82e-03  1.80e-03 -7.03e-01]
 [ 1.67e-05  1.68e-05 -5.53e-03]]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.keypoint as ktool
>>> import wbia.plottool as pt
>>> xy1_man, xy2_man, rchip1, rchip2, T1, T2 = testdata_matching_affine_inliers_
    ↪normalized()
>>> H_prime = compute_homog(xy1_man, xy2_man)
>>> H = npl.solve(T2, H_prime).dot(T1)
>>> H /= H[2, 2]
>>> result = np.array_str(H, precision=2)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> rchip2_blendH = pt.draw_sv.get_blended_chip(rchip1, rchip2, H)
>>> pt.imshow(rchip2_blendH)
>>> ut.show_if_requested()
[[ 9.22e-01 -2.50e-01  2.75e+01]
 [-2.04e-01  8.79e-01 -7.94e+00]
 [-1.82e-04 -5.99e-04  1.00e+00]]
```

`vtool.spatial_verification.estimate_refined_transform(kpts1, kpts2, fm, aff_inliers,
refine_method='homog')`
 estimates final transformation using normalized affine inliers

References

http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

`vtool.spatial_verification.get_affine_inliers(kpts1, kpts2, fm, fs, xy_thresh_sqrd,
scale_thresh_sqrd, ori_thresh)`

Estimates inliers deterministically using elliptical shapes

Compute all transforms from *kpts1* to *kpts2* (enumerate all hypothesis) We transform from chip1 -> chip2 The determinants are squared keypoint scales

Returns *aff_inliers_list*, *aff_errors_list*, *Aff_mats*

Return type tuple

Notes

FROM PERDOCH 2009: $H = \text{inv}(A_j) \cdot \text{dot}(R_j, T) \cdot \text{dot}(R_i) \cdot \text{dot}(A_i)$ $H = \text{inv}(A_j) \cdot \text{dot}(A_i)$ The input *invVs* = perdoch.*invA*'s

CommandLine: `python2 -m vtool.spatial_verification -test-get_affine_inliers` `python3 -m vtool.spatial_verification -test-get_affine_inliers`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.demodata as demodata
>>> import vtool.keypoint as ktool
>>> kpts1, kpts2 = demodata.get_dummy_kpts_pair((100, 100))
>>> fm = demodata.make_dummy_fm(len(kpts1)).astype(np.int32)
>>> fs = np.ones(len(fm), dtype=np.float64)
>>> xy_thresh_sqrd = ktool.KPTS_DTYPE(.009) ** 2
>>> scale_thresh_sqrd = ktool.KPTS_DTYPE(2)
>>> ori_thresh = ktool.KPTS_DTYPE(TAU / 4)
>>> output = get_affine_inliers(kpts1, kpts2, fm, fs, xy_thresh_sqrd,
>>>                             scale_thresh_sqrd, ori_thresh)
>>> output_str = ut.repr3(output, precision=2, suppress_small=True)
>>> print('output_str = %s' % (output_str,))
>>> aff_inliers_list, aff_errors_list, Aff_mats = output
>>> result = 'nInliers=%r hash=%s' % (len(aff_inliers_list), ut.hash_data(output_
↪str))
>>> print(result)
```

`vtool.spatial_verification.get_best_affine_inliers(kpts1, kpts2, fm, fs, xy_thresh_sqrd, scale_thresh, ori_thresh, forcepy=False)`

Tests each hypothesis and returns only the best transformation and inliers

`vtool.spatial_verification.get_best_affine_inliers(kpts1, kpts2, fm, fs, xy_thresh_sqrd, scale_thresh, ori_thresh)`

`vtool.spatial_verification.get_normalized_affine_inliers(kpts1, kpts2, fm, aff_inliers)`

returns xy-inliers that are normalized to have a mean of 0 and std of 1 as well as the transformations so the inverse can be taken

`vtool.spatial_verification.refine_inliers(kpts1, kpts2, fm, aff_inliers, xy_thresh_sqrd, scale_thresh=2.0, ori_thresh=1.57, full_homog_checks=True, refine_method='homog')`

Given a set of hypothesis inliers, computes a homography and refines inliers returned homography maps image1 space into image2 space

CommandLine: `python -m vtool.spatial_verification --test-refine_inliers python -m vtool.spatial_verification --test-refine_inliers:0 python -m vtool.spatial_verification --test-refine_inliers:1 --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.demodata as demodata
>>> import vtool.keypoint as ktool
>>> kpts1, kpts2 = demodata.get_dummy_kpts_pair((100, 100))
>>> fm = demodata.make_dummy_fm(len(kpts1)).astype(np.int32)
>>> aff_inliers = np.arange(len(fm))
>>> xy_thresh_sqrd = .01 * ktool.get_kpts_dlen_sqrd(kpts2)
>>> homogtup = refine_inliers(kpts1, kpts2, fm, aff_inliers, xy_thresh_sqrd)
>>> refined_inliers, refined_errors, H = homogtup
```

(continues on next page)

(continued from previous page)

```
>>> import ubelt as ub
>>> result = ub.repr2(homog_tup, precision=2, nl=True, suppress_small=True,
↳ nobr=True)
>>> print(result)
```

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.keypoint as ktool
>>> import wbia.plottool as pt
>>> kpts1, kpts2, fm, aff_inliers, rchip1, rchip2, xy_thresh_sqrd = testdata_
↳ matching_affine_inliers()
>>> homog_tup1 = refine_inliers(kpts1, kpts2, fm, aff_inliers, xy_thresh_sqrd)
>>> homog_tup = (homog_tup1[0], homog_tup1[2])
>>> # xdoctest: +REQUIRES(--show)
>>> pt.draw_sv.show_sv(rchip1, rchip2, kpts1, kpts2, fm, homog_tup=homog_tup)
>>> ut.show_if_requested()
```

```
vtool.spatial_verification.spatially_verify_kpts(kpts1, kpts2, fm,
xy_thresh=0.01, scale_thresh=2.0,
ori_thresh=1.5707963267948966,
dlen_sqrd2=None, min_nInliers=4,
match_weights=None,
returnAff=False,
full_homog_checks=True,
refine_method='homog',
max_nInliers=5000)
```

Driver function Spatially validates feature matches

FIXME: there is a non-determinism here

Returned homography maps image1 space into image2 space.

Parameters

- **kpts1** (*ndarray* [*ndim*=2]) – all keypoints in image 1
- **kpts2** (*ndarray* [*ndim*=2]) – all keypoints in image 2
- **fm** (*ndarray* [*ndim*=2]) – matching keypoint indexes [..., (kp1x, kp2x), ...]
- **xy_thresh** (*float*) – spatial distance threshold under affine transform to be considered a match
- **scale_thresh** (*float*) –
- **ori_thresh** (*float*) –
- **dlen_sqrd2** (*float*) – diagonal length squared of image/chip 2
- **min_nInliers** (*int*) – default=4
- **returnAff** (*bool*) – returns best affine hypothesis as well
- **max_nInliers** (*int*) – homog is not considered after this threshold

Returns (refined_inliers, refined_errors, H, aff_inliers, aff_errors, Aff) if success else None

Return type *tuple*

CommandLine: python -m xdoctest vtool.spatial_verification spatially_verify_kpts:0 -show python -m xdoctest vtool.spatial_verification spatially_verify_kpts:0 -show -refine-method='affine' python -m xdoctest vtool.spatial_verification spatially_verify_kpts:0 -dpath figures -show -save ~/latex/crall-candidacy-2015/figures/sver_kpts.jpg # NOQA python -m xdoctest vtool.spatial_verification spatially_verify_kpts:0

Example

```
>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.spatial_verification import *
>>> import vtool.demodata as demodata
>>> import vtool as vt
>>> fname1 = ut.get_argval('--fname1', type_=str, default='easy1.png')
>>> fname2 = ut.get_argval('--fname2', type_=str, default='easy2.png')
>>> default_dict = vt.get_extract_features_default_params()
>>> default_dict['ratio_thresh'] = .625
>>> kwargs = ut.parse_dict(default_dict)
>>> (kpts1, kpts2, fm, fs, rchip1, rchip2) = demodata.testdata_ratio_
↳ matches(fname1, fname2, **kwargs)
>>> xy_thresh = .01
>>> dlen_sqrd2 = 447271.015
>>> ori_thresh = 1.57
>>> min_nInliers = 4
>>> returnAff = True
>>> scale_thresh = 2.0
>>> match_weights = np.ones(len(fm), dtype=np.float64)
>>> refine_method = ut.get_argval('--refine-method', default='homog')
>>> svtup = spatially_verify_kpts(kpts1, kpts2, fm, xy_thresh,
>>>                               scale_thresh, ori_thresh, dlen_sqrd2,
>>>                               min_nInliers, match_weights, returnAff,
>>>                               refine_method=refine_method)
>>> assert svtup is not None and len(svtup) == 6, 'sver failed'
>>> refined_inliers, refined_errors, H = svtup[0:3]
>>> aff_inliers, aff_errors, Aff = svtup[3:6]
>>> #print('aff_errors = %r' % (aff_errors,))
>>> print('aff_inliers = %r' % (aff_inliers,))
>>> print('refined_inliers = %r' % (refined_inliers,))
>>> #print('refined_errors = %r' % (refined_errors,))
>>> import ubelt as ub
>>> result = ut.list_type_profile(svtup, with_dtype=False)
>>> #result = ub.repr2(svtup, precision=3)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> homog_tup = (refined_inliers, H)
>>> aff_tup = (aff_inliers, Aff)
>>> pt.draw_sv.show_sv(rchip1, rchip2, kpts1, kpts2, fm, aff_tup=aff_tup, homog_
↳ tup=homog_tup, refine_method=refine_method)
>>> pt.show_if_requested()
tuple(numpy.ndarray, tuple(numpy.ndarray*3), numpy.ndarray, numpy.ndarray,
↳ tuple(numpy.ndarray*3), numpy.ndarray)
```

vtool.spatial_verification.**test_affine_errors**(H, kpts1, kpts2, fm, xy_thresh_sqrd,
scale_thresh_sqrd, ori_thresh)

used for refinement as opposed to initial estimation

```
vtool.spatial_verification.test_homog_errors(H, kpts1, kpts2, fm, xy_thresh_sqrd,
                                             scale_thresh, ori_thresh,
                                             full_homog_checks=True)
```

Test to see which keypoints the homography correctly maps

Parameters

- **H** (`ndarray[float64_t, ndim=2]`) – homography/perspective matrix
- **kpts1** (`ndarray[float32_t, ndim=2]`) – keypoints
- **kpts2** (`ndarray[float32_t, ndim=2]`) – keypoints
- **fm** (`list`) – list of feature matches as tuples (qfx, dfx)
- **xy_thresh_sqrd** (`float`) –
- **scale_thresh** (`float`) –
- **ori_thresh** (`float`) – angle in radians
- **full_homog_checks** (`bool`) –

Returns `homog_tup1`

Return type `tuple`

CommandLine: `python -m vtool.spatial_verification -test-test_homog_errors:0 -show python -m vtool.spatial_verification -test-test_homog_errors:0 -show -rotation_invariance python -m vtool.spatial_verification -test-test_homog_errors:0 -show -rotation_invariance -no-affine-invariance -xy-thresh=.001 python -m vtool.spatial_verification -test-test_homog_errors:0 -show -rotation_invariance -no-affine-invariance -xy-thresh=.001 -no-full-homog-checks python -m vtool.spatial_verification -test-test_homog_errors:0 -show -no-full-homog-checks # _____ # Shows (sorta) how inliers are computed python -m vtool.spatial_verification -test-test_homog_errors:1 -show python -m vtool.spatial_verification -test-test_homog_errors:1 -show -rotation_invariance python -m vtool.spatial_verification -test-test_homog_errors:1 -show -rotation_invariance -no-affine-invariance -xy-thresh=.001 python -m vtool.spatial_verification -test-test_homog_errors:1 -show -rotation_invariance -xy-thresh=.001 python -m vtool.spatial_verification -test-test_homog_errors:0 -show -rotation_invariance -xy-thresh=.001`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.spatial_verification import * # NOQA
>>> import wbia.plottool as pt
>>> kpts1, kpts2, fm, aff_inliers, rchip1, rchip2, xy_thresh_sqrd = testdata_
↳matching_affine_inliers()
>>> H = estimate_refined_transform(kpts1, kpts2, fm, aff_inliers)
>>> scale_thresh, ori_thresh = 2.0, 1.57
>>> full_homog_checks = not ut.get_argflag('--no-full-homog-checks')
>>> homog_tup1 = test_homog_errors(H, kpts1, kpts2, fm, xy_thresh_sqrd, scale_
↳thresh, ori_thresh, full_homog_checks)
>>> homog_tup = (homog_tup1[0], homog_tup1[2])
>>> # xdoctest: +REQUIRES(--show)
>>> pt.draw_sv.show_sv(rchip1, rchip2, kpts1, kpts2, fm, homog_tup=homog_tup)
>>> ut.show_if_requested()
```

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.spatial_verification import * # NOQA
>>> import wbia.plottool as pt
>>> kpts1, kpts2, fm_, aff_inliers, rchip1, rchip2, xy_thresh_sqrd = testdata_
↳matching_affine_inliers()
>>> H = estimate_refined_transform(kpts1, kpts2, fm_, aff_inliers)
>>> scale_thresh, ori_thresh = 2.0, 1.57
>>> full_homog_checks = not ut.get_argflag('--no-full-homog-checks')
>>> # -----
>>> # Take subset of feature matches
>>> fm = fm_
>>> scale_err, xy_err, ori_err = \
...     ut.exec_func_src(test_homog_errors, globals(), locals(),
...     'scale_err, xy_err, ori_err'.split(', '))
>>> # we only care about checking out scale and orientation here. ignore bad xy_
↳points
>>> xy_inliers_flag = np.less(xy_err, xy_thresh_sqrd)
>>> scale_err[~xy_inliers_flag] = 0
>>> # filter
>>> fm = fm_[np.array(scale_err).argsort()[::-1][:10]]
>>> fm = fm_[np.array(scale_err).argsort()[::-1][:10]]
>>> # Exec sourcecode
>>> kpts1_m, kpts2_m, off_xy1_m, off_xy1_mt, dxy1_m, dxy1_mt, xy2_m, xy1_m, xy1_
↳mt, scale_err, xy_err, ori_err = \
...     ut.exec_func_src(test_homog_errors, globals(), locals(),
...     'kpts1_m, kpts2_m, off_xy1_m, off_xy1_mt, dxy1_m, dxy1_mt, xy2_m, xy1_m,
↳xy1_mt, scale_err, xy_err, ori_err'.split(', '))
>>> #-----
>>> # xdoctest: +REQUIRES(--show)
>>> pt.figure(fnum=1, pnum=(1, 2, 1), title='orig points and offset point')
>>> segments_list1 = np.array(list(zip(xy1_m.T.tolist(), off_xy1_m.T.tolist())))
>>> pt.draw_line_segments(segments_list1, color=pt.LIGHT_BLUE)
>>> pt.dark_background()
>>> #-----
>>> pt.figure(fnum=1, pnum=(1, 2, 2), title='transformed points and matching_
↳points')
>>> #-----
>>> # first have to make corresponding offset points
>>> # Use reference point for scale and orientation tests
>>> oris2_m = ktool.get_oris(kpts2_m)
>>> scales2_m = ktool.get_scales(kpts2_m)
>>> dxy2_m = np.vstack((np.sin(oris2_m), -np.cos(oris2_m)))
>>> scaled_dxy2_m = dxy2_m * scales2_m[None, :]
>>> off_xy2_m = xy2_m + scaled_dxy2_m
>>> # Draw transformed segments
>>> segments_list2 = np.array(list(zip(xy2_m.T.tolist(), off_xy2_m.T.tolist())))
>>> pt.draw_line_segments(segments_list2, color=pt.GREEN)
>>> # Draw corresponding matches segments
>>> segments_list3 = np.array(list(zip(xy1_mt.T.tolist(), off_xy1_mt.T.tolist())))
>>> pt.draw_line_segments(segments_list3, color=pt.RED)
>>> # Draw matches between correspondences
>>> segments_list4 = np.array(list(zip(xy1_mt.T.tolist(), xy2_m.T.tolist())))
>>> pt.draw_line_segments(segments_list4, color=pt.ORANGE)
>>> pt.dark_background()
>>> #-----

```

(continues on next page)

(continued from previous page)

```
>>> #vt.get_xy_axis_extents(kpts1_m)
>>> #pt.draw_sv.show_sv(rchip1, rchip2, kpts1, kpts2, fm, homog_tup=homog_tup)
>>> ut.show_if_requested()
```

```
vtool.spatial_verification.testdata_matching_affine_inliers()
```

```
vtool.spatial_verification.testdata_matching_affine_inliers_normalized()
```

```
vtool.spatial_verification.try_svd(M)
```

CommandLine: python -m vtool.spatial_verification try_svd

Example

```
>>> # SLOW_DOCTEST
>>> # xdoctest: +SKIP
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.demodata as demodata
>>> rng = np.random.RandomState(42)
>>> num = 1000
>>> xy1_mn = rng.randn(2, num)
>>> xy2_mn = rng.randn(2, num)
>>> M = build_lstsqrs_Mx9(xy1_mn, xy2_mn)
>>> print('M.shape = %r' % (M.shape,))
>>> USV = npl.svd(M, full_matrices=True, compute_uv=True)
>>> USV = try_svd(M)
```

Example

```
>>> # SLOW_DOCTEST
>>> # xdoctest: +SKIP
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.demodata as demodata
>>> num = np.ceil(np.sqrt(2000))
>>> kpts1, kpts2 = demodata.get_dummy_kpts_pair(wh_num=(num, num))
>>> xy1_mn = ktool.get_xys(kpts1).astype(np.float64)
>>> xy2_mn = ktool.get_xys(kpts2).astype(np.float64)
>>> M = build_lstsqrs_Mx9(xy1_mn, xy2_mn)
>>> print('M.shape = %r' % (M.shape,))
>>> USV = npl.svd(M, full_matrices=True, compute_uv=True)
>>> USV = try_svd(M)
```

```
vtool.spatial_verification.unnormalize_transform(M_prime, T1, T2)
```

1.37 vtool.sver_c_wrapper module

1.38 vtool.symbolic module

Sympy helpers

```
vtool.symbolic.check_expr_eq(expr1, expr2, verbose=True)
```

Does not work in general. Problem is not decidable. Thanks Richard.

Parameters

- **expr1** –
- **expr2** –

CommandLine: `python -m vtool.symbolic --test-check_expr_eq`

SeeAlso: `vt.symbolic.randcheck`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.symbolic import * # NOQA
>>> expr1 = sympy.Matrix([ [sx*x + 1.0*tx + w1*y], [sy*y + 1.0*ty + w2*x], [1.0]])
>>> expr2 = sympy.Matrix([ [sx*x + tx + w1*y], [sy*y + ty + w2*x], [1]])
>>> result = check_expr_eq(expr1, expr2)
>>> print(result)
```

`vtool.symbolic.custom_sympy_attrs(mat)`

`vtool.symbolic.evalprint(str_, globals_=None, locals_=None, simplify=False)`

`vtool.symbolic.symbolic_randcheck(expr1, expr2, domain={}, n=10)`

`vtool.symbolic.sympy_latex_repr(expr1)`

`vtool.symbolic.sympy_mat(arr)`

`vtool.symbolic.sympy_numpy_repr(expr1)`

1.39 vtool.trig module

`vtool.trig.atan2(y, x)`

does atan2 but returns from 0 to TAU

Example

```
>>> from vtool.trig import * # NOQA
>>> import utool
>>> rng = np.random.RandomState(0)
>>> y = rng.rand(1000).astype(np.float64)
>>> x = rng.rand(1000).astype(np.float64)
>>> theta = atan2(y, x)
>>> assert np.all(theta >= 0)
>>> assert np.all(theta < 2 * np.pi)
>>> import ubelt as ub
>>> assert ub.hash_data(theta) in [
>>> ↪ '6bfc86a2e94dd2dafbf501035719a7873d57f5f8e9cde88c4ccc35e98bb9e7b82abf6230803a923be7060866d66b8
>>> ↪ ',
>>> ↪ '90fe55311562f1c3ae451d5c4f27573259fed96752a5bd03f0f1216b46cf5b4b48024dcc744bfc6df7e6f8d6eb2a2
>>> ↪ ',
>>> ]
```

1.40 vtool.util_math module

LICENCE Apache 2 or whatever

FIXME: monotization functions need more hueristics

`vtool.util_math.beaton_tukey_loss (u, a=1)`

CommandLine: `python -m wbia.plottool.draw_func2 -exec-plot_func -show -range=-8,8 -func=vt.beaton_tukey_weight,vt.beaton_tukey_loss`

References

Steward_Robust%20parameter%20estimation%20in%20computer%20vision.pdf

`vtool.util_math.beaton_tukey_weight (u, a=1)`

CommandLine: `python -m wbia.plottool.draw_func2 -exec-plot_func -show -range=-8,8 -func=vt.beaton_tukey_weight`

References

Steward_Robust%20parameter%20estimation%20in%20computer%20vision.pdf

`vtool.util_math.breakup_equal_streak (arr_in, left_endpoint=None, right_endpoint=None)`

Breaks up streaks of equal values by interpolating between the next lowest and next highest value

Parameters

- **arr_in** -
- **left_endpoint** (*None*) - (default = None)
- **right_endpoint** (*None*) - (default = None)

Returns arr -

Return type ndarray

CommandLine: `python -m vtool.util_math -exec-breakup_equal_streak python -m vtool.util_math -test-ensure_monotone_strictly_increasing -show -offset=0`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> arr_in = np.array([0, 0, 1, 1, 2, 2], dtype=np.float32)
>>> arr_in = np.array([ 1.20488135,  1.2529297 ,  1.27306686,  1.29859663,
>>>    1.31769871,  1.37102388,  1.38114004,  1.45732054,  1.48119571,  1.48119571,
>>>    1.5381895 ,  1.54162741,  1.57492901,  1.61129523,  1.61129523,
>>>    1.61270343,  1.63377551,  1.7423034 ,  1.76364247,  1.79908459,
>>>    1.83564709,  1.83819742,  1.83819742,  1.86786967,  1.86786967,
>>>    1.90720142,  1.90720142,  1.92293973,  1.92293973, ]) / 2
>>> left_endpoint = 0
>>> right_endpoint = 1.0
>>> arr = breakup_equal_streak(arr_in, left_endpoint, right_endpoint)
>>> assert strictly_increasing(arr)
```

(continues on next page)

(continued from previous page)

```
>>> result = ('arr = %s' % (str(arr),))
>>> print(result)
```

`vtool.util_math.ensure_monotone_decreasing(arr_, fromleft=True, fromright=True)`

Parameters `arr` (ndarray) –

Returns `arr`

Return type ndarray

CommandLine: `python -m vtool.util_math --test-ensure_monotone_decreasing --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> rng = np.random.RandomState(0)
>>> size_ = 100
>>> domain = np.arange(size_)
>>> arr_ = np.sin(np.pi * (domain / 100) ) + (rng.rand(len(domain)) - .5) * .1
>>> arr = ensure_monotone_decreasing(arr_, fromright=True, fromleft=True)
>>> result = str(arr)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.plot2(domain, arr_, 'r-', fnum=1, pnum=(2, 1, 1), title='before', equal_
↳ aspect=False)
>>> pt.plot2(domain, arr, 'r-', fnum=1, pnum=(2, 1, 2), title='after_
↳ monotoneization (decreasing)', equal_aspect=False)
>>> ut.show_if_requested()
```

`vtool.util_math.ensure_monotone_increasing(arr_, fromright=True, fromleft=True, new-mode=True)`

Parameters `arr` (ndarray) –

Returns `arr`

Return type ndarray

CommandLine: `python -m vtool.util_math --test-ensure_monotone_increasing --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> rng = np.random.RandomState(0)
>>> size_ = 100
>>> domain = np.arange(size_)
>>> offset = float(ub.argval('--offset', default=2.3))
>>> arr_ = np.sin(np.pi * (domain / 100) - offset) + (rng.rand(len(domain)) - .5)
↳ * .1
>>> arr = ensure_monotone_increasing(arr_, fromleft=False, fromright=True)
>>> result = str(arr)
>>> print(result)
```

(continues on next page)

(continued from previous page)

```

>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.plot2(domain, arr_, 'r-', fnum=1, pnum=(2, 1, 1), title='before', equal_
↳ aspect=False)
>>> pt.plot2(domain, arr, 'r-', fnum=1, pnum=(2, 1, 2), title='after_
↳ monotonization (increasing)', equal_aspect=False)
>>> ut.show_if_requested()

```

```

vtool.util_math.ensure_monotone_strictly_decreasing(arr_, left_endpoint=None,
right_endpoint=None)

```

Parameters

- **arr** (ndarray) –
- **left_endpoint** (None) –
- **right_endpoint** (None) –

Returns arr**Return type** ndarray**CommandLine:** python -m vtool.util_math --test-ensure_monotone_strictly_decreasing --show**Example**

```

>>> # DISABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> import vtool as vt
>>> domain = np.arange(100)
>>> rng = np.random.RandomState(0)
>>> arr_ = np.sin(np.pi * (domain / 75) + 1.3) + (rng.rand(len(domain)) - .5) * .
↳ 05 + 1.0
>>> #arr_ = vt.demodata.testdata_nonmonotonic()
>>> #domain = np.arange(len(arr_))
>>> left_endpoint = 2.5
>>> right_endpoint = 0.25
>>> arr = ensure_monotone_strictly_decreasing(arr_, left_endpoint, right_endpoint)
>>> result = str(arr)
>>> print(result)
>>> assert strictly_decreasing(arr), 'ensure strict monotonic failed'
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.plot2(domain, arr_, 'r-', fnum=1, pnum=(3, 1, 1), title='before', equal_
↳ aspect=False)
>>> arr2 = ensure_monotone_decreasing(arr_)
>>> pt.plot2(domain, arr, 'b-', fnum=1, pnum=(3, 1, 2), equal_aspect=False)
>>> pt.plot2(domain, arr2, 'r-', fnum=1, pnum=(3, 1, 2), title='after_
↳ monotonization (decreasing)', equal_aspect=False)
>>> pt.plot2(domain, arr, 'r-', fnum=1, pnum=(3, 1, 3), title='after_
↳ monotonization (strictly decreasing)', equal_aspect=False)
>>> ut.show_if_requested()

```

```

vtool.util_math.ensure_monotone_strictly_increasing(arr_, left_endpoint=None,
right_endpoint=None, zero_hack=False, onehack=False,
newmode=True)

```


Parameters

- **arr** (*ndarray*) – sequence to monotone
- **zerohack** (*bool*) – default False, if True sets the first element to be zero and linearly interpolates to the first nonzero item
- **onehack** (*bool*) – default False, if True one will not be in the resulting array (replaced with number very close to one)

References

<http://mathoverflow.net/questions/17464/making-a-non-monotone-function-monotone> <http://stackoverflow.com/questions/28563711/make-a-numpy-array-monotonic-without-a-python-loop> https://en.wikipedia.org/wiki/Isotonic_regression http://scikit-learn.org/stable/auto_examples/plot_isotonic_regression.html

CommandLine: `python -m vtool.util_math -test-ensure_monotone_strictly_increasing -show python -m vtool.util_math -test-ensure_monotone_strictly_increasing -show -offset=0`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> import numpy as np
>>> arr_ = np.array([0.4, 0.4, 0.4, 0.5, 0.6, 0.6, 0.6, 0.7, 0.9, 0.9, 0.91, 0.92,
↪ 1.0, 1.0])
>>> arr = ensure_monotone_strictly_increasing(arr_)
>>> assert strictly_increasing(arr), 'ensure strict monotonic failed1'
```

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> import vtool as vt
>>> left_endpoint = None
>>> rng = np.random.RandomState(0)
>>> right_endpoint = None
>>> domain = np.arange(100)
>>> offset = vt.get_argval('--offset', type_=float, default=2.3)
>>> arr_ = np.sin(np.pi * (domain / 100) - offset) + (rng.rand(len(domain)) - .5)
↪ * .1 + 1.2
>>> #arr_ = vt.demodata.testdata_nonmonotonic()
>>> #domain = np.arange(len(arr_))
>>> arr = ensure_monotone_strictly_increasing(arr_, left_endpoint, right_endpoint)
>>> result = str(arr)
>>> print(result)
>>> print('arr = %r' % (arr,))
>>> print('arr = %r' % (np.diff(arr),))
>>> assert non_decreasing(arr), 'ensure nondecreasing failed2'
>>> assert strictly_increasing(arr), 'ensure strict monotonic failed2'
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.plot2(domain, arr_, 'r-', fnum=1, pnum=(3, 1, 1), title='before', equal_
↪ aspect=False)
>>> arr2 = ensure_monotone_increasing(arr_)
```

(continues on next page)

(continued from previous page)

```

>>> pt.plot2(domain, arr, 'b-', fnum=1, pnum=(3, 1, 2), equal_aspect=False)
>>> pt.plot2(domain, arr2, 'r-', fnum=1, pnum=(3, 1, 2), title='after_
↳monotonization (decreasing)', equal_aspect=False)
>>> pt.plot2(domain, arr, 'r-', fnum=1, pnum=(3, 1, 3), title='after_
↳monotonization (strictly decreasing)', equal_aspect=False)
>>> ut.show_if_requested()

```

`vtool.util_math.gauss_func1d(x, mu=0.0, sigma=1.0)`

Parameters

- **x** –
- **mu** (*float*) –
- **sigma** (*float*) –

CommandLine: `python -m vtool.util_math --test-gauss_func1d`

CommandLine: `python -m vtool.util_math --test-gauss_func1d --show`

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> # build test data
>>> x = np.array([-2, -1, -.5, 0, .5, 1, 2])
>>> mu = 0.0
>>> sigma = 1.0
>>> # execute function
>>> gaussval = gauss_func1d(x, mu, sigma)
>>> # verify results
>>> result = np.array_repr(gaussval, precision=2)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.plot(x, gaussval)
>>> ut.show_if_requested()
array([ 0.05,  0.24,  0.35,  0.4 ,  0.35,  0.24,  0.05])

```

`vtool.util_math.gauss_func1d_unnormalized(x, sigma=1.0)`

faster version of `gauss_func1d` with no normalization. So the maximum point will have a value of 1.0

CommandLine: `python -m vtool.util_math --test-gauss_func1d_unnormalized --show`

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> # build test data
>>> x = np.array([-2, -1, -.5, 0, .5, 1, 2])
>>> sigma = 1.0
>>> # execute function
>>> gaussval = gauss_func1d_unnormalized(x, sigma)
>>> # verify results

```

(continues on next page)

(continued from previous page)

```

>>> result = np.array_repr(gaussval, precision=2)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.plot(x, gaussval)
>>> ut.show_if_requested()
array([ 0.05,  0.24,  0.35,  0.4 ,  0.35,  0.24,  0.05])

```

`vtool.util_math.gauss_parzen_est` (*dist*, *L=1*, *sigma=0.38*)

python -m wbia.plottool.draw_func2 --exec-plot_func --show --range=-.2,.2 --func=vt.gauss_parzen_est python
 -m wbia.plottool.draw_func2 --exec-plot_func --show --range=0,1 --func=vt.gauss_parzen_est

`vtool.util_math.group_consecutive` (*arr*)

Returns lists of consecutive values

References

<http://stackoverflow.com/questions/7352684/how-to-find-the-groups-of-consecutive-elements-from-an-array-in-numpy>

Parameters *arr* (*ndarray*) – must be integral and unique

Returns *arr* -

Return type *ndarray*

CommandLine: python -m vtool.util_math --exec-group_consecutive

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> arr = np.array([1, 2, 3, 5, 6, 7, 8, 9, 10, 15, 99, 100, 101])
>>> groups = group_consecutive(arr)
>>> result = ('groups = %s' % (str(groups),))
>>> print(result)
groups = [array([1, 2, 3]), array([ 5,  6,  7,  8,  9, 10]), array([15]), array([
↪ 99, 100, 101])]

```

`vtool.util_math.iceil` (*num*, *dtype=<class 'numpy.int32'>*)

Integer ceiling. (because numpy doesn't seem to have it!)

Parameters *num* (*ndarray* or *scalar*) -

Returns

Return type *ndarray* or *scalar*

CommandLine: python -m vtool.util_math --test-iceil

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> num = 1.5

```

(continues on next page)

(continued from previous page)

```
>>> result = repr(iceil(num))
>>> print(result)
2
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> import ubelt as ub
>>> num = [1.5, 2.9]
>>> result = ub.repr2(iceil(num), with_dtype=True)
>>> print(result)
np.array([2, 3], dtype=np.int32)
```

`vtool.util_math.interpolate_nans(arr)`
replaces nans with interpolated values or 0

Parameters `arr` (`ndarray`) –

Returns `new_arr`

Return type `ndarray`

CommandLine: `python -m vtool.util_math --exec-interpolate_nans`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> arr = np.array([np.nan, np.nan, np.nan, np.nan])
>>> new_arr = interpolate_nans(arr)
>>> result = ('new_arr = %s' % (str(new_arr),))
>>> print(result)
new_arr = [ 0.  0.  0.  0.]
```

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> arr = np.array([np.nan, 1, np.nan, np.nan, np.nan, np.nan, 10, np.nan, 5])
>>> new_arr = interpolate_nans(arr)
>>> result = ('new_arr = %s' % (str(new_arr),))
>>> print(result)
new_arr = [ 1.    1.    2.8   4.6   6.4   8.2  10.    7.5   5. ]
```

`vtool.util_math.iound(num, dtype=<class 'numpy.int32'>)`
Integer round. (because numpy doesn't seem to have it!)

`vtool.util_math.logistic_01(x)`

Parameters `x` –

CommandLine: `python -m vtool.util_math --exec-logistic_01 --show`

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> x = np.linspace(0, 1)
>>> y = logistic_01(x)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.plot(x, y)
>>> ut.show_if_requested()

```

`vtool.util_math.logit(x)`

`vtool.util_math.non_decreasing(L)`

References

<http://stackoverflow.com/questions/4983258/python-how-to-check-list-monotonicity>

`vtool.util_math.non_increasing(L)`

References

<http://stackoverflow.com/questions/4983258/python-how-to-check-list-monotonicity>

`vtool.util_math.strictly_decreasing(L)`

References

<http://stackoverflow.com/questions/4983258/python-how-to-check-list-monotonicity>

`vtool.util_math.strictly_increasing(L)`

References

<http://stackoverflow.com/questions/4983258/python-how-to-check-list-monotonicity>

`vtool.util_math.test_language_modulus()`

References

http://en.wikipedia.org/wiki/Modulo_operation

1.41 Module contents

VTool - Computer vision tools

Autogenerate Command: `mkinit vtool -i`

class `vtool.AnnotPairFeatInfo` (*columns=None, importances=None*)

Bases: `object`

Information class about feature dimensions of PairwiseMatch.

Notes

Can be used to compute marginal importances over groups of features used in the pairwise one-vs-one scoring algorithm

Can be used to construct an appropriate cfgdict for a new PairwiseMatch.

CommandLine: python -m vtool.matching AnnotPairFeatInfo

Example

```
>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.matching import * # NOQA
>>> import vtool as vt
>>> match = demodata_match({})
>>> match.add_global_measures(['time', 'gps'])
>>> index = pd.MultiIndex.from_tuples([(1, 2)], names=('aid1', 'aid2'))
>>> # Feat info without bins
>>> feat = match.make_feature_vector()
>>> X = pd.DataFrame(feat, index=index)
>>> print(X.keys())
>>> featinfo = AnnotPairFeatInfo(X)
>>> pairfeat_cfg, global_keys = featinfo.make_pairfeat_cfg()
>>> print('pairfeat_cfg = %r' % (pairfeat_cfg,))
>>> print('global_keys = %r' % (global_keys,))
>>> assert 'delta' not in global_keys
>>> assert 'max' not in global_keys
>>> ut.cprint(featinfo.get_infostr(), 'blue')
>>> # Feat info with bins
>>> feat = match.make_feature_vector(indices=0, bins=[.7, .8], bin_key='ratio')
>>> X = pd.DataFrame(feat, index=index)
>>> print(X.keys())
>>> featinfo = AnnotPairFeatInfo(X)
>>> pairfeat_cfg, global_keys = featinfo.make_pairfeat_cfg()
>>> print('pairfeat_cfg = %s' % (ut.repr4(pairfeat_cfg),))
>>> print('global_keys = %r' % (global_keys,))
>>> ut.cprint(featinfo.get_infostr(), 'blue')
```

binsum_fmt = '{op}({measure}[{bin_key}<{binval}])'

dimkey_grammar()

CommandLine: python -m vtool.matching AnnotPairFeatInfo.dimkey_grammar

Example

```
>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.matching import * # NOQA
>>> import vtool as vt
>>> match = demodata_match({})
>>> match.add_global_measures(['view', 'qual', 'gps', 'time'])
>>> index = pd.MultiIndex.from_tuples([(1, 2)], names=('aid1', 'aid2'))
>>> # Feat info without bins
>>> feat = match.make_feature_vector()
>>> X = pd.DataFrame(feat, index=index)
>>> featinfo = AnnotPairFeatInfo(X)
```

(continues on next page)

(continued from previous page)

```

>>> feat_grammar = featinfo.dimkey_grammar()
>>> for key in X.keys():
>>>     print(key)
>>>     print(feat_grammar.parseString(key))

```

feature (*key*)

find (*group_id*, *op*, *value*, *hack=False*)

groupid options: *measure_type* *global_measure*, *local_sorter*, *local_rank*, *local_measure*, *summary_measure*, *summary_op*, *summary_bin*, *summary_binval*, *summary_binkey*,

Ignore: *group_id* = 'summary_op' *op* = '==' *value* = 'len'

get_infostr ()

Summarizes the types (global, local, summary) of features in X based on standardized dimension names.

global_measure (*key*)

group_counts (*item*)

group_importance (*item*)

loc_fmt = 'loc[{**sorter**},{**rank**]}({**measure**})'

local_measure (*key*)

local_rank (*key*)

local_sorter (*key*)

make_pairfeat_cfg ()

measure (*key*)

measure_type (*key*)

print_counts (*group_id*)

print_margins (*group_id*, *ignore_trivial=True*)

rrr (*verbose=True*, *reload_module=True*)

special class reloading function This function is often injected as rrr of classes

select_columns (*criteria*, *op='and'*)

Parameters **criteria** (*list*) – list of tokens denoting selection constraints can be one of:
measure_type *global_measure*, *local_sorter*, *local_rank*, *local_measure*, *summary_measure*,
summary_op, *summary_bin*, *summary_binval*, *summary_binkey*,

Ignore:

```

>>> featinfo.select_columns([
>>>     ('measure_type', '==', 'local'),
>>>     ('local_sorter', 'in', ['weighted_ratio_score', 'lnbnn_norm_dist
↪']),
>>> ], op='and')

```

sum_fmt = '{**op**}({**measure**})'

summary_binkey (*key*)

summary_binval (*key*)

```
summary_measure (key)
summary_op (key)
class vtool.AnnoyWrapper
    Bases: object
    flann-like interface to annnoy
    build_annoy (centroids, trees=3)
    nn (data_vecs, query_vecs, num, trees=3, checks=-1)
    query_annoy (query_vecs, num, checks=-1)
class vtool.AnnoyWrapper
    Bases: object
    Wrapper for annoy to use the FLANN api
    build_index (dvecs, **kwargs)
    nn_index (qvecs, num_neighbs, checks=None)
class vtool.AssignTup (fm, match_dist, norm_fx1, norm_dist)
    Bases: tuple
    fm
        Alias for field number 0
    match_dist
        Alias for field number 1
    norm_dist
        Alias for field number 3
    norm_fx1
        Alias for field number 2
class vtool.ConfusionMetrics
    Bases: ubelt.util_mixins.NiceRepr
    Can compute average percision using the PASCAL definition
```

References

http://www.flinders.edu.au/science_engineering/fms/School-CSEM/publications/tech_reps-research_artfcts/TRRA_2007.pdf http://www.alta.asn.au/events/altss_w2003_proc/altss/courses/powers/Bookmaker-all/200302-ICCS-Bookmaker.pdfcs <http://www.cs.bris.ac.uk/Publications/Papers/1000704.pdf> http://en.wikipedia.org/wiki/Information_retrieval http://en.wikipedia.org/wiki/Precision_and_recall https://en.wikipedia.org/wiki/Confusion_matrix http://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve

SeeAlso: `sklearn.metrics.ranking._binary_clf_curve`

Notes

From oxford: Precision is defined as the ratio of retrieved positive images to the total number retrieved. Recall is defined as the ratio of the number of retrieved positive images to the total number of positive images in the corpus.

Ignore: varname_list = 'tp, fp, fn, tn, fpr, tpr, tpa'.split(', ') lines = ['self.{varname} = {varname}'.format(varname=varname) for varname in varname_list] print(ut.indent('n'.join(lines)))

CommandLine: python -m vtool.confusion ConfusionMetrics --show

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.confusion import * # NOQA
>>> scores, labels = testdata_scores_labels()
>>> c = self = confusions = ConfusionMetrics().fit(scores, labels)
>>> assert np.all(c.n_pos == c.n_tp + c.n_fn)
>>> assert np.all(c.n_neg == c.n_tn + c.n_fp)
>>> assert np.all(np.isclose(c.rp + c.rn, 1.0))
>>> assert np.all(np.isclose(c.pp + c.pn, 1.0))
>>> assert np.all(np.isclose(c.fpr, 1 - c.tnr))
>>> assert np.all(np.isclose(c.fnr, 1 - c.tpr))
>>> assert np.all(np.isclose(c.tpr, c.tp / c.rp))
>>> assert np.all(np.isclose(c.tpa, c.tp / c.pp))
>>> assert np.all(np.isclose(c.jacc, c.tp / (c.tp + c.fn + c.fp)))
>>> assert np.all(np.isclose(c.mcc, np.sqrt(c.mk * c.bm)))
>>> assert np.all(np.isclose(
>>>     c.acc, (c.tpr + c.c * (1 - c.fpr)) / (1 + c.c)))
>>> assert np.all(np.isclose(c.ppv, c.recall * c.prev / c.bias))
>>> assert np.all(np.isclose(
>>>     c.wracc, 4 * c.c * (c.tpr - c.fpr) / (1 + c.c) ** 2))
>>> # xdoctest: +REQUIRES(--show)
>>> confusions.draw_roc_curve()
>>> ut.show_if_requested()
```

acc

accuracy

aliases = {'acc': {'accuracy', 'rand_accuracy', 'tea', 'ter'}, 'bm': {'bookmaker_informedness'}}

auc

The AUC is a standard measure used to evaluate a binary classifier and represents the probability that a random correct case will receive a higher score than a random incorrect case.

References

https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve

auc_trap

bm

bookmaker informedness

c

cs

class ratio

cv

ratio of cost of making a mistake

draw_precision_recall_curve (nSamples=11, **kwargs)

draw_roc_curve (**kwargs)

fit (*scores, labels, verbose=False*)

fn
false negative probability

fnr
miss rate, false negative rate

fp
false positive probability

fpr
fallout, false positive rate

classmethod from_tp_and_tn_scores (*tp_scores, tn_scores, verbose=False*)

get_ave_precision ()

get_fpr_at_recall (*target_recall*)

get_index_at_metric (*at_metric, at_value, subindex=False, tiebreaker='maxthresh'*)
Finds the index that is closet to the metric at a given value

Parameters **tiebreaker** (*str*) – either ‘minimize’ or ‘maximize’ if ‘maximize’, then a larger threshold is considered better when resolving ambiguities. Otherwise a smaller thresh is better.

Doctest:

```
>>> from vtool.confusion import *
>>> pat1 = [0, 0, 0, 0]
>>> pat2 = [0, 0, 1, 1]
>>> pat3 = [0, 1, 1, 1]
>>> pat4 = [1, 1, 1, 1]
>>> pats = [pat1, pat2, pat3, pat4]
>>> n = 4
>>> import itertools as it
>>> s = it.count(0)
>>> # Create places of ambiguity and unambiguity
>>> x = list(ub.flatten([[next(s)] * len(pat) for pat in pats for _ in
↳ range(n)]))
>>> y = list(ub.flatten([pat for pat in pats for _ in range(n)]))
>>> self = ConfusionMetrics().fit(x, y)
>>> at_metric = 'n_false_pos'
>>> at_value = 0
>>> subindex = False
>>> idx1 = self.get_index_at_metric(at_metric, at_value, subindex=False,
↳ tiebreaker='minthresh')
>>> idx2 = self.get_index_at_metric(at_metric, at_value, subindex=False,
↳ tiebreaker='maxthresh')
>>> assert idx1 == 3
>>> assert idx2 == 0
```

get_metric_at_index (*metric, subindex*)

get_metric_at_metric (*get_metric, at_metric, at_value, subindex=False, tiebreaker='maxthresh'*)

Finds the corresponding value of *get_metric* at a specific value of *at_metric*.

get_metric = ‘fpr’ *at_metric* = ‘tpr’ *at_value* = .25 *self.rrr()*

```
self.get_metric_at_metric('fpr', 'tpr', .25) self.get_metric_at_metric('n_false_pos', 'tpr', .25)
self.get_metric_at_metric('n_true_pos', 'tpr', .25)
```

```
get_metric = 'n_true_pos' at_metric = 'n_false_pos' at_value = 0 subindex = False
```

get_metric_at_thresh (*metric, thresh*)

Parameters

- **metric** (*str*) – name of a metric
- **thresh** (*float*) – desired threshold

Returns value - metric value

Return type float

CommandLine: python -m vtool.confusion -exec-get_metric_at_threshold

Ignore:

```
>>> self = cfms
>>> metric = 'fpr'
>>> thresh = 0
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.confusion import * # NOQA
>>> scores, labels = testdata_scores_labels()
>>> self = ConfusionMetrics().fit(scores, labels)
>>> metric = 'tpr'
>>> thresh = .8
>>> thresh = [0, .1, .9, 1.0]
>>> value = self.get_metric_at_thresh(metric, thresh)
>>> result = ('(None, None) = %s' % (str((None, None)),))
>>> print(result)
```

get_recall_at_fpr (*target_fpr*)

get_thresh_at_metric (*metric, value, maximize=None*)

Gets a threshold for a binary classifier using a target metric and value

Parameters

- **metric** (*str*) – name of metric like tpr or fpr
- **value** (*float*) – corresponding numeric value

Returns thresh

Return type float

CommandLine: python -m vtool.confusion get_thresh_at_metric python -m vtool.confusion -exec-interact_roc_factory -show

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.confusion import * # NOQA
>>> scores, labels = testdata_scores_labels()
>>> self = ConfusionMetrics().fit(scores, labels)
>>> metric = 'tpr'
>>> value = .85
>>> thresh = self.get_thresh_at_metric(metric, value)
>>> print('%s = %r' % (metric, value,))
>>> result = ('thresh = %s' % (str(thresh),))
>>> print(result)
thresh = 22.5
```

Ignore: metric = 'fpr' value = 1e-4 self = cfms maximize = False

```
interpolate_replbounds(metric_values, self.thresholds, 0, maximize=maximize) inter-
polate_replbounds(metric_values, self.thresholds, 1e-4, maximize=maximize) interpo-
late_replbounds(metric_values, self.thresholds, 1e-3, maximize=maximize) interpo-
late_replbounds(metric_values, self.thresholds, 1e-2, maximize=maximize) interpo-
late_replbounds(metric_values, self.thresholds, 1e-2, maximize=maximize)
```

get_thresh_at_metric_max(metric)
metric = 'mcc' metric = 'fnr'

inv_aliases = {'acc': 'acc', 'accuracy': 'acc', 'bias': 'pp', 'bm': 'bm', 'bookmak

jacc
jaccard coefficient

mcc
matthews correlation coefficient

Also true that: `mcc == np.sqrt(self.bm * self.mk)`

minimizing_metrics = {'fn', 'fnr', 'fp', 'fpr'}

mk
markedness

paper_alias = [['dtp', 'determinant'], ['lir', 'likelihood-ratio'], ['nlr', 'negative-li

paper_relations = {'BMG': ['dtp / evenness_G'], 'BiasG2': ['bias * 1 - bias'], 'IBias

plot_metrics()

plot_vs(x_metric, y_metric)
x_metric = 'thresholds' y_metric = 'fpr'

pn
predicted negative probability

pp
predicted positive probability

rn
real negative probability

rp
real positive probability

show_mcc()

sqrd_error
squared error

thresh

tn
true negative probability

tna
negative predictive value, inverse precision

tnr
true negative rate, inverse recall

tp
true positive probability

tpa
miss rate, false negative rate

tpr
sensitivity, recall, hit rate, tpr

wracc
weighted relative accuracy

vtool.DEFAULT_DTYPE
alias of `numpy.float32`

vtool.GaussianBlurInplace (*img*, *sigma*, *size=None*)
simulates code from `helpers.cpp` in `hesaff`

Parameters

- **img** (*ndarray*) –
- **sigma** (*float*) –

CommandLine: `python -m vtool.patch --test-GaussianBlurInplace:0 --show python -m vtool.patch --test-GaussianBlurInplace:1 --show`

References; http://www.cse.usf.edu/~rlk/MachineVisionBook/MachineVision.files/MachineVision_Chapter4.pdf http://en.wikipedia.org/wiki/Scale_space_implementation http://www.cse.psu.edu/~rtc12/CSE486/lecture10_6pp.pdf

Notes

The product of the convolution of two Gaussian functions with spread σ is a Gaussian function with spread $\sqrt{2} \cdot \sigma$ scaled by the area of the Gaussian filter

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> from mpl_toolkits.mplot3d import Axes3D # NOQA
>>> import wbia.plottool as pt
>>> img = get_test_patch('star2')
>>> img_orig = img.copy()
>>> sigma = .8
```

(continues on next page)

(continued from previous page)

```

>>> GaussianBlurInplace(img, sigma)
>>> fig = plt.figure(fnum=1, pnum=(1, 3, 1))
>>> size = int((2.0 * 3.0 * sigma + 1.0))
>>> if not size & 1: # check if even
>>>     size += 1
>>> ksize = (size, size)
>>> fig.add_subplot(1, 3, 1, projection='3d')
>>> show_gaussian_patch(ksize, sigma, sigma)
>>> plt.imshow(img_orig * 255, fnum=1, pnum=(1, 3, 2))
>>> plt.imshow(img * 255, fnum=1, pnum=(1, 3, 3))
>>> plt.show_if_requested()

```

Example

```

>>> # DISABLE_DOCTEST
>>> # demonstrate cascading smoothing property
>>> # THIS ISNT WORKING WHY???
>>> from vtool.patch import * # NOQA
>>> from mpl_toolkits.mplot3d import Axes3D # NOQA
>>> import wbia.plottool as pt
>>> img = get_test_patch('star2')
>>> img1 = img.copy()
>>> img2 = img.copy()
>>> img3 = img.copy()
>>> img4 = img.copy()
>>> img_orig = img.copy()
>>> sigma1 = .6
>>> sigma2 = .9
>>> sigma3 = sigma1 + sigma2
>>> size = 7
>>> # components
>>> GaussianBlurInplace(img1, sigma1, size)
>>> GaussianBlurInplace(img2, sigma2, size)
>>> # all in one shot
>>> GaussianBlurInplace(img3, sigma3, size)
>>> # additive
>>> GaussianBlurInplace(img4, sigma1, size)
>>> GaussianBlurInplace(img4, sigma2, size)
>>> print((img4 - img3).sum())
>>> # xdoctest: +REQUIRES(--show)
>>> fig = plt.figure(fnum=1, pnum=(2, 4, 1))
>>> ksize = (size, size)
>>> #fig.add_subplot(1, 3, 1, projection='3d')
>>> fig.add_subplot(2, 4, 1, projection='3d')
>>> show_gaussian_patch(ksize, sigma1, sigma1)
>>> fig.add_subplot(2, 4, 2, projection='3d')
>>> show_gaussian_patch(ksize, sigma2, sigma2)
>>> fig.add_subplot(2, 4, 3, projection='3d')
>>> show_gaussian_patch(ksize, sigma3, sigma3)
>>> plt.imshow(img_orig * 255, fnum=1, pnum=(2, 4, 4))
>>> plt.imshow(img1 * 255, fnum=1, pnum=(2, 4, 5), title='%r' % (sigma1))
>>> plt.imshow(img2 * 255, fnum=1, pnum=(2, 4, 6), title='%r' % (sigma2))
>>> plt.imshow(img3 * 255, fnum=1, pnum=(2, 4, 7), title='%r' % (sigma3))
>>> plt.imshow(img4 * 255, fnum=1, pnum=(2, 4, 8), title='%r + %r' % (sigma1,
↵sigma2))

```

(continues on next page)

(continued from previous page)

```
>>> pt.show_if_requested()
```

vtool.INDEX_DTYPE

alias of `numpy.int32`

vtool.KPTS_DTYPE

alias of `numpy.float32`

vtool.L1 (*hist1*, *hist2*, *dtype=<class 'numpy.float64'>*)

returns L1 (aka manhattan or grid) distance between two histograms

vtool.L2 (*hist1*, *hist2*)

returns L2 (aka euclidean or standard) distance between two histograms

vtool.L2_root_sift (*hist1*, *hist2*)

Normalized Root-SIFT L2

Parameters

- **hist1** (*ndarray*) – Nx128 array of uint8 with pseudomax trick
- **hist2** (*ndarray*) – Nx128 array of uint8 with pseudomax trick

Returns euclidean distance between 0-1 normalized sift descriptors

Return type `ndarray`

vtool.L2_sift (*hist1*, *hist2*)

Normalized SIFT L2

Parameters

- **hist1** (*ndarray*) – Nx128 array of uint8 with pseudomax trick
- **hist2** (*ndarray*) – Nx128 array of uint8 with pseudomax trick

Returns euclidean distance between 0-1 normalized sift descriptors

Return type `ndarray`

CommandLine: `python -m vtool.distance --test-L2_sift`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> hist1, hist2 = testdata_hist()
>>> sift1, sift2, sift3, sift4, sift5 = testdata_sift2()
>>> l2_dist = L2_sift(hist1, hist2)
>>> max_dist = L2_sift(sift4, sift5)
>>> assert np.isclose(max_dist, 1.0)
>>> result = ub.repr2(l2_dist, precision=2)
```

vtool.L2_sift_sqrd (*hist1*, *hist2*)

Normalized SIFT L2**2

Parameters

- **hist1** (*ndarray*) – Nx128 array of uint8 with pseudomax trick
- **hist2** (*ndarray*) – Nx128 array of uint8 with pseudomax trick

Returns squared euclidean distance between 0-1 normalized sift descriptors

Return type ndarray

`vtool.L2_sqrd(hist1, hist2, dtype=<class 'numpy.float64'>)`

returns the squared L2 distance

FIXME: if `hist1.shape = (0,)` and `hist.shape = (0,)` then `result=0.0`

SeeAlso: L2

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> import numpy
>>> ut.exec_func(kw(L2_sqrd, globals()))
>>> rng = np.random.RandomState(53)
>>> hist1 = rng.rand(5, 2)
>>> hist2 = rng.rand(5, 2)
>>> l2dist = L2_sqrd(hist1, hist2)
>>> result = ub.repr2(l2dist, precision=2, threshold=2)
```

exception `vtool.MatchingError`

Bases: `Exception`

class `vtool.PairwiseMatch(annot1=None, annot2=None)`

Bases: `ubelt.util_mixins.NiceRepr`

Newest (Sept-16-2016) object oriented one-vs-one matching interface

Creates an object holding two annotations Then a pipeline of operations can be applied to generate score and refine the matches

Note: The annotation dictionaries are required to have certain attributes.

Required annotation attributes: (kpts, vecs) OR `rchip` OR `rchip_fpath`

Optional annotation attributes: `aid`, `nid`, `flann`, `rchip`, `dlen_sqrd`, `weight`

Ignore:

```
>>> from vtool.matching import * # NOQA
>>> import vtool as vt
>>> imgR = vt.imread(ut.grab_test_imgpath('easy1.png'))
>>> imgL = vt.imread(ut.grab_test_imgpath('easy2.png'))
>>> annot1 = {'rchip': imgR}
>>> annot2 = {'rchip': imgL}
>>> match = vt.PairwiseMatch(annot1, annot2)
>>> match.apply_all({'refine_method': 'affine', 'affine_invariance': False,
↪ 'rotation_invariance': False})
>>> dsize = imgR.shape[0:2][::-1]
>>> imgR_warp = vt.warpHomog(imgR, match.H_12, dsize)
>>> # xdoctest: +REQUIRES(--show)
>>> import kwplot
>>> kwplot.autompl()
>>> kwplot.imshow(imgL, pnum=(2, 1, 1))
```

(continues on next page)

(continued from previous page)

```

>>> kwplot.imshow(imgR_warp, pnum=(2, 1, 2))
>>> kwplot.imshow(imgL, pnum=(2, 1, 1))
>>> kwplot.imshow(imgR_warp, pnum=(2, 1, 2))
>>> # xdoctest: +REQUIRES(--gui)
>>> import wbia.guitool as gt
>>> gt.ensure_qapp()
>>> match.ishow()
>>> from vtool.matching import * # NOQA
>>> import vtool as vt
>>> imgR = vt.imread(ut.grab_test_imgpath('easy1.png'))
>>> imgL = vt.imread(ut.grab_test_imgpath('easy2.png'))
>>> annot1 = {'rchip': imgR}
>>> annot2 = {'rchip': imgL}
>>> match = vt.PairwiseMatch(annot1, annot2)
>>> match.apply_all({'refine_method': 'affine', 'affine_invariance': False,
↪ 'rotation_invariance': False})
>>> dsize = imgR.shape[0:2][::-1]
>>> imgR_warp = vt.warpHomog(imgR, match.H_12, dsize)
>>> # xdoctest: +REQUIRES(--show)
>>> import kwplot
>>> kwplot.autompl()
>>> kwplot.imshow(imgL, pnum=(2, 1, 1))
>>> kwplot.imshow(imgR_warp, pnum=(2, 1, 2))
>>> kwplot.imshow(imgL, pnum=(2, 1, 1))
>>> kwplot.imshow(imgR_warp, pnum=(2, 1, 2))
>>> # xdoctest: +REQUIRES(--gui)
>>> import wbia.guitool as gt
>>> gt.ensure_qapp()
>>> match.ishow()

```

add_global_measures (global_keys)

add_local_measures (xy=True, scale=True)

apply_all (cfgdict)

apply_ratio_test (cfgdict={}, inplace=None)

apply_sver (cfgdict={}, inplace=None)

Ignore:

```

>>> from vtool.matching import * # NOQA
>>> cfgdict = {'symmetric': True, 'ratio_thresh': .8,
>>>             'thresh_bins': [.5, .6, .7, .8]}
>>> match = demodata_match(cfgdict, apply=False)
>>> match = match.assign(cfgbase)
>>> match.apply_ratio_test(cfgdict, inplace=True)
>>> flags1 = match.apply_sver(cfgdict)

```

assign (cfgdict={}, verbose=None)

Assign feature correspondences between annots

Example

```
>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.matching import * # NOQA
>>> cfgdict = {'symmetric': True}
>>> match = demodata_match({}, apply=False)
>>> m1 = match.copy().assign({'symmetric': False})
>>> m2 = match.copy().assign({'symmetric': True})
```

Example

```
>>> from vtool.matching import * # NOQA
>>> grid = {
>>>     'symmetric': [True, False],
>>> }
>>> for cfgdict in ut.all_dict_combinations(grid):
>>>     match = demodata_match(cfgdict, apply=False)
>>>     match.assign()
```

compress (*flags*, *inplace=None*)

copy ()

ishow ()

CommandLine: python -m vtool.matching ishow --show

Example

```
>>> # SCRIPT
>>> from vtool.matching import * # NOQA
>>> import vtool as vt
>>> import wbia.guitool as gt
>>> gt.ensure_qapp()
>>> match = demodata_match(use_cache=False)
>>> self = match.ishow()
>>> self.disp_config['show_homog'] = True
>>> self.update()
>>> # xdoctest: +REQUIRES(--show)
>>> gt.qtapp_loop(qwin=self, freq=10)
```

make_feature_vector (*local_keys=None*, *global_keys=None*, *summary_ops=None*,
sorters='ratio', *indices=3*, *bin_key=None*, *bins=None*)

Constructs the pairwise feature vector that represents a match

Parameters

- **local_keys** (*None*) – (default = None)
- **global_keys** (*None*) – (default = None)
- **summary_ops** (*None*) – (default = None)
- **sorters** (*str*) – (default = 'ratio')
- **indices** (*int*) – (default = 3)

Returns feat

Return type dict

CommandLine: `python -m vtool.matching make_feature_vector`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.matching import * # NOQA
>>> import vtool as vt
>>> match = demodata_match({})
>>> feat = match.make_feature_vector(indices=[0, 1])
>>> result = ('feat = %s' % (ub.repr2(feat, nl=2),))
>>> print(result)
```

matched_vecs2()

ratio_test_flags (cfgdict={})

show (ax=None, show_homog=False, show_ori=False, show_ell=True, show_pts=False, show_lines=True, show_rect=False, show_eig=False, show_all_kpts=False, mask_blend=0, ell_alpha=0.6, line_alpha=0.35, modifysize=False, vert=None, overlay=True, heatmask=False, line_lw=1.4)

sver_flags (cfgdict={}, return_extra=False)

Example

```
>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.matching import * # NOQA
>>> cfgdict = {'symmetric': True, 'newsym': True}
>>> match = demodata_match(cfgdict, apply=False)
>>> cfgbase = {'symmetric': True, 'ratio_thresh': .8}
>>> cfgdict = ut.dict_union(cfgbase, dict(thresh_bins=[.5, .6, .7, .8]))
>>> match = match.assign(cfgbase)
>>> match.apply_ratio_test(cfgdict, inplace=True)
>>> flags1 = match.sver_flags(cfgdict)
>>> flags2 = match.sver_flags(cfgbase)
```

take (indicies, inplace=None)

vtool.SV_DTYPE

alias of `numpy.float64`

class vtool.ScaleStrat

Bases: `object`

Scaling strategies

static area (target, orig_wh, tol=0)

The area becomes target

Parameters **target** (`int`) – target size

Example

```
>>> # ENABLE_DOCTEST
>>> import vtool as ut
>>> ut.assert_eq(ScaleStrat.area(800 ** 2, (190, 220)), (743, 861))
>>> ut.assert_eq(ScaleStrat.area(800 ** 2, (220, 190)), (861, 743))
```

static maxwh (*target, orig_wh, tol=0*)

The maximum dimension becomes target

Parameters **target** (*int*) – target size

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> ut.assert_eq(ScaleStrat.maxwh(800, (190, 220)), (691, 800))
>>> ut.assert_eq(ScaleStrat.maxwh(800, (220, 190)), (800, 691))
```

static width (*target, orig_wh, tol=0*)

The width becomes target

Parameters **target** (*int*) – target size

Example

```
>>> # ENABLE_DOCTEST
>>> import utool as ut
>>> ut.assert_eq(ScaleStrat.width(800, (190, 220)), (800, 926))
>>> ut.assert_eq(ScaleStrat.width(800, (220, 190)), (800, 691))
```

class vtool.**ScoreNormVisualizeClass**

Bases: *object*

HACK; eventually move all individual plots into a class structure

class vtool.**ScoreNormalizer** (***kwargs*)

Bases: *utool.util_cache.Cachable, vtool.score_normalization.ScoreNormVisualizeClass*

Conforms to scikit-learn Estimator interface

CommandLine: `python -m vtool.score_normalization -test-ScoreNormalizer -show -cmd`

Kwargs: *tpr* (float): target true positive rate (default .90) *fpr* (float): target false positive rate (default None)
reverse (bool): True if lower scores are better, False if higher scores are better (default=None)

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.score_normalization import * # NOQA
>>> import vtool as vt
>>> encoder = ScoreNormalizer()
>>> X, y = vt.demodata.testdata_binary_scores()
>>> attrs = {'index': np.arange(len(y)) * ((2 * y) - 1)}
>>> encoder.fit(X, y, attrs)
>>> # xdoctest: +REQUIRES(--show)
>>> encoder.visualize()
>>> ut.show_if_requested()
```

fit (*X, y, attrs=None, verbose=False, finite_only=True*)

Fits estimator to data

Parameters

- **X** (*ndarray*) – one dimensional scores
- **y** (*ndarray*) – binary labels
- **attrs** (*dict*) – dictionary of data attributes

fit_partitioned (*tp_scores, tn_scores, part_attrs=None, **kwargs*)
 convenience func to fit only scores that have been separated instead of labeled

get_accuracy (*X, y*)

get_confusion_indicies (*X, y*)
 combination of `get_correct_indices` and `get_error_indicies`

get_correct_indices (*X, y*)

Parameters

- **X** (*ndarray*) – data
- **y** (*ndarray*) – labels

Returns (*fp_indicies, fn_indicies*)

Return type `tuple`

CommandLine: `python -m vtool.score_normalization --test-get_correct_indices`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.score_normalization import * # NOQA
>>> encoder, X, y = testdata_score_normalier()
>>> (tp_indicies, tn_indicies) = encoder.get_correct_indices(X, y)
>>> tp_X = X.take(tp_indicies)[0:3]
>>> tn_X = X.take(tn_indicies)[0:3]
>>> result = 'tp_X = ' + ub.repr2(tp_X)
>>> result += '\ntn_X = ' + ub.repr2(tn_X)
>>> print(result)
tp_X = np.array([ 8.883,  8.77 ,  8.759])
tn_X = np.array([ 0.727,  0.76 ,  0.841])
```

get_error_indicies (*X, y*)

Returns the indicies of the most difficult type I and type II errors.

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.score_normalization import * # NOQA
>>> encoder, X, y = testdata_score_normalier()
>>> (fp_indicies, fn_indicies) = encoder.get_error_indicies(X, y)
>>> fp_X = X.take(fp_indicies)[0:3]
>>> fn_X = X.take(fn_indicies)[0:3]
>>> result = 'fp_X = ' + ub.repr2(fp_X)
>>> result += '\nfn_X = ' + ub.repr2(fn_X)
>>> print(result)
```

(continues on next page)

(continued from previous page)

```
fp_X = np.array([ 6.196,  5.912,  5.804])
fn_X = np.array([ 3.947,  4.277,  4.43 ])
```

get_partitioned_support()
convenience get prepartitioned data

get_prefix()

get_support (*finite_only=True*)
return X, y, and attrs

inverse_normalize (*probs*)

learn_probabilities (*verbose=False*)
Kernel density estimation

learn_threshold (*verbose=False, **thresh_kw*)
Learns cutoff threshold that achieves the target confusion metric Typically a desired false positive rate (recall) is specified

learn_threshold2 ()
Finds a cutoff where the probability of a truepos stats becoming greater than probability of trueneg

CommandLine: python -m vtool.score_normalization --exec-learn_threshold2 --show

Example

```
>>> from vtool.score_normalization import * # NOQA
>>> import vtool as vt
>>> #encoder, X, y = testdata_score_normalier([(3.5, 256), (9.5, 1024), (15.5,
↳ 2048)], [(6.5, 256), (12.5, 5064), (18.5, 128)], adjust=1, p_tp_method=
↳ 'ratio')
>>> encoder, X, y = testdata_score_normalier([(3.5, 64), (9.5, 1024), (15.5,
↳ 5064)], [(6.5, 256), (12.5, 2048), (18.5, 128)], adjust=1, p_tp_method=
↳ 'ratio')
>>> #encoder, X, y = testdata_score_normalier(adjust=1)
>>> #encoder, X, y = testdata_score_normalier([(3.5, 2048)], [(30.5, 128)],
↳ tn_scale=1, adjust=1)
>>> #encoder, X, y = testdata_score_normalier([(0, 64)], [(-.1, 12)],
↳ adjust=8, min_clip=0)
>>> locals_ = ut.exec_func_src(encoder.learn_threshold2)
>>> exec(ut.execstr_dict(locals_))
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.ensureqt()
>>> #pt.plot(xdata[0:-2], np.diff(np.diff(closeness)))
>>> #maxima_x, maxima_y, argmaxima = vt.hist_argmaxima(closeness)
>>> fnum = 100
>>> pt.multi_plot(xdata, [tp_curve, tn_curve, closeness, ],
>>>                 label_list=['p(tp | s)', 'p(tn | s)', 'closeness', ],
↳ marker='|',
>>>                 linewidth_list=[4, 4, 1,], title='intersection points',
>>>                 pnum=(4, 1, 1), fnum=fnum, xmax=xdata.max(), xmin=0)
>>> pt.plot(xdata[argmaxima], closeness[argmaxima], 'rx', label='closeness_
↳ maxima')
>>> pt.plot(x_submax, y_submax, 'o', label='chosen')
>>> #pt.plot(xdata[argmaxima], curveness[argmaxima], 'rx', label='curveness_
↳ maxima')
```

(continues on next page)

(continued from previous page)

```

>>> pt.legend()
>>> #pt.plot(x_submax, y_submax, 'o')
>>> pt.plot(xdata[argmaxima], tp_curve[argmaxima], 'rx')
>>> pt.plot(xdata[argmaxima], tn_curve[argmaxima], 'rx')
>>> pt.plot(xdata[argmaxima], tp_curve[argmaxima], 'rx')
>>> pt.plot(xdata[argmaxima], tn_curve[argmaxima], 'rx')
>>> #pt.plot(xdata[argmaxima], encoder.interp_fn(x_submax), 'rx')
>>> _mkinterp = ut.partial(
>>>     scipy.interpolate.interpld, kind='linear', copy=False,
>>>     assume_sorted=False, bounds_error=False)
>>> _interp_sgt_n = _mkinterp(xdata, tn_curve)
>>> _interp_sgt_p = _mkinterp(xdata, tp_curve)
>>> pt.plot(x_submax, _interp_sgt_n(x_submax), 'go')
>>> pt.plot(x_submax, _interp_sgt_p(x_submax), 'bx')
>>> #
>>> pt.multi_plot(xdata[argmaxima], [tp_area, fp_area, tn_area, fn_area],
>>>     title='intersection areas',
>>>     label_list=['tp_area', 'fp_area', 'tn_area', 'fn_area'],
>>>     markers=['o', 'd', 'o', '.'],
>>>     pnum=(4, 1, 2), fnum=fnum, xmax=xdata.max(), xmin=0)
>>> #
>>> pt.multi_plot(xdata[argmaxima], [lr_pos, lr_neg, acc], title=
>>>     'intersection quality (likelihood ratios)',
>>>     label_list=['lr_pos=tp/fp', 'lr_neg=fn/tn', 'acc'],
>>>     markers=['o', 'o', '*'],
>>>     pnum=(4, 1, 3), fnum=fnum, xmax=xdata.max(), xmin=0)
>>> #
>>> pnum_ = pt.make_pnum_nextgen(4, 3, start=9)
>>> encoder._plot_score_support_hist(fnum=fnum, pnum=pnum_())
>>> #encoder._plot_prebayes(fnum=fnum, pnum=pnum_())
>>> encoder._plot_postbayes(fnum=fnum, pnum=pnum_())
>>> encoder._plot_roc(fnum=fnum, pnum=pnum_())
>>> pt.adjust_subplots(hspace=.5, top=.95, bottom=.08)
>>> pt.show_if_requested()

```

normalize_scores (*X*)**predict** (*X*)Predict true or false of *X*.**visualize** (***kwargs*)

shows details about the score normalizer

Kwargs: fnum figtitle with_hist interactive with_scores with_roc with_precision_recall**CommandLine:** python -m vtool.score_normalization --exec-ScoreNormalizer.visualize:0 --show python
-m vtool.score_normalization --exec-ScoreNormalizer.visualize:1 --show

Example

```

>>> # UNSTABLE_DOCTEST
>>> from vtool.score_normalization import * # NOQA
>>> import vtool as vt
>>> encoder = ScoreNormalizer()
>>> X, y = vt.demodata.testdata_binary_scores()
>>> encoder.fit(X, y)

```

(continues on next page)

(continued from previous page)

```
>>> kwargs = dict(
>>>     with_pr=True, interactive=True, with_roc=True,
>>>     with_hist=True)
>>> encoder.visualize(**kwargs)
>>> ut.show_if_requested()
```

Example

```
>>> # UNSTABLE_DOCTEST
>>> from vtool.score_normalization import * # NOQA
>>> import vtool as vt
>>> encoder = ScoreNormalizer()
>>> X, y = vt.demodata.testdata_binary_scores()
>>> encoder.fit(X, y)
>>> kwargs = dict(
>>>     with_pr=True, interactive=True, with_roc=True, with_hist=True,
>>>     with_scores=False, with_prebayes=False, with_postbayes=False)
>>> encoder.visualize(target_tpr=.95, **kwargs)
>>> ut.show_if_requested()
```

`vtool.TEMP_VEC_DTYPE`
alias of `numpy.float64`

`vtool.TRANSFORM_DTYPE`
alias of `numpy.float64`

`vtool.adaptive_scale` (*img_fpath*, *kpts*, *nScales*=4, *low*=-0.5, *high*=0.5, *nSamples*=16)

`vtool.add_homogenous_coordinate` (*_xys*)

CommandLine: `python -m vtool.linalg --test-add_homogenous_coordinate`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.linalg import * # NOQA
>>> _xys = np.array([[ 2.,  0.,  0.,  2.],
...                 [ 2.,  2.,  0.,  0.]], dtype=np.float32)
>>> _xyzs = add_homogenous_coordinate(_xys)
>>> assert np.all(_xys == remove_homogenous_coordinate(_xyzs))
>>> result = ub.repr2(_xyzs, with_dtype=True)
>>> print(result)
```

`vtool.affine_around_mat3x3` (*x*, *y*, *sx*=1.0, *sy*=1.0, *theta*=0.0, *shear*=0.0, *tx*=0.0, *ty*=0.0, *x2*=None, *y2*=None)

Executes an affine transform around center point (*x*, *y*). Equivalent to `translation.dot(affine).dot(inv(translation))`

Parameters

- **x** (*float*) – center x location in input space
- **y** (*float*) – center y location in input space
- **sx** (*float*) – x scale factor (default = 1)
- **sy** (*float*) – y scale factor (default = 1)

- **theta** (*float*) – counter-clockwise rotation angle in radians(default = 0)
- **shear** (*float*) – counter-clockwise shear angle in radians(default = 0)
- **tx** (*float*) – x-translation (default = 0)
- **ty** (*float*) – y-translation (default = 0)
- **x2** (*float*, *optional*) – center x location in output space (default = x)
- **y2** (*float*, *optional*) – center y location in output space (default = y)

CommandLine: python -m vtool.linalg affine_around_mat3x3 --show

CommandLine: xdoctest -m ~/code/vtool/vtool/linalg.py affine_around_mat3x3

Example

```
>>> from vtool.linalg import * # NOQA
>>> import vtool as vt
>>> orig_pts = np.array(vt.verts_from_bbox([10, 10, 20, 20]))
>>> x, y = vt.bbox_center(vt.bbox_from_verts(orig_pts))
>>> sx, sy = 0.5, 1.0
>>> theta = 1 * np.pi / 4
>>> shear = .1 * np.pi / 4
>>> tx, ty = 5, 0
>>> x2, y2 = None, None
>>> Aff = affine_around_mat3x3(x, y, sx, sy, theta, shear,
>>>                             tx, ty, x2, y2)
>>> trans_pts = vt.transform_points_with_homography(Aff, orig_pts.T).T
>>> # xdoc: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.ensureqt()
>>> pt.plt.plot(x, y, 'bx', label='center')
>>> pt.plt.plot(orig_pts.T[0], orig_pts.T[1], 'b-', label='original')
>>> pt.plt.plot(trans_pts.T[0], trans_pts.T[1], 'r-', label='transformed')
>>> pt.plt.legend()
>>> pt.plt.title('Demo of affine_around_mat3x3')
>>> pt.plt.axis('equal')
>>> pt.plt.xlim(0, 40)
>>> pt.plt.ylim(0, 40)
>>> ut.show_if_requested()
```

Ignore:

```
>>> from vtool.linalg import * # NOQA
>>> x, y, sx, sy, theta, shear, tx, ty, x2, y2 = (
>>>     256.0, 256.0, 1.5, 1.0, 0.78, 0.2, 0, 100, 500.0, 500.0)
>>> for timer in ub.Timerit(1000, 'old'): # 19.0697 µs
>>>     with timer:
>>>         tr1_ = translation_mat3x3(-x, -y)
>>>         Aff_ = affine_mat3x3(sx, sy, theta, shear, tx, ty)
>>>         tr2_ = translation_mat3x3(x2, y2)
>>>         Aff1 = tr2_.dot(Aff_).dot(tr1_)
>>> for timer in ub.Timerit(1000, 'new'): # 11.0242 µs
>>>     with timer:
>>>         Aff2 = affine_around_mat3x3(x, y, sx, sy, theta, shear,
```

(continues on next page)

(continued from previous page)

```
>>>                                     tx, ty, x2, y2)
>>> assert np.all(np.isclose(Aff2, Aff1))
```

Ignore:

```
>>> from vtool.linalg import * # NOQA
>>> import vtool as vt
>>> import sympy
>>> # Shows the symbolic construction of the code
>>> # https://groups.google.com/forum/#!topic/sympy/k1HnZK_bNNA
>>> from sympy.abc import theta
>>> x, y, sx, sy, theta, shear, tx, ty, x2, y2 = sympy.symbols(
>>>     'x, y, sx, sy, theta, shear, tx, ty, x2, y2')
>>> theta = sx = sy = tx = ty = 0
>>> # move to center xy, apply affine transform, move center xy2
>>> tr1_ = translation_mat3x3(-x, -y, dtype=None)
>>> Aff_ = affine_mat3x3(sx, sy, theta, shear, tx, ty, trig=sympy)
>>> tr2_ = translation_mat3x3(x2, y2, dtype=None)
>>> # combine transformations
>>> Aff = vt.sympy_mat(tr2_.dot(Aff_).dot(tr1_))
>>> vt.evalprint('Aff')
>>> print('-----')
>>> print('Numpy')
>>> vt.sympy_numpy_repr(Aff)
```

```
vtool.affine_mat3x3(sx=1, sy=1, theta=0, shear=0, tx=0, ty=0, trig=<module 'numpy'
from
/home/docs/checkouts/readthedocs.org/user_builds/wbia-
vtool/envs/stable/lib/python3.7/site-packages/numpy/__init__.py'>)
```

Parameters

- **sx** (*float*) – x scale factor (default = 1)
- **sy** (*float*) – y scale factor (default = 1)
- **theta** (*float*) – rotation angle (radians) in counterclockwise direction
- **shear** (*float*) – shear angle (radians) in counterclockwise directions
- **tx** (*float*) – x-translation (default = 0)
- **ty** (*float*) – y-translation (default = 0)

References

https://github.com/scikit-image/scikit-image/blob/master/skimage/transform/_geometric.py

```
vtool.affine_warp_around_center(img, sx=1, sy=1, theta=0, shear=0, tx=0, ty=0, dsize=None,
borderMode=0, flags=4, out=None, **kwargs)
```

CommandLine: python -m vtool.image --test-affine_warp_around_center --show

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
```

(continues on next page)

(continued from previous page)

```

>>> img_fpath = ut.grab_test_imgpath('carl.jpg')
>>> img = vt.imread(img_fpath) / 255.0
>>> img = img.astype(np.float32)
>>> dsize = (1000, 1000)
>>> shear = .2
>>> theta = np.pi / 4
>>> tx = 0
>>> ty = 100
>>> sx = 1.5
>>> sy = 1.0
>>> borderMode = cv2.BORDER_CONSTANT
>>> flags = cv2.INTER_LANCZOS4
>>> img_warped = affine_warp_around_center(img, sx=sx, sy=sy,
...     theta=theta, shear=shear, tx=tx, ty=ty, dsize=dsize,
...     borderMode=borderMode, flags=flags, borderValue=(.5, .5, .5))
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow((img_warped * 255.0).astype(np.uint8))
>>> ut.show_if_requested()

```

`vtool.and_lists(*args)`

Like `np.logical_and`, but can take more than 2 arguments

CommandLine: `python -m vtool.other --test-and_lists`

SeeAlso: `or_lists`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> arg1 = np.array([1, 1, 1, 1,])
>>> arg2 = np.array([1, 1, 0, 1,])
>>> arg3 = np.array([0, 1, 0, 1,])
>>> args = (arg1, arg2, arg3)
>>> flags = and_lists(*args)
>>> result = str(flags)
>>> print(result)
[False True False True]

```

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> size = 10000
>>> rng = np.random.RandomState(0)
>>> arg1 = rng.randint(2, size=size)
>>> arg2 = rng.randint(2, size=size)
>>> arg3 = rng.randint(2, size=size)
>>> args = (arg1, arg2, arg3)
>>> flags = and_lists(*args)
>>> # ensure equal division
>>> segments = 5
>>> validx = np.where(flags)[0]

```

(continues on next page)

(continued from previous page)

```
>>> endx = int(segments * (validx.size // (segments)))
>>> parts = np.split(validx[:endx], segments)
>>> result = str(list(map(np.sum, parts)))
>>> print(result)
[243734, 714397, 1204989, 1729375, 2235191]
```

`%timeit reduce(np.logical_and, args) %timeit np.logical_and.reduce(args) # wins with more data`

`vtool.ann_flann_once(dpts, qpts, num_neighbors, flann_params={})`

Finds the approximate nearest neighbors of qpts in dpts

CommandLine: `xdoctest -m ~/code/vtool/vtool/nearest_neighbors.py ann_flann_once:0`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.nearest_neighbors import * # NOQA
>>> np.random.seed(1)
>>> dpts = np.random.randint(0, 255, (5, 128)).astype(np.uint8)
>>> qpts = np.random.randint(0, 255, (5, 128)).astype(np.uint8)
>>> qx2_dx, qx2_dist = ann_flann_once(dpts, qpts, 2)
>>> import ubelt as ub
>>> result = ub.repr2((qx2_dx.T, qx2_dist.T), precision=2, with_dtype=True, nl=2)
>>> print(result)
(
  np.array([[3, 3, 3, 3, 0],
            [2, 0, 1, 4, 4]], dtype=np.int32),
  np.array([[1037329., 1235876., 1168550., 1286435., 1075507.],
            [1038324., 1243690., 1304896., 1320598., 1369036.]], dtype=np.
→float32),
)
```

Example

```
>>> # ENABLE_DOCTEST
>>> # Test upper bounds on sift descriptors
>>> # SeeAlso distance.understanding_pseudomax_props
>>> from vtool.nearest_neighbors import * # NOQA
>>> import vtool as vt
>>> import numpy as np
>>> np.random.seed(1)
>>> # get points on unit sphere
>>> nDpts = 5000 # 5
>>> nQpts = 10000 # 10
>>> dpts = vt.normalize_rows(np.random.rand(nDpts, 128))
>>> qpts = vt.normalize_rows(np.random.rand(nQpts, 128))
>>> qmag = np.sqrt(np.power(qpts, 2).sum(1))
>>> dmag = np.sqrt(np.power(dpts, 2).sum(1))
>>> assert np.all(np.allclose(qmag, 1)), 'not on unit sphere'
>>> assert np.all(np.allclose(dmag, 1)), 'not on unit sphere'
>>> # cast to uint8
>>> uint8_max = 512 # hack
>>> uint8_min = 0 # hack
>>> K = 100 # 2
```

```

>>> qpts8 = np.clip(np.round(qpts * uint8_max), uint8_min, uint8_max).astype(np.
↳uint8)
>>> dpts8 = np.clip(np.round(dpts * uint8_max), uint8_min, uint8_max).astype(np.
↳uint8)
>>> qmag8 = np.sqrt(np.power(qpts8.astype(np.float32), 2).sum(1))
>>> dmag8 = np.sqrt(np.power(dpts8.astype(np.float32), 2).sum(1))
>>> # test
>>> qx2_dx, qx2_dist = ann_flann_once(dpts8, qpts8, K)
>>> biggest_dist = np.sqrt(qx2_dist.max())
>>> print('biggest_dist = %r' % (biggest_dist))
>>> # Get actual distance by hand
>>> hand_dist = np.sum((qpts8 - dpts8[qx2_dx.T[0]]) ** 2, 0)
>>> # Seems like flann returns squared distance. makes sense
>>> result = ub.hash_data(repr((qx2_dx, qx2_dist)))
>>> print(result)

```

Example:

```

>>> # Build theoretically maximally distant vectors
>>> b = 512
>>> D = 128
>>> x = np.sqrt((float(b) ** 2) / float(D - 1))
>>> dpts = np.ones((2, 128)) * x
>>> qpts = np.zeros((2, 128))
>>> dpts[:, 0] = 0
>>> qpts[:, 0] = 512
>>> qpts[:, 0::2] = 1
>>> dpts[:, 1::2] = 1
>>> qpts[:, 1::2] = 0
>>> dpts[:, 0::2] = 0
>>> qmag = np.sqrt(np.power(qpts.astype(np.float64), 2).sum(1))
>>> dmag = np.sqrt(np.power(dpts.astype(np.float64), 2).sum(1))
>>> # FIX TO ACTUALLY BE AT THE RIGHT NORM
>>> dpts = (dpts * (512 / np.linalg.norm(dpts, axis=1))[:, None]).
↳astype(np.float32)
>>> qpts = (qpts * (512 / np.linalg.norm(qpts, axis=1))[:, None]).
↳astype(np.float32)
>>> print(np.linalg.norm(dpts))
>>> print(np.linalg.norm(qpts))
>>> dist = np.sqrt(np.sum((qpts - dpts) ** 2, 1))
>>> # Because of norm condition another maximally disant pair of_
↳vectors
>>> # is [1, 0, 0, ... 0] and [0, 1, .. 0, 0, 0]
>>> # verifythat this gives you same dist.
>>> dist2 = np.sqrt((512 ** 2 + 512 ** 2))
>>> print(dist2)
>>> print(dist)

```

`vtool.apply_filter_funcs(chipBGR, filter_funcs)`
applies a list of preprocessing filters to a chip

DEPRICATE

`vtool.apply_grouping(items, groupxs, axis=0)`
applies grouping from group_indicies apply_grouping

Parameters

- **items** (*ndarray*) –
- **groupxs** (*list of ndarrays*) –

Returns grouped items

Return type list of ndarrays

SeeAlso: `group_indices` `invert_apply_grouping`

CommandLine: `python -m vtool.clustering2 -test-apply_grouping`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.clustering2 import * # NOQA
>>> idx2_groupid = np.array([2, 1, 2, 1, 2, 1, 2, 3, 3, 3])
>>> items = np.array([1, 8, 5, 5, 8, 6, 7, 5, 3, 0, 9])
>>> (keys, groupxs) = group_indices(idx2_groupid)
>>> grouped_items = apply_grouping(items, groupxs)
>>> result = str(grouped_items)
>>> print(result)
[array([8, 5, 6]), array([1, 5, 8, 7]), array([5, 3, 0, 9])]
```

`vtool.apply_grouping_` (*items*, *groupxs*)
non-optimized version

`vtool.apply_grouping_iter` (*items*, *groupxs*)

`vtool.apply_grouping_iter2` (*items*, *groupxs*)

`vtool.apply_jagged_grouping` (*unflat_items*, *groupxs*)
takes *unflat_list* and *flat_group_indices*. Returns the *unflat grouping*

`vtool.argsort_groups` (*scores_list*, *reverse=False*, *rng=<module 'numpy.random' from '/home/docs/checkouts/readthedocs.org/user_builds/wbia-vtool/envs/stable/lib/python3.7/site-packages/numpy/random/__init__.py'>*, *randomize_levels=True*)

Sorts each group normally, but randomizes order of level values.

TODO: move to `vtool`

Parameters

- **scores_list** (*list*) –
- **reverse** (*bool*) – (default = `True`)
- **rng** (*module*) – random number generator (default = `numpy.random`)

CommandLine: `python -m wbia.init.filter_annots -exec-argsort_groups`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> scores_list = [
>>>     np.array([np.nan, np.nan], dtype=np.float32),
>>>     np.array([np.nan, 2], dtype=np.float32),
```

(continues on next page)

(continued from previous page)

```

>>> np.array([4, 1, 1], dtype=np.float32),
>>> np.array([7, 3, 3, 0, 9, 7, 5, 8], dtype=np.float32),
>>> np.array([2, 4], dtype=np.float32),
>>> np.array([np.nan, 4, np.nan, 8, np.nan, 9], dtype=np.float32),
>>> ]
>>> reverse = True
>>> rng = np.random.RandomState(0)
>>> idxs_list = argsort_groups(scores_list, reverse, rng)
>>> result = 'idxs_list = %s' % (ut.repr4(idxs_list, with_dtype=False),)
>>> print(result)

```

`vtool.argsort_records` (*arrays*, *reverse=False*)

Sorts arrays that form records. Same as `lexsort(arrays[::-1])` — ie. rows are reversed.

Parameters

- **arrays** (*ndarray*) – array of records
- **reverse** (*bool*) – (default = False)

Returns `sortx` - sorted indicies

Return type `ndarray`

CommandLine: `python -m vtool.other --exec-argsort_records`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> arrays = np.array([
>>>     [1, 1, 1, 2, 2, 2, 3, 4, 5],
>>>     [2, 0, 2, 6, 4, 3, 2, 5, 6],
>>>     [1, 1, 0, 2, 3, 4, 5, 6, 7],
>>> ],)
>>> reverse = False
>>> sortx = argsort_records(arrays, reverse)
>>> result = ('sortx = %s' % (str(sortx),))
>>> print('lxsrt = %s' % (np.lexsort(arrays[::-1]),))
>>> print(result)
sortx = [1 2 0 5 4 3 6 7 8]

```

`vtool.argsubextrema2` (*op*, *ydata*, *xdata=None*, *thresh_factor=None*, *normalize_x=True*, *flat=True*)

Determines approximate maxima values to subindex accuracy.

Parameters

- **ydata** (*ndarray*) – ydata, histogram frequencies
- **xdata** (*ndarray*) – xdata, histogram labels
- **thresh_factor** (*float*) – cutoff point for labeling a value as a maxima
- **flat** (*bool*) – if True allows for flat extrema to be found.

Returns (`submaxima_x`, `submaxima_y`)

Return type `tuple`

CommandLine: `python -m vtool.histogram argsubmaxima python -m vtool.histogram argsubmaxima --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> thresh_factor = .8
>>> ydata = np.array([6.73, 8.69, 0.00, 0.00, 34.62, 29.16, 0.00, 0.00, 6.73, 8.
↳69])
>>> xdata = np.array([-0.39, 0.39, 1.18, 1.96, 2.75, 3.53, 4.32, 5.11, 5.89, 6.
↳68])
>>> op = 'min'
>>> (subextrema_x, subextrema_y) = argsubextrema2(op, ydata, xdata, thresh_factor)
>>> result = str((subextrema_x, subextrema_y))
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.draw_hist_subbin_maxima(ydata, xdata)
>>> pt.show_if_requested()
```

Doctest:

```
>>> from vtool.histogram import * # NOQA
>>> thresh_factor = .8
>>> ydata = np.array([1, 1, 1, 2, 1, 2, 3, 2, 4, 1.1, 5, 1.2, 1.1, 1.1, 1.2,
↳1.1])
>>> op = 'max'
>>> thresh_factor = .8
>>> (subextrema_x, subextrema_y) = argsubextrema2(op, ydata, thresh_
↳factor=thresh_factor)
>>> result = str((subextrema_x, subextrema_y))
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.qensure()
>>> xdata = np.arange(len(ydata))
>>> pt.figure(fnum=1, doclf=True)
>>> pt.plot(xdata, ydata)
>>> pt.plot(subextrema_x, subextrema_y, 'o')
>>> ut.show_if_requested()
```

`vtool.argsubmax(ydata, xdata=None)`

Finds a single submaximum value to subindex accuracy. If `xdata` is not specified, `submax_x` is a fractional index. Otherwise, `submax_x` is sub-`xdata` (essentially doing the index interpolation for you)

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> import ubelt as ub
>>> ydata = [0, 1, 2, 1.5, 0]
>>> xdata = [00, 10, 20, 30, 40]
>>> result1 = argsubmax(ydata, xdata=None)
>>> result2 = argsubmax(ydata, xdata=xdata)
>>> result = ub.repr2([result1, result2], precision=4, nl=1, nobr=True)
>>> print(result)
```

(continues on next page)

(continued from previous page)

```
2.1667, 2.0208,
21.6667, 2.0208,
```

Example

```
>>> from vtool.histogram import * # NOQA
>>> hist_ = np.array([0, 1, 2, 3, 4])
>>> centers = None
>>> maxima_thresh=None
>>> argsubmax(hist_)
(4.0, 4.0)
```

`vtool.argsubmax2(ydata, xdata=None)`

Finds a single submaximum value to subindex accuracy. If xdata is not specified, submax_x is a fractional index. This version always normalizes x-coordinates.

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> import ubelt as ub
>>> ydata = [ 0, 1, 2, 1.5, 0]
>>> xdata = [00, 10, 20, 30, 40]
>>> result1 = argsubmax(ydata, xdata=None)
>>> result2 = argsubmax(ydata, xdata=xdata)
>>> result = ub.repr2([result1, result2], precision=4, nl=1, nobr=True)
>>> print(result)
2.1667, 2.0208,
21.6667, 2.0208,
```

Example

```
>>> from vtool.histogram import * # NOQA
>>> hist_ = np.array([0, 1, 2, 3, 4])
>>> centers = None
>>> thresh_factor = None
>>> argsubmax(hist_)
(4.0, 4.0)
```

`vtool.argsubmaxima(hist, centers=None, maxima_thresh=None, _debug=False)`

Determines approximate maxima values to subindex accuracy.

Parameters

- **hist_** (*ndarray*) – ydata, histogram frequencies
- **centers** (*ndarray*) – xdata, histogram labels
- **maxima_thresh** (*float*) – cutoff point for labeling a value as a maxima

Returns (submaxima_x, submaxima_y)

Return type `tuple`

CommandLine: python -m vtool.histogram argsubmaxima python -m vtool.histogram argsubmaxima --show

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> maxima_thresh = .8
>>> hist = np.array([6.73, 8.69, 0.00, 0.00, 34.62, 29.16, 0.00, 0.00, 6.73, 8.
↳69])
>>> centers = np.array([-0.39, 0.39, 1.18, 1.96, 2.75, 3.53, 4.32, 5.11, 5.89,
↳6.68])
>>> (submaxima_x, submaxima_y) = argsubmaxima(hist, centers, maxima_thresh)
>>> result = str((submaxima_x, submaxima_y))
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.draw_hist_subbin_maxima(hist, centers)
>>> pt.show_if_requested()
(array([ 3.0318792]), array([ 37.19208239]))
```

`vtool.argsubmaxima2(ydata, xdata=None, thresh_factor=None, normalize_x=True)`

`vtool.argsubmin2(ydata, xdata=None)`

`vtool.argsubminima2(ydata, xdata=None, thresh_factor=None, normalize_x=True)`

`vtool.assert_zipcompress(arr_list, flags_list, axis=None)`

`vtool.asserteq(output1, output2, thresh=1e-08, nestpath=None, level=0, lbl1=None, lbl2=None, out-
put_lbl=None, verbose=True, iswarning=False)`
recursive equality checks

asserts that output1 and output2 are close to equal.

`vtool.assign_symmetric_matches(fx2_to_fx1, fx2_to_dist, fx1_to_fx2, fx1_to_dist, K,
Knorm=None)`

Ignore:

```
>>> import vtool as vt
>>> from vtool.matching import *
>>> K = 2
>>> Knorm = 1
>>> feat1 = np.random.rand(5, 3)
>>> feat2 = np.random.rand(7, 3)
>>>
>>> # Assign distances
>>> distmat = vt.L2(feat1[:, None], feat2[None, :])
>>>
>>> # Find nearest K
>>> fx1_to_fx2 = distmat.argsort()[:, 0:K + Knorm]
>>> fx2_to_fx1 = distmat.T.argsort()[:, 0:K + Knorm]
>>> # and order their distances
>>> fx1_to_dist = np.array([distmat[i].take(col) for i, col in enumerate(fx1_
↳to_fx2)])
>>> fx2_to_dist = np.array([distmat.T[j].take(row) for j, row in
↳enumerate(fx2_to_fx1)])
>>>
>>> # flat_matx1 = fx1_to_fx2 + np.arange(distmat.shape[0])[:, None] *
↳distmat.shape[1]
```

(continues on next page)

(continued from previous page)

```

>>> # fx1_to_dist = distmat.take(flat_matx1).reshape(fx1_to_fx2.shape)
>>>
>>> fx21 = pd.DataFrame(fx2_to_fx1)
>>> fx21.columns.name = 'K'
>>> fx21.index.name = 'fx1'
>>>
>>> fx12 = pd.DataFrame(fx1_to_fx2)
>>> fx12.columns.name = 'K'
>>> fx12.index.name = 'fx2'
>>>
>>> fx12 = fx12.T[0:K].T.astype(np.float)
>>> fx21 = fx21.T[0:K].T.astype(np.float)
>>>
>>> fx12.values[~fx1_to_flags] = np.nan
>>> fx21.values[~fx2_to_flags] = np.nan
>>>
>>> print('fx12.values =\n%r' % (fx12,))
>>> print('fm_ =\n%r' % (fm_,))
>>>
>>> print('fx21.values =\n%r' % (fx21,))
>>> print('fm =\n%r' % (fm,))
>>>
>>> unflat_match_idx2 = -np.ones(fx2_to_fx1.shape)
>>> unflat_match_idx2.ravel()[flat_match_idx2] = flat_match_idx2
>>> inv_lookup21 = unflat_match_idx2.T[0:K].T
>>>
>>> for fx2 in zip(fx12.values[fx1_to_flags]:
>>>
>>> for fx1, fx2 in zip(match_fx1_, match_fx2_):
>>>     cx = np.where(fx2_to_fx1[fx2][0:K] == fx1)[0][0]
>>>     inv_idx = inv_lookup21[fx2][cx]
>>>     print('inv_idx = %r' % (inv_idx,))

```

`vtool.assign_to_centroids(dpts, qpts, num_neighbors=1, flann_params={})`

Helper for akmeans

`vtool.assign_unconstrained_matches(fx2_to_fx1, fx2_to_dist, K, Knorm=None, fx2_to_flags=None)`

assigns vsone matches using results of nearest neighbors.

Ignore: `fx2_to_dist = np.arange(fx2_to_fx1.size).reshape(fx2_to_fx1.shape)`

CommandLine: `python -m vtool.matching -test-assign_unconstrained_matches -show python -m vtool.matching assign_unconstrained_matches:0 python -m vtool.matching assign_unconstrained_matches:1`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.matching import * # NOQA
>>> fx2_to_fx1, fx2_to_dist = empty_neighbors(0, 0)
>>> K = 1
>>> Knorm = 1
>>> fx2_to_flags = None
>>> assigntup = assign_unconstrained_matches(fx2_to_fx1, fx2_to_dist, K,
>>>                                           Knorm, fx2_to_flags)

```

(continues on next page)

(continued from previous page)

```
>>> fm, match_dist, norm_fx1, norm_dist = assigntup
>>> result = ub.repr2(assigntup, precision=3, nobr=True, with_dtype=True)
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.matching import * # NOQA
>>> fx2_to_fx1 = np.array([[ 77, 971, 22],
>>>                        [116, 120, 34],
>>>                        [122, 128, 99],
>>>                        [1075, 692, 102],
>>>                        [ 530, 45, 120],
>>>                        [ 45, 530, 77]], dtype=np.int32)
>>> fx2_to_dist = np.array([[ 0.059, 0.238, .3],
>>>                          [ 0.021, 0.240, .4],
>>>                          [ 0.039, 0.247, .5],
>>>                          [ 0.149, 0.151, .6],
>>>                          [ 0.226, 0.244, .7],
>>>                          [ 0.215, 0.236, .8]], dtype=np.float32)
>>> K = 1
>>> Knorm = 1
>>> fx2_to_flags = np.array([[1, 1], [0, 1], [1, 1], [0, 1], [1, 1], [1, 1]])
>>> fx2_to_flags = fx2_to_flags[:, 0:K]
>>> assigntup = assign_unconstrained_matches(fx2_to_fx1, fx2_to_dist, K,
>>>                                         Knorm, fx2_to_flags)
>>> fm, match_dist, norm_fx1, norm_dist = assigntup
>>> result = ub.repr2(assigntup, precision=3, nobr=True, with_dtype=True)
>>> print(result)
>>> assert len(fm.shape) == 2 and fm.shape[1] == 2
>>> assert ub.allsame(list(map(len, assigntup)))
```

`vtool.asymmetric_correspondence` (*annot1, annot2, K, Knorm, checks, allow_shrink=True*)
Find symmetric feature correspondences

`vtool.atan2` (*y, x*)
does atan2 but returns from 0 to TAU

Example

```
>>> from vtool.trig import * # NOQA
>>> import utool
>>> rng = np.random.RandomState(0)
>>> y = rng.rand(1000).astype(np.float64)
>>> x = rng.rand(1000).astype(np.float64)
>>> theta = atan2(y, x)
>>> assert np.all(theta >= 0)
>>> assert np.all(theta < 2 * np.pi)
>>> import ubelt as ub
>>> assert ub.hash_data(theta) in [
>>>
→ '6bfc86a2e94dd2dafbf501035719a7873d57f5f8e9cde88c4ccc35e98bb9e7b82abf6230803a923be7060866d66b8
→ ',
>>>
→ '90fe55311562f1c3ae451d5c4f27573259fed96752a5bd03f0f1216b46cf5b4b40024dc714b5c0df7e6f8d6eb2a2
→ ',
```

(continues on next page)

(continued from previous page)

```
>>> ]
```

`vtool.atleast_3channels(arr, copy=True)`

Ensures that there are 3 channels in the image

Parameters

- **arr** (`ndarray[N, M, ...]`) – the image
- **copy** (`bool`) – Always copies if True, if False, then copies only when the size of the array must change.

Returns with shape (N, M, C), where C in {3, 4}**Return type** `ndarray`**CommandLine:** `python -m vtool.other atleast_3channels`**Doctest:**

```
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> assert atleast_3channels(np.zeros((10, 10))).shape[-1] == 3
>>> assert atleast_3channels(np.zeros((10, 10, 1))).shape[-1] == 3
>>> assert atleast_3channels(np.zeros((10, 10, 3))).shape[-1] == 3
>>> assert atleast_3channels(np.zeros((10, 10, 4))).shape[-1] == 4
```

`vtool.atleast_nd(arr, n, tofront=False)`

View inputs as arrays with at least n dimensions. TODO: Submit as a PR to numpy

Parameters

- **arr** (`array_like`) – One array-like object. Non-array inputs are converted to arrays. Arrays that already have n or more dimensions are preserved.
- **n** (`int`) – number of dimensions to ensure
- **tofront** (`bool`) – if True new dimensions are added to the front of the array. otherwise they are added to the back.

CommandLine: `python -m vtool.numpy_utils atleast_nd`**Returns** An array with `a.ndim >= n`. Copies are avoided where possible, and views with three or more dimensions are returned. For example, a 1-D array of shape `(N,)` becomes a view of shape `(1, N, 1)`, and a 2-D array of shape `(M, N)` becomes a view of shape `(M, N, 1)`.**Return type** `ndarray`**See also:**`ensure_shape`, `np.atleast_1d`, `np.atleast_2d`, `np.atleast_3d`**Example**

```
>>> # ENABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> import ubelt as ub
```

(continues on next page)

(continued from previous page)

```
>>> n = 2
>>> arr = np.array([1, 1, 1])
>>> arr_ = atleast_nd(arr, n)
>>> result = ub.repr2(arr_.tolist())
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> import ubelt as ub
>>> n = 4
>>> arr1 = [1, 1, 1]
>>> arr2 = np.array(0)
>>> arr3 = np.array([[[[1]]]])
>>> arr1_ = atleast_nd(arr1, n)
>>> arr2_ = atleast_nd(arr2, n)
>>> arr3_ = atleast_nd(arr3, n)
>>> result1 = ub.repr2(arr1_.tolist())
>>> result2 = ub.repr2(arr2_.tolist())
>>> result3 = ub.repr2(arr3_.tolist())
>>> result = '\n'.join([result1, result2, result3])
>>> print(result)
```

`vtool.atleast_nd(arr, n, tofront=False)`

View inputs as arrays with at least n dimensions. TODO: Submit as a PR to numpy

Parameters

- **arr** (*array_like*) – One array-like object. Non-array inputs are converted to arrays. Arrays that already have n or more dimensions are preserved.
- **n** (*int*) – number of dimensions to ensure
- **tofront** (*bool*) – if True new dimensions are added to the front of the array. otherwise they are added to the back.

CommandLine: `python -m vtool.numpy_utils atleast_nd`

Returns An array with `a.ndim >= n`. Copies are avoided where possible, and views with three or more dimensions are returned. For example, a 1-D array of shape `(N,)` becomes a view of shape `(1, N, 1)`, and a 2-D array of shape `(M, N)` becomes a view of shape `(M, N, 1)`.

Return type ndarray

See also:

`ensure_shape`, `np.atleast_1d`, `np.atleast_2d`, `np.atleast_3d`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> import ubelt as ub
```

(continues on next page)

(continued from previous page)

```
>>> n = 2
>>> arr = np.array([1, 1, 1])
>>> arr_ = atleast_nd(arr, n)
>>> result = ub.repr2(arr_.tolist())
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> import ubelt as ub
>>> n = 4
>>> arr1 = [1, 1, 1]
>>> arr2 = np.array(0)
>>> arr3 = np.array([[[[1]]]])
>>> arr1_ = atleast_nd(arr1, n)
>>> arr2_ = atleast_nd(arr2, n)
>>> arr3_ = atleast_nd(arr3, n)
>>> result1 = ub.repr2(arr1_.tolist())
>>> result2 = ub.repr2(arr2_.tolist())
>>> result3 = ub.repr2(arr3_.tolist())
>>> result = '\n'.join([result1, result2, result3])
>>> print(result)
```

`vtool.atleast_shape(arr, dimshape)`

Ensures that an array takes a certain shape. The total size of the array must not change.

Parameters

- **arr** (*ndarray*) – array to change the shape of
- **dimshape** (*tuple*) – desired shape (Nones can be used to broadcast dimensions)

Returns *ndarray* - the input array, which has been modified inplace.

CommandLine: `python -m vtool.other ensure_shape`

Doctest:

```
>>> from vtool.other import * # NOQA
>>> arr = np.zeros((7, 7))
>>> assert atleast_shape(arr, (1, 1, 3,)).shape == (7, 7, 3)
>>> assert atleast_shape(arr, (1, 1, 2, 4,)).shape == (7, 7, 2, 4)
>>> assert atleast_shape(arr, (1, 1,)).shape == (7, 7,)
>>> assert atleast_shape(arr, (1, 1, 1)).shape == (7, 7, 1)
>>> assert atleast_shape(np.zeros(()), (1,)).shape == (1,)
>>> assert atleast_shape(np.zeros(()), tuple()).shape == tuple()
>>> assert atleast_shape(np.zeros(()), (1, 2, 3,)).shape == (1, 2, 3)
>>> ut.assert_raises(ValueError, atleast_shape, arr, (2, 2))
>>> assert atleast_shape(np.zeros((7, 7, 3)), (1, 1, 3)).shape == (7, 7, 3)
>>> ut.assert_raises(ValueError, atleast_shape, np.zeros((7, 7, 3)), (1, 1, 4))
```

`vtool.augment_2x2_with_translation(kpts, _mat2x2)`

helper function to augment shape matrix with a translation component.

`vtool.bar_L2_sift(hist1, hist2)`
Normalized SIFT L2

Parameters

- **hist1** (*ndarray*) – Nx128 array of uint8 with pseudomax trick
- **hist2** (*ndarray*) – Nx128 array of uint8 with pseudomax trick

CommandLine: `python -m vtool.distance --test-bar_L2_sift`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> hist1, hist2 = testdata_hist()
>>> barl2_dist = bar_L2_sift(hist1, hist2)
>>> result = ub.repr2(barl2_dist, precision=2)
```

`vtool.bar_cos_sift(hist1, hist2)`
1 - cos dist

`vtool.bbox_center(bbox)`

`vtool.bbox_from_center_wh(center_xy, wh)`

`vtool.bbox_from_extent(extent)`

Parameters **extent** (*ndarray*) – tl_x, br_x, tl_y, br_y

Returns tl_x, tl_y, w, h

Return type `bbox` (*ndarray*)

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> import ubelt as ub
>>> extent = [0, 10, 0, 10]
>>> bbox = bbox_from_extent(extent)
>>> result = ('bbox = %s' % (ub.repr2(bbox, nl=0),))
>>> print(result)
bbox = [0, 0, 10, 10]
```

`vtool.bbox_from_verts(verts, castint=False)`

`vtool.bbox_from_xywh(xy, wh, xy_rel_pos=[0, 0])`
need to specify `xy_rel_pos` if `xy` is not in tl already

`vtool.bboxes_from_vert_list(verts_list, castint=False)`
Fit the bounding polygon inside a rectangle

`vtool.beaton_tukey_loss(u, a=1)`

CommandLine: `python -m wbia.plottool.draw_func2 --exec-plot_func --show --range=-8,8 --func=vt.beaton_tukey_weight,vt.beaton_tukey_loss`

References

Steward_Robust%20parameter%20estimation%20in%20computer%20vision.pdf

`vtool.beaton_tukey_weight(u, a=1)`

CommandLine: `python -m wbia.plottool.draw_func2 --exec-plot_func --show --range=-8,8 --func=vt.beaton_tukey_weight`

References

Steward_Robust%20parameter%20estimation%20in%20computer%20vision.pdf

`vtool.blend_images(img1, img2, mode='average', **kwargs)`

Parameters

- **img1** (`np.ndarray`) – first image
- **img2** (`np.ndarray`) – second image
- **mode** (`str`) – can be average, multiply, or overlay

`vtool.blend_images_average(img1, img2, alpha=0.5)`

Parameters

- **img1** (`ndarray[uint8_t, ndim=2]`) – image data
- **img2** (`ndarray[uint8_t, ndim=2]`) – image data
- **alpha** (`float`) – (default = 0.5)

Returns `imgB`

Return type `ndarray`

References

https://en.wikipedia.org/wiki/Blend_modes

CommandLine: `python -m vtool.blend blend_images_average:0 --show python -m vtool.blend blend_images_average:1 --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.blend import * # NOQA
>>> alpha = 0.8
>>> img1, img2 = testdata_blend()
>>> imgB = blend_images_average(img1, img2, alpha)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(imgB)
>>> ut.show_if_requested()
```

Ignore:

```
>>> # GRIDSEARCH
>>> from vtool.blend import * # NOQA
>>> test_func = blend_images_average
>>> args = testdata_blend()
>>> param_info = ut.ParamInfoList('blend_params', [
...     ut.ParamInfo('alpha', .8, 'alpha=',
...                   varyvals=np.linspace(0, 1.0, 25).tolist()),
... ])
>>> gridsearch_image_function(param_info, test_func, args)
>>> ut.show_if_requested()
```

`vtool.blend_images_average_stack(images, alpha=None)`

Parameters

- **img1** (`ndarray[uint8_t, ndim=2]`) – image data
- **img2** (`ndarray[uint8_t, ndim=2]`) – image data
- **alpha** (`float`) – (default = 0.5)

Returns `imgB`

Return type `ndarray`

References

https://en.wikipedia.org/wiki/Blend_modes

CommandLine: `python -m vtool.blend --test-blend_images_average:0 --show python -m vtool.blend --test-blend_images_average:1 --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.blend import * # NOQA
>>> alpha = 0.8
>>> img1, img2 = testdata_blend()
>>> imgB = blend_images_average(img1, img2, alpha)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(imgB)
>>> ut.show_if_requested()
```

`vtool.blend_images_mult_average(img1, img2, alpha=0.5)`

Parameters

- **img1** (`ndarray[uint8_t, ndim=2]`) – image data
- **img2** (`ndarray[uint8_t, ndim=2]`) – image data
- **alpha** (`float`) – (default = 0.5)

Returns `imgB`

Return type `ndarray`

References

https://en.wikipedia.org/wiki/Blend_modes

CommandLine: python -m vtool.blend -test-blend_images_mult_average:0 -show python -m vtool.blend -test-blend_images_mult_average:1 -show

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.blend import * # NOQA
>>> alpha = 0.8
>>> img1, img2 = testdata_blend()
>>> imgB = blend_images_mult_average(img1, img2, alpha)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(imgB)
>>> ut.show_if_requested()
```

Ignore:

```
>>> # GRIDSEARCH
>>> from vtool.blend import * # NOQA
>>> test_func = blend_images_mult_average
>>> args = testdata_blend()
>>> param_info = ut.ParamInfoList('blend_params', [
...     ut.ParamInfo('alpha', .8, 'alpha=',
...                   varyvals=np.linspace(0, 1.0, 9).tolist()),
... ])
>>> gridsearch_image_function(param_info, test_func, args)
>>> ut.show_if_requested()
```

`vtool.blend_images_multiply(img1, img2, alpha=0.5)`

Parameters

- **img1** (`ndarray[uint8_t, ndim=2]`) – image data
- **img2** (`ndarray[uint8_t, ndim=2]`) – image data
- **alpha** (`float`) – (default = 0.5)

Returns `imgB`

Return type `ndarray`

References

https://en.wikipedia.org/wiki/Blend_modes

CommandLine: python -m vtool.blend -test-blend_images_multiply:0 -show python -m vtool.blend -test-blend_images_multiply:1 -show

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.blend import * # NOQA
>>> alpha = 0.8
>>> img1, img2 = testdata_blend()
>>> imgB = blend_images_multiply(img1, img2, alpha)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(imgB)
>>> ut.show_if_requested()
```

Ignore:

```
>>> # GRIDSEARCH
>>> from vtool.blend import * # NOQA
>>> test_func = blend_images_multiply
>>> args = testdata_blend(scale=128)
>>> param_info = ut.ParamInfoList('blend_params', [
...     ut.ParamInfo('alpha', .8, 'alpha=',
...                 varyvals=np.linspace(0, 1.0, 9).tolist()),
... ])
>>> gridsearch_image_function(param_info, test_func, args)
>>> ut.show_if_requested()
```

`vtool.breakup_equal_streak(arr_in, left_endpoint=None, right_endpoint=None)`

Breaks up streaks of equal values by interpolating between the next lowest and next highest value

Parameters

- **arr_in** –
- **left_endpoint** (*None*) – (default = None)
- **right_endpoint** (*None*) – (default = None)

Returns arr -

Return type ndarray

CommandLine: `python -m vtool.util_math --exec-breakup_equal_streak python -m vtool.util_math --test-ensure_monotone_strictly_increasing --show --offset=0`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> arr_in = np.array([0, 0, 1, 1, 2, 2], dtype=np.float32)
>>> arr_in = np.array([ 1.20488135,  1.2529297 ,  1.27306686,  1.29859663,
>>>    1.31769871,  1.37102388,  1.38114004,  1.45732054,  1.48119571,  1.48119571,
>>>    1.5381895 ,  1.54162741,  1.57492901,  1.61129523,  1.61129523,
>>>    1.61270343,  1.63377551,  1.7423034 ,  1.76364247,  1.79908459,
>>>    1.83564709,  1.83819742,  1.83819742,  1.86786967,  1.86786967,
>>>    1.90720142,  1.90720142,  1.92293973,  1.92293973, ]) / 2
>>> left_endpoint = 0
>>> right_endpoint = 1.0
>>> arr = breakup_equal_streak(arr_in, left_endpoint, right_endpoint)
```

(continues on next page)

(continued from previous page)

```
>>> assert strictly_increasing(arr)
>>> result = ('arr = %s' % (str(arr),))
>>> print(result)
```

`vtool.build_affine_lstsqrs_Mx6(xy1_man, xy2_man)`

CURRENTLY NOT WORKING

CommandLine: `python -m vtool.spatial_verification --test-build_affine_lstsqrs_Mx6`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.demodata as demodata
>>> kpts1, kpts2 = demodata.get_dummy_kpts_pair()
>>> xy1_man = ktool.get_xys(kpts1).astype(np.float64)
>>> xy2_man = ktool.get_xys(kpts2).astype(np.float64)
>>> Mx6 = build_affine_lstsqrs_Mx6(xy1_man, xy2_man)
>>> import ubelt as ub
>>> print(ub.repr2(Mx6))
>>> result = ut.hashstr(Mx6)
>>> print(result)
```

Ignore:

```
>>> import sympy as sym
>>> x1, y1, x2, y2 = sym.symbols('x1, y1, x2, y2')
>>> A = sym.Matrix([
>>>     [x1, y1, 0, 0, 1, 0],
>>>     [0, 0, x1, y1, 0, 1],
>>> ])
>>> b = sym.Matrix([[x2], [y2]])
>>> x = (A.T.multiply(A)).inv().multiply(A.T.multiply(b))
>>> x = (A.T.multiply(A)).pinv().multiply(A.T.multiply(b))
```

References

<https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf> page 22

`vtool.build_lstsqrs_Mx9(xy1_mn, xy2_mn)`

Builds the M x 9 least squares matrix

CommandLine: `python -m vtool.spatial_verification --test-build_lstsqrs_Mx9`

Example

```
>>> # DISABLE_DOCTEST
>>> # xdoctest: +SKIP
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.demodata as demodata
>>> kpts1, kpts2 = demodata.get_dummy_kpts_pair()
>>> xy1_mn = ktool.get_xys(kpts1).astype(np.float64)
```

(continues on next page)

(continued from previous page)

```

>>> xy2_mn = ktool.get_xys(kpts2).astype(np.float64)
>>> Mx9 = build_lstsqrs_Mx9(xy1_mn, xy2_mn)
>>> import ubelt as ub
>>> result = (ub.repr2(Mx9[0:2], suppress_small=True, precision=2, with_
↳dtype=True))
>>> print(result)
np.array([[ 0.00e+00,  0.00e+00,  0.00e+00, -3.20e+01, -2.72e+01,
          -1.00e+00,  8.82e+02,  7.49e+02,  2.76e+01],
         [ 3.20e+01,  2.72e+01,  1.00e+00,  0.00e+00,  0.00e+00,
          0.00e+00, -1.09e+03, -9.28e+02, -3.42e+01]], dtype=np.float64)

```

References

http://dip.sun.ac.za/~stefan/TW793/attach/notes/homography_estimation.pdf http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf Page 317 <http://vision.ece.ucsb.edu/~zuliani/Research/RANSAC/docs/RANSAC4Dummies.pdf> page 53

vtool.calc_error_bars_from_sample (*sample_size, num_positive, pop, conf_level=0.95*)
Determines a error bars of sample

References

<https://www.qualtrics.com/blog/determining-sample-size/> <http://www.surveysystem.com/sscalc.htm> https://en.wikipedia.org/wiki/Sample_size_determination <http://www.surveysystem.com/sample-size-formula.htm> http://courses.wcupa.edu/rbove/Berenson/10th%20ed%20CD-ROM%20topics/section8_7.pdf https://en.wikipedia.org/wiki/Standard_normal_table <https://www.unc.edu/~rls/s151-2010/class23.pdf>

vtool.calc_sample_from_error_bars (*err_frac, pop, conf_level=0.95, prior=0.5*)
Determines a reasonable sample size to achieve desired error bars.

```

import sympy p, n, N, z = sympy.symbols('prior, ss, pop, zval') me = sympy.symbols('err_frac') expr = (z *
sympy.sqrt((p * (1 - p) / n) * ((N - n) / (N - 1)))) equation = sympy.Eq(me, expr) nexpr = sympy.solve(equation,
[n])[0] nexpr = sympy.simplify(nexpr)

```

```

import autopep8 print(autopep8.fix_lines(['ss = ' + str(nexpr)], autopep8._get_options({ }, False)))

```

```

ss = -pop * prior* (zval**2) (prior - 1) / ((err_frac * 2) * pop - (err_frac**2) - prior * (zval**2) * (prior - 1)) ss
= pop * prior * zval ** 2 * (prior - 1) / (-err_frac ** 2 * pop + err_frac ** 2 + prior * zval ** 2 * (prior - 1))

```

vtool.cast_split (*kpts, dtype=<class 'numpy.float32'>*)
breakup keypoints into location, shape, and orientation

vtool.check_exif_keys (*pil_img*)

vtool.check_expr_eq (*expr1, expr2, verbose=True*)
Does not work in general. Problem is not decidable. Thanks Richard.

Parameters

- **expr1** –
- **expr2** –

CommandLine: python -m vtool.symbolic --test-check_expr_eq

SeeAlso: vt.symbolic_randcheck

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.symbolic import * # NOQA
>>> expr1 = sympy.Matrix([ [sx*x + 1.0*tx + w1*y], [sy*y + 1.0*ty + w2*x], [1.0]])
>>> expr2 = sympy.Matrix([ [sx*x + tx + w1*y], [sy*y + ty + w2*x], [1]])
>>> result = check_expr_eq(expr1, expr2)
>>> print(result)
```

`vtool.check_kpts_in_bounds(kpts_, width, height)`

`vtool.check_sift_validity(sift_uint8, lbl=None, verbose=True)`
checks if a SIFT descriptor is valid

`vtool.check_unused_kwargs(kwargs, expected_keys)`

`vtool.circular_distance(arr=None)`

`vtool.clipnorm(arr, min_, max_, out=None)`
normalizes arr to the range 0 to 1 using min_ and max_ as clipping bounds

`vtool.clipwhite(img)`
Strips white borders off an image

`vtool.clipwhite_ondisk(fpath_in, fpath_out=None, verbose=True)`
Strips white borders off an image on disk

Parameters

- `fpath_in` (*str*) –
- `fpath_out` (*None*) – (default = None)
- `verbose` (*bool*) – verbosity flag (default = True)

Returns `fpath_out`

Return type `str`

CommandLine: `python -m vtool.image clipwhite_ondisk`

`vtool.closest_point(pt, pt_arr, distfunc=<function L2_sqrd>)`
finds the nearest point(s) in pts to (x, y) `pt = np.array([1])` `pt_arr = np.array([1.1, 2, .95, 20])[:, None]` `distfunc = vt.L2_sqrd`

`vtool.closest_point_on_bbox(p, bbox)`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> p_list = np.array([[19, 7], [7, 14], [14, 11], [8, 7], [23, 21]], dtype=np.
    ↪ float)
>>> bbox = np.array([10, 10, 10, 10], dtype=np.float)
>>> [closest_point_on_bbox(p, bbox) for p in p_list]
```

`vtool.closest_point_on_line(p, e1, e2)`
`e1` and `e2` define two points on the line. Does not clip to the segment.

CommandLine: `python -m vtool.geometry closest_point_on_line --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> import vtool as vt
>>> verts = np.array([[ 21.83012702, 13.16987298],
>>>                    [ 16.83012702, 21.83012702],
>>>                    [  8.16987298, 16.83012702],
>>>                    [ 13.16987298,  8.16987298],
>>>                    [ 21.83012702, 13.16987298]])
>>> rng = np.random.RandomState(0)
>>> p_list = rng.rand(64, 2) * 20 + 5
>>> close_pts = []
>>> for p in p_list:
>>>     candidates = [closest_point_on_line(p, e1, e2) for e1, e2 in ut.
→itertwo(verts)]
>>>     dists = np.array([vt.L2_sqrd(p, new_pt) for new_pt in candidates])
>>>     close_pts.append(candidates[dists.argmin()])
>>> close_pts = np.array(close_pts)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.ensureqt()
>>> pt.plt.plot(p_list.T[0], p_list.T[1], 'ro', label='original point')
>>> pt.plt.plot(close_pts.T[0], close_pts.T[1], 'rx', label='closest point on_
→shape')
>>> for x, y in list(zip(p_list, close_pts)):
>>>     z = np.array(list(zip(x, y)))
>>>     pt.plt.plot(z[0], z[1], 'r--')
>>> pt.plt.legend()
>>> pt.plt.plot(verts.T[0], verts.T[1], 'b-')
>>> pt.plt.xlim(0, 30)
>>> pt.plt.ylim(0, 30)
>>> pt.plt.axis('equal')
>>> ut.show_if_requested()
```

`vtool.closest_point_on_line_segment(p, e1, e2)`

Finds the closet point from p on line segment (e1, e2)

Parameters

- **p** (*ndarray*) – and xy point
- **e1** (*ndarray*) – the first xy endpoint of the segment
- **e2** (*ndarray*) – the second xy endpoint of the segment

Returns `pt_on_seg` - the closest xy point on (e1, e2) from p

Return type `ndarray`

References

http://en.wikipedia.org/wiki/Distance_from_a_point_to_a_line <http://stackoverflow.com/questions/849211/shortest-distance-between-a-point-and-a-line-segment>

CommandLine: `python -m vtool.geometry --exec-closest_point_on_line_segment --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> import vtool as vt
>>> #bbox = np.array([10, 10, 10, 10], dtype=np.float)
>>> #verts_ = np.array(vt.verts_from_bbox(bbox, close=True))
>>> #R = vt.rotation_around_bbox_mat3x3(vt.TAU / 3, bbox)
>>> #verts = vt.transform_points_with_homography(R, verts_.T).T
>>> verts = np.array([[ 21.83012702,  13.16987298],
>>>                    [ 16.83012702,  21.83012702],
>>>                    [  8.16987298,  16.83012702],
>>>                    [ 13.16987298,  8.16987298],
>>>                    [ 21.83012702,  13.16987298]])
>>> rng = np.random.RandomState(0)
>>> p_list = rng.rand(64, 2) * 20 + 5
>>> close_pts = np.array([closest_point_on_vert_segments(p, verts) for p in p_
↪list])
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.ensureqt()
>>> pt.plt.plot(p_list.T[0], p_list.T[1], 'ro', label='original point')
>>> pt.plt.plot(close_pts.T[0], close_pts.T[1], 'rx', label='closest point on_
↪shape')
>>> for x, y in list(zip(p_list, close_pts)):
>>>     z = np.array(list(zip(x, y)))
>>>     pt.plt.plot(z[0], z[1], 'r--')
>>> pt.plt.legend()
>>> pt.plt.plot(verts.T[0], verts.T[1], 'b-')
>>> pt.plt.xlim(0, 30)
>>> pt.plt.ylim(0, 30)
>>> pt.plt.axis('equal')
>>> ut.show_if_requested()
```

`vtool.closest_point_on_vert_segments(p, verts)`

`vtool.colwise_operation(arr1, arr2, op)`

`vtool.combine_offset_lists(offsets_list, sfs_list, offset_tups, sf_tups)`
 Helper for stacking

`vtool.compare_implementations(func1, func2, args, show_output=False, lbl1="", lbl2="", out-
 put_lbl=None)`
 tests two different implementations of the same function

`vtool.compare_matrix_columns(matrix, columns, comp_op=<ufunc 'equal'>, logic_op=<ufunc
 'logical_or'>)`

REPLACE WITH: `qfx2_invalid = logic_op.reduce([comp_op[:, None], qfx2_normnid] for col1 in
 qfx2_topnid.T])`

`vtool.compare_matrix_to_rows(row_matrix, row_list, comp_op=<ufunc 'equal'>,
 logic_op=<ufunc 'logical_or'>)`

Compares each row in row_list to each row in row matrix using comp_op Both must have the same number of
 columns. Performs logic_op on the results of each individual row

SeeAlso: `wbia.algo.hots.nn_weights.mark_name_valid_normalizers`

`compop = np.equal logic_op = np.logical_or`

`vtool.componentwise_dot(arr1, arr2)`

a dot product is a componentwise multiplication of two vector and then a sum.

Parameters

- **arr1** (*ndarray*) –
- **arr2** (*ndarray*) –

Returns cosangle

Return type ndarray

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> np.random.seed(0)
>>> arr1 = np.random.rand(3, 128)
>>> arr1 = arr1 / np.linalg.norm(arr1, axis=1)[:, None]
>>> arr2 = arr1
>>> cosangle = componentwise_dot(arr1, arr2)
>>> result = str(cosangle)
>>> print(result)
[ 1.  1.  1.]
```

`vtool.compress2(arr, flag_list, axis=None, out=None)`

Wrapper around numpy compress that makes the signature more similar to take

`vtool.compute_affine(xyl_man, xy2_man)`

Parameters

- **xy1_mn** (*ndarray* [*ndim=2*]) – xy points in image1
- **xy2_mn** (*ndarray* [*ndim=2*]) – corresponding xy points in image 2

Returns A - affine matrix

Return type ndarray[shape=(3,3)]

CommandLine: `python -m vtool.spatial_verification --test-compute_affine:1 --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.demodata as demodata
>>> import vtool.keypoint as ktool
>>> kpts1, kpts2 = demodata.get_dummy_kpts_pair()
>>> xyl_mn = ktool.get_xys(kpts1)
>>> xy2_mn = ktool.get_xys(kpts2)
>>> A = compute_affine(xyl_mn, xy2_mn)
>>> result = str(A)
>>> result = np.array_str(A, precision=2)
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.demodata as demodata
>>> import vtool.keypoint as ktool
>>> import wbia.plottool as pt
>>> xyl_man, xy2_man, rchip1, rchip2, T1, T2 = testdata_matching_affine_inliers_
↳normalized()
>>> A_prime = compute_affine(xyl_man, xy2_man)
>>> A = npl.solve(T2, A_prime).dot(T1)
>>> A /= A[2, 2]
>>> result = np.array_str(A, precision=2)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> rchip2_blendA = pt.draw_sv.get_blended_chip(rchip1, rchip2, A)
>>> pt.imshow(rchip2_blendA)
>>> ut.show_if_requested()
[[ 1.19e+00 -1.06e-02 -4.49e+01]
 [ -2.22e-01  1.12e+00 -2.78e+01]
 [ 0.00e+00  0.00e+00  1.00e+00]]
```

`vtool.compute_chip(gfpath, bbox, theta, new_size, filter_list=[], interpolation=4)`
 Extracts a chip and applies filters

DEPRICATE

Parameters

- **gfpath** (*str*) – image file path string
- **bbox** (*tuple*) – bounding box in the format (x, y, w, h)
- **theta** (*float*) – angle in radians
- **new_size** (*tuple*) – must maintain the same aspect ratio or else you will get weirdness
- **filter_list** (*list*) –

Returns chipBGR - cropped image

Return type ndarray

CommandLine: `python -m vtool.chip --test-compute_chip --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.chip import * # NOQA
>>> from vtool.util_math import TAU
>>> # build test data
>>> gfpath = ut.grab_test_imgpath('car1.jpg')
>>> bbox = (100, 3, 100, 100)
>>> theta = TAU / 8
>>> new_size = (32, 32)
>>> filter_list = []
>>> # execute function
>>> chipBGR = compute_chip(gfpath, bbox, theta, new_size, filter_list)
>>> # verify results
```

(continues on next page)

(continued from previous page)

```

>>> assert chipBGR.shape[0:2] == new_size[::-1], 'did not resize correctly'
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> import vtool as vt
>>> pt.imshow(vt.draw_verts(vt.imread(gfpath), vt.scaled_verts_from_bbox(bbox,
↳ theta, 1, 1)), pnum=(1, 2, 1))
>>> pt.imshow(chipBGR, pnum=(1, 2, 2))
>>> pt.show_if_requested()

```

`vtool.compute_distances` (*hist1*, *hist2*, *dist_list*=['L1', 'L2'])

Parameters

- **hist1** (*ndarray*) –
- **hist2** (*ndarray*) –
- **dist_list** (*list*) – (default = ['L1', 'L2'])

Returns *dist_dict*

Return type *dict*

CommandLine: `python -m vtool.distance --test-compute_distances`

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> hist1 = np.array([[1, 2], [2, 1], [0, 0]])
>>> hist2 = np.array([[1, 2], [3, 1], [2, 2]])
>>> dist_list = ['L1', 'L2']
>>> dist_dict = compute_distances(hist1, hist2, dist_list)
>>> result = ub.repr2(dist_dict, precision=3)
>>> print(result)

```

`vtool.compute_homog` (*xy1_mn*, *xy2_mn*)

Generate 6 degrees of freedom homography transformation Computes homography from normalized (0 to 1) point correspondences from 2 -> 1 (database->query)

Parameters

- **xy1_mn** (*ndarray* [*ndim*=2]) – xy points in image1
- **xy2_mn** (*ndarray* [*ndim*=2]) – corresponding xy points in image 2

Returns *H* - homography matrix

Return type *ndarray*[*shape*=(3,3)]

CommandLine: `python -m vtool.spatial_verification --test-compute_homog:1 --show`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.keypoint as ktool

```

(continues on next page)

(continued from previous page)

```

>>> import vtool.demodata as demodata
>>> kpts1, kpts2 = demodata.get_dummy_kpts_pair()
>>> xy1_mn = ktool.get_xys(kpts1)
>>> xy2_mn = ktool.get_xys(kpts2)
>>> H = compute_homog(xy1_mn, xy2_mn)
>>> #result = ut.hashstr(H)
>>> result = np.array_str(H, precision=2)
>>> print(result)
[[ 1.83e-03  2.85e-03 -7.11e-01]
 [ 2.82e-03  1.80e-03 -7.03e-01]
 [ 1.67e-05  1.68e-05 -5.53e-03]]

```

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.keypoint as ktool
>>> import wbia.plottool as pt
>>> xy1_man, xy2_man, rchip1, rchip2, T1, T2 = testdata_matching_affine_inliers_
↳ normalized()
>>> H_prime = compute_homog(xy1_man, xy2_man)
>>> H = npl.solve(T2, H_prime).dot(T1)
>>> H /= H[2, 2]
>>> result = np.array_str(H, precision=2)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> rchip2_blendH = pt.draw_sv.get_blended_chip(rchip1, rchip2, H)
>>> pt.imshow(rchip2_blendH)
>>> ut.show_if_requested()
[[ 9.22e-01 -2.50e-01  2.75e+01]
 [ -2.04e-01  8.79e-01 -7.94e+00]
 [ -1.82e-04 -5.99e-04  1.00e+00]]

```

```

vtool.compute_ndarray_unique_rowids_unsafe(arr)
arr = np.random.randint(2, size=(10000, 10)) vt.compute_unique_data_ids_(list(map(tuple, arr)))
len(vt.compute_unique_data_ids_(list(map(tuple, arr)))) len(np.unique(vt.compute_unique_data_ids_(list(map(tuple,
arr))))))

```

```

%timeit vt.compute_unique_data_ids_(list(map(tuple, arr))) %timeit com-
pute_ndarray_unique_rowids_unsafe(arr)

```

```

vtool.compute_unique_arr_dataids(arr)
specialized version for speed when arr is an ndarray

```

```

vtool.compute_unique_data_ids(data)
This is actually faster than compute_unique_integer_data_ids it seems

```

CommandLine: python -m vtool.other --test-compute_unique_data_ids

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> data = np.array([[0, 0], [0, 1], [1, 0], [1, 1], [0, 0], [.534, .432], [.534,
↳ .432], [1, 0], [0, 1]])

```

(continues on next page)

(continued from previous page)

```

>>> dataid_list = compute_unique_data_ids(data)
>>> result = 'dataid_list = ' + ub.repr2(dataid_list, with_dtype=True)
>>> print(result)
dataid_list = np.array([0, 1, 2, 3, 0, 4, 4, 2, 1], dtype=np.int32)

```

`vtool.compute_unique_data_ids_(hashable_rows, iddict_=None)`

`vtool.compute_unique_integer_data_ids(data)`

This is actually slower than `compute_unique_data_ids` it seems

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> # build test data
>>> data = np.array([[0, 0], [0, 1], [1, 1], [0, 0], [0, 0], [0, 1], [1, 1], [0, 0], [9, 0]])
>>> data = np.random.randint(1000, size=(1000, 2))
>>> # execute function
>>> result1 = compute_unique_data_ids(data)
>>> result2 = compute_unique_integer_data_ids(data)
>>> # verify results
>>> print(result)

```

`%timeit compute_unique_data_ids(data) %timeit compute_unique_integer_data_ids(data)`

`vtool.convert_colorspace(img, colorspace, src_colorspace='BGR')`

Converts colorspace of `img`. Convenience function around `cv2.cvtColor`

Parameters

- **img** (`ndarray[uint8_t, ndim=2]`) – image data
- **colorspace** (`str`) – RGB, LAB, etc
- **src_colorspace** (`unicode`) – (default = u'BGR')

Returns `img` - image data

Return type `ndarray[uint8_t, ndim=2]`

CommandLine: `python -m vtool.image convert_colorspace --show`

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> img_fpath = ut.grab_test_imgpath('zebra.png')
>>> img_fpath = ut.grab_file_url('http://itsnasb.com/wp-content/uploads/2013/03/lisa-frank-logol.jpg')
>>> img_fpath = ut.grab_test_imgpath('carl.jpg')
>>> img = vt.imread(img_fpath)
>>> img_float = vt.rectify_to_float01(img, np.float32)
>>> colorspace = 'LAB'
>>> src_colorspace = 'BGR'

```

(continues on next page)

(continued from previous page)

```

>>> imgLAB = convert_colorspace(img, colorspace, src_colorspace)
>>> imgL = imgLAB[:, :, 0]
>>> fillL = imgL.mean()
>>> fillAB = 0 if ut.is_float(img) else 128
>>> imgAB_LAB = vt.embed_channels(imgLAB[:, :, 1:3], (1, 2), fill=fillL)
>>> imgA_LAB = vt.embed_channels(imgLAB[:, :, 1], (1,), fill=(fillL, fillAB))
>>> imgB_LAB = vt.embed_channels(imgLAB[:, :, 2], (2,), fill=(fillL, fillAB))
>>> imgAB_BGR = convert_colorspace(imgAB_LAB, src_colorspace, colorspace)
>>> imgA_BGR = convert_colorspace(imgA_LAB, src_colorspace, colorspace)
>>> imgB_BGR = convert_colorspace(imgB_LAB, src_colorspace, colorspace)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> #imgAB_HSV = convert_colorspace(convert_colorspace(imgAB_LAB, 'LAB', 'BGR'),
↳ 'BGR', 'HSV')
>>> imgAB_HSV = convert_colorspace(img, 'HSV', 'BGR')
>>> imgAB_HSV[:, :, 1:3] = .6 if ut.is_float(img) else 128
>>> imgCOLOR_BRG = convert_colorspace(imgAB_HSV, 'BGR', 'HSV')
>>> pt.imshow(img, pnum=(3, 4, 1), title='input')
>>> pt.imshow(imgL, pnum=(3, 4, 2), title='L (lightness)')
>>> pt.imshow((imgLAB[:, :, 1]), pnum=(3, 4, 3), title='A (grayscale)')
>>> pt.imshow((imgLAB[:, :, 2]), pnum=(3, 4, 4), title='B (grayscale)')
>>> pt.imshow(imgCOLOR_BRG, pnum=(3, 4, 5), title='Hue')
>>> pt.imshow(imgAB_BGR, pnum=(3, 4, 6), title='A+B (color overlay)')
>>> pt.imshow(imgA_BGR, pnum=(3, 4, 7), title='A (Red-Green)')
>>> pt.imshow(imgB_BGR, pnum=(3, 4, 8), title='B (Blue-Yellow)')
>>> rgbblind_LAB = vt.embed_channels(imgLAB[:, :, (0, 2)], (0, 2), fill=fillAB)
>>> rgbblind_BRG = convert_colorspace(rgbblind_LAB, src_colorspace, colorspace)
>>> byblind_LAB = vt.embed_channels(imgLAB[:, :, (0, 1)], (0, 1), fill=fillAB)
>>> byblind_BGR = convert_colorspace(byblind_LAB, src_colorspace, colorspace)
>>> pt.imshow(byblind_BGR, title='colorblind B-Y', pnum=(3, 4, 11))
>>> pt.imshow(rgbblind_BRG, title='colorblind R-G', pnum=(3, 4, 12))
>>> ut.show_if_requested()

```

vtool.convert_degrees (*value*)

Helper function to convert the GPS coordinates stored in the EXIF to degrees in float format

References

http://en.wikipedia.org/wiki/Geographic_coordinate_conversion

vtool.convert_image_list_colorspace (*image_list*, *colorspace*, *src_colorspace*='BGR')

converts a list of images from <src_colorspace> to <colorspace>

vtool.convert_kptsZ_to_kpts (*kpts_Z*)

Convert keypoints in Z format to invV format

vtool.cos_sift (*hist1*, *hist2*)

cos dist

CommandLine: python -m vtool.distance --test-cos_sift

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA

```

(continues on next page)

(continued from previous page)

```
>>> hist1, hist2 = testdata_hist()
>>> l2_dist = cos_sift(hist1, hist2)
```

`vtool.cosine_dist(hist1, hist2)`

`vtool.crop_out_imgfill(img, fillval=None, thresh=0, channel=None)`

Crops image to remove fillval

Parameters

- **img** (`ndarray[uint8_t, ndim=2]`) – image data
- **fillval** (`None`) – (default = None)
- **thresh** (`int`) – (default = 0)

Returns `cropped_img`

Return type `ndarray`

CommandLine: `python -m vtool.image --exec-crop_out_imgfill`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> img = vt.get_stripe_patch()
>>> img = (img * 255).astype(np.uint8)
>>> print(img)
>>> img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
>>> fillval = np.array([25, 25, 25])
>>> thresh = 0
>>> cropped_img = crop_out_imgfill(img, fillval, thresh)
>>> cropped_img2 = cv2.cvtColor(cropped_img, cv2.COLOR_RGB2GRAY)
>>> result = ('cropped_img2 = \n%s' % (str(cropped_img2),))
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> img = vt.get_stripe_patch()
>>> img = (img * 255).astype(np.uint8)
>>> print(img)
>>> fillval = 25
>>> thresh = 0
>>> cropped_img = crop_out_imgfill(img, fillval, thresh)
>>> result = ('cropped_img = \n%s' % (str(cropped_img),))
>>> print(result)
```

`vtool.csum(x)`

`vtool.custom_sympy_attrs(mat)`

`vtool.cvt_BGR2L(imgBGR)`

`vtool.cvt_BGR2RGB (imgBGR)`

`vtool.cvt_bbox_xywh_to_pt1pt2 (xywh, sx=1.0, sy=1.0, round_=True)`
 Converts bbox to thumb format with a scale factor

`vtool.cyclic_distance (arr1, arr2, modulo, out=None)`
 returns an unsigned distance

Parameters

- **arr1** (*ndarray*) –
- **arr2** (*ndarray*) –
- **modulo** (*float or int*) –
- **out** (*ndarray*) – (default = None)

Returns arr_dist

Return type ndarray

CommandLine: `python -m vtool.distance cyclic_distance`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> out = None
>>> modulo = 8
>>> offset = 0 # doesnt matter what offset is
>>> arr1 = np.hstack([np.arange(offset, modulo + offset), np.nan])
>>> arr2 = arr1[:, None]
>>> arr_dist = cyclic_distance(arr1, arr2, modulo, out)
>>> result = ('arr_dist =\n%s' % (ub.repr2(arr_dist),))
```

`vtool.decompose_Z_to_RV_mats2x2 (Z_mats2x2)`

A, B, C = [0.016682, 0.001693, 0.014927] # A, B, C = [0.010141, -1.1e-05, 0.02863] Z = np.array([[A, B], [B, C]])

A, B, C = 0.010141, -1.1e-05, 0.02863

Ignore:

```
>>> # Working on figuring relationship between us and VGG
>>> A, B, _, C = Z_mats2x2[0].ravel()
>>> X, Y = 0, 0
>>> theta = np.linspace(0, np.pi * 2)
>>> circle_xy = np.vstack([np.cos(theta), np.sin(theta)])
>>> invV = invV_mats[0, 0:2, 0:2]
>>> x, y = invV.dot(circle_xy)
>>> V = np.linalg.inv(invV)
>>> E = V.T.dot(V)
>>> [[A, B], [_, C]] = E
>>> [[A_, B_], [_, C_]] = E
>>> print(A*(x-X) ** 2 + 2*B*(x-X)*(y-Y) + C*(y-Y) ** 2)
>>>
>>> Z_mats2x2 = np.array([
>>>     [[ .016682, .001693],
>>>     [ .001693, .014927]],
```

(continues on next page)

(continued from previous page)

```

>>> [[ .01662, .001693],
>>> [ .001693, .014927]],
>>> [[ .016682, .00193],
>>> [ .00193, .01492]],
>>> ])
>>>
>>> import scipy.linalg
>>> %timeit np.array([scipy.linalg.sqrtm(Z) for Z in Z_mats2x2])
>>> %timeit decompose_Z_to_VR_mats2x2(Z_mats2x2)

```

`vtool.decompose_Z_to_V_2x2(Z_2x2)`

`vtool.decompose_Z_to_invV_2x2(Z_2x2)`

`vtool.decompose_Z_to_invV_mats2x2(Z_mats2x2)`

`vtool.demodata_match(cfgdict={}, apply=True, use_cache=True, recompute=False)`

`vtool.det_distance(det1, det2)`

Returns how far off determinants are from one another

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> rng = np.random.RandomState(53)
>>> det1 = rng.rand(5)
>>> det2 = rng.rand(5)
>>> scaledist = det_distance(det1, det2)
>>> result = ub.repr2(scaledist, precision=2, threshold=2)

```

`vtool.det_ltri(ltri)`

Lower triangular determinant

`vtool.detect_opencv_keypoints()`

`vtool.distance_to_lineseg(p, e1, e2)`

`vtool.dot_ltri(ltri1, ltri2)`

Lower triangular dot product

`vtool.draw_border(img_in, color=(0, 128, 255), thickness=2, out=None)`

Parameters

- `img_in` (`ndarray[uint8_t, ndim=2]`) – image data
- `color` (`tuple`) – in bgr
- `thickness` (`int`) –
- `out` (`None`) –

CommandLine: `python -m vtool.geometry --test-draw_border --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> import vtool as vt
>>> img_in = vt.imread(ut.grab_test_imgpath('car1.jpg'))
>>> color = (0, 128, 255)
>>> thickness = 20
>>> out = None
>>> # xdoctest: +REQUIRES(module:plottool)
>>> img = draw_border(img_in, color, thickness, out)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(img)
>>> pt.show_if_requested()
```

`vtool.draw_kp_ori_steps()`

Shows steps in orientation estimation

CommandLine: `python -m vtool.patch -test-draw_kp_ori_steps -show -fname=zebra.png -fx=121`
`python -m vtool.patch -test-draw_kp_ori_steps -show -interact python -m vtool.patch -test-draw_kp_ori_steps -save ~/latex/crall-candidacy-2015/figures/test_fint_kp_direction.jpg -dpath figures -caption=visualization of the steps in the computation of the dominant gradient orientations. -figsize=14,9 -dpi=160 -height=2.65 -left=.04 -right=.96 -top=.95 -bottom=.05 -wspace=.1 -hspace=.1`
`python -m vtool.patch -test-draw_kp_ori_steps -dpath ~/latex/crall-candidacy-2015/ -save figures/draw_kp_ori_steps.jpg -figsize=14,9 -dpi=180 -height=2.65 -left=.04 -right=.96 -top=.95 -bottom=.05 -wspace=.1 -hspace=.1 -diskshow`
`python -m vtool.patch -test-draw_kp_ori_steps -dpath ~/latex/crall-candidacy-2015/ -save figures/draw_kp_ori_steps.jpg -figsize=14,9 -dpi=180 -djust=.04,.05,.1 -diskshow -fname=zebra.png -fx=121`

Example

```
>>> # DISABLE_DOCTEST
>>> import wbia.plottool as pt
>>> from vtool.patch import * # NOQA
>>> draw_kp_ori_steps()
>>> pt.show_if_requested()
```

`vtool.draw_precision_recall_curve(recall_domain, p_interp, title_pref=None, fnum=1, pnum=None, color=None)`

`vtool.draw_roc_curve(fpr, tpr, fnum=None, pnum=None, marker="", target_tpr=None, target_fpr=None, thresholds=None, color=None, name=None, label=None, show_operating_point=False)`

Parameters

- **fpr** –
- **tpr** –
- **fnum** (*int*) – figure number(default = None)
- **pnum** (*tuple*) – plot number(default = None)
- **marker** (*str*) – (default = '-x')

- **target_tpr** (*None*) – (default = None)
- **target_fpr** (*None*) – (default = None)
- **thresholds** (*None*) – (default = None)
- **color** (*None*) – (default = None)
- **show_operating_point** (*bool*) – (default = False)

CommandLine: python -m vtool.confusion --exec-draw_roc_curve --show --lightbg

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.confusion import * # NOQA
>>> scores, labels = testdata_scores_labels()
>>> confusions = ConfusionMetrics().fit(scores, labels)
>>> fpr = confusions.fpr
>>> tpr = confusions.tpr
>>> thresholds = confusions.thresholds
>>> fnum = None
>>> pnum = None
>>> marker = 'x'
>>> target_tpr = .85
>>> target_fpr = None
>>> color = None
>>> show_operating_point = True
>>> draw_roc_curve(fpr, tpr, fnum, pnum, marker, target_tpr, target_fpr,
>>>     thresholds, color, show_operating_point)
>>> ut.show_if_requested()
```

vtool.draw_text (*img, text, org, textcolor_rgb=[0, 0, 0], fontScale=1, thickness=2, fontFace=0, lineType=16, bottomLeftOrigin=False*)

CommandLine: python -m vtool.image --test-draw_text:0 --show python -m vtool.image --test-draw_text:1 --show

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> font_names = sorted([key for key in cv2.__dict__.keys() if key.startswith(
>>>     ↪ 'FONT_H')])
>>> text = 'opencv'
>>> img = np.zeros((400, 1024), dtype=np.uint8)
>>> thickness = 2
>>> fontScale = 1.0
>>> lineType = 4
>>> lineType = 8
>>> lineType = cv2.CV_AA
>>> for count, font_name in enumerate(font_names, start=1):
>>>     print(font_name)
>>>     fontFace = cv2.__dict__[font_name]
>>>     org = (10, count * 45)
>>>     text = 'opencv - ' + font_name
```

(continues on next page)

(continued from previous page)

```

>>> vt.draw_text(img, text, org,
...             fontFace=fontFace, textcolor_rgb=[255, 255, 255],
...             fontScale=fontScale, thickness=thickness)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(img)
>>> ut.show_if_requested()

```

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> font_names = sorted([key for key in cv2.__dict__.keys() if key.startswith(
↳ 'FONT_')])
>>> text = 'opencv'
>>> img = np.zeros((400, 1024, 3), dtype=np.uint8)
>>> img[:200, :512, 0] = 255
>>> img[200:, 512:, 2] = 255
>>> thickness = 2
>>> fontScale = 1.0
>>> lineType = 4
>>> lineType = 8
>>> lineType = cv2.CV_AA
>>> for count, font_name in enumerate(font_names, start=1):
>>>     print(font_name)
>>>     fontFace = cv2.__dict__[font_name]
>>>     org = (10, count * 45)
>>>     text = 'opencv - ' + font_name
>>>     vt.draw_text(img, text, org,
...                 fontFace=fontFace, textcolor_rgb=[255, 255, 255],
...                 fontScale=fontScale, thickness=thickness)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(img)
>>> ut.show_if_requested()

```

where each of the font IDs can be combined with FONT_ITALIC to get the slanted letters.

`vtool.draw_verts` (*img_in*, *verts*, *color*=(0, 128, 255), *thickness*=2, *out*=None)

Parameters

- **img_in** –
- **verts** –
- **color** (*tuple*) –
- **thickness** (*int*) –

Returns *img* - image data

Return type ndarray[uint8_t, ndim=2]

CommandLine: python -m vtool.geometry --test-draw_verts --show python -m vtool.geometry --test-draw_verts:0 --show python -m vtool.geometry --test-draw_verts:1 --show

References

http://docs.opencv.org/modules/core/doc/drawing_functions.html#line

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> import vtool as vt
>>> # build test data
>>> img_in = vt.imread(ut.grab_test_imgpath('car1.jpg'))
>>> verts = ((10, 10), (10, 100), (100, 100), (100, 10))
>>> color = (0, 128, 255)
>>> thickness = 2
>>> # execute function
>>> out = None
>>> img = draw_verts(img_in, verts, color, thickness, out)
>>> assert img_in is not img
>>> assert out is not img
>>> assert out is not img_in
>>> # verify results
>>> # xdoctest: +REQUIRES(--show)
>>> pt.imshow(img)
>>> pt.show_if_requested()
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> import vtool as vt
>>> # build test data
>>> img_in = vt.imread(ut.grab_test_imgpath('car1.jpg'))
>>> verts = ((10, 10), (10, 100), (100, 100), (100, 10))
>>> color = (0, 128, 255)
>>> thickness = 2
>>> out = img_in
>>> # execute function
>>> img = draw_verts(img_in, verts, color, thickness, out)
>>> assert img_in is img, 'should be in place'
>>> assert out is img, 'should be in place'
>>> # verify results
>>> # xdoctest: +REQUIRES(--show)
>>> pt.imshow(img)
>>> pt.show_if_requested()
```

```
out = img_in = np.zeros((500, 500, 3), dtype=np.uint8)
```

```
vtool.dummy_img(w, h, intensity=200)
```

Creates a demodata test image

```
vtool.dummy_seed(seed=None)
```

`vtool.embed_channels (img, input_channels=(0,), nchannels=3, fill=0)`

Parameters

- **img** (`ndarray [uint8_t, ndim=2]`) – image data
- **input_channels** (`tuple`) – (default = (0,))
- **nchannels** (`int`) – (default = 3)

CommandLine: `python -m vtool.image embed_channels --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> # Embed a (N,M,2) image into an (N,M,3) image
>>> img_fpath = ut.grab_test_imgpath('carl.jpg')
>>> img = vt.imread(img_fpath).T[1:3].T
>>> input_channels = (1, 2)
>>> nchannels = 3
>>> newimg = embed_channels(img, input_channels, nchannels)
>>> assert newimg.shape[-1] == 3
>>> assert np.all(newimg[:, :, input_channels] == img)
```

`vtool.embed_in_square_image (img, target_size, img_origin=(0.5, 0.5), target_origin=(0.5, 0.5))`

Embeds an image in the center of an empty image

Parameters

- **img** (`ndarray [uint8_t, ndim=2]`) – image data
- **target_size** (`tuple`) –
- **offset** (`tuple`) – position of

Returns `img_square`

Return type `ndarray`

CommandLine: `python -m vtool.image embed_in_square_image --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> img_fpath = ut.grab_test_imgpath('carl.jpg')
>>> img = vt.imread(img_fpath)
>>> target_size = tuple(np.array(vt.get_size(img)) * 3)
>>> img_origin = (.5, .5)
>>> target_origin = (.5, .5)
>>> img_square = embed_in_square_image(img, target_size, img_origin, target_
↪origin)
>>> assert img_square.sum() == img.sum()
>>> assert vt.get_size(img_square) == target_size
>>> img_origin = (0, 0)
```

(continues on next page)

(continued from previous page)

```
>>> target_origin = (0, 0)
>>> img_square2 = embed_in_square_image(img, target_size, img_origin, target_
↳ origin)
>>> assert img_square.sum() == img.sum()
>>> assert vt.get_size(img_square) == target_size
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(img_square, pnum=(1, 2, 1))
>>> pt.imshow(img_square2, pnum=(1, 2, 2))
>>> ut.show_if_requested()
```

`vtool.emd(hist1, hist2, cost_matrix='sift')`

earth mover's distance by robjects(lpSolve::lp.transport) require: lpsolve55-5.5.0.9.win32-py2.7.exe

CommandLine: python -m vtool.distance --test-emd

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> hist1, hist2 = testdata_hist()
>>> emd_dists = emd(hist1, hist2)
>>> result = ub.repr2(emd_dists, precision=2)
```

`vtool.empty_assign()`

`vtool.empty_neighbors(num_vecs=0, K=0)`

`vtool.ensure_3channel(patch)`

DEPRICATE IN FAVOR OF `atleast_3channels?`

Ensures that there are 3 channels in the image

Parameters `patch` (`ndarray[N, M, ...]`) – the image

Returns [N, M, 3]

Return type `ndarray`

CommandLine: python -m vtool.image --exec-ensure_3channel --show

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> patch1 = vt.imread(ut.grab_test_imgpath('astro.png'))[0:512, 0:500, :]
>>> patch2 = vt.imread(ut.grab_test_imgpath('ada.jpg'))[:, :, 0:1]
>>> patch3 = vt.imread(ut.grab_test_imgpath('jeff.png'))[0:390, 0:400, 0]
>>> res1 = ensure_3channel(patch1)
>>> res2 = ensure_3channel(patch2)
>>> res3 = ensure_3channel(patch3)
>>> assert res1.shape[0:2] == patch1.shape[0:2], 'failed test1'
>>> assert res2.shape[0:2] == patch2.shape[0:2], 'failed test2'
>>> assert res3.shape[0:2] == patch3.shape[0:2], 'failed test3'
>>> assert res1.shape[-1] == 3
```

(continues on next page)

(continued from previous page)

```
>>> assert res2.shape[-1] == 3
>>> assert res3.shape[-1] == 3
```

```
vtool.ensure_4channel(img)
vtool.ensure_alpha_channel(img, alpha=1.0)
vtool.ensure_grayscale(img, colorspace_hint='BGR')
vtool.ensure_metadata_dlen_sqrd(annot)
vtool.ensure_metadata_feats(annot, cfgdict={})
    Adds feature evaluation keys to a lazy dictionary
```

Parameters

- **annot** (*utool.LazyDict*) –
- **suffix** (*str*) – (default = '')
- **cfgdict** (*dict*) – (default = {})

CommandLine: python -m vtool.matching --exec-ensure_metadata_feats

Example

```
>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.matching import * # NOQA
>>> rchip_fpath = ut.grab_test_imgpath('easy1.png')
>>> annot = ut.LazyDict({'rchip_fpath': rchip_fpath})
>>> cfgdict = {}
>>> ensure_metadata_feats(annot, cfgdict)
>>> assert len(annot._stored_results) == 1
>>> annot['kpts']
>>> assert len(annot._stored_results) >= 4
>>> annot['vecs']
>>> assert len(annot._stored_results) >= 5
```

```
vtool.ensure_metadata_flann(annot, cfgdict)
    setup lazy flann evaluation
vtool.ensure_metadata_normxy(annot, cfgdict={})
vtool.ensure_metadata_vsone(annot1, annot2, cfgdict={})
vtool.ensure_monotone_decreasing(arr_, fromleft=True, fromright=True)
```

Parameters **arr** (*ndarray*) –

Returns *arr*

Return type *ndarray*

CommandLine: python -m vtool.util_math --test-ensure_monotone_decreasing --show

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> rng = np.random.RandomState(0)
>>> size_ = 100
>>> domain = np.arange(size_)
>>> arr_ = np.sin(np.pi * (domain / 100) ) + (rng.rand(len(domain)) - .5) * .1
>>> arr = ensure_monotone_decreasing(arr_, fromright=True, fromleft=True)
>>> result = str(arr)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.plot2(domain, arr_, 'r-', fnum=1, pnum=(2, 1, 1), title='before', equal_
↳ aspect=False)
>>> pt.plot2(domain, arr, 'r-', fnum=1, pnum=(2, 1, 2), title='after_
↳ monotonization (decreasing)', equal_aspect=False)
>>> ut.show_if_requested()

```

`vtool.ensure_monotone_increasing(arr_, fromright=True, fromleft=True, newmode=True)`

Parameters `arr` (ndarray) –

Returns `arr`

Return type ndarray

CommandLine: `python -m vtool.util_math --test-ensure_monotone_increasing --show`

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> rng = np.random.RandomState(0)
>>> size_ = 100
>>> domain = np.arange(size_)
>>> offset = float(ub.argval('--offset', default=2.3))
>>> arr_ = np.sin(np.pi * (domain / 100) - offset) + (rng.rand(len(domain)) - .5)
↳ * .1
>>> arr = ensure_monotone_increasing(arr_, fromleft=False, fromright=True)
>>> result = str(arr)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.plot2(domain, arr_, 'r-', fnum=1, pnum=(2, 1, 1), title='before', equal_
↳ aspect=False)
>>> pt.plot2(domain, arr, 'r-', fnum=1, pnum=(2, 1, 2), title='after_
↳ monotonization (increasing)', equal_aspect=False)
>>> ut.show_if_requested()

```

`vtool.ensure_monotone_strictly_decreasing(arr_, left_endpoint=None, right_endpoint=None)`

Parameters

- `arr` (ndarray) –
- `left_endpoint` (None) –
- `right_endpoint` (None) –

Returns arr

Return type ndarray

CommandLine: python -m vtool.util_math --test-ensure_monotone_strictly_decreasing --show

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> import vtool as vt
>>> domain = np.arange(100)
>>> rng = np.random.RandomState(0)
>>> arr_ = np.sin(np.pi * (domain / 75) + 1.3) + (rng.rand(len(domain)) - .5) * .
↳05 + 1.0
>>> #arr_ = vt.demodata.testdata_nonmonotonic()
>>> #domain = np.arange(len(arr_))
>>> left_endpoint = 2.5
>>> right_endpoint = 0.25
>>> arr = ensure_monotone_strictly_decreasing(arr_, left_endpoint, right_endpoint)
>>> result = str(arr)
>>> print(result)
>>> assert strictly_decreasing(arr), 'ensure strict monotonic failed'
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.plot2(domain, arr_, 'r-', fnum=1, pnum=(3, 1, 1), title='before', equal_
↳aspect=False)
>>> arr2 = ensure_monotone_decreasing(arr_)
>>> pt.plot2(domain, arr, 'b-', fnum=1, pnum=(3, 1, 2), equal_aspect=False)
>>> pt.plot2(domain, arr2, 'r-', fnum=1, pnum=(3, 1, 2), title='after_
↳monotonization (decreasing)', equal_aspect=False)
>>> pt.plot2(domain, arr, 'r-', fnum=1, pnum=(3, 1, 3), title='after_
↳monotonization (strictly decreasing)', equal_aspect=False)
>>> ut.show_if_requested()
```

```
vtool.ensure_monotone_strictly_increasing(arr_, left_endpoint=None,
right_endpoint=None, zerohack=False, onehack=False, newmode=True)
```

Parameters

- **arr** (*ndarray*) – sequence to monotonize
- **zerohack** (*bool*) – default False, if True sets the first element to be zero and linearly interpolates to the first nonzero item
- **onehack** (*bool*) – default False, if True one will not be in the resulting array (replaced with number very close to one)

References

<http://mathoverflow.net/questions/17464/making-a-non-monotone-function-monotone> <http://stackoverflow.com/questions/28563711/make-a-numpy-array-monotonic-without-a-python-loop> https://en.wikipedia.org/wiki/Isotonic_regression http://scikit-learn.org/stable/auto_examples/plot_isotonic_regression.html

CommandLine: python -m vtool.util_math --test-ensure_monotone_strictly_increasing --show python -m vtool.util_math --test-ensure_monotone_strictly_increasing --show --offset=0

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> import numpy as np
>>> arr_ = np.array([0.4, 0.4, 0.4, 0.5, 0.6, 0.6, 0.6, 0.7, 0.9, 0.9, 0.91, 0.92,
↳ 1.0, 1.0])
>>> arr = ensure_monotone_strictly_increasing(arr_)
>>> assert strictly_increasing(arr), 'ensure strict monotonic failed1'
```

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> import vtool as vt
>>> left_endpoint = None
>>> rng = np.random.RandomState(0)
>>> right_endpoint = None
>>> domain = np.arange(100)
>>> offset = ut.get_argval('--offset', type_=float, default=2.3)
>>> arr_ = np.sin(np.pi * (domain / 100) - offset) + (rng.rand(len(domain)) - .5)
↳ .1 + 1.2
>>> #arr_ = vt.demodata.testdata_nonmonotonic()
>>> #domain = np.arange(len(arr_))
>>> arr = ensure_monotone_strictly_increasing(arr_, left_endpoint, right_endpoint)
>>> result = str(arr)
>>> print(result)
>>> print('arr = %r' % (arr,))
>>> print('arr = %r' % (np.diff(arr),))
>>> assert non_decreasing(arr), 'ensure nondecreasing failed2'
>>> assert strictly_increasing(arr), 'ensure strict monotonic failed2'
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.plot2(domain, arr_, 'r-', fnum=1, pnum=(3, 1, 1), title='before', equal_
↳ aspect=False)
>>> arr2 = ensure_monotone_increasing(arr_)
>>> pt.plot2(domain, arr, 'b-', fnum=1, pnum=(3, 1, 2), equal_aspect=False)
>>> pt.plot2(domain, arr2, 'r-', fnum=1, pnum=(3, 1, 2), title='after_
↳ monotonization (decreasing)', equal_aspect=False)
>>> pt.plot2(domain, arr, 'r-', fnum=1, pnum=(3, 1, 3), title='after_
↳ monotonization (strictly decreasing)', equal_aspect=False)
>>> ut.show_if_requested()
```

`vtool.ensure_rng(seed=None)`

Returns a numpy random number generator given a seed.

`vtool.ensure_shape(arr, dimshape)`

TODO: Submit as a PR to numpy?

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> ensure_shape(np.array([[1, 2]]), (None, 2))
>>> ensure_shape(np.array([]), (None, 2))
```

`vtool.ensure_shape(arr, dimshape)`
 TODO: Submit as a PR to numpy?

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> ensure_shape(np.array([[1, 2]]), (None, 2))
>>> ensure_shape(np.array([]), (None, 2))
```

`vtool.estimate_pdf(data, gridsize=1024, adjust=1)`

References; <http://statsmodels.sourceforge.net/devel/generated/statsmodels.nonparametric.kde.KDEUnivariate.html> <https://jakevdp.github.io/blog/2013/12/01/kernel-density-estimation/>

Parameters

- **data** (*ndarray*) – 1 dimensional data of float64
- **gridsize** (*int*) – domain size
- **adjust** (*int*) – smoothing factor

Returns data_pdf

Return type ndarray

Example

```
>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.score_normalization import * # NOQA
>>> import vtool as vt
>>> rng = np.random.RandomState(0)
>>> data = rng.randn(1000)
>>> data_pdf = vt.estimate_pdf(data)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.plot(data_pdf.support[:-1], np.diff(data_pdf.cdf))
>>> ut.show_if_requested()
```

`vtool.estimate_refined_transform(kpts1, kpts2, fm, aff_inliers, refine_method='homog')`
 estimates final transformation using normalized affine inliers

References

http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

`vtool.evalprint(str_, globals_=None, locals_=None, simplify=False)`

`vtool.example_binary()`

`vtool.expand_kpts(kpts, scales)`

`vtool.expand_scales(kpts, nScales, low, high)`

`vtool.expand_subscales(kpts, subscale_list)`

`vtool.extent_from_bbox(bbox)`

Parameters `bbox` (*ndarray*) – tl_x, tl_y, w, h

Returns tl_x, br_x, tl_y, br_y

Return type extent (*ndarray*)

CommandLine: `xdoctest -m ~/code/vtool/vtool/geometry.py extent_from_bbox`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> import ubelt as ub
>>> bbox = [0, 0, 10, 10]
>>> extent = extent_from_bbox(bbox)
>>> result = ('extent = %s' % (ub.repr2(extent, nl=0),))
>>> print(result)
extent = [0, 10, 0, 10]
```

`vtool.extent_from_verts` (*verts*)

`vtool.extract_chip_from_gpath` (*gpath, bbox, theta, new_size, interpolation=4*)

`vtool.extract_chip_from_gpath_into_square` (*args*)

`vtool.extract_chip_from_img` (*imgBGR, bbox, theta, new_size, interpolation=4*)

Crops chip from image ; Rotates and scales;

`ibs.show_annot_image(aid)[0].pt_save_and_view()`

Parameters

- `gpath` (*str*) –
- `bbox` (*tuple*) – xywh
- `theta` (*float*) –
- `new_size` (*tuple*) – wy

Returns chipBGR

Return type *ndarray*

CommandLine: `python -m vtool.chip -test-extract_chip_from_img python -m vtool.chip -test-extract_chip_from_img -show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.chip import * # NOQA
>>> # build test data
>>> imgBGR = gtool.imread(ut.grab_test_imgpath('car1.jpg'))
>>> bbox = (100, 3, 100, 100)
>>> theta = 0.0
>>> new_size = (58, 34)
>>> # execute function
>>> chipBGR = extract_chip_from_img(imgBGR, bbox, theta, new_size)
>>> # verify results
```

(continues on next page)

(continued from previous page)

```

>>> assert chipBGR.shape[0:2] == new_size[:::-1], 'did not resize correctly'
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(chipBGR)
>>> pt.show_if_requested()

```

`vtool.extract_chip_into_square` (*imgBGR*, *bbox*, *theta*, *target_size*)

`vtool.extract_feature_from_patch` (*patch*)

`vtool.extract_features` (*img_or_fpath*, *feat_type*=*'hesaff+sift'*, ***kwargs*)

calls pyhesaff's main driver function for detecting hessian affine keypoints. extra parameters can be passed to the hessian affine detector by using kwargs.

Parameters

- `img_or_fpath` (*str*) – image file path on disk
- `use_adaptive_scale` (*bool*) –
- `nogravity_hack` (*bool*) –

Returns (kpts, vecs)

Return type tuple

CommandLine: `python -m vtool.features -test-extract_features python -m vtool.features -test-extract_features -show`

Example

```

>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.features import * # NOQA
>>> import vtool as vt
>>> # build test data
>>> img_fpath = ut.grab_test_imgpath(ut.get_argval('--fname', default='lena.png'))
>>> imgBGR = vt.imread(img_fpath)
>>> feat_type = ut.argval('--feat_type', default='hesaff+sift')
>>> import pyhesaff
>>> kwargs = ut.parse_dict_from_argv(pyhesaff.get_hesaff_default_params())
>>> # execute function
>>> #(kpts, vecs) = extract_features(img_fpath)
>>> (kpts, vecs) = extract_features(imgBGR, feat_type, **kwargs)
>>> # verify results
>>> result = str((kpts, vecs))
>>> print(result)
>>> # Show keypoints
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> #pt.figure(fnum=1, doclf=True, docla=True)
>>> #pt.imshow(imgBGR)
>>> #pt.draw_kpts2(kpts, ori=True)
>>> pt.interact_keypoints.isshow_keypoints(imgBGR, kpts, vecs, ori=True, ell_
↪alpha=.4, color='distinct')
>>> pt.show_if_requested()

```

`vtool.extrema_neighbors` (*extrema_list*, *nBins*)

`vtool.filterflags_valid_images(gpaths, valid_formats=None, invalid_formats=None, verbose=True)`

Flags images with a format that disagrees with its extension

Parameters

- **gpaths** (*list*) – list of image paths
- **valid_formats** (*None*) – (default = None)
- **invalid_formats** (*None*) – (default = None)
- **verbose** (*bool*) – verbosity flag (default = True)

Returns `isvalid_flags`

Return type `list`

CommandLine: `python -m vtool.image filterflags_valid_images --show`

Notes

An MPO (Multi Picture Object) file is a stereoscopic image and contains two JPG images side-by-side, and allows them to be viewed as a single 3D image.

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> gpaths = [ut.grab_test_imgpath('car1.jpg'),
>>>            ut.grab_test_imgpath('astro.png')]
>>> flags = filterflags_valid_images(gpaths)
>>> assert all(flags)
```

`vtool.find_best_undirected_edge_indexes(directed_edges, score_arr=None)`

Parameters

- **directed_edges** (*ndarray [ndims=2]*) –
- **score_arr** (*ndarray*) –

Returns `unique_edge_xs`

Return type `list`

CommandLine: `python -m vtool.other --test-find_best_undirected_edge_indexes`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> directed_edges = np.array([[1, 2], [2, 1], [2, 3], [3, 1], [1, 1], [2, 3], [3,
>>> ↪ 2]])
>>> score_arr = np.array([1, 1, 1, 1, 1, 1, 2])
>>> unique_edge_xs = find_best_undirected_edge_indexes(directed_edges, score_arr)
>>> result = str(unique_edge_xs)
```

(continues on next page)

(continued from previous page)

```
>>> print(result)
[0 3 4 6]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> directed_edges = np.array([[1, 2], [2, 1], [2, 3], [3, 1], [1, 1], [2, 3], [3,
↪ 2]])
>>> score_arr = None
>>> unique_edge_xs = find_best_undirected_edge_indexes(directed_edges, score_arr)
>>> result = str(unique_edge_xs)
>>> print(result)
[0 2 3 4]
```

`vtool.find_clip_range(tp_support, tn_support, clip_factor=2.6180339887499997, reverse=None)`
 TODO: generalize to arbitrary domains (not just 0->inf)

Finds score to clip true positives past. This is useful when the highest true positive scores can be much larger than the highest true negative score.

Parameters

- **tp_support** (*ndarray*) –
- **tn_support** (*ndarray*) –
- **clip_factor** (*float*) – factor of the true negative domain to search for true positives

Returns min_score, max_score

Return type tuple

CommandLine: `python -m vtool.score_normalization --test-find_clip_range`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.score_normalization import * # NOQA
>>> tp_support = np.array([100, 200, 50000])
>>> tn_support = np.array([10, 30, 110])
>>> clip_factor = ut.PHI + 1
>>> min_score, max_score = find_clip_range(tp_support, tn_support, clip_factor)
>>> result = '%.4f, %.4f' % (min_score, max_score)
>>> print(result)
10.0000, 287.9837
```

`vtool.find_dominant_kp_orientations(imgBGR, kp, bins=36, maxima_thresh=0.8, DEBUG_ROTINVAR=False)`

References

http://szeliski.org/Book/drafts/SzeliskiBook_20100903_draft.pdf page 219

<http://www.cs.berkeley.edu/~malik/cs294/lowe-ijcv04.pdf> page 13.

Lowe uses a 36-bin histogram of edge orientations weighted by a gaussian distance to the center and gradient magnitude. He finds all peaks within 80% of the global maximum. Then he fine tunes the orientation using a 3-binned parabolic fit. Multiple orientations (and hence multiple keypoints) can be returned, but empirically only about 15% will have these and they do tend to be important.

Returns ori_offset - offset of current orientation to dominant orientation

Return type float

CommandLine: python -m vtool.patch --test-find_dominant_kp_orientations

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> import vtool as vt
>>> # build test data
>>> np.random.seed(0)
>>> #imgBGR = get_test_patch('cross', jitter=False)
>>> img_fpath = ut.grab_test_imgpath('star.png')
>>> imgBGR = vt.imread(img_fpath)
>>> kpts, vecs = vt.extract_features(img_fpath)
>>> assert len(kpts) == 1
>>> kp = kpts[0]
>>> print('kp = \n' + (vt.kp_cpp_infostr(kp)))
>>> bins = 36
>>> maxima_thresh = .8
>>> # execute function
>>> new_oris = find_dominant_kp_orientations(imgBGR, kp, bins,
>>>                                         maxima_thresh,
>>>                                         DEBUG_ROTINVAR=True)
>>> # verify results
>>> result = 'new_oris = %r' % (new_oris,)
```

vtool.find_duplicate_items(item_arr)

Parameters item_arr –

Returns duplicate_items

Return type

?

CommandLine: python -m vtool.clustering2 --test-find_duplicate_items

References

<http://stackoverflow.com/questions/21888406/getting-the-indexes-to-the-duplicate-columns-of-a-numpy-array>

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.clustering2 import * # NOQA
>>> np.random.seed(0)
```

(continues on next page)

(continued from previous page)

```

>>> item_arr = np.random.randint(100, size=30)
>>> duplicate_items = find_duplicate_items(item_arr)
>>> assert duplicate_items == list(six.iterkeys(ut.find_duplicate_items(item_
    ↪arr)))
>>> result = str(duplicate_items)
>>> print(result)
[9, 67, 87, 88]

```

vtool.find_elbow_point (*curve*)

Finds the on the curve point furthest from the line defined by the endpoints of the curve.

Parameters **curve** (*ndarray*) – a monotonic curve

Returns tradeoff_idx - this is an elbow point in the curve

Return type int

References

<http://stackoverflow.com/questions/2018178/trade-off-point-on-curve>

CommandLine: python -m vtool.other find_elbow_point --show

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> curve = np.exp(np.linspace(0, 10, 100))
>>> tradeoff_idx = find_elbow_point(curve)
>>> result = ('tradeoff_idx = %s' % (ub.repr2(tradeoff_idx),))
>>> print(result)
>>> assert tradeoff_idx == 76
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> import vtool as vt
>>> point = [tradeoff_idx, curve[tradeoff_idx]]
>>> segment = np.array([[0, len(curve) - 1], [curve[0], curve[-1]]])
>>> e1, e2 = segment.T
>>> dist_point = vt.closest_point_on_line_segment(point, e1, e2)
>>> dist_line = np.array([dist_point, point]).T
>>> pt.plot(curve, 'r', label='curve')
>>> pt.plot(point[0], point[1], 'go', markersize=10, label='tradeoff point')
>>> pt.plot(dist_line[0], dist_line[1], '-xb')
>>> pt.plot(segment[0], segment[1], '-xb')
>>> pt.legend()
>>> ut.show_if_requested()

```

vtool.find_first_true_indices (*flags_list*)

TODO: move to vtool

returns a list of indexes where the index is the first True position in the corresponding sublist or None if it does not exist

in other words: for each row finds the smallest True column number or None

Parameters **flags_list** (*list*) – list of lists of booleans

CommandLine: python -m utool.util_list --test-find_first_true_indices

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> # build test data
>>> flags_list = [[True, False, True],
...               [False, False, False],
...               [False, True, True],
...               [False, False, True]]
>>> # execute function
>>> index_list = find_first_true_indices(flags_list)
>>> # verify results
>>> result = str(index_list)
>>> print(result)
[0, None, 1, 2]
```

`vtool.find_k_true_indicies(flags_list, k)`

Uses output of either this function or `find_first_true_indices` to find the next index of true flags

Parameters `flags_list` (*list*) – list of lists of booleans

CommandLine: python -m utool.util_list --test-find_next_true_indices

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> flags_list = [[False, False, True],
...               [False, False, False],
...               [False, True, True],
...               [True, True, True]]
>>> k = 2
>>> indices = find_k_true_indicies(flags_list, k)
>>> result = str(indices)
>>> print(result)
[array([2]), None, array([1, 2]), array([0, 1])]
```

`vtool.find_kpts_direction(imgBGR, kpts, DEBUG_ROTINVAR=False)`

Parameters

- `imgBGR` (`ndarray[uint8_t, ndim=2]`) – image data in opencv format (blue, green, red)
- `kpts` (`ndarray[float32_t, ndim=2]`) – keypoints

Returns kpts - keypoints

Return type `ndarray[float32_t, ndim=2]`

CommandLine: python -m vtool.patch --test-find_kpts_direction

`vtool.find_maxima(y_list)`

`vtool.find_maxima_with_neighbors(scalar_list)`

`vtool.find_next_true_indices(flags_list, offset_list)`

Uses output of either this function or `find_first_true_indices` to find the next index of true flags

Parameters `flags_list` (*list*) – list of lists of booleans

CommandLine: `python -m vtool.util_list --test-find_next_true_indices`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> # build test data
>>> flags_list = [[True, False, True],
...               [False, False, False],
...               [False, True, True],
...               [False, False, True]]
>>> offset_list = find_first_true_indices(flags_list)
>>> # execute function
>>> index_list = find_next_true_indices(flags_list, offset_list)
>>> # verify results
>>> result = str(index_list)
>>> print(result)
[2, None, 2, None]
```

`vtool.find_patch_dominant_orientations(patch, bins=36, maxima_thresh=0.8, DE-BUG_ROTINVAR=False)`

helper

`vtool.find_pixel_value_index(img, pixel)`

Parameters

- `img` (`ndarray[uint8_t, ndim=2]`) – image data
- `pixel` (`ndarray` or `scalar`) –

CommandLine: `python -m vtool.util_math --test-find_pixel_value_index`

References

<http://stackoverflow.com/questions/21407815/get-column-row-index-from-numpy-array-that-meets-a-boolean-condition>

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> # build test data
>>> img = np.random.rand(10, 10, 3) + 1.0
>>> pixel = np.array([0, 0, 0])
>>> img[5, 5, :] = pixel
>>> img[2, 3, :] = pixel
>>> img[1, 1, :] = pixel
>>> img[0, 0, :] = pixel
>>> img[2, 0, :] = pixel
>>> # execute function
```

(continues on next page)

(continued from previous page)

```

>>> result = find_pixel_value_index(img, pixel)
>>> # verify results
>>> print(result)
[[0 0]
 [1 1]
 [2 0]
 [2 3]
 [5 5]]

```

`vtool.flag_intersection(arr1, arr2)`

Flags the rows in *arr1* that contain items in *arr2*

Returns flags where `len(flags) == len(arr1)`

Return type ndarray

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> arr1 = np.array([0, 1, 2, 3, 4, 5])
>>> arr2 = np.array([2, 6, 4])
>>> flags = flag_intersection(arr1, arr2)
>>> assert len(flags) == len(arr1)
>>> result = ('flags = %s' % (ub.repr2(flags),))
>>> print(result)

```

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> import vtool as vt
>>> arr1 = np.array([[0, 0], [0, 1], [0, 2], [0, 3], [0, 4], [0, 5]])
>>> arr2 = np.array([[0, 2], [0, 6], [0, 4], [3, 0]])
>>> arr1, arr2 = vt.structure_rows(arr1, arr2)
>>> flags = flag_intersection(arr1, arr2)
>>> assert len(flags) == len(arr1)
>>> result = ('flags = %s' % (ub.repr2(flags),))
>>> print(result)

```

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> arr1 = np.array([0, 1, 2, 3, 4, 5])
>>> arr2 = np.array([])
>>> flags = flag_intersection(arr1, arr2)
>>> assert len(flags) == len(arr1)
>>> flags = flag_intersection(np.array([]), np.array([2, 6, 4]))
>>> assert len(flags) == 0

```

Ignore:

```

>>> setup = ut.codeblock(
>>>     r'''
import vtool as vt
import numpy as np
rng = np.random.RandomState(0)
arr1 = rng.randint(0, 100, 100000).reshape(-1, 2)
arr2 = rng.randint(0, 100, 1000).reshape(-1, 2)
arr1_, arr2_ = vt.structure_rows(arr1, arr2)
'''
>>> stmt_list = ut.codeblock(
>>>     '''
np.array([row in arr2_ for row in arr1_])
np.logical_or.reduce([arr1_ == row_ for row_ in arr2_]).ravel()
vt.iter_reduce_ufunc(np.logical_or, (arr1_ == row_ for row_ in arr2_
↵)).ravel()
''').split('\n')
>>> out = ut.timeit_compare(stmt_list, setup=setup, iterations=3)

```

vtool.flag_sym_slow (*fx1_to_fx2, fx2_to_fx1, K*)

Returns flags indicating if the matches in *fx1_to_fx2* are reciprocal with the matches in *fx2_to_fx1*.

Much slower version of `flag_symmetric_matches`, but more clear

vtool.flag_symmetric_matches (*fx2_to_fx1, fx1_to_fx2, K=2*)

Returns flags indicating if the matches in *fx2_to_fx1* are reciprocal with the matches in *fx1_to_fx2*.

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.matching import * # NOQA
>>> K = 2
>>> fx2_to_fx1 = np.array([[ 0,  1], # 0
>>>                        [ 1,  4], # 1
>>>                        [ 3,  4], # 2
>>>                        [ 2,  3]], dtype=np.int32) # 3
>>> fx1_to_fx2 = np.array([[ 0, 1], # 0
>>>                        [ 2, 1], # 1
>>>                        [ 0, 1], # 2
>>>                        [ 3, 1], # 3
>>>                        [ 0, 1]], dtype=np.int32) # 4
>>> fx2_to_flagsA = flag_symmetric_matches(fx2_to_fx1, fx1_to_fx2, K)
>>> fx2_to_flagsB = flag_sym_slow(fx2_to_fx1, fx1_to_fx2, K)
>>> assert np.all(fx2_to_flagsA == fx2_to_flagsB)
>>> result = ub.repr2(fx2_to_flagsB)
>>> print(result)

```

vtool.flann_augment (*dpts, new_dpts, cache_dir, cfgstr, new_cfgstr, flann_params, use_cache=True, save=True*)

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.nearest_neighbors import * # NOQA
>>> import vtool.demodata as demodata # NOQA

```

(continues on next page)

(continued from previous page)

```

>>> dpts = demodata.get_dummy_dpts(ut.get_nth_prime(10))
>>> new_dpts = demodata.get_dummy_dpts(ut.get_nth_prime(9))
>>> cache_dir = ut.get_app_resource_dir('vtool')
>>> cfgstr = '_testcfg'
>>> new_cfgstr = '_new_testcfg'
>>> flann_params = get_kdtree_flann_params()
>>> use_cache = False
>>> save = False

```

vtool.flann_cache (*dpts*, *cache_dir*='default', *cfgstr*="", *flann_params*={}, *use_cache*=True, *save*=True, *use_params_hash*=True, *use_data_hash*=True, *appname*='vtool', *verbose*=None)
Tries to load a cached flann index before doing anything from vtool.nn

vtool.flann_index_time_experiment ()
Shows a plot of how long it takes to build a flann index for a given number of KD-trees
CommandLine: python -m vtool.nearest_neighbors --test-flann_index_time_experiment

Example

```

>>> # SLOW_DOCTEST
>>> # xdoctest: +SKIP
>>> from vtool.nearest_neighbors import * # NOQA
>>> result = flann_index_time_experiment()
>>> print(result)

```

vtool.flatten_invV_mats_to_kpts (*invV_mats*)
flattens invV matrices into kpts format

vtool.flatten_scores (*tp_scores*, *tn_scores*, *part_attrs*=None)
convenience helper to translate partitioned to unpartitioned data

Parameters

- **tp_scores** (*ndarray*) –
- **tn_scores** (*ndarray*) –
- **part_attrs** (*dict*) – (default = None)

Returns (scores, labels, attrs)

Return type `tuple`

CommandLine: python -m vtool.score_normalization --test-flatten_scores

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.score_normalization import * # NOQA
>>> tp_scores = np.array([5, 6, 6, 7])
>>> tn_scores = np.array([1, 2, 2])
>>> part_attrs = {
...     1: {'qaid': [21, 24, 25, 26]},
...     0: {'qaid': [11, 14, 15]},
... }

```

(continues on next page)

(continued from previous page)

```

>>> tup = flatten_scores(
...   tp_scores, tn_scores, part_attrs)
>>> (X, y, attrs) = tup
>>> y = y.astype(np.int)
>>> resdict = ut.odict(zip(['X', 'y', 'attrs'], [X, y, attrs]))
>>> result = ub.repr2(resdict, nobraces=True, with_dtype=False,
>>>                  explicit=1, nl=1)
>>> print(result)
X=np.array([5, 6, 6, 7, 1, 2, 2]),
y=np.array([1, 1, 1, 1, 0, 0, 0]),
attrs='qaids': np.array([21, 24, 25, 26, 11, 14, 15]),

```

`vtool.force_kpts_feasibility(kpts, xys_nonneg=False)`

`vtool.fromiter_nd(iter_, shape, dtype)`

Like `np.fromiter` but handles iterators that generated n-dimensional arrays. Slightly faster than `np.array`.

Note: `np.vstack(list_)` is still faster than `vt.fromiter_nd(ut.iflatten(list_))`

Parameters

- **iter_** (*iter*) – an iterable that generates homogenous ndarrays
- **shape** (*tuple*) – the expected output shape
- **dtype** (*dtype*) – the numpy datatype of the generated ndarrays

Note: The iterable must yeild a numpy array. It cannot yeild a Python list.

CommandLine: `python -m vtool.numpy_utils fromiter_nd`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> dtype = np.float
>>> total = 11
>>> rng = np.random.RandomState(0)
>>> iter_ = (rng.rand(5, 7, 3) for _ in range(total))
>>> shape = (total, 5, 7, 3)
>>> result = fromiter_nd(iter_, shape, dtype)
>>> assert result.shape == shape

```

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> import utool as ut
>>> dtype = np.int
>>> qfxs = np.array([1, 2, 3])

```

(continues on next page)

(continued from previous page)

```

>>> dfxs = np.array([4, 5, 6])
>>> iter_ = (np.array(x) for x in ut.product(qfxs, dfxs))
>>> total = len(qfxs) * len(dfxs)
>>> shape = (total, 2)
>>> result = fromiter_nd(iter_, shape, dtype)
>>> assert result.shape == shape

```

Ignore:

```

>>> dtype = np.uint8
>>> feat_dim = 128
>>> mu = 1000
>>> sigma = 500
>>> n_data = 1000
>>> rng = np.random.RandomState(42)
>>> n_feat_list = np.clip(rng.randn(n_data) * sigma + mu, 0, np.inf).
↳astype(np.int)
>>> # Make a large list of vectors of various sizes
>>> print('Making random vectors')
>>> vecs_list = [(rng.rand(num, feat_dim) * 255).astype(dtype) for num in n_
↳feat_list]
>>> mega_bytes = sum([x.nbytes for x in vecs_list]) / 2 ** 20
>>> print('mega_bytes = %r' % (mega_bytes,))
>>> import itertools as it
>>> import vtool as vt
>>> n_total = n_feat_list.sum()
>>> target1 = np.vstack(vecs_list)
>>> iter_ = it.chain.from_iterable(vecs_list)
>>> shape = (n_total, feat_dim)
>>> target2 = vt.fromiter_nd(it.chain.from_iterable(vecs_list), shape,
↳dtype=dtype)
>>> assert np.all(target1 == target2)
>>>
>>> %timeit np.vstack(vecs_list)
>>> 20.4ms
>>> %timeit vt.fromiter_nd(it.chain.from_iterable(vecs_list), shape, dtype)
>>> 102ms
>>>
>>> iter_ = it.chain.from_iterable(vecs_list)
>>> %time vt.fromiter_nd(iter_, shape, dtype)
>>> %time np.vstack(vecs_list)

```

`vtool.fromiter_nd(iter_, shape, dtype)`

Like `np.fromiter` but handles iterators that generated n-dimensional arrays. Slightly faster than `np.array`.

Note: `np.vstack(list_)` is still faster than `vt.fromiter_nd(ut.flatten(list_))`

Parameters

- **iter_** (*iter*) – an iterable that generates homogenous ndarrays
- **shape** (*tuple*) – the expected output shape
- **dtype** (*dtype*) – the numpy datatype of the generated ndarrays

Note: The iterable must yeild a numpy array. It cannot yeild a Python list.

CommandLine: python -m vtool.numpy_utils fromiter_nd

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> dtype = np.float
>>> total = 11
>>> rng = np.random.RandomState(0)
>>> iter_ = (rng.rand(5, 7, 3) for _ in range(total))
>>> shape = (total, 5, 7, 3)
>>> result = fromiter_nd(iter_, shape, dtype)
>>> assert result.shape == shape
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> import utool as ut
>>> dtype = np.int
>>> qfxs = np.array([1, 2, 3])
>>> dfxs = np.array([4, 5, 6])
>>> iter_ = (np.array(x) for x in ut.product(qfxs, dfxs))
>>> total = len(qfxs) * len(dfxs)
>>> shape = (total, 2)
>>> result = fromiter_nd(iter_, shape, dtype)
>>> assert result.shape == shape
```

Ignore:

```
>>> dtype = np.uint8
>>> feat_dim = 128
>>> mu = 1000
>>> sigma = 500
>>> n_data = 1000
>>> rng = np.random.RandomState(42)
>>> n_feat_list = np.clip(rng.randn(n_data) * sigma + mu, 0, np.inf).
↳ astype(np.int)
>>> # Make a large list of vectors of various sizes
>>> print('Making random vectors')
>>> vecs_list = [(rng.rand(num, feat_dim) * 255).astype(dtype) for num in n_
↳ feat_list]
>>> mega_bytes = sum([x.nbytes for x in vecs_list]) / 2 ** 20
>>> print('mega_bytes = %r' % (mega_bytes,))
>>> import itertools as it
>>> import vtool as vt
>>> n_total = n_feat_list.sum()
>>> target1 = np.vstack(vecs_list)
>>> iter_ = it.chain.from_iterable(vecs_list)
>>> shape = (n_total, feat_dim)
```

(continues on next page)

(continued from previous page)

```

>>> target2 = vt.fromiter_nd(it.chain.from_iterable(vecs_list), shape,
↳dtype=dtype)
>>> assert np.all(target1 == target2)
>>>
>>> %timeit np.vstack(vecs_list)
>>> 20.4ms
>>> %timeit vt.fromiter_nd(it.chain.from_iterable(vecs_list), shape, dtype)
>>> 102ms
>>>
>>> iter_ = it.chain.from_iterable(vecs_list)
>>> %time vt.fromiter_nd(iter_, shape, dtype)
>>> %time np.vstack(vecs_list)

```

`vtool.gamma_adjust (img, gamma=1.0)`

CommandLine: `python -m vtool.blend --test-gamma_adjust:0 --show`

Ignore:

```

>>> # DISABLE_DOCTEST
>>> from vtool.blend import * # NOQA
>>> import vtool as vt
>>> test_func = gamma_adjust
>>> img_fpath = ut.grab_test_imgpath('carl.jpg')
>>> img = vt.rectify_to_float01(vt.imread(img_fpath))
>>> args = (img,)
>>> param_info = ut.ParamInfoList('blend_params', [
...     ut.ParamInfo('gamma', .8, 'gamma=',
...                   varyvals=np.linspace(.1, 2.5, 25).tolist()),
... ])
>>> gridsearch_image_function(param_info, test_func, args)
>>> ut.show_if_requested()

```

`vtool.gauss2d_pdf (x_, y_, sigma=None, mu=None)`

Parameters

- **x** – x coordinate of a 2D Gaussian
- **y** – y coordinate of a 2D Gaussian
- **sigma** – covariance of vector
- **mu** – mean of vector

Returns float - The probability density at that point

`vtool.gauss_func1d (x, mu=0.0, sigma=1.0)`

Parameters

- **x** –
- **mu** (*float*) –
- **sigma** (*float*) –

CommandLine: `python -m vtool.util_math --test-gauss_func1d`

CommandLine: `python -m vtool.util_math --test-gauss_func1d --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> # build test data
>>> x = np.array([-2, -1, -.5, 0, .5, 1, 2])
>>> mu = 0.0
>>> sigma = 1.0
>>> # execute function
>>> gaussval = gauss_func1d(x, mu, sigma)
>>> # verify results
>>> result = np.array_repr(gaussval, precision=2)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.plot(x, gaussval)
>>> ut.show_if_requested()
array([ 0.05,  0.24,  0.35,  0.4 ,  0.35,  0.24,  0.05])
```

`vtool.gauss_func1d_unnormalized(x, sigma=1.0)`

faster version of `gauss_func1d` with no normalization. So the maximum point will have a value of 1.0

CommandLine: `python -m vtool.util_math -test-gauss_func1d_unnormalized --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> # build test data
>>> x = np.array([-2, -1, -.5, 0, .5, 1, 2])
>>> sigma = 1.0
>>> # execute function
>>> gaussval = gauss_func1d_unnormalized(x, sigma)
>>> # verify results
>>> result = np.array_repr(gaussval, precision=2)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.plot(x, gaussval)
>>> ut.show_if_requested()
array([ 0.05,  0.24,  0.35,  0.4 ,  0.35,  0.24,  0.05])
```

`vtool.gauss_parzen_est(dist, L=1, sigma=0.38)`

`python -m wbia.plottool.draw_func2 -exec-plot_func -show -range=-.2,.2 -func=vt.gauss_parzen_est python`
`-m wbia.plottool.draw_func2 -exec-plot_func -show -range=0,1 -func=vt.gauss_parzen_est`

`vtool.gaussian_average_patch(patch, sigma=None, copy=True)`

Parameters

- `patch` (`ndarray`) –
- `sigma` (`float`) –

CommandLine: `python -m vtool.patch -test-gaussian_average_patch`

References

<http://docs.opencv.org/modules/imgproc/doc/filtering.html#getgaussiankernel>

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> patch = get_star_patch()
>>> #sigma = 1.6
>>> sigma = None
>>> result = gaussian_average_patch(patch, sigma)
>>> print(result)
0.414210641527
```

Ignore: import utool as ut import wbia.plottool as pt import vtool as vt import cv2
 gauss_kernel_d0 = (cv2.getGaussianKernel(patch.shape[0], sigma)) gauss_kernel_d1 =
 (cv2.getGaussianKernel(patch.shape[1], sigma)) weighted_patch = patch.copy() weighted_patch =
 np.multiply(weighted_patch, gauss_kernel_d0) weighted_patch = np.multiply(weighted_patch.T,
 gauss_kernel_d1).T gaussian_kern2 = gauss_kernel_d0.dot(gauss_kernel_d1.T) fig = pt.figure(fnum=1,
 pnum=(1, 3, 1), doclf=True, docla=True) pt.imshow(patch * 255) fig = pt.figure(fnum=1, pnum=(1,
 3, 2)) pt.imshow(ut.norm_zero_one(gaussian_kern2) * 255.0) fig = pt.figure(fnum=1, pnum=(1, 3, 3))
 pt.imshow(ut.norm_zero_one(weighted_patch) * 255.0) pt.update()

`vtool.gaussian_patch(shape=(7, 7), sigma=1.0)`
 another version of the gaussian_patch function. hopefully better

References

<http://docs.opencv.org/modules/imgproc/doc/filtering.html#getgaussiankernel>

Parameters

- **shape** (*tuple*) – array dimensions
- **sigma** (*float*) –

CommandLine: python -m vtool.patch --test-gaussian_patch --show

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> #shape = (7, 7)
>>> shape = (24, 24)
>>> sigma = None # 1.0
>>> gausspatch = gaussian_patch(shape, sigma)
>>> sum_ = gausspatch.sum()
>>> ut.assert_almost_eq(sum_, 1.0)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(vt.norm01(gausspatch) * 255)
>>> ut.show_if_requested()
```

`vtool.gaussian_weight_patch(patch, sigma=None)`

Applies two one dimensional gaussian operations to a patch which effectively weights it by a 2-dimensional gaussian. This is efficient because the actually 2-d gaussian never needs to be allocated.

`test_show_gaussian_patches`

`vtool.generate_to_patch_transforms(kpts, patch_size=41)`

`vtool.get_RV_mats2x2(kpts)`

Returns sequence of matrices that transform an ellipse to unit circle

Return type V_mats (ndarray)

`vtool.get_RV_mats_3x3(kpts)`

preferred over `get_invV_mats`

Returns sequence of matrices that transform an ellipse to unit circle

Return type V_mats (ndarray)

`vtool.get_V_mats(kpts, **kwargs)`

Returns sequence of matrices that transform an ellipse to unit circle

Return type V_mats (ndarray)

`vtool.get_Z_mats(V_mats)`

transform into conic matrix Z $Z = (V.T).dot(V)$

Returns Z is a conic representation of an ellipse

Return type Z_mats (ndarray)

`vtool.get_affine_inliers(kpts1, kpts2, fm, fs, xy_thresh_sqrd, scale_thresh_sqrd, ori_thresh)`

Estimates inliers deterministically using elliptical shapes

Compute all transforms from `kpts1` to `kpts2` (enumerate all hypothesis) We transform from `chip1` -> `chip2` The determinants are squared keypoint scales

Returns `aff_inliers_list`, `aff_errors_list`, `Aff_mats`

Return type tuple

Notes

FROM PERDOCH 2009: $H = inv(A_j).dot(R_j.T).dot(R_i).dot(A_i)$ $H = inv(A_j).dot(A_i)$ The input `invVs` = perdoch.`invA`'s

CommandLine: `python2 -m vtool.spatial_verification -test-get_affine_inliers python3 -m vtool.spatial_verification -test-get_affine_inliers`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.demodata as demodata
>>> import vtool.keypoint as ktool
>>> kpts1, kpts2 = demodata.get_dummy_kpts_pair((100, 100))
>>> fm = demodata.make_dummy_fm(len(kpts1)).astype(np.int32)
>>> fs = np.ones(len(fm), dtype=np.float64)
>>> xy_thresh_sqrd = ktool.KPTS_DTYPE(.009) ** 2
```

(continues on next page)

(continued from previous page)

```

>>> scale_thresh_sqrd = ktool.KPTS_DTYPE(2)
>>> ori_thresh = ktool.KPTS_DTYPE(TAU / 4)
>>> output = get_affine_inliers(kpts1, kpts2, fm, fs, xy_thresh_sqrd,
>>>                             scale_thresh_sqrd, ori_thresh)
>>> output_str = ut.repr3(output, precision=2, suppress_small=True)
>>> print('output_str = %s' % (output_str,))
>>> aff_inliers_list, aff_errors_list, Aff_mats = output
>>> result = 'nInliers=%r hash=%s' % (len(aff_inliers_list), ut.hash_data(output_
↪str))
>>> print(result)

```

vtool.get_best_affine_inliers (*kpts1, kpts2, fm, fs, xy_thresh_sqrd, scale_thresh, ori_thresh, forcepy=False*)

Tests each hypothesis and returns only the best transformation and inliers

vtool.get_best_affine_inliers_ (*kpts1, kpts2, fm, fs, xy_thresh_sqrd, scale_thresh, ori_thresh*)

vtool.get_covered_mask (*covered_array, covering_array*)

vtool.get_crop_slices (*isfill*)

vtool.get_cross_patch (*jitter=False*)
test data patch

vtool.get_dummy_dpts (*num, dtype=<class 'numpy.uint8'>*)
Random SIFTish keypoints

vtool.get_dummy_invV_mats (*dtype=<class 'numpy.float32'>*)

vtool.get_dummy_kpts (*num=1, dtype=<class 'numpy.float32'>*)
Some testing data

Parameters

- **num** (*int*) – number of times to duplicate
- **dtype** (*type*) –

Returns kpts - keypoints

Return type ndarray[float32_t, ndim=2][ndims=2]

CommandLine: xdoctest -m ~/code/vtool/vtool/demodata.py get_dummy_kpts

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.demodata import * # NOQA
>>> num = 1
>>> dtype = ktool.KPTS_DTYPE
>>> kpts = get_dummy_kpts(num, dtype)
>>> import ubelt as ub
>>> result = ub.repr2(kpts, precision=2, with_dtype=False)

```

vtool.get_dummy_kpts_pair (*wh_stride=(30, 30), wh_num=None*)

vtool.get_dummy_matching_kpts (*dtype=<class 'numpy.float32'>*)

vtool.get_dummy_xy (*seed=0*)

`vtool.get_even_point_sample(kpts)`
 gets even points sample along the boundary of the ellipse

SeeAlso: `pyhesaff.tests.test_ellipse`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts()[0:2]
>>> ell_border_pts_list = get_even_point_sample(kpts)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.draw_line_segments(ell_border_pts_list)
>>> pt.set_title('even sample points')
>>> pt.show_if_requested()
```

`vtool.get_exif_dict(pil_img)`
 Returns exif dictionary by TAGID

`vtool.get_exif_dict2(pil_img)`
 Returns exif dictionary by TAG (less efficient)

`vtool.get_exif_tagids(tag_list)`

`vtool.get_exist(data, key)`

`vtool.get_extract_features_default_params()`

Returns

Return type `dict`

CommandLine: `python -m vtool.features --test-get_extract_features_default_params`

Example

```
>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.features import * # NOQA
>>> # build test data
>>> # execute function
>>> param_dict = get_extract_features_default_params()
>>> result = ub.repr2(param_dict)
>>> # verify results
>>> print(result)
```

`vtool.get_extramargin_measures(bbox_gs, new_size, halfoffset_ms=(64, 64))`
 Computes a detection chip with a bit of spatial context so the detection algorithm doesn't clip boundaries

Returns

mbbox_gs, margin_size - margin bounding box in image size, size of entire margined chip,

CommandLine: `python -m vtool.chip --test-get_extramargin_measures --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.chip import * # NOQA
>>> gfilepath = ut.grab_test_imgpath('car1.jpg')
>>> bbox_gs = [40, 40, 150, 150]
>>> theta = .15 * (np.pi * 2)
>>> new_size = (150, 150)
>>> halfoffset_ms = (32, 32)
>>> mbbox_gs, margin_size = get_extramargin_measures(bbox_gs, new_size,
↳halfoffset_ms)
>>> # xdoctest: +REQUIRES(--show)
>>> testshow_extramargin_info(gfilepath, bbox_gs, theta, new_size, halfoffset_ms,
↳mbbox_gs, margin_size)
```

```
vtool.get_flann_cfgstr(dpts, flann_params, cfgstr="", use_params_hash=True,
use_data_hash=True)
```

CommandLine: python -m vtool.nearest_neighbors --test-get_flann_cfgstr

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.nearest_neighbors import * # NOQA
>>> rng = np.random.RandomState(1)
>>> dpts = rng.randint(0, 255, (10, 128)).astype(np.uint8)
>>> cache_dir = '.'
>>> cfgstr = '_FEAT(alg=hshes)'
>>> flann_params = get_kdtree_flann_params()
>>> result = get_flann_cfgstr(dpts, flann_params, cfgstr)
>>> print(result)
_FEAT(alg=hshes)_FLANN(4kdtree)_DPTS((10,128)xxaotseonmfjkzcr)
```

```
vtool.get_flann_fpath(dpts, cache_dir='default', cfgstr="", flann_params={},
use_params_hash=True, use_data_hash=True, appname='vtool', ver-
bose=True)
```

returns filepath for flann index

```
vtool.get_flann_params(algorithm='kdtree', **kwargs)
```

Returns flann params that are relevant to the algorithm

References

http://www.cs.ubc.ca/research/flann/uploads/FLANN/flann_manual-1.8.4.pdf

Parameters `algorithm` (*str*) – (default = 'kdtree')

Returns flann_params

Return type dict

CommandLine: python -m vtool.nearest_neighbors --test-get_flann_params --algo=kdtree python -m vtool.nearest_neighbors --test-get_flann_params --algo=kmeans

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.nearest_neighbors import * # NOQA
>>> algorithm = ut.get_argval('--algo', default='kdtree')
>>> flann_params = get_flann_params(algorithm)
>>> result = ('flann_params = %s' % (ub.repr2(flann_params),))
>>> print(result)
```

`vtool.get_flann_params_cfgstr(flann_params)`

`vtool.get_grid_kpts(wh=(300, 300), wh_stride=None, scale=20, wh_num=None, dtype=<class 'numpy.float32'>, **kwargs)`

Returns a regular grid of keypoints

Parameters

- **wh** (*tuple*) – (default = (300, 300))
- **wh_stride** (*tuple*) – stride of keypoints (defaults to (50, 50))
- **scale** (*int*) – (default = 20)
- **wh_num** (*tuple*) – desired number of keypoints in x and y direction. (incompatible with stride).
- **dtype** (*type*) – (default = <type 'numpy.float32'>)

Returns kpts - keypoints

Return type ndarray[float32_t, ndim=2]

CommandLine: `python -m vtool.keypoint get_grid_kpts --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> wh = (300, 300)
>>> wh_stride = None
>>> scale = 20
>>> wh_num = (3, 3)
>>> dtype = np.float32
>>> kpts = get_grid_kpts(wh, wh_num=wh_num, dtype=dtype)
>>> assert len(kpts) == np.prod(wh_num)
>>> result = ('kpts = %s' % (ub.repr2(kpts.shape),))
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.show_kpts(kpts)
>>> pt.dark_background()
>>> ut.show_if_requested()
```

`vtool.get_histinfo_str(hist, edges)`

`vtool.get_image_to_chip_transform(bbox, chipsize, theta)`
transforms image space into chip space

Parameters

- **– bounding box of chip in image space** (*bbox*) –

- - size of the chip(*chipsz*)-
- - rotation of the bounding box(*theta*)-

Ignore:

```
>>> # https://groups.google.com/forum/#!topic/sympy/k1HnZK_bNNA
>>> from vtool.patch import * # NOQA
>>> import sympy
>>> import sympy.abc
>>> theta = sympy.abc.theta
>>>
>>> x, y, w, h, target_area = sympy.symbols('x y w h, a')
>>> gx, gy = sympy.symbols('gx, gy')
>>>
>>> round = sympy.floor # hack
>>>
>>> ht = sympy.sqrt(target_area * h / w)
>>> wt = w * ht / h
>>> cw_, ch_ = round(wt), round(ht)
>>>
>>> from vtool import ltool
>>> T1 = ltool.translation_mat3x3(tx1, ty1, dtype=None)
>>> S = ltool.scale_mat3x3(sx, sy, dtype=None)
>>> R = ltool.rotation_mat3x3(-theta, sympy.sin, sympy.cos)
>>> T2 = ltool.translation_mat3x3(tx2, ty2, dtype=None)
>>>
>>> def add_matmul_hold_prop(mat):
>>>     #import functools
>>>     mat = sympy.Matrix(mat)
>>>     def matmul_hold(other, hold=False):
>>>         new = sympy.MatMul(mat, other, hold=hold)
>>>         add_matmul_hold_prop(new)
>>>         return new
>>>     setattr(mat, 'matmul_hold', matmul_hold)
>>>     return mat
>>>
>>> T1 = add_matmul_hold_prop(T1)
>>> T2 = add_matmul_hold_prop(T2)
>>> R = add_matmul_hold_prop(R)
>>> S = add_matmul_hold_prop(S)
>>>
>>> C = T2.multiply(R.multiply(S.multiply(T1)))
>>> sympy.simplify(C)
```

`vtool.get_invVR_mats2x2(kpts)`

Returns the keypoint shape+rotation matrix (from unit circle to ellipse) Ignores translation component

Parameters `kpts` (`ndarray[float32_t, ndim=2][ndims=2]`) – keypoints

Returns `invVR_mats`

Return type `ndarray`

CommandLine: `python -m vtool.keypoint --test-get_invVR_mats2x2`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> kpts = np.array([
...     [0, 0, 1, 2, 3, 0],
...     [0, 0, 1, 2, 3, TAU / 4.0],
... ])
>>> invVR_mats2x2 = get_invVR_mats2x2(kpts)
>>> result = kpts_repr(invVR_mats2x2)
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> kpts = np.empty((0, 6))
>>> invVR_mats2x2 = get_invVR_mats2x2(kpts)
>>> assert invVR_mats2x2.shape == (0, 2, 2)
```

`vtool.get_invVR_mats3x3(kpts)`

NEWER FUNCTION

Returns full keypoint transform matrices from a unit circle to an ellipse that has been rotated, scaled, skewed, and translated. Into the image keypoint position.

Parameters `kpts` (`ndarray[float32_t, ndim=2]`) – keypoints

Returns `invVR_mats`

Return type `ndarray[float32_t, ndim=3]`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> kpts = np.array([
...     [10, 20, 1, 2, 3, 0],
...     [30, 40, 1, 2, 3, TAU / 4.0],
... ])
>>> invVR_mats3x3 = get_invVR_mats3x3(kpts)
>>> # verify results
>>> result = kpts_repr(invVR_mats3x3)
>>> print(result)
array([[ [ 1.,  0., 10.],
        [ 2.,  3., 20.],
        [ 0.,  0.,  1.]],
       [[ 0., -1., 30.],
        [ 3., -2., 40.],
        [ 0.,  0.,  1.]])
```

`vtool.get_invVR_mats_oris(invVR_mats)`

extracts orientation from matrix encoding, this is a bit trickier can use `-arctan2` or `(0, 0)` and `(0, 1)`, but then have to normalize

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> np.random.seed(0)
>>> invVR_mats = np.random.rand(7, 2, 2).astype(np.float64)
>>> output = get_invVR_mats_oris(invVR_mats)
>>> result = ub.repr2(output, precision=2, with_dtype=True)
```

`vtool.get_invVR_mats_shape(invVR_mats)`
Extracts keypoint shape components

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> np.random.seed(0)
>>> invVR_mats = np.random.rand(1000, 3, 3).astype(np.float64)
>>> output = get_invVR_mats_shape(invVR_mats)
>>> result = ut.hash_data(output)
>>> print(result)
pibujdiaimwcnmomserkcytyyikahjmp
```

References

TODO (a.ravel()[cols + (rows * a.shape[1]).reshape((-1,1)).ravel()]).reshape(rows.size, cols.size) <http://stackoverflow.com/questions/14386822/fast-numpy-fancy-indexing> # So, this doesn't work # Try this instead <http://docs.cython.org/src/userguide/memoryviews.html#memoryviews>

`vtool.get_invVR_mats_sqrd_scale(invVR_mats)`
Returns the squared scale of the invVR keypoints

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> np.random.seed(0)
>>> invVR_mats = np.random.rand(7, 3, 3).astype(np.float64)
>>> det_arr = get_invVR_mats_sqrd_scale(invVR_mats)
>>> result = ub.repr2(det_arr, precision=2, with_dtype=True)
>>> print(result)
np.array([-0.16, -0.09, -0.34, 0.59, -0.2 , 0.18, 0.06], dtype=np.float64)
```

`vtool.get_invVR_mats_xys(invVR_mats)`
extracts locations extracts xys from matrix encoding, Its just the (0, 2), and (1, 2) components

Parameters `invVR_mats` (*ndarray*) – list of matrices mapping uicircles to ellipses

Returns the xy location

Return type *ndarray*

Ignore:

```

>>> # DISABLE_DOCTEST
>>> import utool as ut
>>> setup = ut.codeblock(
...     '''
...     import numpy as np
...     np.random.seed(0)
...     invVR_mats = np.random.rand(1000, 3, 3).astype(np.float64)
...     '''
>>> stmt_list = ut.codeblock(
...     '''
...     invVR_mats[:, 0:2, 2].T
...     invVR_mats.T[2, 0:2]
...     invVR_mats.T.take(2, axis=0).take([0, 1], axis=0)
...     invVR_mats.T.take(2, axis=0)[0:2]
...     '''
... ).split('\n')
>>> ut.util_dev.timeit_compare(stmt_list, setup, int(1E5))

```

Example

```

>>> from vtool.keypoint import * # NOQA
>>> np.random.seed(0)
>>> invVR_mats = np.random.rand(1000, 3, 3).astype(np.float64)
>>> invVR_mats.T[2, 0:2]

```

`vtool.get_invV_mats(kpts, with_trans=False, with_ori=False, ashomog=False, ascontiguous=False)`

TODO: DEPRICATE. too many conditionals

packs keypoint shapes into affine invV matrixes (default is just the 2x2 shape. But translation, orientation, homogenous, and contiguous flags can be set.)

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> kpts = np.array([[10, 20, 1, 2, 3, 0]])
>>> with_trans=True
>>> with_ori=True
>>> ashomog=True
>>> ascontiguous=False
>>> innVR_mats = get_invV_mats(kpts, with_trans, with_ori, ashomog, ascontiguous)
>>> result = kpts_repr(innVR_mats)
>>> print(result)
array([[ 1.,  0., 10.],
       [ 2.,  3., 20.],
       [ 0.,  0.,  1.]])

```

`vtool.get_invV_mats2x2(kpts)`

Returns the keypoint shape (from unit circle to ellipse) Ignores translation and rotation component

Parameters `kpts` (`ndarray[float32_t, ndim=2]`) – keypoints

Returns `invV_mats`

Return type `ndarray[float32_t, ndim=3]`

CommandLine: `python -m vtool.keypoint --test-get_invV_mats2x2`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> kpts = np.array([
...     [0, 0, 1, 2, 3, 0],
...     [0, 0, 1, 2, 3, TAU / 4.0],
... ])
>>> invV_mats2x2 = get_invV_mats2x2(kpts)
>>> # verify results
>>> result = kpts_repr(invV_mats2x2)
```

`vtool.get_invV_mats3x3(kpts)`

NEWER FUNCTION

Returns full keypoint transform matrices from a unit circle to an ellipse that has been scaled, skewed, and translated. Into the image keypoint position.

DOES NOT INCLUDE ROTATION

Parameters `kpts` (`ndarray[float32_t, ndim=2]`) – keypoints

Returns `invVR_mats` - keypoint shape and rotations (possibly translation)

Return type `ndarray[float32_t, ndim=3]`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> kpts = np.array([
...     [0, 0, 1, 2, 3, 0],
...     [0, 0, 1, 2, 3, TAU / 4.0],
... ])
>>> invV_arrs3x3 = get_invV_mats3x3(kpts)
>>> # verify results
>>> result = kpts_repr(invV_arrs3x3)
```

`vtool.get_invVs(kpts)`

Keypoint shapes (oriented with the gravity vector)

`vtool.get_kdtree_flann_params()`

`vtool.get_kpts_dlen_sqrd(kpts, outer=False)`

returns diagonal length squared of keypoint extent

Parameters

- `kpts` (`ndarray[float32_t, ndim=2]`) – keypoints
- `outer` (`bool`) – loose if False tight if True

Returns `dlen_sqrd`

Return type `float`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts()
>>> dlen_sqrd = get_kpts_dlen_sqrd(kpts)
>>> result = '%.2f' % dlen_sqrd
>>> print(result)
3735.01
```

`vtool.get_kpts_dummy_img(kpts, sf=1.0, intensity=200)`

Parameters

- **kpts** (`ndarray[float32_t, ndim=2]`) – keypoints
- **sf** (`float`) –

Returns `img`

Return type `tuple`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.demodata import * # NOQA
>>> kpts = get_dummy_kpts()
>>> sf = 1.0
>>> img = get_kpts_dummy_img(kpts, sf, 10)
```

`vtool.get_kpts_eccentricity(kpts)`

SeeAlso: `pyhesaff.tests.test_ellipse`

References

[https://en.wikipedia.org/wiki/Eccentricity_\(mathematics\)](https://en.wikipedia.org/wiki/Eccentricity_(mathematics))

Notes

For an ellipse/hyperbola the eccentricity is $\sqrt{1 - (b^2 / a^2)}$

Eccentricity is undefined for parabolas

where *a* is the length of the semi-major axis and *b* is the length of the semi minor axis. The length of the semi-major axis is 2 times the largest eigenvalue. And the length of the semi-minor axis is 2 times the smallest eigenvalue.

Parameters

- **kpts** (`ndarray[float32_t, ndim=2]`) – keypoints
- **offset** (`tuple`) – (default = (0.0, 0.0))
- **scale_factor** (`float`) – (default = 1.0)

CommandLine: `python -m vtool.keypoint --exec-get_kpts_eccentricity --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts_ = vt.demodata.get_dummy_kpts()
>>> kpts = np.append(kpts_, [[10, 10, 5, 0, 5, 0]], axis=0)
>>> ecc = get_kpts_eccentricity(kpts)
>>> result = 'ecc = %s' % (ub.repr2(ecc, precision=2))
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> colors = pt.scores_to_color(ecc)
>>> pt.draw_kpts2(kpts, color=colors, ell_linewidth=6)
>>> extent = vt.get_kpts_image_extent(kpts)
>>> ax = pt.gca()
>>> pt.set_axis_extent(extent, ax)
>>> pt.dark_background()
>>> pt.colorbar(ecc, colors)
>>> ut.show_if_requested()
ecc = np.array([ 0.96, 0.99, 0.87, 0.91, 0.55, 0.  ])
```

`vtool.get_kpts_image_extent(kpts, outer=False, only_xy=False)`

returns the width and height of keypoint bounding box This combines xy and shape information Does not take into account if keypoint extent goes under (0, 0)

Parameters

- **kpts** (`ndarray[float32_t, ndim=2][ndims=2]`) – keypoints
- **outer** – uses outer rectangle if True. Set to false for a tighter extent.

Returns (minx, maxx, miny, maxy)

Return type tuple

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts()
>>> extent = get_kpts_image_extent(kpts, outer=False)
>>> result = ub.repr2(np.array(extent), precision=2)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.draw_kpts2(kpts, bbox=True)
>>> ax = pt.gca()
>>> pt.set_axis_extent(extent, ax)
>>> ut.show_if_requested()
np.array([ 14.78, 48.05,  0.32, 51.58])
```

`vtool.get_kpts_strs(kpts)`

`vtool.get_kpts_wh(kpts, outer=True)`

Gets the width / height diameter of a keypoint ie the diameter of the xaxis and yaxis of the keypoint.

Parameters

- **kpts** (`ndarray[float32_t, ndim=2][ndims=2]`) – keypoints
- **outer** (`bool`) – if `True` returns wh of bounding box. This is useful because extracting a patch needs a rectangle. If `false` it returns the otherwise gets the extent of the ellipse.

Returns (2xN) column1 is X extent and column2 is Y extent

Return type `ndarray`

Ignore:

```
>>> # Determine formula for min/maxing x and y
>>> import sympy
>>> x, y = sympy.symbols('x, y', real=True)
>>> a, d = sympy.symbols('a, d', real=True, positive=True)
>>> c = sympy.symbols('c', real=True)
>>> theta = sympy.symbols('theta', real=True, nonnegative=True)
>>> xeqn = sympy.Eq(x, a * sympy.cos(theta))
>>> yeqn = sympy.Eq(y, c * sympy.sin(theta) + v * d)
>>> dxdt = sympy.solve(sympy.diff(xeqn, theta), 0)
>>> dydt = sympy.solve(sympy.diff(yeqn, theta), 0)
>>>
>>> # Ugg, cant get sympy to do trig derivative, do it manually
>>> dxdt = -a * sin(theta)
>>> dydt = d * cos(theta) - c * sin(theta)
>>> critical_thetas = solve(Eq(dxdt, 0), theta)
>>> critical_thetas += solve(Eq(dydt, 0), theta)
>>> [a, _, c, d] = invV.ravel()
>>> critical_thetas = [
>>>     0, np.pi,
>>>     -2 * np.arctan((c + np.sqrt(c ** 2 + d ** 2)) / d),
>>>     -2 * np.arctan((c - np.sqrt(c ** 2 + d ** 2)) / d),
>>> ]
>>> critical_uvs = np.vstack([np.cos(critical_thetas),
>>>                             np.sin(critical_thetas)])
>>> critical_xys = invV.dot(critical_uvs)
```

SeeAlso: `get_kpts_major_minor`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts()[0:5]
>>> kpts[:, 0] += np.arange(len(kpts)) * 30
>>> kpts[:, 1] += np.arange(len(kpts)) * 30
>>> xyexnts = get_kpts_wh(kpts)
>>> result = ub.repr2(xyexnts)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.cla()
>>> pt.draw_kpts2(kpts, color='red', ell_linewidth=6, rect=True)
>>> ax = pt.gca()
>>> extent = np.array(get_kpts_image_extent(kpts))
>>> extent = vt.scale_extents(extent, 1.1)
```

(continues on next page)

(continued from previous page)

```

>>> pt.set_axis_extent(extent, ax)
>>> xs, ys = vt.get_xys(kpts)
>>> radii = xyexnts / 2
>>> horiz_pts1 = np.array([(xs - radii.T[0]), ys]).T
>>> horiz_pts2 = np.array([(xs + radii.T[0]), ys]).T
>>> vert_pts1 = np.array([xs, (ys - radii.T[1])]).T
>>> vert_pts2 = np.array([xs, (ys + radii.T[1])]).T
>>> pt.draw_line_segments2(horiz_pts1, horiz_pts2, color='g')
>>> pt.draw_line_segments2(vert_pts1, vert_pts2, color='b')
>>> ut.show_if_requested()
np.array([[10.43315411, 58.5216589 ],
          [ 4.71017647, 58.5216589 ],
          [24.43314171, 45.09558868],
          [26.71114159, 63.47679138],
          [32.10540009, 30.28536987]])

```

vttool.get_lat_lon (exif_dict, default=(-1, -1))

Returns the latitude and longitude, if available, from the provided exif_data2 (obtained through exif_data2 above)

Notes

Might need to downgrade to Pillow 2.9.0 to solve a bug with getting GPS <https://github.com/python-pillow/Pillow/issues/1477>

python -c "from PIL import Image; print(Image.PILLOW_VERSION)"

pip uninstall Pillow pip install Pillow==2.9

CommandLine: python -m vtool.exif --test-get_lat_lon

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.exif import * # NOQA
>>> import numpy as np
>>> image_fpath = ut.grab_file_url('http://images.summitpost.org/original/769474.
↳JPG')
>>> pil_img = Image.open(image_fpath)
>>> exif_dict = get_exif_dict(pil_img)
>>> latlon = get_lat_lon(exif_dict)
>>> result = np.array_str(np.array(latlon), precision=3)
>>> print(result)

```

vttool.get_left_area (ydata, xdata, index_list)

area to the left of each index point

vttool.get_match_spatial_squared_error (kpts1, kpts2, H, fx2_to_fx1)

transforms img2 to img1 and finds squared spatial error

Parameters

- **kpts1** (ndarray[float32_t, ndim=2]) – keypoints
- **kpts2** (ndarray[float32_t, ndim=2]) – keypoints

- `H(ndarray[float64_t, ndim=2])` – homography/perspective matrix mapping image 1 to image 2 space
- `fx2_to_fx1(ndarray)` – has shape (nMatch, K)

Returns `fx2_to_xyerr_sqrd` has shape (nMatch, K)

Return type `ndarray`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> kpts1 = np.array([[ 129.83,  46.97,  15.84,  4.66,  7.24,  0. ],
...                  [ 137.88,  49.87,  20.09,  5.76,  6.2 ,  0. ],
...                  [ 115.95,  53.13,  12.96,  1.73,  8.77,  0. ],
...                  [ 324.88, 172.58, 127.69, 41.29, 50.5 ,  0. ],
...                  [ 285.44, 254.61, 136.06, -4.77, 76.69,  0. ],
...                  [ 367.72, 140.81, 172.13, 12.99, 96.15,  0. ]],
dtype=np.float64)
>>> kpts2 = np.array([[ 318.93,  11.98,  12.11,  0.38,  8.04,  0. ],
...                  [ 509.47,  12.53,  22.4 ,  1.31,  5.04,  0. ],
...                  [ 514.03,  13.04,  19.25,  1.74,  4.72,  0. ],
...                  [ 490.19, 185.49,  95.67, -4.84, 88.23,  0. ],
...                  [ 316.97, 206.07,  90.87,  0.07, 80.45,  0. ],
...                  [ 366.07, 140.05, 161.27, -47.01, 85.62,  0. ]],
dtype=np.float64)
>>> H = np.array([[ -0.70098,  0.12273,  5.18734],
...               [  0.12444, -0.63474, 14.13995],
...               [  0.00004,  0.00025, -0.64873]])
>>> fx2_to_fx1 = np.array([[5, 4, 1, 0],
...                        [0, 1, 5, 4],
...                        [0, 1, 5, 4],
...                        [2, 3, 1, 5],
...                        [5, 1, 0, 4],
...                        [3, 1, 5, 0]], dtype=np.int32)
>>> fx2_to_xyerr_sqrd = get_match_spatial_squared_error(kpts1, kpts2, H, fx2_to_
dtype=np.float64)
>>> fx2_to_xyerr = np.sqrt(fx2_to_xyerr_sqrd)
>>> result = ub.repr2(fx2_to_xyerr, precision=3)
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> kpts1 = np.array([[ 6.,  4., 15.84,  4.66,  7.24,  0. ],
...                  [ 9.,  3., 20.09,  5.76,  6.2 ,  0. ],
...                  [ 1.,  1., 12.96,  1.73,  8.77,  0. ]])
>>> kpts2 = np.array([[ 2.,  1., 12.11,  0.38,  8.04,  0. ],
...                  [ 5.,  1., 22.4 ,  1.31,  5.04,  0. ],
...                  [ 6.,  1., 19.25,  1.74,  4.72,  0. ]])
>>> H = np.array([[ 2, 0, 0],
...               [ 0, 1, 0],
...               [ 0, 0, 1]])
>>> fx2_to_fx1 = np.array([[2, 1, 0],
```

(continues on next page)

(continued from previous page)

```

>>>             [0, 1, 2],
>>>             [2, 1, 0]], dtype=np.int32)
>>> fx2_to_xyerr_sqrd = get_match_spatial_squared_error(kpts1, kpts2, H, fx2_to_
↳fx1)
>>> fx2_to_xyerr = np.sqrt(fx2_to_xyerr_sqrd)
>>> result = ub.repr2(fx2_to_xyerr, precision=3)
>>> print(result)

```

`vtool.get_no_symbol` (*variant='symbol', size=(100, 100)*)

Returns errorimg

Return type ndarray

CommandLine: `python -m vtool.patch --test-get_no_symbol --show`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> errorimg = get_no_symbol()
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(errorimg)
>>> ut.show_if_requested()

```

`vtool.get_normalized_affine_inliers` (*kpts1, kpts2, fm, aff_inliers*)

returns xy-inliers that are normalized to have a mean of 0 and std of 1 as well as the transformations so the inverse can be taken

`vtool.get_num_channels` (*img*)

Returns the number of color channels

`vtool.get_ori_mats` (*kpts*)

Returns keypoint orientation matrixes

`vtool.get_ori_strs` (*kpts*)

`vtool.get_orientation` (*exif_dict, default=0, on_error='warn'*)

Returns the image orientation, if available, from the provided *exif_data2* (obtained through *exif_data2* above)

CommandLine: `python -m vtool.exif --test-get_orientation`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.exif import * # NOQA
>>> from os.path import join
>>> import numpy as np
>>> url = 'https://wildbookiarepository.azureedge.net/models/orientation.zip'
>>> images_path = ut.grab_zipped_url(url)
>>> result = []
>>> for index in range(3):
>>>     image_filename = 'orientation_%05d.JPG' % (index + 1, )
>>>     pil_img = Image.open(join(images_path, image_filename))

```

(continues on next page)

(continued from previous page)

```

>>> exif_dict = get_exif_dict(pil_img)
>>> orient = get_orientation(exif_dict)
>>> pil_img.close()
>>> result.append(orient)
>>> print(result)
[1, 6, 8]

```

`vtool.get_orientation_histogram(gori, gori_weights, bins=36, DEBUG_ROTINVAR=False)`

Parameters

- **gori** –
- **gori_weights** –
- **bins** (*int*) –

Returns (hist, centers)

Return type `tuple`

CommandLine: `python -m vtool.patch -test-get_orientation_histogram`

Ignore: `print(vt.kpts_docrepr(gori, 'gori = ')) print(vt.kpts_docrepr(gori_weights, 'gori_weights = '))`

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> # build test data
>>> gori = np.array([[ 0. ,  0. ,  3.14,  3.14,  0. ],
...                 [ 4.71,  6.15,  3.13,  3.24,  4.71],
...                 [ 4.71,  4.61,  0.5 ,  4.85,  4.71],
...                 [ 1.57,  6.28,  3.14,  3.14,  1.57],
...                 [ 0. ,  0. ,  3.14,  3.14,  0. ]])
>>> gori_weights = np.array([[ 0. ,  0.11,  0.02,  0.13,  0. ],
...                          [ 0.02,  0.19,  0.02,  0.21,  0.02],
...                          [ 0.11,  0.16,  0. ,  0.13,  0.11],
...                          [ 0. ,  0.17,  0.02,  0.19,  0. ],
...                          [ 0. ,  0.11,  0.02,  0.13,  0. ]])
>>> bins = 36
>>> # execute function
>>> (hist, centers) = get_orientation_histogram(gori, gori_weights, bins)
>>> # verify results
>>> result = str((hist, centers))
>>> print(result)

```

`vtool.get_orientation_str(exif_dict, **kwargs)`

Returns the image orientation strings, if available, from the provided `exif_data2` (obtained through `exif_data2` above)

CommandLine: `python -m vtool.exif -test-get_orientation_str`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.exif import * # NOQA
>>> from os.path import join
>>> import numpy as np
>>> url = 'https://wildbookiarepository.azureedge.net/models/orientation.zip'
>>> images_path = ut.grab_zipped_url(url)
>>> result = []
>>> for index in range(3):
>>>     image_filename = 'orientation_%05d.JPG' % (index + 1, )
>>>     pil_img = Image.open(join(images_path, image_filename))
>>>     exif_dict = get_exif_dict(pil_img)
>>>     orient_str = get_orientation_str(exif_dict)
>>>     pil_img.close()
>>>     result.append(orient_str)
>>> print(result)
['Normal', '90 Clockwise', '90 Counter-Clockwise']
```

`vtool.get_oris(kpts)`

Extracts keypoint orientations for kpts array

(in isotropic guassian space relative to the gravity vector) (in simpler words: the orientation is taken from keypoints warped to the unit circle)

Parameters `kpts` (*ndarray*) – (N x 6) [x, y, a, c, d, theta]

Returns (*ndarray*) theta

`vtool.get_pixel_dist(img, pixel, channel=None)`

pixel = fillval isfill = mask2d

`vtool.get_pointset_extent_wh(pts)`

`vtool.get_pointset_extents(pts)`

`vtool.get_right_area(ydata, xdata, index_list)`

area to the right of each index point

`vtool.get_round_scaled_dsize(dsize, scale)`

Returns an integer size and scale that best approximates the floating point scale on the original size

Parameters

- **dsize** (*tuple*) – original width height
- **scale** (*float or tuple*) – desired floating point scale factor

`vtool.get_scale_factor(src_img, dst_img)`

returns scale factor from one image to the next

`vtool.get_scaled_size_with_dlen(target_dlen, w, h)`

returns new_size which scales (w, h) as close to target_dlen as possible and maintains aspect ratio

`vtool.get_scales(kpts)`

Gets average scale (does not take into account elliptical shape)

`vtool.get_shape_strs(kpts)`

strings debugging and output

`vtool.get_size(img)`

Returns the image size in (width, height)

`vtool.get_sqrd_scales(kpts)`

gets average squared scale (does not take into account elliptical shape)

Parameters `kpts` (`ndarray[float32_t, ndim=2]`) – keypoints

Returns `np.ndarray`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts()
>>> _scales_sqr = get_sqr_scales(kpts)
>>> result = (ub.repr2(_scales_sqr, precision=2))
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> _scales_sqr = get_sqr_scales([])
>>> result = (ub.repr2(_scales_sqr, precision=2))
>>> print(result)
```

```
vtool.get_star2_patch(jitter=False)
test data patch
```

```
vtool.get_star_patch(jitter=False)
test data patch
```

```
vtool.get_stripe_patch(jitter=False)
test data patch
```

```
vtool.get_test_patch(key='star', jitter=False)
```

Parameters

- `key` (`str`) –
- `jitter` (`bool`) –

Returns `patch`

Return type `ndarray`

CommandLine: `python -m vtool.patch --test-get_test_patch --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> import wbia.plottool as pt
>>> key = 'star2'
>>> jitter = False
>>> patch = get_test_patch(key, jitter)
>>> pt.imshow(255 * patch)
>>> pt.show_if_requested()
```

```
vtool.get_testdata_kpts(fname=None, with_vecs=False)
```

`vtool.get_transforms_from_patch_image_kpts(kpts, patch_shape, scale_factor=1.0)`

Given some patch (like a gaussian patch) transforms a patch to be overlayed on top of each keypoint in the image (adjusted for a scale factor)

Parameters

- **kpts** (`ndarray[float32_t, ndim=2]`) – keypoints
- **patch_shape** –
- **scale_factor** (`float`) –

Returns a list of 3x3 tranformation matrices for each keypoint

Return type M_list

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts()
>>> patch_shape = (7, 7)
>>> scale_factor = 1.0
>>> M_list = get_transforms_from_patch_image_kpts(kpts, patch_shape, scale_
↳ factor)
>>> # verify results
>>> result = kpts_repr(M_list)
```

`vtool.get_uncovered_mask(covered_array, covering_array)`

Parameters

- **covered_array** (`ndarray`) –
- **covering_array** (`ndarray`) –

Returns flags

Return type ndarray

CommandLine: `python -m vtool.other -test-get_uncovered_mask`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> covered_array = [1, 2, 3, 4, 5]
>>> covering_array = [2, 4, 5]
>>> flags = get_uncovered_mask(covered_array, covering_array)
>>> result = str(flags)
>>> print(result)
[ True False  True False False]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> covered_array = [1, 2, 3, 4, 5]
>>> covering_array = []
>>> flags = get_uncovered_mask(covered_array, covering_array)
>>> result = str(flags)
>>> print(result)
[ True True True True True]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> covered_array = np.array([
...     [1, 2, 3],
...     [4, 5, 6],
...     [7, 8, 9],
... ], dtype=np.int32)
>>> covering_array = [2, 4, 5]
>>> flags = get_uncovered_mask(covered_array, covering_array)
>>> result = ub.repr2(flags, with_dtype=True)
>>> print(result)
np.array([[ True, False,  True],
         [False, False,  True],
         [ True,  True,  True]], dtype=np.bool)
```

Ignore: covering_array = [1, 2, 3, 4, 5, 6, 7] %timeit get_uncovered_mask(covered_array, covering_array)
 100000 loops, best of 3: 18.6 µs per loop %timeit get_uncovered_mask2(covered_array, covering_array)
 100000 loops, best of 3: 16.9 µs per loop

`vtool.get_undirected_edge_ids` (*directed_edges*)

Parameters *directed_edges* (*ndarray* [*ndims*=2]) –

Returns *edgeid_list*

Return type *list*

CommandLine: `python -m vtool.other --exec-get_undirected_edge_ids`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> directed_edges = np.array([[1, 2], [2, 1], [2, 3], [3, 1], [1, 1], [2, 3], [3,
↪ 2]])
>>> edgeid_list = get_undirected_edge_ids(directed_edges)
>>> result = ('edgeid_list = %s' % (ub.repr2(edgeid_list),))
>>> print(result)
edgeid_list = [0 0 1 2 3 1 1]
```

`vtool.get_uneven_point_sample` (*kpts*)

for each keypoint returns an uneven sample of points along the elliptical boundries.

Parameters `kpts` (`ndarray[float32_t, ndim=2]`) – keypoints

SeeAlso: `pyhesaff.tests.test_ellipse python -m pyhesaff.tests.test_ellipse --test-in_depth_ellipse --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts()[0:2]
>>> ellipse_pts1 = get_uneven_point_sample(kpts)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.draw_line_segments(ellipse_pts1)
>>> pt.set_title('uneven sample points')
>>> pt.show_if_requested()
```

`vtool.get_unixtime(exif_dict, default=-1)`
TODO: Exif.Image.TimeZoneOffset

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.exif import * # NOQA
>>> image_fpath = ut.grab_file_url('http://images.summitpost.org/original/769474.
↳JPG')
>>> pil_img = Image.open(image_fpath)
>>> exif_dict = get_exif_dict(pil_img)
>>> pil_img.close()
```

`vtool.get_unixtime_gps(exif_dict, default=-1)`

`vtool.get_unwarped_patch(imgBGR, kp, gray=False)`
Returns unwarped warped patch around a keypoint

Parameters

- `img` (`ndarray`) – array representing an image
- `kpt` (`ndarray`) – keypoint ndarray in [x, y, a, c, d, theta] format

Returns (`wpatch, wkp`) the normalized 41x41 patches from the `img` corresponding to the keypoint

Return type `tuple`

`vtool.get_unwarped_patches(img, kpts)`

Returns cropped unwarped (keypoint is still elliptical) patch around a keypoint

Parameters

- `img` (`ndarray`) – array representing an image
- `kpts` (`ndarrays`) – keypoint ndarrays in [x, y, a, c, d, theta] format

Returns

(`patches, subkpts`) - the unnormalized patches from the `img` corresponding to the keypoint

Return type `tuple`

`vtool.get_warped_patch (imgBGR, kp, gray=False, flags=4, borderMode=1, patch_size=41)`

Returns warped (into a unit circle) patch around a keypoint

Parameters

- **img** (*ndarray*) – array representing an image
- **kpt** (*ndarray*) – keypoint ndarray in [x, y, a, c, d, theta] format

Returns

(wpatch, wkpt) the normalized 41x41 patches from the img corresponding to the keypoint

Return type (*ndarray, ndarray*)

`vtool.get_warped_patches (img, kpts, flags=4, borderMode=1, patch_size=41, use_cpp=False)`

Returns warped (into a unit circle) patch around a keypoint

FIXME: there is a slight translation difference in the way Python extracts patches and the way C++ extracts patches. C++ should be correct. TODO: have C++ able to extract color.

Parameters

- **img** (*ndarray [uint8_t, ndim=2]*) – array representing an image
- **kpts** (*ndarray [float32_t, ndim=2]*) – list of keypoint ndarrays in [[x, y, a, c, d, theta]] format
- **flags** (*long*) – cv2 interpolation flags
- **borderMode** (*long*) – cv2 border flags
- **patch_size** (*int*) – resolution of resulting image patch

Returns

(warped_patches, warped_subkpts) the normalized 41x41 patches from the img corresponding to the keypoint

Return type (*list, list*)

CommandLine: `python -m vtool.patch --test-get_warped_patches --show --use_cpp python -m vtool.patch --test-get_warped_patches --show --use_python`

Example

```
>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.patch import * # NOQA
>>> import vtool as vt
>>> import ubelt as ub
>>> # build test data
>>> img_fpath = ut.grab_test_imgpath('car1.jpg')
>>> img = vt.imread(img_fpath)
>>> use_cpp = ut.get_argflag('--use_cpp')
>>> kpts, desc = vt.extract_features(img_fpath)
>>> kpts = kpts[0:1]
>>> flags = cv2.INTER_LANCZOS4
>>> borderMode = cv2.BORDER_REPLICATE
>>> # execute function
>>> (warped_patches, warped_subkpts) = get_warped_patches(img, kpts, flags,
↳borderMode, use_cpp=use_cpp)
```

(continues on next page)

(continued from previous page)

```

>>> # verify results
>>> print(np.array(warped_patches).shape)
>>> print(ub.repr2(np.array(warped_subkpts), precision=2))
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(warped_patches[0])
>>> #pt.draw_kpts2(warped_subkpts, pts=True, rect=True)
>>> pt.set_title('use_cpp = %r' % (use_cpp,))
>>> pt.show_if_requested()

```

`vtool.get_xy_strs(kpts)`
strings debugging and output

`vtool.get_xys(kpts)`
Keypoint locations in chip space

`vtool.grab_webcam_image()`

References

http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html

CommandLine: `python -m vtool.other --test-grab_webcam_image --show`

Example

```

>>> # SCRIPT
>>> from vtool.other import * # NOQA
>>> import vtool as vt
>>> img = grab_webcam_image()
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(img)
>>> vt.imwrite('webcap.jpg', img)
>>> ut.show_if_requested()

```

`vtool.gradient_fill(shape, theta=0, flip=False, vert=False, style='linear')`
FIXME: angle does not work properly

CommandLine: `python -m vtool.patch gradient_fill --show`

Example

```

>>> from vtool.patch import * # NOQA
>>> import vtool as vt
>>> shape = (9, 9)
>>> #style = 'linear'
>>> style = 'step'
>>> theta = np.pi / 4
>>> patch = vt.gradient_fill(shape, theta, style=style)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt

```

(continues on next page)

(continued from previous page)

```
>>> pt.imshow(vt.rectify_to_uint8(patch))
>>> ut.show_if_requested()
```

`vtool.gradient_magnitude(img)`

`vtool.greedy_setcover(universe, subsets, weights=None)`

Copied implementation of greedy set cover from stack overflow. Needs work.

References

<http://stackoverflow.com/questions/7942312/of-greedy-set-cover-faster>

Example

```
>>> # SLOW_DOCTEST
>>> # xdoctest: +SKIP
>>> from vtool.other import * # NOQA
>>> import vtool as vt
>>> universe = set([1,2,3,4])
>>> subsets = [set([1,2]), set([1]), set([1,2,3]), set([1]), set([3,4]),
>>>             set([4]), set([1,2]), set([3,4]), set([1,2,3,4])]
>>> weights = [1, 1, 2, 2, 2, 3, 3, 4, 4]
>>> chosen, costs = greedy_setcover(universe, subsets, weights)
>>> print('Cover: %r' % (chosen,))
>>> print('Total Cost: %r=sum(%r)' % (sum(costs), costs))
```

`vtool.gridsearch_addWeighted()`

CommandLine: `xdoctest -m ~/code/vtool/vtool/blend.py gridsearch_addWeighted`

`vtool.gridsearch_chipextract()`

CommandLine: `xdoctest -m ~/code/vtool/vtool/chip.py gridsearch_chipextract --show`

Example

```
>>> # DISABLE_DOCTEST
>>> # GRIDSEARCH
>>> from vtool.chip import * # NOQA
>>> gridsearch_chipextract()
>>> ut.show_if_requested()
```

`vtool.gridsearch_image_function(param_info, test_func, args=(), show_func=None)`
gridsearch for a function that produces a single image

`vtool.group_consecutive(arr)`

Returns lists of consecutive values

References

<http://stackoverflow.com/questions/7352684/how-to-find-the-groups-of-consecutive-elements-from-an-array-in-numpy>

Parameters `arr` (`ndarray`) – must be integral and unique

Returns arr -

Return type ndarray

CommandLine: python -m vtool.util_math --exec-group_consecutive

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> arr = np.array([1, 2, 3, 5, 6, 7, 8, 9, 10, 15, 99, 100, 101])
>>> groups = group_consecutive(arr)
>>> result = ('groups = %s' % (str(groups),))
>>> print(result)
groups = [array([1, 2, 3]), array([ 5,  6,  7,  8,  9, 10]), array([15]), array([
↪99, 100, 101])]
```

vtool.group_indices (idx2_groupid, assume_sorted=False)

Parameters idx2_groupid (ndarray) – numpy array of group ids (must be numeric)

Returns (keys, groupxs)

Return type tuple (ndarray, list of ndarrays)

CommandLine: xdoctest -m ~/code/vtool/vtool/clustering2.py group_indices xdoctest -m ~/code/vtool/vtool/clustering2.py group_indices:0 xdoctest -m ~/code/vtool/vtool/clustering2.py group_indices:1

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.clustering2 import * # NOQA
>>> idx2_groupid = np.array([2, 1, 2, 1, 2, 1, 2, 3, 3, 3, 3])
>>> (keys, groupxs) = group_indices(idx2_groupid)
>>> result = ut.repr2((keys, groupxs), nl=2, nobr=True, with_dtype=True)
>>> print(result)
np.array([1, 2, 3], dtype=np.int64),
[
  np.array([1, 3, 5], dtype=np.int64),
  np.array([0, 2, 4, 6], dtype=np.int64),
  np.array([ 7,  8,  9, 10], dtype=np.int64),
],
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.clustering2 import * # NOQA
>>> idx2_groupid = np.array([[ 24], [ 129], [ 659], [ 659], [ 24],
... [ 659], [ 659], [ 822], [ 659], [ 659], [24]])
>>> # 2d arrays must be flattened before coming into this function so
>>> # information is on the last axis
>>> (keys, groupxs) = group_indices(idx2_groupid.T[0])
>>> result = ut.repr2((keys, groupxs), nl=2, nobr=True, with_dtype=True)
```

(continues on next page)

(continued from previous page)

```
>>> print(result)
np.array([ 24, 129, 659, 822], dtype=np.int64),
[
  np.array([ 0,  4, 10], dtype=np.int64),
  np.array([1], dtype=np.int64),
  np.array([2, 3, 5, 6, 8, 9], dtype=np.int64),
  np.array([7], dtype=np.int64),
],
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.clustering2 import * # NOQA
>>> idx2_groupid = np.array([True, True, False, True, False, False, True])
>>> (keys, groupxs) = group_indices(idx2_groupid)
>>> result = ut.repr2((keys, groupxs), nl=2, nobr=True, with_dtype=True)
>>> print(result)
np.array([False,  True], dtype=np.bool),
[
  np.array([2, 4, 5], dtype=np.int64),
  np.array([0, 1, 3, 6], dtype=np.int64),
],
```

Time:

```
>>> # xdoctest: +SKIP
>>> import vtool as vt
>>> setup = ut.extract_timeit_setup(vt.group_indices, 2, 'groupxs =')
>>> print(setup)
>>> stmt_list = ut.codeblock(
'''
    [sortx[lx:rx] for lx, rx in ut.itertwo(idxs)]
    [sortx[lx:rx] for lx, rx in zip(idxs, idxs[1:])]
    #[sortx[lx:rx] for lx, rx in ut.iter_window(idxs)]
    #[sortx[slice(*_)] for _ in ut.itertwo(idxs)]
    #[sortx[slice(lr, lx)] for lr, lx in ut.itertwo(idxs)]
    #np.split(sortx, idxs[1:-1])
    #np.hsplit(sortx, idxs[1:-1])
    np.array_split(sortx, idxs[1:-1])
''').split('\n')
>>> stmt_list = [x for x in stmt_list if not x.startswith('#')]
>>> passed, times, outputs = ut.timeit_compare(stmt_list, setup,
↳ iterations=10000)
```

```
>>> # xdoctest: +SKIP
>>> stmt_list = ut.codeblock(
'''
    np.diff(groupids_sorted)
    np.ediff1d(groupids_sorted)
    np.subtract(groupids_sorted[1:], groupids_sorted[:-1])
''').split('\n')
>>> stmt_list = [x for x in stmt_list if not x.startswith('#')]
>>> passed, times, outputs = ut.timeit_compare(stmt_list, setup,
↳ iterations=10000)
```

Ignore: `import numba group_indices_numba = numba.jit(group_indices)`
`group_indices_numba(idx2_groupid)`

SeeAlso: `apply_grouping`

References

<http://stackoverflow.com/questions/4651683/numpy-grouping-using-itertools-groupby-performance>

Todo: Look into `np.split` <http://stackoverflow.com/questions/21888406/getting-the-indexes-to-the-duplicate-columns-of-a-numpy-array>

`vtool.groupby(items, idx2_groupid)`

```
>>> items = np.array(np.arange(100))
>>> idx2_groupid = np.array(np.random.randint(0, 4, size=100))
>>> items = idx2_groupid
```

`vtool.groupby_dict(items, idx2_groupid)`

`vtool.groupby_gen(items, idx2_groupid)`

```
>>> items = np.array(np.arange(100))
>>> idx2_groupid = np.array(np.random.randint(0, 4, size=100))
```

`vtool.groupedzip(id_list, datas_list)`

Function for grouping multiple lists of data (stored in `datas_list`) using `id_list`.

Parameters

- `id_list(list)` –
- `datas_list(list)` –

Returns `_iter`

Return type `iterator`

CommandLine: `python -m vtool.clustering2 --test-groupedzip`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.clustering2 import * # NOQA
>>> # build test data
>>> id_list = np.array([1, 2, 1, 2, 1, 2, 3])
>>> datas_list = [
...     ['a', 'b', 'c', 'd', 'e', 'f', 'g'],
...     ['A', 'B', 'C', 'D', 'E', 'F', 'G'],
... ]
>>> # execute function
>>> groupxs, grouped_iter = groupedzip(id_list, datas_list)
>>> grouped_tuples = list(grouped_iter)
>>> # verify results
```

(continues on next page)

(continued from previous page)

```

>>> result = str(groupxs) + '\n'
>>> result += ub.repr2(grouped_tuples, nl=1)
>>> print(result)
[1 2 3]
[
    (['a', 'c', 'e'], ['A', 'C', 'E']),
    (['b', 'd', 'f'], ['B', 'D', 'F']),
    (['g'], ['G']),
]

```

`vtool.haversine(latlon1, latlon2)`

Calculate the great circle distance between two points on the earth (specified in decimal degrees)

Parameters

- `latlon1` (*ndarray*) –
- `latlon2` (*ndarray*) –

References

en.wikipedia.org/wiki/Haversine_formula gis.stackexchange.com/questions/81551/matching-gps-tracks
stackoverflow.com/questions/4913349/haversine-distance-gps-points

CommandLine: `python -m vtool.distance --exec-haversine`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> import scipy.spatial.distance as spdist
>>> import vtool as vt
>>> import functools
>>> gpsarr_track_list_ = [
...     np.array([[ -80.21895315, -158.81099213],
...               [ -12.08338926,  67.50368014],
...               [ -11.08338926,  67.50368014],
...               [ -11.08338926,  67.50368014],
...               ],
...     ),
...     np.array([[  9.77816711, -17.27471498],
...               [ -51.67678814, -158.91065495],
...               ],
...     )
>>> latlon1 = gpsarr_track_list_[0][0]
>>> latlon2 = gpsarr_track_list_[0][1]
>>> kilometers = vt.haversine(latlon1, latlon2)
>>> haversin_pdist = functools.partial(spdist.pdist, metric=vt.haversine)
>>> dist_vector_list = list(map(haversin_pdist, gpsarr_track_list_))
>>> dist_matrix_list = list(map(spdist.squareform, dist_vector_list))

```

`vtool.hist_argmaxima(hist, centers=None, maxima_thresh=None)`
 must take positive only values

CommandLine: `python -m vtool.histogram hist_argmaxima`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> maxima_thresh = .8
>>> hist = np.array([ 6.73, 8.69, 0.00, 0.00, 34.62, 29.16, 0.00, 0.00, 6.73,
↳8.69])
>>> centers = np.array([-0.39, 0.39, 1.18, 1.96, 2.75, 3.53, 4.32, 5.11, 5.89,
↳6.68])
>>> maxima_x, maxima_y, argmaxima = hist_argmaxima(hist, centers)
>>> result = str((maxima_x, maxima_y, argmaxima))
>>> print(result)
```

`vtool.hist_argmaxima2(hist, maxima_thresh=0.8)`
must take positive only values

Setup:

```
>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
```

GridSearch:

```
>>> hist1 = np.array([1, .9, .8, .99, .99, 1.1, .9, 1.0, 1.0])
>>> hist2 = np.array([1, .9, .8, .99, .99, 1.1, 1.0, 1.0])
>>> hist2 = np.array([1, .9, .8, .99, .99, 1.1, 1.0])
>>> hist2 = np.array([1, .9, .8, .99, .99, 1.1, 1.2])
>>> hist2 = np.array([1, 1.2])
>>> hist2 = np.array([1, 1, 1.2])
>>> hist2 = np.array([1])
>>> hist2 = np.array([])
```

Example

```
>>> # ENABLE_DOCTEST
>>> maxima_thresh = .8
>>> hist = np.array([1, .9, .8, .99, .99, 1.1, .9, 1.0, 1.0])
>>> argmaxima = hist_argmaxima2(hist)
>>> print(argmaxima)
```

`vtool.hist_edges_to_centers(edges)`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> edges = [-0.79, 0.00, 0.79, 1.57, 2.36, 3.14, 3.93, 4.71, 5.50, 6.28, 7.07]
>>> centers = hist_edges_to_centers(edges)
>>> result = str(centers)
>>> print(result)
[-0.395 0.395 1.18 1.965 2.75 3.535 4.32 5.105 5.89 6.675]
```

`vtool.hist_isect(hist1, hist2)`
returns histogram intersection distance between two histograms

`vtool.homogenous_circle_pts(nSamples)`

Make a list of homogenous circle points

`vtool.iceil(num, dtype=<class 'numpy.int32'>)`

Integer ceiling. (because numpy doesn't seem to have it!)

Parameters *num* (*ndarray* or *scalar*) –

Returns

Return type *ndarray* or *scalar*

CommandLine: `python -m vtool.util_math -test-iceil`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> num = 1.5
>>> result = repr(iceil(num))
>>> print(result)
2
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> import ubelt as ub
>>> num = [1.5, 2.9]
>>> result = ub.repr2(iceil(num), with_dtype=True)
>>> print(result)
np.array([2, 3], dtype=np.int32)
```

`vtool.imread(img_fpath, grayscale=False, orient=False, flags=None, force_pil=None, delete_if_corrupted=False, **kwargs)`

Wrapper around the opencv imread function. Handles remote uris.

Parameters

- **img_fpath** (*str*) – file path string
- **grayscale** (*bool*) – (default = False)
- **orient** (*bool*) – (default = False)
- **flags** (*None*) – opencv flags (default = None)
- **force_pil** (*bool*) – (default = None)
- **delete_if_corrupted** (*bool*) – (default = False)

Returns *imgBGR*

Return type *ndarray*

CommandLine: `python -m vtool.image -test-imread python -m vtool.image -test-imread:1 python -m vtool.image -test-imread:2`

References

http://docs.opencv.org/modules/core/doc/utility_and_system_functions_and_macros.html#error
<http://stackoverflow.com/questions/23572241/cv2-threshold-error-210>

[http:](http://)

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> img_fpath = ut.grab_test_imgpath('carl.jpg')
>>> imgBGR1 = imread(img_fpath, grayscale=False)
>>> imgBGR2 = imread(img_fpath, grayscale=True)
>>> imgBGR3 = imread(img_fpath, orient=True)
>>> assert imgBGR1.shape == (250, 300, 3)
>>> assert imgBGR2.shape == (250, 300)
>>> # assert np.all(imgBGR1 == imgBGR3)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(imgBGR1, pnum=(2, 2, 1))
>>> pt.imshow(imgBGR2, pnum=(2, 2, 2))
>>> pt.imshow(imgBGR3, pnum=(2, 2, 3))
>>> ut.show_if_requested()
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> img_url = 'http://images.summitpost.org/original/769474.JPG'
>>> img_fpath = ut.grab_file_url(img_url)
>>> imgBGR1 = imread(img_url)
>>> imgBGR2 = imread(img_fpath)
>>> #imgBGR2 = imread(img_fpath, force_pil=False, flags=cv2.IMREAD_UNCHANGED)
>>> print('imgBGR.shape = %r' % (imgBGR1.shape,))
>>> print('imgBGR2.shape = %r' % (imgBGR2.shape,))
>>> result = str(imgBGR1.shape)
>>> diff_pxls = imgBGR1 != imgBGR2
>>> num_diff_pxls = diff_pxls.sum()
>>> print(result)
>>> print('num_diff_pxls=%r/%r' % (num_diff_pxls, diff_pxls.size))
>>> assert num_diff_pxls == 0
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> diffMag = np.linalg.norm(imgBGR2 / 255. - imgBGR1 / 255., axis=2)
>>> pt.imshow(imgBGR1, pnum=(1, 3, 1))
>>> pt.imshow(diffMag / diffMag.max(), pnum=(1, 3, 2))
>>> pt.imshow(imgBGR2, pnum=(1, 3, 3))
>>> ut.show_if_requested()
(2736, 3648, 3)
```

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> url = 'http://www.sherv.net/cm/emo/funny/2/big-dancing-banana-smiley-emoticon.
↳gif'
>>> img_fpath = ut.grab_file_url(url)
>>> delete_if_corrupted = False
>>> grayscale = False
>>> imgBGR = imread(img_fpath, grayscale=grayscale)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(imgBGR)
>>> ut.show_if_requested()

```

`vtool.imread_remote_s3(img_fpath, **kwargs)`

`vtool.imread_remote_url(img_url, **kwargs)`

`vtool.imwrite(img_fpath, imgBGR, fallback=False)`

References

http://docs.opencv.org/modules/highgui/doc/reading_and_writing_images_and_video.html

Parameters

- **img_fpath** (*str*) – file path string
- **imgBGR** (*ndarray[uint8_t, ndim=2]*) – image data in opencv format (blue, green, red)
- **fallback** (*bool*) – (default = False)

CommandLine: `python -m vtool.image --exec-implode`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> import utool as ut
>>> img_fpath1 = ut.grab_test_imgpath('zebra.png')
>>> imgBGR = vt.imread(img_fpath1)
>>> img_dpath = ub.ensure_app_cache_dir('vtool', 'testwrite')
>>> img_fpath2 = ut.unixjoin(img_dpath, 'zebra.png')
>>> fallback = False
>>> imwrite(img_fpath2, imgBGR, fallback=fallback)
>>> imgBGR2 = vt.imread(img_fpath2)
>>> assert np.all(imgBGR2 == imgBGR)

```

`vtool.imwrite_fallback(img_fpath, imgBGR)`

`vtool.inbounds(num, low, high, eq=False)`

Parameters

- **num** (*scalar or ndarray*) –
- **low** (*scalar or ndarray*) –

- **high**(*scalar or ndarray*) –
- **eq**(*bool*) –

Returns *is_inbounds*

Return type *scalar or ndarray*

CommandLine: `xdoctest -m ~/code/vtool/vtool/other.py inbounds`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> import utool as ut
>>> num = np.array([[ 0.    ,  0.431,  0.279],
...                [ 0.204,  0.352,  0.08 ],
...                [ 0.107,  0.325,  0.179]])
>>> low  = .1
>>> high = .4
>>> eq = False
>>> is_inbounds = inbounds(num, low, high, eq)
>>> result = ub.repr2(is_inbounds, with_dtype=True)
>>> print(result)
```

`vtool.index_partition`(*item_list, part1_items*)

returns two lists. The first are the indices of items in *item_list* that are in *part1_items*. the second is the indices in *item_list* that are not in *part1_items*. items in *part1_items* that are not in *item_list* are ignored

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> item_list = ['dist', 'fg', 'distinctiveness']
>>> part1_items = ['fg', 'distinctiveness']
>>> part1_indexes, part2_indexes = index_partition(item_list, part1_items)
>>> ut.assert_eq(part1_indexes.tolist(), [1, 2])
>>> ut.assert_eq(part2_indexes.tolist(), [0])
```

`vtool.index_to_boolmask`(*index_list, maxval=None, isflat=True*)

transforms a list of indices into a boolean mask

Parameters

- **index_list**(*ndarray*) –
- **maxval**(*None*) – (default = None)

Kwargs: *maxval*

Returns *mask*

Return type *ndarray*

CommandLine: `python -m vtool.util_numpy index_to_boolmask`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.util_numpy import * # NOQA
>>> import vtool as vt
>>> index_list = np.array([(0, 0), (1, 1), (2, 1)])
>>> maxval = (3, 3)
>>> mask = vt.index_to_boolmask(index_list, maxval, isflat=False)
>>> result = ('mask =\n%s' % (str(mask.astype(np.uint8)),))
>>> print(result)
[[1 0 0]
 [0 1 0]
 [0 1 0]]
```

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.util_numpy import * # NOQA
>>> import vtool as vt
>>> index_list = np.array([0, 1, 4])
>>> maxval = 5
>>> mask = vt.index_to_boolmask(index_list, maxval, isflat=True)
>>> result = ('mask = %s' % (str(mask.astype(np.uint8)),))
>>> print(result)
mask = [1 1 0 0 1]
```

`vtool.infer_vert (img1, img2, vert)`
which is the better stack dimension

`vtool.inspect_pdfs (tn_support, tp_support, score_domain, p_tp_given_score, p_tn_given_score, p_score_given_tp, p_score_given_tn, p_score, prob_thresh=None, score_thresh=None, with_scores=False, with_roc=False, with_precision_recall=False, with_hist=False, fnum=None, figtitle=None, interactive=None, use_stems=None, part_attrs=None, thresh_kw=None, attr_callback=None, with_prebayes=True, with_postbayes=True, score_range=None, **kwargs)`

Shows plots of learned thresholds

CommandLine: `python -m vtool.score_normalization --test-ScoreNormalizer --show python -m vtool.score_normalization --exec-ScoreNormalizer.visualize --show`

`vtool.interact_roc_factory (confusions, target_tpr=None, show_operating_point=False)`

Parameters `confusions` (`Confusions`) –

CommandLine: `python -m vtool.confusion --exec-interact_roc_factory --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.confusion import * # NOQA
>>> scores, labels = testdata_scores_labels()
>>> print('scores = %r' % (scores,))
>>> confusions = ConfusionMetrics().fit(scores, labels)
>>> print(ut.make_csv_table(
```

(continues on next page)

(continued from previous page)

```

>>> [confusions.fpr, confusions.tpr, confusions.thresholds],
>>> ['fpr', 'tpr', 'thresh'])
>>> # xdoctest: +REQUIRES(--show)
>>> ROCInteraction = interact_roc_factory(confusions, target_tpr=.4, show_
↳operating_point=True)
>>> inter = ROCInteraction()
>>> inter.show_page()
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> ut.show_if_requested()

```

`vtool.intern_warp_single_patch(img, x, y, ori, V, patch_size, flags=2, borderMode=1)`

Ignore:

```

>>> # https://groups.google.com/forum/#!topic/sympy/k1HnZK_bNNA
>>> from vtool.patch import * # NOQA
>>> import sympy
>>> from sympy.abc import theta
>>> ori = theta
>>> x, y, a, c, d, patch_size = sympy.symbols('x y a c d S')
>>> half_patch_size = patch_size / 2
>>>
>>> def sympy_rotation_mat3x3(radians):
>>>     # TODO: handle array inputs
>>>     sin_ = sympy.sin(radians)
>>>     cos_ = sympy.cos(radians)
>>>     R = np.array(((cos_, -sin_, 0),
>>>                   (sin_, cos_, 0),
>>>                   (0, 0, 1)),)
>>>     return sympy.Matrix(R)
>>>
>>> kpts = np.array([[x, y, a, c, d, ori]])
>>> kp = ktool.get_invV_mats(kpts, with_trans=True)[0]
>>> invV = sympy.Matrix(kp)
>>> V = invV.inv()
>>> ss = sympy.sqrt(patch_size) * 3.0
>>> T = sympy.Matrix(ltool.translation_mat3x3(-x, -y, None)) # Center the_
↳patch
>>> R = sympy_rotation_mat3x3(-ori) # Rotate the centered unit circle patch
>>> S = sympy.Matrix(ltool.scale_mat3x3(ss, dtype=None)) # scale from unit_
↳circle to the patch size
>>> X = sympy.Matrix(ltool.translation_mat3x3(half_patch_size, half_patch_
↳size, None)) # Translate back to patch-image coordinates
>>>
>>> sympy.MatMul(X, S, hold=True)
>>>
>>> def add_matmul_hold_prop(mat):
>>>     #import functools
>>>     def matmul_hold(other, hold=True):
>>>         new = sympy.MatMul(mat, other, hold=hold)
>>>         add_matmul_hold_prop(new)
>>>         return new
>>>     #matmul_hold = functools.partial(sympy.MatMul, mat, hold=True)
>>>     setattr(mat, 'matmul_hold', matmul_hold)
>>> add_matmul_hold_prop(X)
>>> add_matmul_hold_prop(S)

```

(continues on next page)

(continued from previous page)

```

>>> add_matmul_hold_prop(R)
>>> add_matmul_hold_prop(V)
>>> add_matmul_hold_prop(T)
>>>
>>> M = X.matmul_hold(S).matmul_hold(R).matmul_hold(V).matmul_hold(T)
>>> #M = X.multiply(S).multiply(R).multiply(V).multiply(T)
>>>
>>>
>>> V_full = R.multiply(V).multiply(T)
>>> sympy.latex(V_full)
>>> print(sympy.latex(R.multiply(V).multiply(T)))
>>> print(sympy.latex(X))
>>> print(sympy.latex(S))
>>> print(sympy.latex(R))
>>> print(sympy.latex(invV) + '^{-1}')
>>> print(sympy.latex(T))

```

`vtool.interpolate_between` (*peak_list*, *nScales*, *high*, *low*)

`vtool.interpolate_maxima` (*scalar_list*)

`vtool.interpolate_nans` (*arr*)

replaces nans with interpolated values or 0

Parameters *arr* (*ndarray*) –

Returns *new_arr*

Return type *ndarray*

CommandLine: `python -m vtool.util_math --exec-interpolate_nans`

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> arr = np.array([np.nan, np.nan, np.nan, np.nan])
>>> new_arr = interpolate_nans(arr)
>>> result = ('new_arr = %s' % (str(new_arr),))
>>> print(result)
new_arr = [ 0.  0.  0.  0.]

```

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> arr = np.array([np.nan, 1, np.nan, np.nan, np.nan, np.nan, 10, np.nan, 5])
>>> new_arr = interpolate_nans(arr)
>>> result = ('new_arr = %s' % (str(new_arr),))
>>> print(result)
new_arr = [ 1.    1.    2.8   4.6   6.4   8.2  10.    7.5   5. ]

```

`vtool.interpolate_peaks` (*x_data_list*, *y_data_list*)

`vtool.interpolate_peaks2` (*x_data_list*, *y_data_list*)

`vtool.interpolate_precision_recall` (*precision, recall, nSamples=11*)

Interpolates precision as a function of recall $p_{\text{interp}}(r)$

Reduce wiggles in average precision curve by taking interpolated values along a uniform sample.

References

http://en.wikipedia.org/wiki/Information_retrieval#Average_precision
http://en.wikipedia.org/wiki/Information_retrieval#Mean_Average_precision

http://en.wikipedia.org/wiki/Information_retrieval#Mean_Average_precision

CommandLine: `python -m vtool.confusion --test-interpolate_precision_recall --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.confusion import * # NOQA
>>> scores, labels = testdata_scores_labels()
>>> nSamples = 11
>>> confusions = ConfusionMetrics().fit(scores, labels)
>>> precision = confusions.precision
>>> recall = confusions.recall
>>> recall_domain, p_interp = interpolate_precision_recall(confusions.precision,
↳ recall, nSamples=11)
>>> result = ub.repr2(p_interp, precision=1, with_dtype=True)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> draw_precision_recall_curve(recall_domain, p_interp)
>>> ut.show_if_requested()
np.array([ 1. ,  1. ,  1. ,  1. ,  1. ,  1. ,  1. ,  0.9,  0.9,  0.8,  0.6],
↳ dtype=np.float64)
```

`vtool.interpolate_replbounds` (*xdata, ydata, pt, maximize=True*)

`xdata = np.array([.1, .2, .3, .4, .5]) ydata = np.array([.1, .2, .3, .4, .5]) pt = .35`

FIXME: if duplicate xdata is given bad things happen.

BUG: in `scipy.interpolate.interp1d` If there is a duplicate xdata, then `assume_sorted=False` will sort ydata by xdata, but xdata should retain its initial ordering in places of ambiguity. Currently it does not.

Parameters

- **xdata** (*ndarray*) –
- **ydata** (*ndarray*) –
- **pt** (*ndarray*) –

Returns `interp_vals`

Return type `float`

CommandLine: `python -m vtool.confusion --exec-interpolate_replbounds`

Example

```

>>> from vtool.confusion import * # NOQA
>>> scores, labels = testdata_scores_labels()
>>> self = ConfusionMetrics().fit(scores, labels)
>>> xdata = self.tpr
>>> ydata = self.thresholds
>>> pt = 1.0
>>> #xdata = self.fpr
>>> #ydata = self.thresholds
>>> #pt = 0.0
>>> thresh = interpolate_replbounds(xdata, ydata, pt, maximize=True)
>>> print('thresh = %r' % (thresh,))
>>> thresh = interpolate_replbounds(xdata, ydata, pt, maximize=False)
>>> print('thresh = %r' % (thresh,))

```

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.confusion import * # NOQA
>>> xdata = np.array([0.7, 0.8, 0.8, 0.9, 0.9, 0.9])
>>> ydata = np.array([34, 26, 23, 22, 19, 17])
>>> pt = np.array([.85, 1.0, -1.0])
>>> interp_vals = interpolate_replbounds(xdata, ydata, pt)
>>> result = ('interp_vals = %s' % (str(interp_vals),))
>>> print(result)
interp_vals = [ 22.5  17.   34. ]

```

`vtool.interpolate_submaxima(argmaxima, hist_, centers=None)`

Parameters

- **argmaxima** (*ndarray*) – indices into ydata / centers that are argmaxima
- **hist_** (*ndarray*) – ydata, histogram frequencies
- **centers** (*ndarray*) – xdata, histogram labels

FIXME: what happens when `argmaxima[i] == len(hist_)`

CommandLine: `python -m vtool.histogram --test-interpolate_submaxima --show`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> import ubelt as ub
>>> argmaxima = np.array([1, 4, 7])
>>> hist_ = np.array([ 6.73, 8.69, 0.00, 0.00, 34.62, 29.16, 0.00, 0.00, 6.73,
↳ 8.69])
>>> centers = np.array([-0.39, 0.39, 1.18, 1.96, 2.75, 3.53, 4.32, 5.11, 5.89,
↳ 6.68])
>>> submaxima_x, submaxima_y = interpolate_submaxima(argmaxima, hist_, centers)
>>> locals_ = ut.exec_func_src(interpolate_submaxima,
>>>                             key_list=['x123', 'y123', 'coeff_list'])
>>> x123, y123, coeff_list = locals_
>>> res = (submaxima_x, submaxima_y)
>>> result = ub.repr2(res, nl=1, nobr=True, precision=2, with_dtype=True)

```

(continues on next page)

(continued from previous page)

```

>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.ensureqt()
>>> pt.figure(fnum=pt.ensure_fnum(None))
>>> pt.plot(centers, hist_, '-')
>>> pt.plot(centers[argmaxima], hist_[argmaxima], 'o', label='argmaxima')
>>> pt.plot(submaxima_x, submaxima_y, 'b*', markersize=20, label='interp maxima')
>>> # Extract parabola points
>>> pt.plt.plot(x123, y123, 'o', label='maxima neighbors')
>>> xpoints = [np.linspace(x1, x3, 50) for (x1, x2, x3) in x123.T]
>>> ypoints = [np.polyval(coeff, x_pts) for x_pts, coeff in zip(xpoints, coeff_
↪list)]
>>> # Draw Submax Parabola
>>> for x_pts, y_pts in zip(xpoints, ypoints):
>>>     pt.plt.plot(x_pts, y_pts, 'g--', lw=2)
>>> pt.show_if_requested()
np.array([ 0.15,  3.03,  5.11], dtype=np.float64),
np.array([ 9.2 , 37.19,  0. ], dtype=np.float64),

```

Example

```

>>> hist_ = np.array([5])
>>> argmaxima = [0]

```

`vtool.interpolated_histogram(data, weights, range_, bins, interpolation_wrap=True, _debug=False)`

Follows `np.histogram`, but does interpolation

Parameters

- **data** (*ndarray*) –
- **weights** (*ndarray*) –
- **range_** (*tuple*) – range from 0 to 1
- **bins** (*int*) –
- **interpolation_wrap** (*bool*) – (default = True)
- **_debug** (*bool*) – (default = False)

CommandLine: `python -m vtool.histogram --test-interpolated_histogram`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> data = np.array([ 0,  1,  2,  3.5,  3,  3,  4,  4])
>>> weights = np.array([1., 1., 1., 1., 1., 1., 1., 1.])
>>> range_ = (0, 4)
>>> bins = 5
>>> interpolation_wrap = False
>>> hist, edges = interpolated_histogram(data, weights, range_, bins,
>>>                                     interpolation_wrap)

```

(continues on next page)

(continued from previous page)

```

>>> assert np.abs(hist.sum() - weights.sum()) < 1E-9
>>> assert hist.size == bins
>>> assert edges.size == bins + 1
>>> result = get_histinfo_str(hist, edges)
>>> print(result)

```

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> data = np.array([ 0, 1, 2, 3.5, 3, 3, 4, 4])
>>> weights = np.array([4.5, 1., 1., 1., 1., 1., 1., 1.])
>>> range_ = (-.5, 4.5)
>>> bins = 5
>>> interpolation_wrap = True
>>> hist, edges = interpolated_histogram(data, weights, range_, bins,
>>>                                     interpolation_wrap)
>>> assert np.abs(hist.sum() - weights.sum()) < 1E-9
>>> assert hist.size == bins
>>> assert edges.size == bins + 1
>>> result = get_histinfo_str(hist, edges)
>>> print(result)

```

`vtool.intersect1d_reduce(arr_list, assume_unique=False)`

`vtool.intersect2d_flags(A, B)`

Checks intersection of rows of A against rows of B

Parameters

- **A** (`ndarray[ndims=2]`) –
- **B** (`ndarray[ndims=2]`) –

Returns (`flag_list1, flag_list2`)

Return type `tuple`

CommandLine: `python -m vtool.other --test-intersect2d_flags`

SeeAlso: `np.in1d` - the one dimensional version

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> A = np.array([[609, 307], [ 95, 344], [ 1, 690]])
>>> B = np.array([[ 422, 1148], [ 422, 968], [ 481, 1148], [ 750, 1132], [ 759,
↪159]])
>>> (flag_list1, flag_list2) = intersect2d_flags(A, B)
>>> result = str((flag_list1, flag_list2))
>>> print(result)

```

`vtool.intersect2d_indices(A, B)`

Parameters

- **A**(*ndarray*[*ndims*=2])–
- **B**(*ndarray*[*ndims*=2])–

Returns (ax_list, bx_list)

Return type tuple

CommandLine: python -m vtool.other --test-intersect2d_indices

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> # build test data
>>> A = np.array([[ 158,  171], [ 542,  297], [ 955, 1113], [ 255, 1254], [ 976,
↪1255], [ 170, 1265]])
>>> B = np.array([[ 117,  211], [ 158,  171], [ 255, 1254], [ 309,  328], [ 447,
↪1148], [ 750,  357], [ 976, 1255]])
>>> # execute function
>>> (ax_list, bx_list) = intersect2d_indices(A, B)
>>> # verify results
>>> result = str((ax_list, bx_list))
>>> print(result)
```

vtool.**intersect2d_numpy**(A, B, *assume_unique=False*, *return_indices=False*)

References

<http://stackoverflow.com/questions/8317022/get-intersecting-rows-across-two-2d-numpy-arrays/8317155#8317155>

Parameters

- **A**(*ndarray*[*ndims*=2])–
- **B**(*ndarray*[*ndims*=2])–
- **assume_unique**(*bool*)–

Returns C

Return type ndarray[*ndims*=2]

CommandLine: python -m vtool.other --test-intersect2d_numpy

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> # build test data
>>> A = np.array([[ 0,  78,  85, 283, 396, 400, 403, 412, 535, 552],
...               [152, 98, 32, 260, 387, 285, 22, 103, 55, 261]]).T
>>> B = np.array([[403, 85, 412, 85, 815, 463, 613, 552],
...               [ 22, 32, 103, 116, 188, 199, 217, 254]]).T
>>> assume_unique = False
>>> # execute function
```

(continues on next page)

(continued from previous page)

```
>>> C, Ax, Bx = intersect2d_numpy(A, B, return_indices=True)
>>> # verify results
>>> result = str((C.T, Ax, Bx))
>>> print(result)
(array([[ 85, 403, 412],
        [ 32,  22, 103]]), array([2, 6, 7]), array([0, 1, 2]))
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> A = np.array([[1, 2, 3], [1, 1, 1]])
>>> B = np.array([[1, 2, 3], [1, 2, 14]])
>>> C, Ax, Bx = intersect2d_numpy(A, B, return_indices=True)
>>> result = str((C, Ax, Bx))
>>> print(result)
(array([[1, 2, 3]]), array([0]), array([0]))
```

`vtool.intersect2d_structured_numpy(arr1, arr2, assume_unique=False)`

Parameters

- **arr1** – unstructured 2d array
- **arr2** – unstructured 2d array

Returns A_, B_, C_ - structured versions of arr1, and arr2, and their structured intersection

References

<http://stackoverflow.com/questions/16970982/find-unique-rows-in-numpy-array>

<http://stackoverflow.com/questions/8317022/get-intersecting-rows-across-two-2d-numpy-arrays>

`vtool.inv_ltri(ltri, det)`
Lower triangular inverse

`vtool.invert_apply_grouping(grouped_items, groupxs)`

Parameters

- **grouped_items** (*list*) – of lists
- **groupxs** (*list*) – of lists

Returns items

Return type *list*

CommandLine: `python -m vtool.clustering2 -test-invert_apply_grouping`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.clustering2 import * # NOQA
>>> grouped_items = [[8, 5, 6], [1, 5, 8, 7], [5, 3, 0, 9]]
>>> groupxs = [np.array([1, 3, 5]), np.array([0, 2, 4, 6]), np.array([ 7,  8,  9,
↪10])]
(continues on next page)
```

(continued from previous page)

```
>>> items = invert_apply_grouping(grouped_items, groupxs)
>>> result = items
>>> print(result)
[1, 8, 5, 5, 8, 6, 7, 5, 3, 0, 9]
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.clustering2 import * # NOQA
>>> grouped_items, groupxs = [], []
>>> result = invert_apply_grouping(grouped_items, groupxs)
>>> print(result)
[]
```

`vtool.invert_apply_grouping2(grouped_items, groupxs, dtype=None)`
use only when ungrouping will be complete

`vtool.invert_apply_grouping3(grouped_items, groupxs, maxval)`

`vtool.invert_invV_mats(invV_mats)`

Parameters `invV_mats` (`ndarray[float32_t, ndim=3]`) – keypoint shapes (possibly translation)

Returns `V_mats`

Return type `ndarray[float32_t, ndim=3]`

```
# Ignore: # >>> from vtool.keypoint import * # >>> invV_mats = np.array([[[ 18.00372824, 1.86434161, 32. ],
# >>> [ -0.61356842, 16.02202028, 27.2 ], # >>> [ 0. , 0. , 1. ]], # >>> # # >>> [[ 17.41989015, 2.51145917,
61. ], # >>> [ -2.94649591, 24.02540959, 22.9 ], # >>> [ 0. , 0. , 1. ]], # >>> # # >>> [[ 20.38098025,
0.88070646, 93.1 ], # >>> [ -0.93778675, 24.78261982, 23.6 ], # >>> [ 0. , 0. , 1. ]], # >>> # # >>> [[
16.25114793, -5.93213207, 120. ], # >>> [ 4.71295477, 21.80597527, 29.5 ], # >>> [ 0. , 0. , 1. ]], # >>> # #
>>> [[ 19.60863253, -11.43641248, 147. ], # >>> [ 8.45128003, 10.69925072, 42. ], # >>> [ 0. , 0. , 1. ]]])
# >>> ut.hash_data(invV_mats) # hcnoknyxgeecfyfrygblbvdeezmiulws # >>> V_mats = npl.inv(invV_mats) #
>>> ut.hash_data(V_mats) # yooneahjgcifojzpovddehyhtkkypldd
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts()
>>> invV_mats = vt.get_invVR_mats3x3(kpts)
>>> V_mats = invert_invV_mats(invV_mats)
>>> test = np.matmul(invV_mats, V_mats)
>>> # This should give us identity
>>> assert np.allclose(test, np.eye(3))
```

`vtool.inverted_sift_patch(sift, dim=32)`

Idea for inverted sift visualization

CommandLine: `python -m vtool.patch test_sift_viz --show --name=star python -m vtool.patch test_sift_viz --show --name=star2 python -m vtool.patch test_sift_viz --show --name=cross python -m vtool.patch test_sift_viz --show --name=stripe`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> import vtool as vt
>>> patch = vt.get_test_patch(ut.get_argval('--name', default='star'))
>>> sift = vt.extract_feature_from_patch(patch)
>>> siftimg = test_sift_viz(sift)
>>> # Need to do some image blending
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.figure(fnum=1, pnum=(1, 2, 1))
>>> pt.mpl_sift.draw_sift_on_patch(siftimg, sift)
>>> pt.figure(fnum=1, pnum=(1, 2, 2))
>>> patch2 = patch
>>> patch2 = vt.rectify_to_uint8(patch2)
>>> patch2 = vt.rectify_to_square(patch2)
>>> pt.mpl_sift.draw_sift_on_patch(patch2, sift)
>>> ut.show_if_requested()
```

`vtool.invertible_stack(vecs_list, label_list)`

Stacks descriptors into a flat structure and returns inverse mapping from flat database descriptor indexes (dx) to annotation ids (label) and feature indexes (fx). Feature indexes are w.r.t. annotation indexes.

Output: `idx2_desc` - flat descriptor stack `idx2_label` - inverted index into annotations `idx2_fx` - inverted index into features

Example with 2D Descriptors

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.nearest_neighbors import * # NOQA
>>> DESC_TYPE = np.uint8
>>> label_list = [1, 2, 3, 4, 5]
>>> vecs_list = [
...     np.array([[0, 0], [0, 1]], dtype=DESC_TYPE),
...     np.array([[5, 3], [2, 30], [1, 1]], dtype=DESC_TYPE),
...     np.empty((0, 2), dtype=DESC_TYPE),
...     np.array([[5, 3], [2, 30], [1, 1]], dtype=DESC_TYPE),
...     np.array([[3, 3], [42, 42], [2, 6]], dtype=DESC_TYPE),
...     ]
>>> idx2_vec, idx2_label, idx2_fx = invertible_stack(vecs_list, label_list)
>>> print(repr(idx2_vec.T))
array([[ 0,  0,  5,  2,  1,  5,  2,  1,  3, 42,  2],
       [ 0,  1,  3, 30,  1,  3, 30,  1,  3, 42,  6]], dtype=uint8)
>>> print(repr(idx2_label))
array([1, 1, 2, 2, 2, 2, 4, 4, 5, 5, 5])
>>> print(repr(idx2_fx))
array([0, 1, 0, 1, 2, 0, 1, 2, 0, 1, 2])
```

`vtool.invsum(x)`

`vtool.iound(num, dtype=<class 'numpy.int32'>)`

Integer round. (because numpy doesn't seem to have it!)

`vtool.iter_reduce_ufunc(ufunc, arr_iter, out=None)`

constant memory iteration and reduction

applies ufunc from left to right over the input arrays

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> arr_list = [
...     np.array([0, 1, 2, 3, 8, 9]),
...     np.array([4, 1, 2, 3, 4, 5]),
...     np.array([0, 5, 2, 3, 4, 5]),
...     np.array([1, 1, 6, 3, 4, 5]),
...     np.array([0, 1, 2, 7, 4, 5])
... ]
>>> memory = np.array([9, 9, 9, 9, 9, 9])
>>> gen_memory = memory.copy()
>>> def arr_gen(arr_list, gen_memory):
...     for arr in arr_list:
...         gen_memory[:] = arr
...         yield gen_memory
>>> print('memory = %r' % (memory,))
>>> print('gen_memory = %r' % (gen_memory,))
>>> ufunc = np.maximum
>>> res1 = iter_reduce_ufunc(ufunc, iter(arr_list), out=None)
>>> res2 = iter_reduce_ufunc(ufunc, iter(arr_list), out=memory)
>>> res3 = iter_reduce_ufunc(ufunc, arr_gen(arr_list, gen_memory), out=memory)
>>> print('res1      = %r' % (res1,))
>>> print('res2      = %r' % (res2,))
>>> print('res3      = %r' % (res3,))
>>> print('memory    = %r' % (memory,))
>>> print('gen_memory = %r' % (gen_memory,))
>>> assert np.all(res1 == res2)
>>> assert np.all(res2 == res3)
```

`vtool.iter_reduce_ufunc(ufunc, arr_iter, out=None)`

constant memory iteration and reduction

applies ufunc from left to right over the input arrays

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> arr_list = [
...     np.array([0, 1, 2, 3, 8, 9]),
...     np.array([4, 1, 2, 3, 4, 5]),
...     np.array([0, 5, 2, 3, 4, 5]),
...     np.array([1, 1, 6, 3, 4, 5]),
...     np.array([0, 1, 2, 7, 4, 5])
... ]
>>> memory = np.array([9, 9, 9, 9, 9, 9])
>>> gen_memory = memory.copy()
>>> def arr_gen(arr_list, gen_memory):
...     for arr in arr_list:
...         gen_memory[:] = arr
...         yield gen_memory
```

(continues on next page)

(continued from previous page)

```

>>> print('memory = %r' % (memory,))
>>> print('gen_memory = %r' % (gen_memory,))
>>> ufunc = np.maximum
>>> res1 = iter_reduce_ufunc(ufunc, iter(arr_list), out=None)
>>> res2 = iter_reduce_ufunc(ufunc, iter(arr_list), out=memory)
>>> res3 = iter_reduce_ufunc(ufunc, arr_gen(arr_list, gen_memory), out=memory)
>>> print('res1      = %r' % (res1,))
>>> print('res2      = %r' % (res2,))
>>> print('res3      = %r' % (res3,))
>>> print('memory     = %r' % (memory,))
>>> print('gen_memory = %r' % (gen_memory,))
>>> assert np.all(res1 == res2)
>>> assert np.all(res2 == res3)

```

`vtool.jagged_group(groupids_list)`
 flattens and returns group indexes into the flattened list

`vtool.kp_cpp_infostr(kp)`
 mirrors c++ debug code

`vtool.kpts_docrepr(arr, name='arr', indent=True, *args, **kwargs)`

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> np.random.seed(0)
>>> arr = np.random.rand(3, 3)
>>> args = tuple()
>>> kwargs = dict()
>>> result = kpts_docrepr(arr)
>>> # verify results
>>> print(result)

```

`vtool.kpts_matrices(kpts)`

`vtool.kpts_repr(arr, precision=2, suppress_small=True, linebreak=False)`

`vtool.learn_score_normalization(tp_support, tn_support, gridsize=1024, adjust=8, return_all=False, monotonize=True, clip_factor=2.6180339887499997, verbose=False, reverse=False, p_tp_method='eq')`

Takes collected data and applies parzen window density estimation and bayes rule.

#True positive scores must be larger than true negative scores. FIXME: might be an issue with pdfs summing to 1 here.

Parameters

- **tp_support** (*ndarray*) –
- **tn_support** (*ndarray*) –
- **gridsize** (*int*) – default 512
- **adjust** (*int*) – default 8
- **return_all** (*bool*) – default False
- **monotonize** (*bool*) – default True

- **clip_factor** (*float*) – default ϕ^2

Returns (score_domain, p_tp_given_score, p_tn_given_score, p_score_given_tp, p_score_given_tn, p_score)

Return type `tuple`

CommandLine: `python -m vtool.score_normalization --test-learn_score_normalization`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.score_normalization import * # NOQA
>>> tp_support = np.linspace(100, 10000, 512)
>>> tn_support = np.linspace(0, 120, 512)
>>> gridsize = 1024
>>> adjust = 8
>>> return_all = False
>>> monotonize = True
>>> clip_factor = 2.6180339887499997
>>> verbose = True
>>> reverse = False
>>> (score_domain, p_tp_given_score) = learn_score_normalization(tp_support, tn_
↪support)
>>> result = '%.2f' % (np.diff(p_tp_given_score).sum())
>>> print(result)
0.99
```

`vtool.linear_interpolation(arr, subindices)`

Does linear interpolation to lookup subindex values

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> arr = np.array([0, 1, 2, 3])
>>> subindices = np.array([0, .1, 1, 1.8, 2, 2.5, 3] )
>>> subvalues = linear_interpolation(arr, subindices)
>>> assert np.allclose(subindices, subvalues)
>>> assert np.allclose(2.3, linear_interpolation(arr, 2.3))
```

`vtool.list_compress_(list_, flag_list)`

`vtool.list_take_(list_, index_list)`

`vtool.logistic_01(x)`

Parameters **x** –

CommandLine: `python -m vtool.util_math --exec-logistic_01 --show`

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.util_math import * # NOQA
>>> x = np.linspace(0, 1)
>>> y = logistic_01(x)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.plot(x, y)
>>> ut.show_if_requested()

```

`vtool.logit(x)`

`vtool.make_channels_comparable(img1, img2)`

Broadcasts image arrays so they can have elementwise operations applied

CommandLine: `python -m vtool.image make_channels_comparable`

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> wh_basis = [(5, 5), (3, 5), (5, 3), (1, 1), (1, 3), (3, 1)]
>>> for w, h in wh_basis:
>>>     shape_basis = [(w, h), (w, h, 1), (w, h, 3)]
>>>     # Test all permutations of shap inputs
>>>     for shapel, shape2 in ut.product(shape_basis, shape_basis):
>>>         print('*   input shapes: %r, %r' % (shapel, shape2))
>>>         img1 = np.empty(shapel)
>>>         img2 = np.empty(shape2)
>>>         img1, img2 = make_channels_comparable(img1, img2)
>>>         print('... output shapes: %r, %r' % (img1.shape, img2.shape))
>>>         elem = (img1 + img2)
>>>         print('... elem(+) shape: %r' % (elem.shape,))
>>>         assert elem.size == img1.size, 'outputs should have same size'
>>>         assert img1.size == img2.size, 'new imgs should have same size'
>>>         print('-----')

```

`vtool.make_dummy_fm(nKpts)`

`vtool.make_exif_dict_human_readable(exif_dict)`

`vtool.make_test_image_keypoints(imgBGR, scale=1.0, skew=0, theta=0, shift=(0, 0))`

`vtool.make_video(images, outvid=None, fps=5, size=None, is_color=True, format='XVID')`

Create a video from a list of images.

References

http://www.xavierdupre.fr/blog/2016-03-30_nojs.html http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_gui/py_video_display/py_video_display.html

@param outvid output video @param images list of images to use in the video @param fps frame per second @param size size of each frame @param is_color color @param format see <http://www.fourcc.org/codecs.php>

The function relies on <http://opencv-python-tutroals.readthedocs.org/en/latest/>. By default, the video will have the size of the first image. It will resize every image to this size before adding them to the video.

`vtool.make_video2(images, outdir)`

`vtool.make_white_transparent (imgBGR)`

Parameters `imgBGR` (`ndarray [uint8_t, ndim=2]`) – image data (blue, green, red)

Returns `imgBGRA`

Return type `ndarray`

CommandLine: `python -m vtool.image make_white_transparent --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> imgBGR = imread(ut.get_argval('--fpath', type=str))
>>> imgBGRA = make_white_transparent(imgBGR)
>>> result = ('imgBGRA = %s' % (ub.repr2(imgBGRA),))
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> ut.show_if_requested()
```

`vtool.maxima_neighbors (argmaxima, hist_, centers=None)`

`vtool.maximum_parabola_point (A, B, C)`

Maximum x point is where the derivative is 0

`vtool.median_abs_dev (arr_list, **kwargs)`

References

https://en.wikipedia.org/wiki/Median_absolute_deviation

`vtool.montage (img_list, dsize, rng=<module 'numpy.random' from
'/home/docs/checkouts/readthedocs.org/user_builds/wbia-
vtool/envs/stable/lib/python3.7/site-packages/numpy/random/__init__.py'>, method='random', return_debug=False)`

Creates a montage / collage from a set of images

CommandLine: `python -m vtool.image --exec-montage:0 --show python -m vtool.image --exec-montage:1`

Example

```
>>> # SLOW_DOCTEST
>>> # xdoctest: +SKIP
>>> from vtool.image import * # NOQA
>>> img_list0 = testdata_imglist()
>>> img_list1 = [resize_to_maxdims(img, (256, 256)) for img in img_list0]
>>> num = 4
>>> img_list = list(ub.flatten([img_list1] * num))
>>> dsize = (700, 700)
>>> rng = np.random.RandomState(42)
>>> method = 'unused'
>>> #method = 'random'
>>> dst, debug_info = montage(img_list, dsize, rng, method=method,
>>>                             return_debug=True)
```

(continues on next page)

(continued from previous page)

```
>>> place_img = debug_info.get('place_img_', np.ones((2, 2)))
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(dst, pnum=(1, 2, 1))
>>> pt.imshow(place_img / place_img.max(), pnum=(1, 2, 2))
>>> ut.show_if_requested()
```

Example

```
>>> # SLOW_DOCTEST
>>> # xdoctest: +SKIP
>>> import wbia
>>> import random
>>> from os.path import join, expanduser, abspath
>>> from vtool.image import * # NOQA
>>> ibs = wbia.opendb('GZC')
>>> gid_list0 = ibs.get_valid_gids()
>>> img_list = []
>>> for i in range(6000):
>>>     print(i)
>>>     try:
>>>         gid = random.choice(gid_list0)
>>>         image = ibs.get_images(gid)
>>>         image = resize_to_maxdims(image, (512, 512))
>>>         img_list.append(image)
>>>     except Exception:
>>>         pass
>>> dsize = (19200, 10800)
>>> rng = np.random.RandomState(42)
>>> dst = montage(img_list, dsize, rng)
>>> filepath = abspath(expanduser(join('~', 'Desktop', 'montage.jpg')))
>>> print('Writing to: %r' % (filepath, ))
>>> imwrite(filepath, dst)
```

`vtool.mult_lists(*args)`

`vtool.multiaxis_reduce(ufunc, arr, startaxis=0)`
used to get max/min over all axes after <startaxis>

CommandLine: `python -m vtool.numpy_utils --test-multiaxis_reduce`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> rng = np.random.RandomState(0)
>>> arr = (rng.rand(4, 3, 2, 1) * 255).astype(np.uint8)
>>> ufunc = np.amax
>>> startaxis = 1
>>> out_ = multiaxis_reduce(ufunc, arr, startaxis)
>>> result = out_
>>> print(result)
[182 245 236 249]
```

`vttool.multigroup_lookup` (*lazydict*, *keys_list*, *subkeys_list*, *custom_func*)

Efficiently calls `custom_func` for each item in `zip(keys_list, subkeys_list)` by grouping subkeys to minimize the number of calls to `custom_func`.

We are given multiple lists of keys, and subvals. The goal is to group the subvals by keys and apply the subval lookups (a call to a function) to the key only once and at the same time.

Parameters

- **lazydict** (*dict* of `utool.LazyDict`) –
- **keys_list** (*list*) –
- **subkeys_list** (*list*) –
- **custom_func** (*func*) – must have signature `custom_func(lazydict, key, subkeys)`

SeeAlso: `vt.multigroup_lookup_naive` - unoptimized version, but simple to read

Example

```
>>> # SLOW_DOCTEST
>>> # xdoctest: +SKIP
>>> from vtool.other import * # NOQA
>>> import vtool as vt
>>> fpath_list = [ut.grab_test_imgpath(key) for key in ut.util_grabdata.get_valid_
↳ test_imgkeys()]
>>> lazydict = {count: vt.testdata_annot_metadata(fpath) for count, fpath in_
↳ enumerate(fpath_list)}
>>> aids_list = np.array([(3, 2), (0, 2), (1, 2), (2, 3)])
>>> fms = np.array([[2, 5], [2, 3], [2, 1], [3, 4]])
>>> keys_list = aids_list.T
>>> subkeys_list = fms.T
>>> def custom_func(lazydict, key, subkeys):
>>>     annot = lazydict[key]
>>>     kpts = annot['kpts']
>>>     rchip = annot['rchip']
>>>     kpts_m = kpts.take(subkeys, axis=0)
>>>     warped_patches = vt.get_warped_patches(rchip, kpts_m)[0]
>>>     return warped_patches
>>> data_lists1 = multigroup_lookup(lazydict, keys_list, subkeys_list, custom_
↳ func)
>>> data_lists2 = multigroup_lookup_naive(lazydict, keys_list, subkeys_list,
↳ custom_func)
>>> vt.sver_c_wrapper.assertseq(data_lists1, data_lists2)
```

Example

```
>>> keys_list = [np.array([]), np.array([]), np.array([])]
>>> subkeys_list = [np.array([]), np.array([]), np.array([])]
```

`vttool.multigroup_lookup_naive` (*lazydict*, *keys_list*, *subkeys_list*, *custom_func*)

Slow version of `multigroup_lookup`. Makes a call to `custom_func` for each item in `zip(keys_list, subkeys_list)`.

SeeAlso: `vt.multigroup_lookup`

`vttool.nan_to_num` (*arr*, *num*)

`vtool.nearest_point(x, y, pts, mode='random')`
finds the nearest point(s) in pts to (x, y)

`vtool.nearest_point(x, y, pts, mode='random')`
finds the nearest point(s) in pts to (x, y)

`vtool.non_decreasing(L)`

References

<http://stackoverflow.com/questions/4983258/python-how-to-check-list-monotonicity>

`vtool.non_increasing(L)`

References

<http://stackoverflow.com/questions/4983258/python-how-to-check-list-monotonicity>

`vtool.nonunique_row_flags(arr)`

`vtool.nonunique_row_indexes(arr)`
rows that are not unique (does not include the first instance of each pattern)

Parameters `arr` (*ndarray*) – 2d array

Returns `nonunique_rowx`

Return type `ndarray`

SeeAlso: `unique_row_indexes` `nonunique_row_flags`

CommandLine: `python -m vtool.other --test-unique_row_indexes`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> arr = np.array([[0, 0], [0, 1], [1, 0], [1, 1], [0, 0], [.534, .432], [.534, .
↪432], [1, 0], [0, 1]])
>>> nonunique_rowx = unique_row_indexes(arr)
>>> result = ('nonunique_rowx = %s' % (ub.repr2(nonunique_rowx),))
>>> print(result)
nonunique_rowx = np.array([4, 6, 7, 8], dtype=np.int64)
```

`vtool.norm01(array, dim=None)`
normalizes a numpy array from 0 to 1 based in its extent

Parameters

- `array` (*ndarray*) –
- `dim` (*int*) –

Returns

Return type `ndarray`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> array = np.array([ 22, 1, 3, 2, 10, 42, ])
>>> dim = None
>>> array_norm = norm01(array, dim)
>>> result = ub.repr2(array_norm, precision=3)
>>> print(result)
```

`vtool.normalize` (*arr*, *ord=None*, *axis=None*, *out=None*)

Returns all row vectors normalized by their magnitude.

Parameters

- **arr** (*ndarray*) – row vectors to normalize
- **ord** (*int*) – type of norm to use (defaults to 2-norm) {non-zero int, inf, -inf}
- **axis** (*int*) – axis to normalize
- **out** (*ndarray*) – preallocated output

SeeAlso: `np.linalg.norm`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.linalg import * # NOQA
>>> arr = np.array([[1, 2, 3, 4, 5], [2, 2, 2, 2, 2]])
>>> arr_normed = normalize(arr, axis=1)
>>> result = ub.hzcat(['arr_normed = ', ub.repr2(arr_normed, precision=2, with_
↳dtype=True)])
>>> assert np.allclose((arr_normed ** 2).sum(axis=1), [1, 1])
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.linalg import * # NOQA
>>> arr = np.array([ 0.6, 0.1, -0.6])
>>> arr_normed = normalize(arr)
>>> result = ub.hzcat(['arr_normed = ', ub.repr2(arr_normed, precision=2)])
>>> assert np.allclose((arr_normed ** 2).sum(), [1])
>>> print(result)
```

Example

```
>>> from vtool.linalg import * # NOQA
>>> ord_list = [0, 1, 2, np.inf, -np.inf]
>>> arr = np.array([ 0.6, 0.1, -0.5])
>>> normed = [(ord, normalize(arr, ord=ord)) for ord in ord_list]
>>> result = ub.repr2(normed, precision=2, with_dtype=True)
>>> print(result)
```

`vtool.normalize_rows` (*arr, out=None*)
DEPRICATE

`vtool.normalize_scores` (*score_domain, p_tp_given_score, scores, interp_fn=None*)
Adjusts a raw scores to a probabilities based on a learned normalizer

Parameters

- **score_domain** (*ndarray*) – input score domain
- **p_tp_given_score** (*ndarray*) – learned probability mapping
- **scores** (*ndarray*) – raw scores

Returns probabilities

Return type ndarray

CommandLine: `python -m vtool.score_normalization --test-normalize_scores`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.score_normalization import * # NOQA
>>> score_domain = np.linspace(0, 10, 10)
>>> p_tp_given_score = (score_domain ** 2) / (score_domain.max() ** 2)
>>> scores = np.array([-1, 0.0, 0.01, 2.3, 8.0, 9.99, 10.0, 10.1, 11.1])
>>> prob = normalize_scores(score_domain, p_tp_given_score, scores)
>>> #np.set_printoptions(suppress=True)
>>> result = ub.repr2(prob, precision=2, suppress_small=True)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.plot2(score_domain, p_tp_given_score, 'r-x', equal_aspect=False, label=
↳ 'learned probability')
>>> pt.plot2(scores, prob, 'yo', equal_aspect=False, title='Normalized scores',
↳ pad=.2, label='query points')
>>> pt.legend('upper left')
>>> ut.show_if_requested()
np.array([ 0. ,  0. ,  0. ,  0.05,  0.64,  1. ,  1. ,  1. ,  1. ],
↳ dtype=np.float64)
```

`vtool.normalized_nearest_neighbors` (*flann1, vecs2, K, checks=800*)
Computes matches from vecs2 to flann1.

uses flann index to return nearest neighbors with distances normalized between 0 and 1 using sifts uint8 trick

`vtool.offset_kpts` (*kpts, offset=(0.0, 0.0), scale_factor=1.0*)
Transfoms keypoints by a scale factor and a translation

Parameters

- **kpts** (*ndarray[float32_t, ndim=2]*) – keypoints
- **offset** (*tuple*) –
- **scale_factor** (*float*) –

Returns kpts - keypoints

Return type ndarray[float32_t, ndim=2]

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts().astype(np.float64)
>>> offset = (0.0, 0.0)
>>> scale_factor = (1.5, 0.5)
>>> kpts_ = offset_kpts(kpts, offset, scale_factor)
>>> # verify results (hack + 0. to fix negative 0)
>>> result = ut.repr3((kpts, kpts_ + 0.), precision=2, nobr=True, with_dtype=True)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.draw_kpts2(kpts, color=pt.ORANGE, ell_linewidth=6)
>>> pt.draw_kpts2(kpts_, color=pt.LIGHT_BLUE, ell_linewidth=4)
>>> extent1 = np.array(vt.get_kpts_image_extent(kpts))
>>> extent2 = np.array(vt.get_kpts_image_extent(kpts_))
>>> extent = vt.union_extents([extent1, extent2])
>>> ax = pt.gca()
>>> pt.set_axis_extents(extent)
>>> pt.dark_background()
>>> ut.show_if_requested()
np.array([[20. , 25. , 5.22, -5.11, 24.15, 0. ],
          [29. , 25. , 2.36, -5.11, 24.15, 0. ],
          [30. , 30. , 12.22, 12.02, 10.53, 0. ],
          [31. , 29. , 13.36, 17.63, 14.1 , 0. ],
          [32. , 31. , 16.05, 3.41, 11.74, 0. ]], dtype=np.float64),
np.array([[30. , 12.5 , 7.82, -2.56, 12.07, 0. ],
          [43.5 , 12.5 , 3.53, -2.56, 12.07, 0. ],
          [45. , 15. , 18.32, 6.01, 5.26, 0. ],
          [46.5 , 14.5 , 20.03, 8.82, 7.05, 0. ],
          [48. , 15.5 , 24.08, 1.7 , 5.87, 0. ]], dtype=np.float64),
```

`vtool.open_image_size(image_fpath)`

Gets image size from an image on disk

Parameters `image_fpath` (*str*) –

Returns size (width, height)

Return type tuple

CommandLine: `python -m vtool.image --test-open_image_size`

Doctest:

```
>>> from vtool.image import * # NOQA
>>> image_fpath = ut.grab_test_imgpath('patsy.jpg')
>>> size = open_image_size(image_fpath)
>>> result = ('size = %s' % (str(size),))
>>> print(result)
size = (800, 441)
```

Ignore:

```
>>> # Confirm that Image.open is a lazy load
>>> import vtool as vt
```

(continues on next page)

(continued from previous page)

```

>>> import utool as ut
>>> import timeit
>>> setup = ut.codeblock(
>>>     '''
>>>     from PIL import Image
>>>     import utool as ut
>>>     import vtool as vt
>>>     image_fpath = ut.grab_test_imgpath('patsy.jpg')
>>>     '''
>>> )
>>> t1 = timeit.timeit('Image.open(image_fpath)', setup, number=100)
>>> t2 = timeit.timeit('Image.open(image_fpath).size', setup, number=100)
>>> t3 = timeit.timeit('vt.open_image_size(image_fpath)', setup, number=100)
>>> t4 = timeit.timeit('vt.imread(image_fpath).shape', setup, number=100)
>>> t5 = timeit.timeit('Image.open(image_fpath).getdata()', setup, number=100)
>>> print('t1 = %r' % (t1,))
>>> print('t2 = %r' % (t2,))
>>> print('t3 = %r' % (t3,))
>>> print('t4 = %r' % (t4,))
>>> print('t5 = %r' % (t5,))
>>> assert t2 < t5
>>> assert t3 < t4

```

`vtool.open_pil_image(image_fpath)`

`vtool.or_lists(*args)`

Like `np.logical_and`, but can take more than 2 arguments

SeeAlso: `and_lists`

`vtool.ori_distance(ori1, ori2, out=None)`

Returns the unsigned distance between two angles

References

<http://stackoverflow.com/questions/1878907/the-smallest-difference-between-2-angles>

CommandLine: `python -m vtool.distance --test-ori_distance`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> rng = np.random.RandomState(0)
>>> ori1 = (rng.rand(10) * TAU) - np.pi
>>> ori2 = (rng.rand(10) * TAU) - np.pi
>>> dist_ = ori_distance(ori1, ori2)
>>> result = ub.repr2(ori1, precision=1)
>>> result += '\n' + ub.repr2(ori2, precision=1)
>>> result += '\n' + ub.repr2(dist_, precision=1)
>>> print(result)

```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> ori1 = .3
>>> ori2 = 6.8
>>> dist_ = ori_distance(ori1, ori2)
>>> result = ub.repr2(dist_, precision=2)
>>> print(result)
```

`vtool.overlay_alpha_images (img1, img2)`
places img1 on top of img2 respecting alpha channels

References

<http://stackoverflow.com/questions/25182421/overlay-numpy-alpha>

`vtool.pad_image (imgBGR, pad_, value=0, borderType=0)`

`vtool.pad_image_ondisk (img_fpath, pad_, out_fpath=None, value=0, borderType=0, **kwargs)`

Returns out_fpath - file path string

Return type str

CommandLine: python -m vtool.image pad_image_ondisk

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> img_fpath = ut.get_argval('--fpath', type_=str)
>>> pad_ = '?'
>>> out_fpath = None
>>> value = 0
>>> borderType = 0
>>> out_fpath = pad_image_ondisk(img_fpath, pad_, out_fpath, value, borderType)
>>> result = ('out_fpath = %s' % (ub.repr2(out_fpath),))
>>> print(result)
```

`vtool.pad_vstack (arrs, fill_value=0)`
Stacks values and pads arrays with different lengths with zeros

`vtool.padded_resize (img, target_size=(64, 64), interpolation=None)`
makes the image resize to the target size and pads the rest of the area with a fill value

Parameters

- **img** (`ndarray [uint8_t, ndim=2]`) – image data
- **target_size** (`tuple`) –

CommandLine: python -m vtool.image --test-padded_resize --show

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> imgA = vt.imread(ut.grab_test_imgpath('carl.jpg'))
>>> imgB = vt.imread(ut.grab_test_imgpath('ada.jpg'))
>>> imgC = vt.imread(ut.grab_test_imgpath('carl.jpg'), grayscale=True)
>>> #target_size = (64, 64)
>>> target_size = (1024, 1024)
>>> img3_list = [padded_resize(img, target_size) for img in [imgA, imgB, imgC]]
>>> # verify results
>>> assert ut.allsame([vt.get_size(img3) for img3 in img3_list])
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pnum_ = pt.make_pnum_nextgen(1, 3)
>>> pt.imshow(img3_list[0], pnum=pnum_())
>>> pt.imshow(img3_list[1], pnum=pnum_())
>>> pt.imshow(img3_list[2], pnum=pnum_())
>>> ut.show_if_requested()

```

`vtool.parse_exif_unixtime(image_fpath)`

Parameters `image_fpath` (*str*) –

Returns `unixtime`

Return type `float`

CommandLine: `python -m vtool.exif --test-parse_exif_unixtime`

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.exif import * # NOQA
>>> image_fpath = ut.grab_file_url('http://images.summitpost.org/original/769474.
↳JPG')
>>> unixtime = parse_exif_unixtime(image_fpath)
>>> result = str(unixtime)
>>> print(result)

```

`vtool.parse_exif_unixtime_gps(image_fpath)`

`vtool.partition_scores(X, y, attrs=None)`

convenience helper to translate partitioned to unpartitioned data

Parameters

- `tp_scores` (*ndarray*) –
- `tn_scores` (*ndarray*) –
- `attrs` (*dict*) – (default = None)

Returns (scores, labels, attrs)

Return type `tuple`

CommandLine: `python -m vtool.score_normalization --test-partition_scores`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.score_normalization import * # NOQA
>>> X = np.array([5, 6, 6, 7, 1, 2, 2])
>>> attrs = {'qaid': np.array([21, 24, 25, 26, 11, 14, 15])}
>>> y = np.array([1, 1, 1, 1, 0, 0, 0], dtype=np.bool_)
>>> tup = partition_scores(X, y, attrs)
>>> resdict = ut.odict(zip(
>>>     ['tp_scores', 'tn_scores', 'part_attrs'], tup))
>>> result = ub.repr2(resdict, nobraces=True, with_dtype=False,
>>>                    explicit=1, nl=2)
>>> print(result)
tp_scores=np.array([5, 6, 6, 7]),
tn_scores=np.array([1, 2, 2]),
part_attrs=False: 'qaid': np.array([11, 14, 15]),
                True: 'qaid': np.array([21, 24, 25, 26]),,
```

`vtool.patch_gaussian_weighted_average_intensities` (*probchip, kpts_*)

`vtool.patch_gradient` (*patch, ksize=1, gaussian_weighted=False*)

`vtool.patch_mag` (*gradx, grady*)

`vtool.patch_ori` (*gradx, grady*)

returns patch orientation relative to the x-axis

`vtool.pdist_argsort` (*x*)

Sorts 2d indicies by their distnace matrix output from `scipy.spatial.distance` `x = np.array([3.05555556e-03, 1.47619797e+04, 1.47619828e+04])`

Parameters *x* (*ndarray*) –

Returns `sortx_2d`

Return type *ndarray*

CommandLine: `python -m vtool.distance --test-pdist_argsort`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> x = np.array([ 21695.78, 10943.76, 10941.44, 25867.64, 10752.03,
>>>               10754.35, 4171.86, 2.32, 14923.89, 14926.2 ],
>>>               dtype=np.float64)
>>> sortx_2d = pdist_argsort(x)
>>> result = ('sortx_2d = %s' % (str(sortx_2d),))
>>> print(result)
sortx_2d = [(2, 3), (1, 4), (1, 2), (1, 3), (0, 3), (0, 2), (2, 4), (3, 4), (0, ↵
↵1), (0, 4)]
```

`vtool.pdist_indicies` (*num*)

`vtool.perlin_noise` (*size, scale=32.0, rng=<module 'numpy.random' from
'/home/docs/checkouts/readthedocs.org/user_builds/wbia-
vtool/envs/stable/lib/python3.7/site-packages/numpy/random/__init__.py'>*)

References

<http://www.siafoo.net/snippet/229>

CommandLine: `python -m vtool.image perlin_noise --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> #size = (64, 64)
>>> size = (256, 256)
>>> #scale = 32.0
>>> scale = 64.0
>>> img = perlin_noise(size, scale)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(img, pnum=(1, 1, 1))
>>> ut.show_if_requested()
```

`vtool.perterb_kpts(kpts, xy_std=None, invV_std=None, ori_std=None, damping=None, seed=None, **kwargs)`

Adds normally distributed perturbations to keypoints

`vtool.perterbed_grid_kpts(*args, **kwargs)`

`vtool.plot_centroids(data, centroids, num_pca_dims=3, whiten=False, labels='centroids', fnum=1, prefix='')`

Plots centroids and datapoints. Plots accurately up to 3 dimensions. If there are more than 3 dimensions, PCA is used to reduce the dimensionality to the <num_pca_dims> principal components

`vtool.plot_postbayes_pdf(score_domain, p_tn_given_score, p_tp_given_score, score_thresh=None, prob_thresh=None, cfgstr="", fnum=None, pnum=(1, 1, 1))`

`vtool.plot_prebayes_pdf(score_domain, p_score_given_tn, p_score_given_tp, p_score, cfgstr="", fnum=None, pnum=(1, 1, 1), **kwargs)`

`vtool.point_inside_bbox(point, bbox)`

Flags points that are strictly inside a bounding box. Points on the boundary are not considered inside.

Parameters

- **point** (*ndarray*) – one or more points to test (2xN)
- **bbox** (*tuple*) – a bounding box in (x, y, w, h) format

Returns True if the point is in the bbox

Return type `bool` or `ndarray`

CommandLine: `python -m vtool.geometry point_inside_bbox --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> import ubelt as ub
>>> point = np.array([
```

(continues on next page)

(continued from previous page)

```

>>> [3, 2], [4, 1], [2, 3], [1, 1], [0, 0],
>>> [4, 9.5], [9, 9.5], [7, 2], [7, 8], [9, 3]
>>> ]).T
>>> bbox = (3, 2, 5, 7)
>>> flag = point_inside_bbox(point, bbox)
>>> flag = flag.astype(np.int)
>>> result = ('flag = %s' % (ub.repr2(flag),))
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> verts = np.array(verts_from_bbox(bbox, close=True))
>>> pt.plot(verts.T[0], verts.T[1], 'b-')
>>> pt.plot(point[0][flag], point[1][flag], 'go')
>>> pt.plot(point[0][~flag], point[1][~flag], 'rx')
>>> pt.plt.xlim(0, 10); pt.plt.ylim(0, 10)
>>> pt.show_if_requested()
flag = np.array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0])

```

```
vtool.print_image_checks(img_fpath)
```

```

vtool.random_affine_args (zoom_pdf=None, tx_pdf=None, ty_pdf=None, shear_pdf=None,
                           theta_pdf=None, enable_flip=False, enable_stretch=False,
                           default_distribution='uniform', scalar_anchor='reflect',
                           txy_pdf=None, rng=<module 'numpy.random' from
                           '/home/docs/checkouts/readthedocs.org/user_builds/wbia-
                           vtool/envs/stable/lib/python3.7/site-packages/numpy/random/__init__.py'>)

```

TODO: allow for a pdf of ranges for each dimension

If pdfs are tuples it is interpreted as a default (uniform) distribution between the two points. A single scalar is a default distribution between -scalar and scalar.

Parameters

- **zoom_range** (*tuple*) – (default = (1.0, 1.0))
- **tx_range** (*tuple*) – (default = (0.0, 0.0))
- **ty_range** (*tuple*) – (default = (0.0, 0.0))
- **shear_range** (*tuple*) – (default = (0, 0))
- **theta_range** (*tuple*) – (default = (0, 0))
- **enable_flip** (*bool*) – (default = False)
- **enable_stretch** (*bool*) – (default = False)
- **rng** (*module*) – random number generator (default = numpy.random)

Returns affine_args

Return type *tuple*

CommandLine: xdoctest -m ~/code/vtool/vtool/linalg.py random_affine_args

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.linalg import * # NOQA
>>> import vtool as vt
>>> zoom_range = (0.9090909090909091, 1.1)
>>> tx_pdf = (0.0, 4.0)
>>> ty_pdf = (0.0, 4.0)
>>> shear_pdf = (0, 0)
>>> theta_pdf = (0, 0)
>>> enable_flip = False
>>> enable_stretch = False
>>> rng = np.random.RandomState(0)
>>> affine_args = random_affine_args(
>>>     zoom_range, tx_pdf, ty_pdf, shear_pdf, theta_pdf,
>>>     enable_flip, enable_stretch, rng=rng)
>>> print('affine_args = %s' % (ub.repr2(affine_args),))
>>> (sx, sy, theta, shear, tx, ty) = affine_args
>>> Aff = vt.affine_mat3x3(sx, sy, theta, shear, tx, ty)
>>> result = ub.repr2(Aff, precision=3, nl=1, with_dtype=0)
>>> print(result)
np.array([[ 1.009, -0.    ,  1.695],
          [ 0.    ,  1.042,  2.584],
          [ 0.    ,  0.    ,  1.   ]])

```

`vtool.random_affine_transform(*args, **kwargs)`

`vtool.read_all_exif_tags(pil_img)`

`vtool.read_exif(fpath, tag=None)`

`vtool.read_exif_tags(pil_img, exif_tagid_list, default_list=None)`

`vtool.read_one_exif_tag(pil_img, tag)`

`vtool.rebuild_partition(part1_vals, part2_vals, part1_indexes, part2_indexes)`
 Inverts work done by `index_partition`

Parameters

- `part1_vals` (*list*) –
- `part2_vals` (*list*) –
- `part1_indexes` (*dict*) –
- `part2_indexes` (*dict*) –

CommandLine: `python -m vtool.other --test-rebuild_partition`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> item_list = ['dist', 'fg', 'distinctiveness']
>>> part1_items = ['fg', 'distinctiveness']
>>> part1_indexes, part2_indexes = index_partition(item_list, part1_items)
>>> part1_vals = ut.take(item_list, part1_indexes)
>>> part2_vals = ut.take(item_list, part2_indexes)
>>> val_list = rebuild_partition(part1_vals, part2_vals, part1_indexes, part2_
↪ indexes)

```

(continues on next page)

(continued from previous page)

```
>>> assert val_list == item_list, 'incorrect inversin'
>>> print(val_list)
```

`vtool.rectify_invV_mats_are_up(invVR_mats)`

Useful if invVR_mats is no longer lower triangular rotates affine shape matrixes into downward (lower triangular) position

CommandLine: `python -m vtool.keypoint --exec-rectify_invV_mats_are_up --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> rng = np.random.RandomState(0)
>>> kpts = vt.demodata.get_dummy_kpts()[0:2]
>>> # Shrink x and y scales a bit
>>> kpts.T[2:4] /= 2
>>> kpts[1][3] *= 3 # increase skew
>>> # Set random orientation
>>> kpts.T[5] = TAU * np.array([.2, .6])
>>> invVR_mats = get_invVR_mats3x3(kpts)
>>> invVR_mats2, oris = rectify_invV_mats_are_up(invVR_mats)
>>> kpts2 = flatten_invV_mats_to_kpts(invVR_mats2)
>>> # Scale down in y a bit
>>> kpts2.T[1] += 100
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.show_kpts(np.vstack([kpts, kpts2]), ori=1, eig=True,
>>>               ori_color='green', rect=True)
>>> # Redraw oriented to show difference
>>> pt.draw_kpts2(kpts2, color='red', ell_linewidth=2, ori=1,
>>>               eig=True, ori_color='green', rect=True)
>>> ax = pt.gca()
>>> ax.set_aspect('auto')
>>> pt.dark_background()
>>> ut.show_if_requested()
```

```
pt.figure(doclf=True, fnum=pt.ensure_fnum(None)) ax = pt.gca() #ax.invert_yaxis() #pt.draw_kpts2(kpts,
color='blue', ell_linewidth=3, ori=1, eig=True, ori_color='green', rect=True) pt.draw_kpts2(kpts2,
color='red', ell_linewidth=2, ori=1, eig=True, ori_color='green', rect=True) extents =
np.array(vt.get_kpts_image_extent(np.vstack([kpts, kpts2]))) pt.set_axis_extent(extent, ax)
pt.dark_background() ut.show_if_requested()
```

Example

```
>>> from vtool.keypoint import * # NOQA
>>> rng = np.random.RandomState(0)
>>> invVR_mats = rng.rand(1000, 2, 2).astype(np.float64)
>>> output = rectify_invV_mats_are_up(invVR_mats)
>>> print(ut.hash_data(output))
oxvrkuiaffukpyalgxyhqikxgbuesutz
```

Ignore: `_invRs_2x2 = invVR_mats[:, 0:2, 0:2][0:1] A = _invRs_2x2[0] Q, R = np.linalg.qr(A)`

```

invVR_mats2, oris = rectify_invV_mats_are_up(_invRs_2x2[0:1]) L2, ori2 = invVR_mats2[0], oris[0] Q2
= vt.rotation_mat2x2(ori2)

np.linalg.det(Q)

vecs = np.random.rand(2, 4) Q2.dot(vecs) Q.dot(vecs)

np.linalg.cholesky(_invR_2x2)

```

```

vtool.rectify_to_float01(img, dtype=<class 'numpy.float32'>)

```

Ensure that an image is encoded using a float properly

```

vtool.rectify_to_square(img, extreme='max')

```

```

vtool.rectify_to_uint8(img)

```

Ensure that an image is encoded in uint8 properly

```

vtool.refine_inliers(kpts1, kpts2, fm, aff_inliers, xy_thresh_sqrd, scale_thresh=2.0, ori_thresh=1.57,
                    full_homog_checks=True, refine_method='homog')

```

Given a set of hypothesis inliers, computes a homography and refines inliers returned homography maps image1 space into image2 space

CommandLine: python -m vtool.spatial_verification --test-refine_inliers python -m vtool.spatial_verification --test-refine_inliers:0 python -m vtool.spatial_verification --test-refine_inliers:1 --show

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.demodata as demodata
>>> import vtool.keypoint as ktool
>>> kpts1, kpts2 = demodata.get_dummy_kpts_pair((100, 100))
>>> fm = demodata.make_dummy_fm(len(kpts1)).astype(np.int32)
>>> aff_inliers = np.arange(len(fm))
>>> xy_thresh_sqrd = .01 * ktool.get_kpts_dlen_sqrd(kpts2)
>>> homog_tup = refine_inliers(kpts1, kpts2, fm, aff_inliers, xy_thresh_sqrd)
>>> refined_inliers, refined_errors, H = homog_tup
>>> import ubelt as ub
>>> result = ub.repr2(homog_tup, precision=2, nl=True, suppress_small=True,
↳ nobr=True)
>>> print(result)

```

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.keypoint as ktool
>>> import wbia.plottool as pt
>>> kpts1, kpts2, fm, aff_inliers, rchip1, rchip2, xy_thresh_sqrd = testdata_
↳ matching_affine_inliers()
>>> homog_tup1 = refine_inliers(kpts1, kpts2, fm, aff_inliers, xy_thresh_sqrd)
>>> homog_tup = (homog_tup1[0], homog_tup1[2])
>>> # xdoctest: +REQUIRES(--show)
>>> pt.draw_sv.show_sv(rchip1, rchip2, kpts1, kpts2, fm, homog_tup=homog_tup)
>>> ut.show_if_requested()

```

`vtool.remove_homogenous_coordinate(_xyzs)`
normalizes 3d homogenous coordinates into 2d coordinates

Parameters `_xyzs` (*ndarray*) – of shape (3, N)

Returns `_xys` of shape (2, N)

Return type *ndarray*

CommandLine: `python -m vtool.linalg --test-remove_homogenous_coordinate`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.linalg import * # NOQA
>>> _xyzs = np.array([[ 2.,  0.,  0.,  2.],
...                  [ 2.,  2.,  0.,  0.],
...                  [ 1.2,  1.,  1.,  2.]], dtype=np.float32)
>>> _xys = remove_homogenous_coordinate(_xyzs)
>>> result = ub.repr2(_xys, precision=3, with_dtype=True)
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.linalg import * # NOQA
>>> _xyzs = np.array([[ 140.,  167.,  185.,  185.,  194.],
...                  [ 121.,  139.,  156.,  155.,  163.],
...                  [  47.,   56.,   62.,   62.,   65.]])
>>> _xys = remove_homogenous_coordinate(_xyzs)
>>> result = ub.repr2(_xys, precision=3)
>>> print(result)
```

`vtool.resize(img, dsize, interpolation=None)`

`vtool.resize_image_by_scale(img, scale, interpolation=None)`

`vtool.resize_mask(mask, chip, interpolation=None)`

`vtool.resize_thumb(img, max_dsize=(64, 64), interpolation=None)`

Resize an image such that its max width or height is:

CommandLine: `python -m vtool.image --test-resize_thumb --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> # build test data
>>> img_fpath = ut.grab_test_imgpath('car1.jpg')
>>> img = vt.imread(img_fpath)
>>> max_dsize = (64, 64)
>>> # execute function
>>> img2 = resize_thumb(img, max_dsize)
```

(continues on next page)

(continued from previous page)

```

>>> print('img.shape = %r' % (img.shape,))
>>> print('img2.shape = %r' % (img2.shape,))
>>> # verify results
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(img2)
>>> ut.show_if_requested()

```

`vtool.resize_to_maxdims (img, max_dsize=(64, 64), interpolation=None)`

Parameters

- **img** (`ndarray[uint8_t, ndim=2]`) – image data
- **max_dsize** (`tuple`) –
- **interpolation** (`long`) –

CommandLine: `python -m vtool.image -test-resize_to_maxdims --show`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> img_fpath = ut.grab_test_imgpath('carl.jpg')
>>> img = vt.imread(img_fpath)
>>> max_dsize = (1024, 1024)
>>> img2 = resize_to_maxdims(img, max_dsize)
>>> print('img.shape = %r' % (img.shape,))
>>> print('img2.shape = %r' % (img2.shape,))
>>> # verify results
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(img2)
>>> ut.show_if_requested()

```

`vtool.resize_to_maxdims_ondisk (img_fpath, max_dsize, out_fpath=None)`

Parameters

- **img_fpath** (`str`) – file path string
- **max_dsize** –
- **out_fpath** (`str`) – file path string (default = None)

CommandLine: `python -m vtool.image resize_to_maxdims_ondisk -fpath ~/latex/crall-candidacy-2015/figures3/knormA.png -dsize=417, None python -m vtool.image resize_to_maxdims_ondisk -fpath ~/latex/crall-candidacy-2015/figures3/knormB.png -dsize=417, None python -m vtool.image resize_to_maxdims_ondisk -fpath ~/latex/crall-candidacy-2015/figures3/knormC.png -dsize=417, None python -m vtool.image resize_to_maxdims_ondisk -fpath ~/latex/crall-candidacy-2015/figures3/knormD.png -dsize=417, None python -m vtool.image resize_to_maxdims_ondisk -fpath ~/latex/crall-candidacy-2015/figures3/knormE.png -dsize=417, None python -m vtool.image resize_to_maxdims_ondisk -fpath ~/latex/crall-candidacy-2015/figures3/knormF.png -dsize=417, None python -m vtool.image resize_to_maxdims_ondisk -fpath ~/latex/crall-candidacy-2015/figures3/knormG.png -dsize=417, None python -m vtool.image resize_to_maxdims_ondisk`

```
-fpath ~/latex/crall-candidacy-2015/figures3/knownH.png -dsize=417,None python -m
vtool.image resize_to_maxdims_ondisk -fpath ~/latex/crall-candidacy-2015/figures3/knownI.png
-dsize=417,None python -m vtool.image resize_to_maxdims_ondisk -fpath ~/latex/crall-candidacy-
2015/figures3/knownJ.png -dsize=417,None
```

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> img_fpath = ut.get_argval('--fpath')
>>> max_dsize = ut.get_argval('--dsize', type_=list)
>>> out_fpath = None
>>> resize_to_maxdims_ondisk(img_fpath, max_dsize, out_fpath)
```

`vtool.resized_clamped_thumb_dims(img_size, max_dsize)`

`vtool.resized_dims_and_ratio(img_size, max_dsize)`

returns resized dimensions to get `img_size` to fit into `max_dsize`

FIXME: Should specifying a `None` force the use of the original dim?

Parameters

- `img_size(tuple)` –
- `max_dsize(tuple)` –

Returns (dsize, ratio)

Return type `tuple`

CommandLine: `python -m vtool.image resized_dims_and_ratio --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> img_size = (200, 100)
>>> max_dsize = (150, 150)
>>> (dsize, ratio) = resized_dims_and_ratio(img_size, max_dsize)
>>> result = ('(dsize, ratio) = %s' % (ub.repr2((dsize, ratio), nl=0),))
>>> print(result)
(dsize, ratio) = ((150, 75), 0.75)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> img_size = (200, 100)
>>> max_dsize = (5000, 1000)
>>> (dsize, ratio) = resized_dims_and_ratio(img_size, max_dsize)
>>> result = ('(dsize, ratio) = %s' % (ub.repr2((dsize, ratio), nl=0),))
>>> print(result)
(dsize, ratio) = ((2000, 1000), 10.0)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> img_size = (200, 100)
>>> max_dsize = (5000, None)
>>> (dsize, ratio) = resized_dims_and_ratio(img_size, max_dsize)
>>> result = ('(dsize, ratio) = %s' % (ub.repr2((dsize, ratio), nl=0),))
>>> print(result)
(dsize, ratio) = ((200, 100), 1.0)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> img_size = (200, 100)
>>> max_dsize = (None, None)
>>> (dsize, ratio) = resized_dims_and_ratio(img_size, max_dsize)
>>> result = ('(dsize, ratio) = %s' % (ub.repr2((dsize, ratio), nl=0),))
>>> print(result)
(dsize, ratio) = ((200, 100), 1.0)
```

`vtool.rotate_image(img, theta, border_mode=None, interpolation=None, dsize=None)`

Rotates an image around its center

Parameters

- **img** (`ndarray[uint8_t, ndim=2]`) – image data
- **theta** –

CommandLine: `python -m vtool.image --test-rotate_image`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> # build test data
>>> img = vt.get_test_patch('star2')
>>> theta = TAU / 16.0
>>> # execute function
>>> imgR = rotate_image(img, theta)
>>> if ut.get_argflag('--show') or ut.inIPython():
>>>     import wbia.plottool as pt
>>>     pt.imshow(img * 255, pnum=(1, 2, 1))
>>>     pt.imshow(imgR * 255, pnum=(1, 2, 2))
>>>     pt.show_if_requested()
```

`vtool.rotate_image_ondisk(img_fpath, theta, out_fpath=None, **kwargs)`

Rotates an image on disk

Parameters

- **img_fpath** –
- **theta** –

- `out_fpath` (*None*) –

CommandLine: `python -m vtool.image --test-rotate_image_ondisk`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> # build test data
>>> img_fpath = ut.grab_test_imgpath('star.png')
>>> theta = TAU * 3 / 8
>>> # execute function
>>> out_fpath = None
>>> out_fpath_ = rotate_image_ondisk(img_fpath, theta, out_fpath)
>>> print(out_fpath_)
>>> if ut.get_argflag('--show') or ut.inIPython():
>>>     import wbia.plottool as pt
>>>     pt.imshow(out_fpath_, pnum=(1, 1, 1))
>>>     pt.show_if_requested()
```

`vtool.rotation_around_bbox_mat3x3(theta, bbox0, bbox1=None)`

`vtool.rotation_around_mat3x3(theta, x0, y0, x1=None, y1=None)`

`vtool.rotation_mat2x2(theta)`

`vtool.rotation_mat3x3(radians, sin=<ufunc 'sin'>, cos=<ufunc 'cos'>)`

References

https://en.wikipedia.org/wiki/Rotation_matrix

`vtool.rowwise_operation(arr1, arr2, op)`

DEPRICATE THIS IS POSSIBLE WITH STRICTLY BROADCASTING AND USING `np.newaxis`

DEPRICATE, numpy has better ways of doing this. Is the rowwise name correct? Should it be colwise?

performs an operation between an (N x A x B ... x Z) array with an (N x 1) array

`vtool.safe_argmax(arr, fill=nan, finite=False, nans=True)`

Doctest:

```
>>> from vtool.other import *
>>> assert safe_argmax([np.nan, np.nan], nans=False) == 0
>>> assert safe_argmax([-100, np.nan], nans=False) == 0
>>> assert safe_argmax([np.nan, -100], nans=False) == 1
>>> assert safe_argmax([-100, 0], nans=False) == 1
>>> assert np.isnan(safe_argmax([]))
```

`vtool.safe_cat(tup, axis=0, default_shape=(0,), default_dtype=<class 'numpy.float32'>)`

stacks a tuple even if it is empty Also deals with numpy bug where cat fails if an element in sequence is empty

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> import vtool as vt
>>> # test1
>>> tup = []
>>> ut.assert_eq(vt.safe_cat(tup, axis=0).shape, (0,))
>>> # test2
>>> tup = (np.array([[1, 2, 3]]), np.array([[ ]]))
>>> s = vt.safe_cat(tup, axis=0)
>>> print(ub.hzcat(['s = %s' % (ub.repr2(s), )]))
>>> ut.assert_eq(s.shape, (1, 3))
>>> # test3
>>> tup = (np.array([[1, 2, 3]]), np.array([[3, 4, 5]]))
>>> s = vt.safe_cat(tup, axis=1)
>>> print(ub.hzcat(['s = %s' % (ub.repr2(s), )]))
>>> ut.assert_eq(s.shape, (1, 6))
>>> # test3
>>> tup = (np.array(1), np.array(2), np.array(3))
>>> s = vt.safe_cat(tup, axis=1)
>>> print(ub.hzcat(['s = %s' % (ub.repr2(s), )]))
>>> ut.assert_eq(s.shape, (1, 6))

```

`vtool.safe_div(a, b)`

`vtool.safe_extreme(arr, op, fill=nan, finite=False, nans=True)`

Applies an extreme operation to an 1d array (typically max/min) but ensures a value is always returned even in operations without identities. The default identity must be specified using the *fill* argument.

Parameters

- **arr** (*ndarray*) – 1d array to take extreme of
- **op** (*func*) – vectorized operation like `np.max` to apply to array
- **fill** (*float*) – return type if arr has no elements (default = nan)
- **finite** (*bool*) – if True ignores non-finite values (default = False)
- **nans** (*bool*) – if False ignores nans (default = True)

`vtool.safe_max(arr, fill=nan, finite=False, nans=True)`

Parameters

- **arr** (*ndarray*) – 1d array to take max of
- **fill** (*float*) – return type if arr has no elements (default = nan)
- **finite** (*bool*) – if True ignores non-finite values (default = False)
- **nans** (*bool*) – if False ignores nans (default = True)

CommandLine: `python -m vtool.other safe_max -show`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> arrs = [[], [np.nan], [-np.inf, np.nan, np.inf], [np.inf], [np.inf, 1], [0,
↪ 1]]

```

(continues on next page)

(continued from previous page)

```
>>> arrs = [np.array(arr) for arr in arrs]
>>> fill = np.nan
>>> results1 = [safe_max(arr, fill, finite=False, nans=True) for arr in arrs]
>>> results2 = [safe_max(arr, fill, finite=True, nans=True) for arr in arrs]
>>> results3 = [safe_max(arr, fill, finite=True, nans=False) for arr in arrs]
>>> results4 = [safe_max(arr, fill, finite=False, nans=False) for arr in arrs]
>>> results = [results1, results2, results3, results4]
>>> result = ('results = %s' % (ub.repr2(results, nl=1),))
>>> print(result)
results = [
    [float('nan'), float('nan'), float('nan'), float('inf'), float('inf'), 1],
    [float('nan'), float('nan'), float('nan'), float('nan'), 1.0, 1],
    [float('nan'), float('nan'), float('nan'), float('nan'), 1.0, 1],
    [float('nan'), float('nan'), float('inf'), float('inf'), float('inf'), 1],
]
```

`vtool.safe_min(arr, fill=nan, finite=False, nans=True)`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> arrs = [[], [np.nan], [-np.inf, np.nan, np.inf], [np.inf], [np.inf, 1], [0,
↳ 1]]
>>> arrs = [np.array(arr) for arr in arrs]
>>> fill = np.nan
>>> results1 = [safe_min(arr, fill, finite=False, nans=True) for arr in arrs]
>>> results2 = [safe_min(arr, fill, finite=True, nans=True) for arr in arrs]
>>> results3 = [safe_min(arr, fill, finite=True, nans=False) for arr in arrs]
>>> results4 = [safe_min(arr, fill, finite=False, nans=False) for arr in arrs]
>>> results = [results1, results2, results3, results4]
>>> result = ('results = %s' % (ub.repr2(results, nl=1),))
>>> print(result)
results = [
    [float('nan'), float('nan'), float('nan'), float('inf'), 1.0, 0],
    [float('nan'), float('nan'), float('nan'), float('nan'), 1.0, 0],
    [float('nan'), float('nan'), float('nan'), float('nan'), 1.0, 0],
    [float('nan'), float('nan'), float('-inf'), float('inf'), 1.0, 0],
]
```

`vtool.safe_pdist(arr, *args, **kwargs)`

Kwargs: `metric = ut.absdiff`

SeeAlso: `scipy.spatial.distance.pdist`

`vtool.safe_vstack(tup, default_shape=(0,), default_dtype=<class 'numpy.float32'>)`
stacks a tuple even if it is empty

`vtool.sample_ell_border_pts(expanded_kpts, nSamples)`

`vtool.sample_ell_border_vals(imgBGR, expanded_kpts, nKp, nScales, nSamples)`

`vtool.sample_uniform(kpts, nSamples=128)`

SeeAlso: `python -m pyhesaff.tests.test_ellipse --test-in_depth_ellipse --show`

`vtool.scale_around_mat3x3(sx, sy, x, y)`

`vtool.scale_bbox (bbox, sx, sy=None)`

`vtool.scale_extents (extents, sx, sy=None)`

Parameters `extent (ndarray)` – tl_x, br_x, tl_y, br_y

`vtool.scale_mat3x3 (sx, sy=None, dtype=<class 'numpy.float64'>)`

`vtool.scaled_verts_from_bbox (bbox, theta, sx, sy)`

Helps with drawing scaled bounding boxes on thumbnails

`vtool.scaled_verts_from_bbox_gen (bbox_list, theta_list, sx=1, sy=1)`

Helps with drawing scaled bounding boxes on thumbnails

Parameters

- `bbox_list (list)` – bboxes in x,y,w,h format
- `theta_list (list)` – rotation of bounding boxes
- `sx (float)` – x scale factor
- `sy (float)` – y scale factor

Yields: `new_verts` - vertices of scaled bounding box for every input

CommandLine: `python -m vtool.image --test-scaled_verts_from_bbox_gen`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> # build test data
>>> bbox_list = [(10, 10, 100, 100)]
>>> theta_list = [0]
>>> sx = .5
>>> sy = .5
>>> # execute function
>>> new_verts_list = list(scaled_verts_from_bbox_gen(bbox_list, theta_list, sx, sy))
>>> result = str(new_verts_list)
>>> # verify results
>>> print(result)
[[[5, 5], [55, 5], [55, 55], [5, 55], [5, 5]]]
```

`vtool.shear (img, x_shear, y_shear, dsize=None, **kwargs)`

Parameters

- `img (ndarray[uint8_t, ndim=2])` – image data
- `x_shear` –
- `y_shear` –
- `dsize (tuple)` – width, height

CommandLine: `python -m vtool.image --test-shear --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> img_fpath = ut.grab_test_imgpath('carl.jpg')
>>> img = vt.imread(img_fpath)
>>> x_shear = 0.05
>>> y_shear = -0.05
>>> dsize = None
>>> imgSh = shear(img, x_shear, y_shear, dsize)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(imgSh)
>>> ut.show_if_requested()
```

`vtool.shear_mat3x3 (shear_x, shear_y, dtype=<class 'numpy.float64'>)`

`vtool.show_gaussian_patch (shape, sigma1, sigma2)`

`vtool.show_hist_submaxima (hist_, edges=None, centers=None, maxima_thresh=0.8, pnum=(1, 1, 1))`

For C++ to show data

Parameters

- **hist** –
- **edges** (*None*) –
- **centers** (*None*) –

CommandLine: `python -m vtool.histogram --test-show_hist_submaxima --show python -m pyhesaff._pyhesaff --test-test_rot_invar --show python -m vtool.histogram --test-show_hist_submaxima --dpath figures --save ~/latex/crall-candidacy-2015/figures/show_hist_submaxima.jpg`

Example

```
>>> # xdoctest: +REQUIRES(module:wbia)
>>> import wbia.plottool as pt
>>> from vtool.histogram import * # NOQA
>>> hist_ = np.array(list(map(float, ut.get_argval('--hist', type_=list,
↪ default=[1, 4, 2, 5, 3, 3]))))
>>> edges = np.array(list(map(float, ut.get_argval('--edges', type_=list,
↪ default=[0, 1, 2, 3, 4, 5, 6]))))
>>> maxima_thresh = ut.get_argval('--maxima_thresh', type_=float, default=.8)
>>> centers = None
>>> show_hist_submaxima(hist_, edges, centers, maxima_thresh)
>>> pt.show_if_requested()
```

`vtool.show_ori_image (gori, weights, patch, gradx=None, grady=None, gauss=None, fnum=None)`

CommandLine: `python -m pyhesaff._pyhesaff --test-test_rot_invar --show --nocpp`

`vtool.show_ori_image_ondisk ()`

CommandLine: `python -m vtool.histogram --test-show_ori_image_ondisk --show`

`python -m vtool.histogram --test-show_ori_image_ondisk --show --patch_img_fpath patches/KP_0_PATCH.png --ori_img_fpath patches/KP_0_orientations01.png --weights_img_fpath`


```

patches/KP_0_WEIGHTS.png -grady_img_fpath patches/KP_0_ygradient.png -gradx_img_fpath
patches/KP_0_xgradient.png -title cpp_show_ori_ondisk

python -m pyhesaff._pyhesaff --test-test_rot_invar --show --rebuild-hesaff --no-rmbuild

```

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> import wbia.plottool as pt
>>> import vtool as vt
>>> result = show_ori_image_ondisk()
>>> pt.show_if_requested()

```

```

vtool.show_patch_orientation_estimation(imgBGR, kpts, patch, gradx, grady, gmag, gori,
                                         hist, centers, gori_weights, fx=None)

```

```

vtool.signed_cyclic_distance(arr1, arr2, modulo, out=None)

```

```

vtool.signed_ori_distance(ori1, ori2)

```

Parameters

- **ori1** (*ndarray*) –
- **ori2** (*ndarray*) –

Returns ori_dist

Return type ndarray

CommandLine: python -m vtool.distance --exec-signed_ori_distance

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> ori1 = np.array([0, 0, 3, 4, 0, 0])
>>> ori2 = np.array([3, 4, 0, 0, np.pi, np.pi - .1])
>>> ori_dist = signed_ori_distance(ori1, ori2)
>>> result = ('ori_dist = %s' % (ub.repr2(ori_dist, precision=3),))

```

```

vtool.significant_shape(arr)
    find the shape without trailing 1's

```

```

vtool.sorted_indices_ranges(groupids_sorted)
    Like group sorted indices but returns a list of slices

```

```

vtool.spatially_verify_kpts(kpts1, kpts2, fm, xy_thresh=0.01, scale_thresh=2.0,
                           ori_thresh=1.5707963267948966, dlen_sqrd2=None,
                           min_nInliers=4, match_weights=None, returnAff=False,
                           full_homog_checks=True, refine_method='homog',
                           max_nInliers=5000)

```

Driver function Spatially validates feature matches

FIXME: there is a non-determinism here

Returned homography maps image1 space into image2 space.

Parameters

- **kpts1** (*ndarray [ndim=2]*) – all keypoints in image 1
- **kpts2** (*ndarray [ndim=2]*) – all keypoints in image 2
- **fm** (*ndarray [ndim=2]*) – matching keypoint indexes [..., (kp1x, kp2x), ...]
- **xy_thresh** (*float*) – spatial distance threshold under affine transform to be considered a match
- **scale_thresh** (*float*) –
- **ori_thresh** (*float*) –
- **dlen_sqrd2** (*float*) – diagonal length squared of image/chip 2
- **min_nInliers** (*int*) – default=4
- **returnAff** (*bool*) – returns best affine hypothesis as well
- **max_nInliers** (*int*) – homog is not considered after this threshold

Returns (refined_inliers, refined_errors, H, aff_inliers, aff_errors, Aff) if success else None

Return type *tuple*

CommandLine: python -m xdoctest vtool.spatial_verification spatially_verify_kpts:0 --show python -m xdoctest vtool.spatial_verification spatially_verify_kpts:0 --show --refine-method='affine' python -m xdoctest vtool.spatial_verification spatially_verify_kpts:0 --dpath figures --show --save ~/latex/crall-candidacy-2015/figures/sver_kpts.jpg # NOQA python -m xdoctest vtool.spatial_verification spatially_verify_kpts:0

Example

```
>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.spatial_verification import *
>>> import vtool.demodata as demodata
>>> import vtool as vt
>>> fname1 = ut.get_argval('--fname1', type_=str, default='easy1.png')
>>> fname2 = ut.get_argval('--fname2', type_=str, default='easy2.png')
>>> default_dict = vt.get_extract_features_default_params()
>>> default_dict['ratio_thresh'] = .625
>>> kwargs = ut.argparse_dict(default_dict)
>>> (kpts1, kpts2, fm, fs, rchip1, rchip2) = demodata.testdata_ratio_
↳ matches(fname1, fname2, **kwargs)
>>> xy_thresh = .01
>>> dlen_sqrd2 = 447271.015
>>> ori_thresh = 1.57
>>> min_nInliers = 4
>>> returnAff = True
>>> scale_thresh = 2.0
>>> match_weights = np.ones(len(fm), dtype=np.float64)
>>> refine_method = ut.get_argval('--refine-method', default='homog')
>>> svtup = spatially_verify_kpts(kpts1, kpts2, fm, xy_thresh,
>>>                               scale_thresh, ori_thresh, dlen_sqrd2,
>>>                               min_nInliers, match_weights, returnAff,
>>>                               refine_method=refine_method)
>>> assert svtup is not None and len(svtup) == 6, 'sver failed'
>>> refined_inliers, refined_errors, H = svtup[0:3]
>>> aff_inliers, aff_errors, Aff = svtup[3:6]
>>> #print('aff_errors = %r' % (aff_errors,))
```

(continues on next page)

(continued from previous page)

```

>>> print('aff_inliers = %r' % (aff_inliers,))
>>> print('refined_inliers = %r' % (refined_inliers,))
>>> #print('refined_errors = %r' % (refined_errors,))
>>> import ubelt as ub
>>> result = ut.list_type_profile(svtup, with_dtype=False)
>>> #result = ub.repr2(svtup, precision=3)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> homog_tup = (refined_inliers, H)
>>> aff_tup = (aff_inliers, Aff)
>>> pt.draw_sv.show_sv(rchip1, rchip2, kpts1, kpts2, fm, aff_tup=aff_tup, homog_
↳ tup=homog_tup, refine_method=refine_method)
>>> pt.show_if_requested()
tuple(numpy.ndarray, tuple(numpy.ndarray*3), numpy.ndarray, numpy.ndarray,
↳ tuple(numpy.ndarray*3), numpy.ndarray)

```

`vtool.stack_image_list` (`img_list`, `return_offset=False`, `return_sf=False`, `return_info=False`, `**kwargs`)

CommandLine: `python -m vtool.image --test-stack_image_list --show`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> # build test data
>>> img_list = testdata_imglist()
>>> vert = False
>>> return_offset = True
>>> modifysize = True
>>> return_sf=True
>>> kwargs = dict(modifysize=modifysize, vert=vert, use_larger=False)
>>> # execute function
>>> imgB, offset_list, sf_list = stack_image_list(img_list, return_offset=return_
↳ offset, return_sf=return_sf, **kwargs)
>>> # verify results
>>> result = ub.repr2(np.array(offset_list).T, precision=2, with_dtype=True)
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(imgB)
>>> wh_list = np.array([vt.get_size(img) for img in img_list])
>>> wh_list_ = wh_list * sf_list
>>> for offset, wh, color in zip(offset_list, wh_list_, pt.distinct_
↳ colors(len(offset_list))):
...     pt.draw_bbox((offset[0], offset[1], wh[0], wh[1]), bbox_color=color)
>>> pt.show_if_requested()
>>> #wh1 = img1.shape[0:2][::-1]
>>> #wh2 = img2.shape[0:2][::-1]
>>> #pt.draw_bbox((0, 0) + wh1, bbox_color=(1, 0, 0))
>>> #pt.draw_bbox((woff, hoff) + wh2, bbox_color=(0, 1, 0))
np.array([[ 0. , 76.96, 141.08, 181.87, 246. ],
          [ 0. , 0. , 0. , 0. , 0. ]], dtype=np.float64)

```

```
vtool.stack_image_list_special(img1, img_list, num=1, vert=True, use_larger=True, initial_sf=None, interpolation=None)
```

TODO: add initial scale down factor?

CommandLine: python -m vtool.image --test-stack_image_list_special --show

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> img_list_ = testdata_imglist()
>>> img1 = img_list_[0]
>>> img_list = img_list_[1:]
>>> vert = True
>>> return_offset = True
>>> use_larger = False
>>> num_bot = 1
>>> initial_sf = None
>>> initial_sf = .5
>>> imgB, offset_list, sf_list = stack_image_list_special(img1, img_list, num_bot,
↳ vert, use_larger, initial_sf)
>>> # xdoctest: +REQUIRES(--show)
>>> wh_list = np.array([vt.get_size(img1)] + list(map(vt.get_size, img_list)))
>>> wh_list_ = wh_list * sf_list
>>> import wbia.plottool as pt
>>> pt.imshow(imgB)
>>> print('imgB.shape = %r' % (imgB.shape,))
>>> for offset, wh, color in zip(offset_list, wh_list_, pt.distinct_
↳ colors(len(offset_list))):
...     pt.draw_bbox((offset[0], offset[1], wh[0], wh[1]), bbox_color=color)
>>> ut.show_if_requested()
```

```
vtool.stack_image_recurse(img_list1, img_list2=None, vert=True, modifysize=False, return_offsets=False, interpolation=None)
```

TODO: return offsets as well

Parameters

- **img_list1** (*list*) –
- **img_list2** (*list*) –
- **vert** (*bool*) –

Returns None

Return type ndarray

CommandLine: python -m vtool.image --test-stack_image_recurse --show

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> # build test data
```

(continues on next page)

(continued from previous page)

```

>>> img1 = vt.imread(ut.grab_test_imgpath('carl.jpg'))
>>> img2 = vt.imread(ut.grab_test_imgpath('astro.png'))
>>> img3 = vt.imread(ut.grab_test_imgpath('ada.jpg'))
>>> img4 = vt.imread(ut.grab_test_imgpath('jeff.png'))
>>> img5 = vt.imread(ut.grab_test_imgpath('star.png'))
>>> img_list1 = [img1, img2, img3, img4, img5]
>>> img_list2 = None
>>> vert = True
>>> # execute function
>>> imgB = stack_image_recurse(img_list1, img_list2, vert)
>>> # verify results
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> imshow(imgB)
>>> #wh1 = img1.shape[0:2][::-1]
>>> #wh2 = img2.shape[0:2][::-1]
>>> #pt.draw_bbox((0, 0) + wh1, bbox_color=(1, 0, 0))
>>> #pt.draw_bbox((woff, hoff) + wh2, bbox_color=(0, 1, 0))
>>> pt.show_if_requested()

```

`vtool.stack_images` (*img1*, *img2*, *vert=None*, *modifysize=False*, *return_sf=False*, *use_larger=True*, *interpolation=None*, *white_background=False*, *overlap=0*)

Parameters

- **img1** (*ndarray*[*uint8_t*, *ndim=2*]) – image data
- **img2** (*ndarray*[*uint8_t*, *ndim=2*]) – image data

CommandLine: `python -m vtool.image --test-stack_images --show`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> # build test data
>>> img1 = vt.imread(ut.grab_test_imgpath('carl.jpg'))
>>> img2 = vt.imread(ut.grab_test_imgpath('astro.png'))
>>> vert = True
>>> modifysize = False
>>> # execute function
>>> return_sf = True
>>> #(imgB, woff, hoff) = stack_images(img1, img2, vert, modifysize, return_
↪sf=return_sf)
>>> overlap = 100
>>> imgB, offset2, sf_tup = stack_images(img1, img2, vert, modifysize,
>>>                                     return_sf=return_sf,
>>>                                     overlap=overlap)
>>> woff, hoff = offset2
>>> # verify results
>>> result = str((imgB.shape, woff, hoff))
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(imgB)

```

(continues on next page)

(continued from previous page)

```
>>> wh1 = np.multiply(vt.get_size(img1), sf_tup[0])
>>> wh2 = np.multiply(vt.get_size(img2), sf_tup[1])
>>> pt.draw_bbox((0, 0, wh1[0], wh1[1]), bbox_color=(1, 0, 0))
>>> pt.draw_bbox((woff[1], hoff[1], wh2[0], wh2[0]), bbox_color=(0, 1, 0))
>>> pt.show_if_requested()
((662, 512, 3), (0.0, 0.0), (0, 150))
```

`vtool.stack_multi_images(img1, img2, offset_list1, sf_list1, offset_list2, sf_list2, vert=True, use_larger=False, modifysize=True, interpolation=None)`
 combines images that are already stacked

`vtool.stack_multi_images2(multiimg_list, offsets_list, sfs_list, vert=True, modifysize=True)`

Parameters

- `multiimg_list` (*list*) –
- `offset_lists` –
- `sfs_list` –
- `vert` (*bool*) –

Returns (stacked_img, stacked_img, stacked_sfs)

Return type `tuple`

CommandLine: `python -m vtool.image --test-stack_multi_images2 --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> img_list = testdata_imglist()
>>> img_stack1, offset_list1, sf_list1 = stack_image_list(img_list[:-1],
↳vert=True, return_info=True, modifysize=True)
>>> img_stack2, offset_list2, sf_list2 = stack_image_list(img_list, vert=True,
↳return_info=True, modifysize=True)
>>> img_stack3, offset_list3, sf_list3 = stack_image_list(img_list, vert=True,
↳return_info=True, modifysize=False)
>>> multiimg_list = [img_stack1, img_stack2, img_stack3]
>>> offsets_list = [offset_list1, offset_list2, offset_list3]
>>> sfs_list = [sf_list1, sf_list2, sf_list3]
>>> vert = False
>>> tup = stack_multi_images2(multiimg_list, offsets_list, sfs_list, vert)
>>> (stacked_img, stacked_offsets, stacked_sfs) = tup
>>> result = ut.remove_doublspaces(ub.repr2(np.array(stacked_offsets).T,
↳precision=2, with_dtype=True, linewidth=10000)).replace(' ', ', ')
>>> print(result)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(stacked_img)
>>> wh_list = np.array([vt.get_size(img) for img in img_list[:-1] + img_list +
↳img_list])
>>> wh_list_ = wh_list * stacked_sfs
>>> for offset, wh, color in zip(stacked_offsets, wh_list_, pt.distinct_
↳colors(len(stacked_offsets))):
```

(continues on next page)

(continued from previous page)

```

...     pt.draw_bbox((offset[0], offset[1], wh[0], wh[1]), bbox_color=color)
>>> ut.show_if_requested()
np.array([[ 0.,  0.,  0.,  0.,  0., 512., 512., 512., 512., 512., 1024., 1024., 1024.,
↪ 1024., 1024. ],
[ 0., 512.12, 1024.25, 1827., 2339., 0., 427., 939., 1742., 2254., 0., 373.18,
↪ 1137.45, 2073.38, 2670.47]], dtype=np.float64)

```

vtool.stack_square_images (*img_list*, *return_info=False*, ***kwargs*)

Parameters *img_list* (*list*) –

Returns

Return type ndarray

CommandLine: python -m vtool.image --test-stack_square_images

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> img_list = '?'
>>> result = stack_square_images(img_list)
>>> print(result)

```

vtool.strictly_decreasing (*L*)

References

<http://stackoverflow.com/questions/4983258/python-how-to-check-list-monotonicity>

vtool.strictly_increasing (*L*)

References

<http://stackoverflow.com/questions/4983258/python-how-to-check-list-monotonicity>

vtool.structure_rows (**arrs*)

CommandLine: python -m vtool.other structure_rows

SeeAlso: unstructure_rows

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> arr1 = np.array([[609, 307], [ 95, 344], [ 1, 690]])
>>> arr2 = np.array([[ 422, 1148], [ 422, 968], [ 481, 1148], [ 750, 1132], [
↪ 759, 159]])
>>> arrs = (arr1, arr2)
>>> structured_arrs = structure_rows(*arrs)
>>> unstructured_arrs = unstructure_rows(*structured_arrs)
>>> assert np.all(unstructured_arrs[0] == arrs[0])

```

(continues on next page)

(continued from previous page)

```
>>> assert np.all(unstructured_arrs[1] == arrs[1])
>>> union_ = np.union1d(*structured_arrs)
>>> union, = unstructure_rows(union_)
>>> assert len(union.shape) == 2
```

`vtool.subbin_bounds` (*z*, *radius*, *low*, *high*)

Gets quantized bounds of a sub-bin/pixel point and a radius. Useful for cropping using subpixel points

Parameters

- **z** (*float*) – center of a circle a 1d pixel array
- **radius** (*float*) – radius of the circle
- **low** (*int*) – minimum index of 1d pixel array
- **high** (*int*) – maximum index of 1d pixel array

Returns

(*iz1*, *iz2*, *z_offst*) - **quantized_bounds and subbin_offset** *iz1* - low radius endpoint *iz2* - high radius endpoint *z_offst* - subpixel offset #Returns: quantized_bounds=(*iz1*, *iz2*), subbin_offset

Return type `tuple`

CommandLine: `python -m vtool.histogram --test-subbin_bounds`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> z = 1.5
>>> radius = 5.666
>>> low = 0
>>> high = 7
>>> (iz1, iz2, z_offst) = subbin_bounds(z, radius, low, high)
>>> result = str((iz1, iz2, z_offst))
>>> print(result)
(0, 7, 1.5)
```

`vtool.subpixel_values` (*img*, *pts*)

References

stackoverflow.com/questions/12729228/simple-efficient-bilinear-interpolation-of-images-in-numpy-and-python

SeeAlso: `cv2.getRectSubPix(image, patchSize, center[, patch[, patchType]])`

`vtool.subscale_peaks` (*border_vals_sum*, *kpts*, *nScales*, *low*, *high*)

`vtool.svd` (*M*)

Parameters **M** (*ndarray*) – must be either float32 or float64

Returns (*U*, *s*, *Vt*)

Return type `tuple`

CommandLine: python -m vtool.linalg --test-svd

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.linalg import * # NOQA
>>> # build test data
>>> M = np.array([1, 2, 3], dtype=np.float32)
>>> M = np.array([[20.5812, 0], [3.615, 17.1295]], dtype=np.float64)
>>> # execute function
>>> (U, s, Vt) = svd(M)
```

Ignore: flags = cv2.SVD_FULL_UV %timeit cv2.SVDecomp(M, flags=flags) %timeit npl.svd(M)

vtool.**symbolic_randcheck**(expr1, expr2, domain={}, n=10)

vtool.**symmetric_correspondence**(annot1, annot2, K, Knorm, checks, allow_shrink=True)

Find symmetric feature correspondences

vtool.**sympy_latex_repr**(expr1)

vtool.**sympy_mat**(arr)

vtool.**sympy_numpy_repr**(expr1)

vtool.**take2**(arr, index_list, axis=None, out=None)

Wrapper around numpy compress that makes the signature more similar to take

vtool.**take_col_per_row**(arr, colx_list)

takes a column from each row

Ignore: num_rows = 1000 num_cols = 4

```
arr = np.arange(10 * 4).reshape(10, 4) colx_list = (np.random.rand(10) * 4).astype(np.int)
```

```
%timeit np.array([row[cx] for (row, cx) in zip(arr, colx_list)]) %timeit
arr.ravel().take(np.ravel_multi_index((np.arange(len(colx_list)), colx_list), arr.shape)) %timeit
arr.ravel().take(colx_list + np.arange(arr.shape[0]) * arr.shape[1])
```

vtool.**test_affine_errors**(H, kpts1, kpts2, fm, xy_thresh_sqrd, scale_thresh_sqrd, ori_thresh)

used for refinement as opposed to initial estimation

vtool.**test_annoy**()

vtool.**test_cv2_flann**()

Ignore: [name for name in dir(cv2) if 'create' in name.lower()] [name for name in dir(cv2) if 'stereo' in name.lower()]

```
ut.grab_zipped_url('https://priithon.googlecode.com/archive/a6117f5e81ec00abcfb037f0f9da2937bb2ea47f.
tar.gz', download_dir='.')
```

vtool.**test_homog_errors**(H, kpts1, kpts2, fm, xy_thresh_sqrd, scale_thresh, ori_thresh, full_homog_checks=True)

Test to see which keypoints the homography correctly maps

Parameters

- **H** (ndarray [float64_t, ndim=2]) – homography/perspective matrix
- **kpts1** (ndarray [float32_t, ndim=2]) – keypoints
- **kpts2** (ndarray [float32_t, ndim=2]) – keypoints

- **fm**(*list*) – list of feature matches as tuples (qfx, dfx)
- **xy_thresh_sqrd**(*float*) –
- **scale_thresh**(*float*) –
- **ori_thresh**(*float*) – angle in radians
- **full_homog_checks**(*bool*) –

Returns homog_tup1

Return type tuple

CommandLine: python -m vtool.spatial_verification -test-test_homog_errors:0 -show python -m vtool.spatial_verification -test-test_homog_errors:0 -show -rotation_invariance python -m vtool.spatial_verification -test-test_homog_errors:0 -show -rotation_invariance -no-affine-invariance -xy-thresh=.001 python -m vtool.spatial_verification -test-test_homog_errors:0 -show -rotation_invariance -no-affine-invariance -xy-thresh=.001 -no-full-homog-checks python -m vtool.spatial_verification -test-test_homog_errors:0 -show -no-full-homog-checks # _____ # Shows (sorta) how inliers are computed python -m vtool.spatial_verification -test-test_homog_errors:1 -show python -m vtool.spatial_verification -test-test_homog_errors:1 -show -rotation_invariance python -m vtool.spatial_verification -test-test_homog_errors:1 -show -rotation_invariance -no-affine-invariance -xy-thresh=.001 python -m vtool.spatial_verification -test-test_homog_errors:1 -show -rotation_invariance -xy-thresh=.001 python -m vtool.spatial_verification -test-test_homog_errors:0 -show -rotation_invariance -xy-thresh=.001

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.spatial_verification import * # NOQA
>>> import wbia.plottool as pt
>>> kpts1, kpts2, fm, aff_inliers, rchip1, rchip2, xy_thresh_sqrd = testdata_
↳matching_affine_inliers()
>>> H = estimate_refined_transform(kpts1, kpts2, fm, aff_inliers)
>>> scale_thresh, ori_thresh = 2.0, 1.57
>>> full_homog_checks = not ut.get_argflag('--no-full-homog-checks')
>>> homog_tup1 = test_homog_errors(H, kpts1, kpts2, fm, xy_thresh_sqrd, scale_
↳thresh, ori_thresh, full_homog_checks)
>>> homog_tup = (homog_tup1[0], homog_tup1[2])
>>> # xdoctest: +REQUIRES(--show)
>>> pt.draw_sv.show_sv(rchip1, rchip2, kpts1, kpts2, fm, homog_tup=homog_tup)
>>> ut.show_if_requested()
```

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.spatial_verification import * # NOQA
>>> import wbia.plottool as pt
>>> kpts1, kpts2, fm_, aff_inliers, rchip1, rchip2, xy_thresh_sqrd = testdata_
↳matching_affine_inliers()
>>> H = estimate_refined_transform(kpts1, kpts2, fm_, aff_inliers)
>>> scale_thresh, ori_thresh = 2.0, 1.57
>>> full_homog_checks = not ut.get_argflag('--no-full-homog-checks')
>>> # -----
>>> # Take subset of feature matches
```

(continues on next page)

(continued from previous page)

```

>>> fm = fm_
>>> scale_err, xy_err, ori_err = \
...     ut.exec_func_src(test_homog_errors, globals(), locals(),
...     'scale_err, xy_err, ori_err'.split(', '))
>>> # we only care about checking out scale and orientation here. ignore bad xy_
↳points
>>> xy_inliers_flag = np.less(xy_err, xy_thresh_sqrd)
>>> scale_err[~xy_inliers_flag] = 0
>>> # filter
>>> fm = fm[np.array(scale_err).argsort()[::-1][:10]]
>>> fm = fm[np.array(scale_err).argsort()[::-1][:10]]
>>> # Exec sourcecode
>>> kpts1_m, kpts2_m, off_xy1_m, off_xy1_mt, dxy1_m, dxy1_mt, xy2_m, xyl_m, xyl_
↳mt, scale_err, xy_err, ori_err = \
...     ut.exec_func_src(test_homog_errors, globals(), locals(),
...     'kpts1_m, kpts2_m, off_xy1_m, off_xy1_mt, dxy1_m, dxy1_mt, xy2_m, xyl_m,
↳xyl_mt, scale_err, xy_err, ori_err'.split(', '))
>>> #-----
>>> # xdoctest: +REQUIRES(--show)
>>> pt.figure(fnum=1, pnum=(1, 2, 1), title='orig points and offset point')
>>> segments_list1 = np.array(list(zip(xyl_m.T.tolist(), off_xy1_m.T.tolist())))
>>> pt.draw_line_segments(segments_list1, color=pt.LIGHT_BLUE)
>>> pt.dark_background()
>>> #-----
>>> pt.figure(fnum=1, pnum=(1, 2, 2), title='transformed points and matching_
↳points')
>>> #-----
>>> # first have to make corresponding offset points
>>> # Use reference point for scale and orientation tests
>>> oris2_m = ktool.get_oris(kpts2_m)
>>> scales2_m = ktool.get_scales(kpts2_m)
>>> dxy2_m = np.vstack((np.sin(oris2_m), -np.cos(oris2_m)))
>>> scaled_dxy2_m = dxy2_m * scales2_m[None, :]
>>> off_xy2_m = xy2_m + scaled_dxy2_m
>>> # Draw transformed segments
>>> segments_list2 = np.array(list(zip(xy2_m.T.tolist(), off_xy2_m.T.tolist())))
>>> pt.draw_line_segments(segments_list2, color=pt.GREEN)
>>> # Draw corresponding matches segments
>>> segments_list3 = np.array(list(zip(xyl_mt.T.tolist(), off_xy1_mt.T.tolist())))
>>> pt.draw_line_segments(segments_list3, color=pt.RED)
>>> # Draw matches between correspondences
>>> segments_list4 = np.array(list(zip(xyl_mt.T.tolist(), xy2_m.T.tolist())))
>>> pt.draw_line_segments(segments_list4, color=pt.ORANGE)
>>> pt.dark_background()
>>> #-----
>>> #vt.get_xy_axis_extents(kpts1_m)
>>> #pt.draw_sv.show_sv(rchip1, rchip2, kpts1, kpts2, fm, homog_tup=homog_tup)
>>> ut.show_if_requested()

```

vtool.test_language_modulus()

References

http://en.wikipedia.org/wiki/Modulo_operation

vtool.test_mser()

```
vtool.test_ondisk_find_patch_fpath_dominant_orientations(patch_fpath, bins=36,
                                                         maxima_thresh=0.8, DE-
                                                         BUG_ROTINVAR=True)
```

Parameters

- **patch_fpath** –
- **bins** (*int*) –
- **maxima_thresh** (*float*) –

CommandLine: `python -m vtool.patch -test-test_ondisk_find_patch_fpath_dominant_orientations`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> import wbia.plottool as pt
>>> # build test data
>>> patch_fpath = ut.get_argval('--patch-fpath', type=str, default=ut.grab_test_
↳imgpath('star.png'))
>>> bins = 36
>>> maxima_thresh = 0.8
>>> test_ondisk_find_patch_fpath_dominant_orientations(patch_fpath, bins, maxima_
↳thresh)
>>> pt.show_if_requested()
```

```
vtool.test_score_normalization(tp_support, tn_support, with_scores=True, verbose=True,
                              with_roc=True, with_precision_recall=False, figtitle=None,
                              normkw_varydict=None)
```

Gives an overview of how well threshold can be learned from raw scores.

DEPRICATE

CommandLine: `python -m vtool.score_normalization -test-test_score_normalization -show`

CommandLine: `xdoctest -m ~/code/vtool/vtool/score_normalization.py test_score_normalization`

Ignore:

```
>>> # GUI_DOCTEST
>>> # Shows how score normalization works with gaussian noise
>>> from vtool.score_normalization import * # NOQA
>>> verbose = True
>>> randstate = np.random.RandomState(seed=0)
>>> # Get a training sample
>>> tp_support = randstate.normal(loc=6.5, size=(256,))
>>> tn_support = randstate.normal(loc=3.5, size=(256,))
>>> # xdoctest: +REQUIRES(module:plottool)
>>> test_score_normalization(tp_support, tn_support, verbose=verbose)
>>> ut.show_if_requested()
```

```
vtool.test_show_gaussian_patches(shape=(19, 19))
```

CommandLine: `python -m vtool.patch -test-test_show_gaussian_patches -show python -m vtool.patch -test-test_show_gaussian_patches -show -shape=7,7 python -m vtool.patch -test-test_show_gaussian_patches -show -shape=17,17 python -m vtool.patch -test-test_show_gaussian_patches -show -shape=41,41 python -m vtool.patch -test-test_show_gaussian_patches -show -shape=29,29 python -m vtool.patch -test-test_show_gaussian_patches -show -shape=41,7`

References

http://matplotlib.org/examples/mplot3d/surface3d_demo.html

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> from mpl_toolkits.mplot3d import Axes3D # NOQA
>>> import wbia.plottool as pt
>>> shape = ut.get_argval('--shape', type_=list, default=[19, 19])
>>> test_show_gaussian_patches(shape=shape)
>>> pt.show_if_requested()
```

`vtool.test_show_gaussian_patches2(shape=(19, 19))`

CommandLine: `python -m vtool.patch -test-test_show_gaussian_patches2 -show python -m vtool.patch -test-test_show_gaussian_patches2 -show -shape=7,7 python -m vtool.patch -test-test_show_gaussian_patches2 -show -shape=19,19 python -m vtool.patch -test-test_show_gaussian_patches2 -show -shape=41,41 python -m vtool.patch -test-test_show_gaussian_patches2 -show -shape=41,7`

References

http://matplotlib.org/examples/mplot3d/surface3d_demo.html

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.patch import * # NOQA
>>> from mpl_toolkits.mplot3d import Axes3D # NOQA
>>> import wbia.plottool as pt
>>> shape = ut.get_argval('--shape', type_=list, default=[19, 19])
>>> test_show_gaussian_patches2(shape=shape)
>>> pt.show_if_requested()
```

`vtool.testdata_annot_metadata(rchip_fpath, cfgdict={})`

`vtool.testdata_binary_scores()`

`vtool.testdata_blend(scale=128)`

`vtool.testdata_dummy_matches()`

Returns `matches_testup`

Return type `tuple`

CommandLine: `python -m vtool.demodata -test-testdata_dummy_matches -show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.demodata import * # NOQA
>>> matches_testsetup = testdata_dummy_matches()
>>> (kpts1, kpts2, fm, fs, rchip1, rchip2) = matches_testsetup
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.show_chipmatch2(rchip1, rchip2, kpts1, kpts2, fm, fs)
>>> pt.set_figtitle('Dummy matches')
>>> pt.show_if_requested()
```

`vtool.testdata_dummy_sift (nPts=10, asint=True, rng=None)`

Makes a demodata sift descriptor that has the uint8 * 512 hack like hesaff returns

Parameters `nPts` (*int*) – (default = 10)

CommandLine: `python -m vtool.demodata --test-testdata_dummy_sift`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.demodata import * # NOQA
>>> import vtool as vt
>>> nPts = 10
>>> rng = np.random.RandomState(0)
>>> sift = testdata_dummy_sift(nPts, rng)
>>> assert vt.check_sift_validity(sift), 'bad SIFT properties'
>>> #assert np.allclose(((sift / 512) ** 2).sum(axis=1), 1, rtol=.01), 'bad SIFT_
↳property'
>>> #assert np.all(sift / 512 < .2), 'bad SIFT property'
```

`vtool.testdata_hist()`

`vtool.testdata_imglist()`

`vtool.testdata_matching_affine_inliers()`

`vtool.testdata_matching_affine_inliers_normalized()`

`vtool.testdata_nonmonotonic()`

`vtool.testdata_patch()`

`vtool.testdata_ratio_matches (fname1='easy1.png', fname2='easy2.png', **kwargs)`

Runs simple ratio-test matching between two images. Technically this is not demodata data.

Parameters

- `fname1` (*str*) –
- `fname2` (*str*) –

Returns `matches_testsetup`

Return type `tuple`

CommandLine: `python -m vtool.demodata --test-testdata_ratio_matches python -m vtool.demodata --test-testdata_ratio_matches --help python -m vtool.demodata --test-testdata_ratio_matches --show python -m vtool.demodata --test-testdata_ratio_matches --show --ratio_thresh=1.1 --rotation_invariance`

```
python -m vtool.demodata -test-testdata_ratio_matches -show -ratio_thresh=.625 -rotation_invariance
-fname1 easy1.png -fname2 easy3.png python -m vtool.demodata -test-testdata_ratio_matches -show
-ratio_thresh=.625 -no-rotation_invariance -fname1 easy1.png -fname2 easy3.png
```

Example

```
>>> # xdoctest: +REQUIRES(module:pyhesaff)
>>> from vtool.demodata import * # NOQA
>>> import vtool as vt
>>> fname1 = ut.get_argval('--fname1', type=str, default='easy1.png')
>>> fname2 = ut.get_argval('--fname2', type=str, default='easy2.png')
>>> default_dict = vt.get_extract_features_default_params()
>>> default_dict['ratio_thresh'] = .625
>>> kwargs = ut.parse_dict(default_dict)
>>> matches_testtuple = testdata_ratio_matches(fname1, fname2, **kwargs)
>>> (kpts1, kpts2, fm_RAT, fs_RAT, rchip1, rchip2) = matches_testtuple
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.show_chipmatch2(rchip1, rchip2, kpts1, kpts2, fm_RAT, fs_RAT, ori=True)
>>> num_matches = len(fm_RAT)
>>> score_sum = sum(fs_RAT)
>>> title = 'Simple matches using the Lowe\'s ratio test'
>>> title += '\n num_matches=%r, score_sum=%.2f' % (num_matches, score_sum)
>>> pt.set_figtitle(title)
>>> pt.show_if_requested()
```

```
vtool.testdata_score_normalier(tp_bumps=[(6.5, 256)], tn_bumps=[(3.5, 256)], tp_scale=1.0,
tn_scale=1.0, min_clip=None, **kwargs)
```

```
vtool.testdata_scores_labels()
```

```
vtool.testdata_sift2()
```

```
vtool.testshow_extramargin_info(gfpath, bbox_gs, theta, new_size, halfoffset_ms, mbbbox_gs,
margin_size)
```

```
vtool.to_undirected_edges(directed_edges, upper=False)
```

```
vtool.transform_around(M, x, y)
translates to origin, applies transform and then translates back
```

```
vtool.transform_kpts(kpts, M)
returns M.dot(kpts_mat) Currently, only works if M is affine.
```

Parameters

- **kpts** (*ndarray[float32_t, ndim=2]*) – keypoints
- **M** (*ndarray*) – affine transform matrix

Returns *ndarray*

Ignore:

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts()
>>> M = np.array([[10, 0, 0], [10, 10, 0], [0, 0, 1]], dtype=np.float64)
>>> kpts = transform_kpts(kpts, M)
```

(continues on next page)

(continued from previous page)

```
>>> # verify results
>>> result = ub.repr2(kpts, precision=3, with_dtype=True).replace('-0. ', ' 0. ')
↪ '
```

`vtool.transform_kpts_to_imgspace(kpts, bbox, bbox_theta, chipsz)`

Transforms keypoints so they are plotable in imagespace kpts - xyacdo keypoints bbox - chip bounding boxes in image space theta - chip rotations invC chipsz - chip extent (in keypoint / chip space)

`vtool.transform_kpts_xys(H, kpts)`

Parameters

- **kpts** (`ndarray[float32_t, ndim=2]`) – keypoints
- **H** (`ndarray[float64_t, ndim=2]`) – homography/perspective matrix

Returns `xy_t`

Return type `ndarray`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.keypoint import * # NOQA
>>> import vtool as vt
>>> kpts = vt.demodata.get_dummy_kpts()
>>> H = np.array([[ 3.,  3.,  5.],
...              [ 2.,  3.,  6.],
...              [ 1.,  1.,  2.]])
>>> xy_t = transform_kpts_xys(H, kpts)
>>> # verify results
```

`vtool.transform_points_with_homography(H, _xys)`

Parameters

- **H** (`ndarray[float64_t, ndim=2]`) – homography/perspective matrix
- **_xys** (`ndarray[ndim=2]`) – (2 x N) array

`vtool.translation_mat3x3(x, y, dtype=<class 'numpy.float64'>)`

`vtool.try_svd(M)`

CommandLine: `python -m vtool.spatial_verification try_svd`

Example

```
>>> # SLOW_DOCTEST
>>> # xdoctest: +SKIP
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.demodata as demodata
>>> rng = np.random.RandomState(42)
>>> num = 1000
>>> xy1_mn = rng.randn(2, num)
>>> xy2_mn = rng.randn(2, num)
>>> M = build_lstsqrs_Mx9(xy1_mn, xy2_mn)
```

(continues on next page)

(continued from previous page)

```
>>> print('M.shape = %r' % (M.shape,))
>>> USV = npl.svd(M, full_matrices=True, compute_uv=True)
>>> USV = try_svd(M)
```

Example

```
>>> # SLOW_DOCTEST
>>> # xdoctest: +SKIP
>>> from vtool.spatial_verification import * # NOQA
>>> import vtool.demodata as demodata
>>> num = np.ceil(np.sqrt(2000))
>>> kpts1, kpts2 = demodata.get_dummy_kpts_pair(wh_num=(num, num))
>>> xy1_mn = ktool.get_xys(kpts1).astype(np.float64)
>>> xy2_mn = ktool.get_xys(kpts2).astype(np.float64)
>>> M = build_lstsqrs_Mx9(xy1_mn, xy2_mn)
>>> print('M.shape = %r' % (M.shape,))
>>> USV = npl.svd(M, full_matrices=True, compute_uv=True)
>>> USV = try_svd(M)
```

`vtool.trytake(list_, index_list)`

`vtool.tune_flann(dpts, target_precision=0.9, build_weight=0.5, memory_weight=0.0, sample_fraction=0.01)`

References

http://www.cs.ubc.ca/research/flann/uploads/FLANN/flann_pami2014.pdf http://www.cs.ubc.ca/research/flann/uploads/FLANN/flann_manual-1.8.4.pdf http://docs.opencv.org/trunk/modules/flann/doc/flann_fast_approximate_nearest_neighbor_search.html

Math: cost of an algorithm is:

LaTeX:

$$\text{cost} = \text{frac} \{ \text{search} + \text{build_weight} * \text{build} \} \{ \text{minoverparams}(\text{search} + \text{build_weight} * \text{build}) \} + \text{memory_weight} * \text{memory}$$

Parameters

- **dpts** (*ndarray*) –
- **target_precision** (*float*) – number between 0 and 1 representing desired accuracy. Higher values are more accurate.
- **build_weight** (*float*) – importance weight given to minimizing build time relative to search time. This number can range from 0 to infinity. typically because building is a more complex computation you want to keep the number relatively low, (less than 1) otherwise you'll end up getting a linear search (no build time).
- **memory_weight** (*float*) – Importance of memory relative to total speed. A value less than 1 gives more importance to the time spent and a value greater than 1 gives more importance to the memory usage.
- **sample_fraction** (*float*) – number between 0 and 1 representing the fraction of the input data to use in the optimization. A higher number uses more data.

Returns `tuned_params`

Return type dict

CommandLine: python -m vtool.nearest_neighbors --test-tune_flann

`vtool.tune_flann2(data)`

`vtool.understanding_pseudomax_props(mode=2)`

Function showing some properties of distances between normalized pseudomax vectors

CommandLine: python -m vtool.distance --test-understanding_pseudomax_props

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.distance import * # NOQA
>>> for mode in [0, 1, 2, 3]:
...     print('+---')
...     print('mode = %r' % (mode,))
...     result = understanding_pseudomax_props(mode)
...     print('L___')
>>> print(result)
```

`vtool.uniform_sample_hypersphere(num, ndim=2, only_quadrent_1=False)`

Not quite done yet

References

https://en.wikipedia.org/wiki/Regular_polytope

https://en.wikipedia.org/wiki/Platonic_solid#Higher_dimensions

<https://en.wikipedia.org/wiki/Cross-polytope>

Parameters

- **num** –
- **ndim** (*int*) – (default = 2)

CommandLine: python -m vtool.clustering2 --test-uniform_sampe_hypersphere

Ignore: #pip install polytope sudo pip install cvxopt --no-deps

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.clustering2 import * # NOQA
>>> num = 100
>>> ndim = 3
>>> pts = uniform_sampe_hypersphere(num, ndim)
>>> print(pts)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> if ndim == 2:
>>>     pt.plot(pts.T[0], pts.T[1], 'gx')
>>> elif ndim == 3:
>>>     #pt.plot_surface3d(pts.T[0], pts.T[1], pts.T[2])
>>>     from mpl_toolkits.mplot3d import Axes3D # NOQA
```

(continues on next page)

(continued from previous page)

```

>>> fig = plt.figure(1, doclf=True, docla=True)
>>> ax = fig.add_subplot(111, projection='3d')
>>> ax.scatter(pts.T[0], pts.T[1], pts.T[2], s=20, marker='o', alpha=1)
>>> ax.autoscale(enable=False)
>>> ax.set_aspect('equal')
>>> df2.dark_background(ax)
>>> plt.dark_background()
>>> plt.show_if_requested()

```

`vtool.union_extents` (*extents*)

`vtool.unique_row_indexes` (*arr*)

np.unique on rows

Parameters *arr* (*ndarray*) – 2d array

Returns *unique_rowx*

Return type *ndarray*

References

<http://stackoverflow.com/questions/16970982/find-unique-rows-in-numpy-array>

CommandLine: `python -m vtool.numpy_utils --test-unique_row_indexes`

Example

```

>>> # DISABLE_DOCTEST
>>> from vtool.numpy_utils import * # NOQA
>>> import ubelt as ub
>>> arr = np.array([[0, 0], [0, 1], [1, 0], [1, 1], [0, 0], [.534, .432], [.534, .
↪432], [1, 0], [0, 1]])
>>> unique_rowx = unique_row_indexes(arr)
>>> result = ('unique_rowx = %s' % (ub.repr2(unique_rowx),))
>>> print(result)
unique_rowx = np.array([0, 1, 2, 3, 5], dtype=np.int64)

```

Ignore: `%timeit unique_row_indexes(arr)` `%timeit compute_unique_data_ids(arr)` `%timeit compute_unique_integer_data_ids(arr)`

`vtool.unique_rows` (*arr*, *directed=True*)

Order or columns does not matter if *directed* = False

`vtool.unnormalize_transform` (*M_prime*, *T1*, *T2*)

`vtool.unstructure_rows` (**structured_arrs*)

SeeAlso: `structure_rows`

`vtool.unsupervised_multicut_labeling` (*cost_matrix*, *thresh=0*)

Notes

requires CPLEX

CommandLine: python -m vtool.clustering2 unsupervised_multicut_labeling -show

Ignore:

```
>>> # synthetic data
>>> import vtool as vt
>>> size = 100
>>> thresh = 50
>>> np.random.randint(0, 1)
>>> np.zeros((size, size))
>>> #np.random.rand(size, size)
>>> size = 45
>>> #size = 10
>>> size = 5
>>> aids = np.arange(size)
>>> rng = np.random.RandomState(443284320)
>>> encounter_lbls = rng.randint(0, size, size)
>>> separation = 5.0
>>> separation = 1.10
>>> grid1 = np.tile(encounter_lbls, (size, 1))
>>> is_match = grid1.T == grid1
>>> good_pos = np.where(is_match)
>>> bad_pos = np.where(~is_match)
>>> cost_matrix_ = np.zeros((size, size))
>>> cost_matrix_[good_pos] = rng.randn(len(good_pos[0])) + separation
>>> cost_matrix_[bad_pos] = rng.randn(len(bad_pos[0])) - separation
>>> false_val = min(cost_matrix_.min(), np.min(rng.randn(1000) -
↳separation))
>>> true_val = max(cost_matrix_.max(), np.max(rng.randn(500) +
↳separation))
>>> cost_matrix_[np.diag_indices_from(cost_matrix_)] = true_val
>>> #cost_matrix_[np.diag_indices_from(cost_matrix_)] = np.inf
>>> cost_matrix = (cost_matrix_ - false_val) / (true_val - false_val)
>>> cost_matrix = 2 * (cost_matrix - .5)
>>> thresh = 0
>>> labels = vt.unsupervised_multicut_labeling(cost_matrix, thresh)
>>> diff = ut.find_group_differences(
>>>     list(ut.group_items(aids, encounter_lbls).values()),
>>>     list(ut.group_items(aids, labels).values()))
>>> print('diff = %r' % (diff,))
```

```
#gm, = ut.exec_func_src(vt.unsupervised_multicut_labeling, #key_list=['gm'], sentinel='inf =
opengm') #parameter = opengm.InfParam() #%timeit opengm.inference.Multicut(gm, parame-
ter=parameter).infer()
```

Example

```
>>> # SCRIPT
>>> from vtool.clustering2 import * # NOQA
>>> import networkx as nx
>>> import wbia.plottool as pt
>>> rng = np.random.RandomState(443284320)
>>> pt.ensureqt()
>>> #
>>> def make_test_costmatrix(name_labels, view_labels, separation=2):
>>>     is_same = name_labels == name_labels[:, None]
>>>     is_comp = np.abs(view_labels - view_labels[:, None]) <= 1
```

(continues on next page)

(continued from previous page)

```

>>> good_pos = np.where(is_same)
>>> bad_pos = np.where(~is_same)
>>> cost_matrix_ = np.zeros((len(name_labels), len(name_labels)))
>>> cost_matrix_[good_pos] = rng.randn(len(good_pos[0])) + separation
>>> cost_matrix_[bad_pos] = rng.randn(len(bad_pos[0])) - separation
>>> cost_matrix_ = (cost_matrix_.T + cost_matrix_) / 2
>>> false_val = min(cost_matrix_.min(), np.min(rng.randn(1000) - separation))
>>> true_val = max(cost_matrix_.max(), np.max(rng.randn(500) + separation))
>>> cost_matrix_[np.diag_indices_from(cost_matrix_)] = true_val
>>> cost_matrix = (cost_matrix_ - false_val) / (true_val - false_val)
>>> cost_matrix = 2 * (cost_matrix - .5)
>>> cost_matrix[np.where(~is_comp)] = 0
>>> return cost_matrix
>>> #
>>> view_labels = np.array([0, 0, 2, 2, 1, 0, 0, 0])
>>> name_labels = np.array([0, 0, 0, 0, 0, 1, 1, 1])
>>> #cost_matrix = make_test_costmatrix(name_labels, view_labels, 2)
>>> cost_matrix = make_test_costmatrix(name_labels, view_labels, .9)
>>> #
>>> def multicut_value(cost_matrix, name_labels):
>>>     grid1 = np.tile(name_labels, (len(name_labels), 1))
>>>     isdiff = grid1.T != grid1
>>>     cut_value = cost_matrix[isdiff].sum()
>>>     return cut_value
>>> #
>>> aids = np.arange(len(name_labels))
>>> #
>>> graph = ut.nx_from_matrix(cost_matrix)
>>> weights = nx.get_edge_attributes(graph, 'weight')
>>> #
>>> floatfmt1 = ut.partial(ub.map_vals, lambda x: 'w=%.2f' % x)
>>> floatfmt2 = ut.partial(ub.map_vals, lambda x: 'l=%.2f' % x)
>>> #
>>> lens = ub.map_vals(lambda x: (1 - ((x + 1) / 2)) / 2, weights)
>>> labels = floatfmt1(weights)
>>> #labels = floatfmt2(lens)
>>> nx.set_edge_attributes(graph, name='label', values=labels)
>>> #nx.set_edge_attributes(graph, name='len', values=lens)
>>> nx.set_node_attributes(graph, name='shape', values='ellipse')
>>> encounter_lbls_str = [str(x) for x in name_labels]
>>> node_name_lbls = dict(zip(aids, encounter_lbls_str))
>>> import vtool as vt
>>> #
>>> mcut_labels = vt.unsupervised_multicut_labeling(cost_matrix, thresh=vt.eps)
>>> diff = ut.find_group_differences(
>>>     list(ut.group_items(aids, name_labels).values()),
>>>     list(ut.group_items(aids, mcut_labels).values()))
>>> print('diff = %r' % (diff,))
>>> #
>>> nx.set_node_attributes(graph, name='label', values=node_name_lbls)
>>> node_mcut_lbls = dict(zip(aids, mcut_labels))
>>> nx.set_node_attributes(graph, name='mcut_label', values=node_mcut_lbls)
>>> #
>>> print('mc_val(name) ' + str(multicut_value(cost_matrix, name_labels)))
>>> print('mc_val(mcut) ' + str(multicut_value(cost_matrix, mcut_labels)))
>>> #
>>> ut.color_nodes(graph, 'mcut_label')

```

(continues on next page)

(continued from previous page)

```

>>> #
>>> # remove noncomparable edges
>>> is_comp = np.abs(view_labels - view_labels[:, None]) <= 1
>>> #
>>> noncomp_edges = list(zip(*np.where(~is_comp)))
>>> graph.remove_edges_from(noncomp_edges)
>>> #
>>> layoutkw = {
>>>     'sep' : 5,
>>>     'prog': 'neato',
>>>     'overlap': 'false',
>>>     'splines': 'spline',
>>> }
>>> pt.show_nx(graph, layoutkw=layoutkw)
>>> ut.show_if_requested()

```

`vtool.verts_from_bbox(bbox, close=False)`

Parameters

- **bbox** (*tuple*) – bounding box in the format (x, y, w, h)
- **close** (*bool*) – (default = False)

Returns `verts`

Return type `list`

CommandLine: `python -m vtool.geometry --test-verts_from_bbox`

Example

```

>>> # ENABLE_DOCTEST
>>> from vtool.geometry import * # NOQA
>>> bbox = (10, 10, 50, 50)
>>> close = False
>>> verts = verts_from_bbox(bbox, close)
>>> result = ('verts = %s' % (str(verts),))
>>> print(result)
verts = ((10, 10), (60, 10), (60, 60), (10, 60))

```

`vtool.verts_list_from_bboxes_list(bboxes_list)`

Create a four-vertex polygon from the bounding rectangle

`vtool.warpAffine(img, Aff, dsize, assume_float01=True)`

`dsize` = (width, height) of return image

Parameters

- **img** (`ndarray[uint8_t, ndim=2]`) – image data
- **Aff** (`ndarray`) – affine matrix
- **dsize** (*tuple*) – width, height

Returns `warped_img`

Return type `ndarray`

CommandLine: `python -m vtool.image --test-warpAffine --show`

Example

```
>>> # DISABLE_DOCTEST
>>> from vtool.image import * # NOQA
>>> import vtool as vt
>>> img_fpath = ut.grab_test_imgpath('carl.jpg')
>>> img = vt.imread(img_fpath)
>>> Aff = vt.rotation_mat3x3(TAU / 8)
>>> dsize = vt.get_size(img)
>>> warped_img = warpAffine(img, Aff, dsize)
>>> # xdoctest: +REQUIRES(--show)
>>> import wbia.plottool as pt
>>> pt.imshow(warped_img)
>>> ut.show_if_requested()
```

Ignore:

```
>>> import skimage.transform
>>> %timeit cv2.warpAffine(img, Aff[0:2], tuple(dsize), **CV2_WARP_KWARGS)
>>> 100 loops, best of 3: 7.95 ms per loop
>>> skimage.transform.AffineTransform
>>> tf = skimage.transform.AffineTransform(rotation=TAU / 8)
>>> Aff_ = tf.params
>>> out = skimage.transform._warps_cy._warp_fast(img[:, :, 0], Aff_, output_
↳ shape=dsize, mode='constant', order=1)
>>> %timeit skimage.transform._warps_cy._warp_fast(img[:, :, 0], Aff_, output_
↳ shape=dsize, mode='constant', order=1)
>>> 100 loops, best of 3: 5.74 ms per loop
>>> %timeit cv2.warpAffine(img[:, :, 0], Aff[0:2], tuple(dsize), **CV2_WARP_
↳ KWARGS)
>>> 100 loops, best of 3: 5.13 ms per loop
>>> CONCLUSION, cv2 transforms are better
```

`vtool.warpHomog` (*img*, *Homog*, *dsize*, *assume_float01=True*)
dsize = (width, height) of return image

Example

```
>>> img = np.random.rand(224, 224)
>>> Homog = np.random.rand(3, 3)
>>> dsize = (128, 128)
>>> warped_img = warpHomog(img, Homog, dsize)
```

`vtool.weighted_average_scoring` (*fsv*, *weight_filtxs*, *nonweight_filtxs*)
 does $\frac{\sum_i w^f_i * w^d_i * r_i}{\sum_i w^f_i, w^d_i}$ to get a weighed average of ratio scores

If we normalize the weight part to add to 1 then we can get per-feature scores.

References

http://en.wikipedia.org/wiki/Weighted_arithmetic_mean

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> fsv = np.array([
...     [ 0.82992172,  1.56136119,  0.66465378],
...     [ 0.8000412 ,  2.14719748,  1.          ],
...     [ 0.80848503,  2.6816361 ,  1.          ],
...     [ 0.86761665,  2.70189977,  1.          ],
...     [ 0.8004055 ,  1.58753884,  0.92178345],])
>>> weight_filtxs = np.array([1, 2], dtype=np.int32)
>>> nonweight_filtxs = np.array([0], dtype=np.int32)
>>> new_fs = weighted_average_scoring(fsv, weight_filtxs, nonweight_filtxs)
>>> result = new_fs
>>> print(result)
```

`vtool.weighted_geometric_mean(data, weights)`

Parameters

- **data** (list of ndarrays) –
- **weights** (ndarray) –

Returns ndarray

CommandLine: `python -m vtool.other --test-weighted_geometric_mean`

References

https://en.wikipedia.org/wiki/Weighted_geometric_mean

SeeAlso: `scipy.stats.mstats.gmean`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> data = [.9, .5]
>>> weights = np.array([1.0, .5])
>>> gmean_ = weighted_geometric_mean(data, weights)
>>> result = ('gmean_ = %.3f' % (gmean_,))
>>> print(result)
gmean_ = 0.740
```

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> rng = np.random.RandomState(0)
>>> img1 = rng.rand(4, 4)
>>> img2 = rng.rand(4, 4)
>>> data = [img1, img2]
>>> weights = np.array([.5, .5])
```

(continues on next page)

(continued from previous page)

```
>>> gmean_ = weighted_geometric_mean(data, weights)
>>> result = ub.hzcat(['gmean_ = %s' % (ub.repr2(gmean_, precision=2, with_
↳dtype=True), )])
>>> print(result)
```

Ignore: `res1 = ((img1 ** .5 * img2 ** .5)) ** 1` `res2 = np.sqrt(img1 * img2)`

`vtool.weighted_geometric_mean_unnormalized(data, weights)`

`vtool.whiten_xy_points(xy_m)`

whitens points to mean=0, stddev=1 and returns transformation

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.linalg import * # NOQA
>>> from vtool import demodata
>>> xy_m = demodata.get_dummy_xy()
>>> tup = whiten_xy_points(xy_m)
>>> xy_norm, T = tup
>>> result = (ub.hash_data(tup))
>>> print(result)
```

`vtool.wrap_histogram(hist_, edges_, _debug=False)`

Simulates the first and last histogram bin being adjacent to one another by replicating those bins at the last and first positions respectively.

Parameters

- **hist** (*ndarray*) –
- **edges** (*ndarray*) –

Returns (hist_wrap, edge_wrap)

Return type tuple

CommandLine: `python -m vtool.histogram --test-wrap_histogram`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.histogram import * # NOQA
>>> import ubelt as ub
>>> hist_ = np.array([8., 0., 0., 34.32, 29.45, 0., 0., 6.73])
>>> edges_ = np.array([ 0.,          ,  0.78539816,  1.57079633,
...                    2.35619449,  3.14159265,  3.92699081,
...                    4.71238898,  5.49778714,  6.2831853 ])
>>> (hist_wrap, edge_wrap) = wrap_histogram(hist_, edges_)
>>> tup = (hist_wrap.tolist(), edge_wrap.tolist())
>>> result = ub.repr2(tup, nl=1, nobr=True, precision=2)
>>> print(result)
6.73, 8.00, 0.00, 0.00, 34.32, 29.45, 0.00, 0.00, 6.73, 8.00,
-0.79, 0.00, 0.79, 1.57, 2.36, 3.14, 3.93, 4.71, 5.50, 6.28, 7.07,
```

`vtool.wrapped_distance (arr1, arr2, base, out=None)`
base = TAU corresponds to ori diff

`vtool.zipcat (arr1_list, arr2_list, axis=None)`

Parameters

- **arr1_list** (*list*) –
- **arr2_list** (*list*) –
- **axis** (*None*) – (default = None)

Returns

Return type `list`

CommandLine: `python -m vtool.other --exec-zipcat --show`

Example

```
>>> # ENABLE_DOCTEST
>>> from vtool.other import * # NOQA
>>> arr1_list = [np.array([0, 0, 0]), np.array([0, 0, 0, 0])]
>>> arr2_list = [np.array([1, 1, 1]), np.array([1, 1, 1, 1])]
>>> axis = None
>>> arr3_list = zipcat(arr1_list, arr2_list, axis)
>>> arr3_list0 = zipcat(arr1_list, arr2_list, axis=0)
>>> arr3_list1 = zipcat(arr1_list, arr2_list, axis=1)
>>> arr3_list2 = zipcat(arr1_list, arr2_list, axis=2)
>>> print('arr3_list = %s' % (ut.repr3(arr3_list),))
>>> print('arr3_list0 = %s' % (ut.repr3(arr3_list0),))
>>> print('arr3_list2 = %s' % (ut.repr3(arr3_list2),))
```

`vtool.zipcompress (arr_list, flags_list, axis=None)`

`vtool.zipcompress_safe (arr_list, flags_list, axis=None)`

`vtool.ziptake (arr_list, indices_list, axis=None)`

`vtool.zstar_value (conf_level=0.95)`

References

<http://stackoverflow.com/questions/28242593/correct-way-to-obtain-confidence-interval-with-sciPy>

CHAPTER 2

Indices and tables

- `genindex`
- `modindex`
- `search`

V

- `vtool`, 201
- `vtool.__main__`, 1
- `vtool._grave`, 1
- `vtool._old_matching`, 2
- `vtool._pyflann_backend`, 2
- `vtool._rhomb_dist`, 2
- `vtool.blend`, 2
- `vtool.chip`, 6
- `vtool.clustering2`, 10
- `vtool.confusion`, 19
- `vtool.coverage_grid`, 27
- `vtool.coverage_kpts`, 29
- `vtool.demodata`, 33
- `vtool.deprecated`, 35
- `vtool.distance`, 36
- `vtool.ellipse`, 42
- `vtool.exif`, 43
- `vtool.features`, 46
- `vtool.fontdemo`, 47
- `vtool.geometry`, 48
- `vtool.histogram`, 55
- `vtool.image`, 64
- `vtool.image_filters`, 88
- `vtool.image_shared`, 89
- `vtool.inspect_matches`, 89
- `vtool.keypoint`, 91
- `vtool.linalg`, 109
- `vtool.matching`, 116
- `vtool.nearest_neighbors`, 125
- `vtool.numpy_utils`, 130
- `vtool.other`, 135
- `vtool.patch`, 160
- `vtool.quality_classifier`, 172
- `vtool.score_normalization`, 173
- `vtool.segmentation`, 182
- `vtool.spatial_verification`, 183
- `vtool.symbolic`, 192
- `vtool.trig`, 193

- `vtool.util_math`, 194

A

- `acc` (*vtool.confusion.ConfusionMetrics* attribute), 20
- `acc` (*vtool.ConfusionMetrics* attribute), 205
- `adapteq()` (*vtool.image_filters.IntensityPreproc* method), 88
- `adapteq_fn()` (in module *vtool.image_filters*), 88
- `adaptive_scale()` (in module *vtool*), 220
- `adaptive_scale()` (in module *vtool.ellipse*), 42
- `add_global_measures()` (*vtool.matching.PairwiseMatch* method), 119
- `add_global_measures()` (*vtool.PairwiseMatch* method), 213
- `add_homogenous_coordinate()` (in module *vtool*), 220
- `add_homogenous_coordinate()` (in module *vtool.linalg*), 110
- `add_local_measures()` (*vtool.matching.PairwiseMatch* method), 119
- `add_local_measures()` (*vtool.PairwiseMatch* method), 213
- `affine_around_mat3x3()` (in module *vtool*), 220
- `affine_around_mat3x3()` (in module *vtool.linalg*), 110
- `affine_mat3x3()` (in module *vtool*), 222
- `affine_mat3x3()` (in module *vtool.linalg*), 111
- `affine_warp_around_center()` (in module *vtool*), 222
- `affine_warp_around_center()` (in module *vtool.image*), 64
- `aliases` (*vtool.confusion.ConfusionMetrics* attribute), 20
- `aliases` (*vtool.ConfusionMetrics* attribute), 205
- `and_lists()` (in module *vtool*), 223
- `and_lists()` (in module *vtool.other*), 135
- `ann_flann_once()` (in module *vtool*), 224
- `ann_flann_once()` (in module *vtool.nearest_neighbors*), 125
- `AnnotPairFeatInfo` (class in *vtool*), 201
- `AnnotPairFeatInfo` (class in *vtool.matching*), 116
- `AnnoyWrapper` (class in *vtool*), 204
- `AnnoyWrapper` (class in *vtool.clustering2*), 10
- `AnnoyWrapper` (class in *vtool*), 204
- `AnnoyWrapper` (class in *vtool.nearest_neighbors*), 125
- `apply_all()` (*vtool.matching.PairwiseMatch* method), 119
- `apply_all()` (*vtool.PairwiseMatch* method), 213
- `apply_filter_funcs()` (in module *vtool*), 225
- `apply_filter_funcs()` (in module *vtool.chip*), 7
- `apply_grouping()` (in module *vtool*), 225
- `apply_grouping()` (in module *vtool.clustering2*), 10
- `apply_grouping_()` (in module *vtool*), 226
- `apply_grouping_()` (in module *vtool.clustering2*), 11
- `apply_grouping_iter()` (in module *vtool*), 226
- `apply_grouping_iter()` (in module *vtool.clustering2*), 11
- `apply_grouping_iter2()` (in module *vtool*), 226
- `apply_grouping_iter2()` (in module *vtool.clustering2*), 11
- `apply_jagged_grouping()` (in module *vtool*), 226
- `apply_jagged_grouping()` (in module *vtool.clustering2*), 11
- `apply_ratio_test()` (*vtool.matching.PairwiseMatch* method), 119
- `apply_ratio_test()` (*vtool.PairwiseMatch* method), 213
- `apply_sver()` (*vtool.matching.PairwiseMatch* method), 119
- `apply_sver()` (*vtool.PairwiseMatch* method), 213
- `area()` (*vtool.chip.ScaleStrat* static method), 6
- `area()` (*vtool.ScaleStrat* static method), 215
- `argsort_groups()` (in module *vtool*), 226
- `argsort_groups()` (in module *vtool.other*), 136
- `argsort_records()` (in module *vtool*), 227
- `argsort_records()` (in module *vtool.other*), 137
- `argsubextrema2()` (in module *vtool*), 227

[argsubextrema2\(\)](#) (in module *vtool.histogram*), 55
[argsubmax\(\)](#) (in module *vtool*), 228
[argsubmax\(\)](#) (in module *vtool.histogram*), 56
[argsubmax2\(\)](#) (in module *vtool*), 229
[argsubmax2\(\)](#) (in module *vtool.histogram*), 57
[argsubmaxima\(\)](#) (in module *vtool*), 229
[argsubmaxima\(\)](#) (in module *vtool.histogram*), 57
[argsubmaxima2\(\)](#) (in module *vtool*), 230
[argsubmaxima2\(\)](#) (in module *vtool.histogram*), 58
[argsubmin2\(\)](#) (in module *vtool*), 230
[argsubmin2\(\)](#) (in module *vtool.histogram*), 58
[argsubminima2\(\)](#) (in module *vtool*), 230
[argsubminima2\(\)](#) (in module *vtool.histogram*), 58
[assert_zipcompress\(\)](#) (in module *vtool*), 230
[assert_zipcompress\(\)](#) (in module *vtool.other*), 137
[asserteq\(\)](#) (in module *vtool*), 230
[asserteq\(\)](#) (in module *vtool.other*), 137
[assign\(\)](#) (*vtool.matching.PairwiseMatch* method), 120
[assign\(\)](#) (*vtool.PairwiseMatch* method), 213
[assign_symmetric_matches\(\)](#) (in module *vtool*), 230
[assign_symmetric_matches\(\)](#) (in module *vtool.matching*), 121
[assign_to_centroids\(\)](#) (in module *vtool*), 231
[assign_to_centroids\(\)](#) (in module *vtool.nearest_neighbors*), 127
[assign_unconstrained_matches\(\)](#) (in module *vtool*), 231
[assign_unconstrained_matches\(\)](#) (in module *vtool.matching*), 122
[AssignTup](#) (class in *vtool*), 204
[AssignTup](#) (class in *vtool.matching*), 118
[asymmetric_correspondence\(\)](#) (in module *vtool*), 232
[asymmetric_correspondence\(\)](#) (in module *vtool.matching*), 123
[atan2\(\)](#) (in module *vtool*), 232
[atan2\(\)](#) (in module *vtool.trig*), 193
[atleast_3channels\(\)](#) (in module *vtool*), 233
[atleast_3channels\(\)](#) (in module *vtool.other*), 137
[atleast_nd\(\)](#) (in module *vtool*), 233, 234
[atleast_nd\(\)](#) (in module *vtool.numpy_utils*), 130
[atleast_nd\(\)](#) (in module *vtool.other*), 138
[atleast_shape\(\)](#) (in module *vtool*), 235
[atleast_shape\(\)](#) (in module *vtool.other*), 139
[auc](#) (*vtool.confusion.ConfusionMetrics* attribute), 20
[auc](#) (*vtool.ConfusionMetrics* attribute), 205
[auc_trap](#) (*vtool.confusion.ConfusionMetrics* attribute), 20
[auc_trap](#) (*vtool.ConfusionMetrics* attribute), 205
[augment_2x2_with_translation\(\)](#) (in module *vtool*), 235

[augment_2x2_with_translation\(\)](#) (in module *vtool.keypoint*), 93

B

[bar_cos_sift\(\)](#) (in module *vtool*), 236
[bar_cos_sift\(\)](#) (in module *vtool.distance*), 37
[bar_L2_sift\(\)](#) (in module *vtool*), 235
[bar_L2_sift\(\)](#) (in module *vtool.distance*), 37
[bbox_center\(\)](#) (in module *vtool*), 236
[bbox_center\(\)](#) (in module *vtool.geometry*), 48
[bbox_from_center_wh\(\)](#) (in module *vtool*), 236
[bbox_from_center_wh\(\)](#) (in module *vtool.geometry*), 48
[bbox_from_extent\(\)](#) (in module *vtool*), 236
[bbox_from_extent\(\)](#) (in module *vtool.geometry*), 48
[bbox_from_verts\(\)](#) (in module *vtool*), 236
[bbox_from_verts\(\)](#) (in module *vtool.geometry*), 49
[bbox_from_xywh\(\)](#) (in module *vtool*), 236
[bbox_from_xywh\(\)](#) (in module *vtool.geometry*), 49
[bboxes_from_vert_list\(\)](#) (in module *vtool*), 236
[bboxes_from_vert_list\(\)](#) (in module *vtool.geometry*), 49
[beaton_tukey_loss\(\)](#) (in module *vtool*), 236
[beaton_tukey_loss\(\)](#) (in module *vtool.util_math*), 194
[beaton_tukey_weight\(\)](#) (in module *vtool*), 237
[beaton_tukey_weight\(\)](#) (in module *vtool.util_math*), 194
[binsum_fmt](#) (*vtool.AnnotPairFeatInfo* attribute), 202
[binsum_fmt](#) (*vtool.matching.AnnotPairFeatInfo* attribute), 117
[bitblt\(\)](#) (*vtool.fontdemo.Bitmap* method), 47
[Bitmap](#) (class in *vtool.fontdemo*), 47
[blend_images\(\)](#) (in module *vtool*), 237
[blend_images\(\)](#) (in module *vtool.blend*), 2
[blend_images_average\(\)](#) (in module *vtool*), 237
[blend_images_average\(\)](#) (in module *vtool.blend*), 2
[blend_images_average_stack\(\)](#) (in module *vtool*), 238
[blend_images_average_stack\(\)](#) (in module *vtool.blend*), 3
[blend_images_mult_average\(\)](#) (in module *vtool*), 238
[blend_images_mult_average\(\)](#) (in module *vtool.blend*), 4
[blend_images_multiply\(\)](#) (in module *vtool*), 239
[blend_images_multiply\(\)](#) (in module *vtool.blend*), 5
[bm](#) (*vtool.confusion.ConfusionMetrics* attribute), 20
[bm](#) (*vtool.ConfusionMetrics* attribute), 205
[bow_test\(\)](#) (in module *vtool.grave*), 1
[breakup_equal_streak\(\)](#) (in module *vtool*), 240

`breakup_equal_streak()` (in module `vtool.util_math`), 194
`build_affine_lstsqrs_Mx6()` (in module `vtool`), 241
`build_affine_lstsqrs_Mx6()` (in module `vtool.spatial_verification`), 183
`build_annoy()` (`vtool.AnnoyWrapper` method), 204
`build_annoy()` (`vtool.clustering2.AnnoyWrapper` method), 10
`build_index()` (`vtool.AnnoyWrapper` method), 204
`build_index()` (`vtool.nearest_neighbors.AnnoyWrapper` method), 125
`build_lstsqrs_Mx9()` (in module `vtool`), 241
`build_lstsqrs_Mx9()` (in module `vtool.spatial_verification`), 184

C

`c` (`vtool.confusion.ConfusionMetrics` attribute), 20
`c` (`vtool.ConfusionMetrics` attribute), 205
`calc_errorBars_from_sample()` (in module `vtool`), 242
`calc_errorBars_from_sample()` (in module `vtool.other`), 139
`calc_sample_from_errorBars()` (in module `vtool`), 242
`calc_sample_from_errorBars()` (in module `vtool.other`), 139
`cast_split()` (in module `vtool`), 242
`cast_split()` (in module `vtool.keypoint`), 93
`check_exif_keys()` (in module `vtool`), 242
`check_exif_keys()` (in module `vtool.exif`), 43
`check_expr_eq()` (in module `vtool`), 242
`check_expr_eq()` (in module `vtool.symbolic`), 192
`check_kpts_in_bounds()` (in module `vtool`), 243
`check_kpts_in_bounds()` (in module `vtool.ellipse`), 42
`check_sift_validity()` (in module `vtool`), 243
`check_sift_validity()` (in module `vtool.other`), 139
`check_unused_kwargs()` (in module `vtool`), 243
`check_unused_kwargs()` (in module `vtool.score_normalization`), 177
`circular_distance()` (in module `vtool`), 243
`circular_distance()` (in module `vtool.ellipse`), 42
`clean_mask()` (in module `vtool.image_filters`), 88
`clean_mask()` (in module `vtool.segmentation`), 182
`clipnorm()` (in module `vtool`), 243
`clipnorm()` (in module `vtool.other`), 139
`clipwhite()` (in module `vtool`), 243
`clipwhite()` (in module `vtool.image`), 64
`clipwhite_ondisk()` (in module `vtool`), 243
`clipwhite_ondisk()` (in module `vtool.image`), 64
`closeEvent()` (`vtool.inspect_matches.MatchInspector` method), 90
`closest_point()` (in module `vtool`), 243
`closest_point()` (in module `vtool.distance`), 37
`closest_point_on_bbox()` (in module `vtool`), 243
`closest_point_on_bbox()` (in module `vtool.geometry`), 49
`closest_point_on_line()` (in module `vtool`), 243
`closest_point_on_line()` (in module `vtool.geometry`), 49
`closest_point_on_line_segment()` (in module `vtool`), 244
`closest_point_on_line_segment()` (in module `vtool.geometry`), 50
`closest_point_on_vert_segments()` (in module `vtool`), 245
`closest_point_on_vert_segments()` (in module `vtool.geometry`), 51
`colwise_operation()` (in module `vtool`), 245
`colwise_operation()` (in module `vtool.other`), 140
`combine_offset_lists()` (in module `vtool`), 245
`combine_offset_lists()` (in module `vtool.image`), 64
`compare_implementations()` (in module `vtool`), 245
`compare_implementations()` (in module `vtool.other`), 140
`compare_matrix_columns()` (in module `vtool`), 245
`compare_matrix_columns()` (in module `vtool.other`), 140
`compare_matrix_to_rows()` (in module `vtool`), 245
`compare_matrix_to_rows()` (in module `vtool.other`), 140
`componentwise_dot()` (in module `vtool`), 245
`componentwise_dot()` (in module `vtool.other`), 140
`compress()` (`vtool.matching.PairwiseMatch` method), 120
`compress()` (`vtool.PairwiseMatch` method), 214
`compress2()` (in module `vtool`), 246
`compress2()` (in module `vtool.other`), 140
`compute_affine()` (in module `vtool`), 246
`compute_affine()` (in module `vtool.spatial_verification`), 184
`compute_average_contrast()` (in module `vtool.quality_classifier`), 172
`compute_chip()` (in module `vtool`), 247
`compute_chip()` (in module `vtool.chip`), 7
`compute_distances()` (in module `vtool`), 248
`compute_distances()` (in module `vtool.distance`), 37
`compute_homog()` (in module `vtool`), 248
`compute_homog()` (in module `vtool.spatial_verification`), 185
`compute_ndarray_unique_rowids_unsafe()`

(in module *vtool*), 249
 compute_ndarray_unique_rowids_unsafe() (in module *vtool.other*), 140
 compute_subbin_to_bins_dist() (in module *vtool.coverage_grid*), 27
 compute_unique_arr_dataids() (in module *vtool*), 249
 compute_unique_arr_dataids() (in module *vtool.other*), 140
 compute_unique_data_ids() (in module *vtool*), 249
 compute_unique_data_ids() (in module *vtool.other*), 141
 compute_unique_data_ids_() (in module *vtool*), 250
 compute_unique_data_ids_() (in module *vtool.other*), 141
 compute_unique_integer_data_ids() (in module *vtool*), 250
 compute_unique_integer_data_ids() (in module *vtool.other*), 141
 ConfusionMetrics (class in *vtool*), 204
 ConfusionMetrics (class in *vtool.confusion*), 19
 contrast_measures() (in module *vtool.quality_classifier*), 172
 convert_colorspace() (in module *vtool*), 250
 convert_colorspace() (in module *vtool.image*), 64
 convert_degrees() (in module *vtool*), 251
 convert_degrees() (in module *vtool.exif*), 43
 convert_image_list_colorspace() (in module *vtool*), 251
 convert_image_list_colorspace() (in module *vtool.image*), 65
 convert_kptsZ_to_kpts() (in module *vtool*), 251
 convert_kptsZ_to_kpts() (in module *vtool.keypoint*), 93
 copy() (*vtool.matching.PairwiseMatch* method), 120
 copy() (*vtool.PairwiseMatch* method), 214
 cos_sift() (in module *vtool*), 251
 cos_sift() (in module *vtool.distance*), 38
 cosine_dist() (in module *vtool*), 252
 cosine_dist() (in module *vtool.distance*), 38
 crop_out_imgfill() (in module *vtool*), 252
 crop_out_imgfill() (in module *vtool.image*), 66
 cs (*vtool.confusion.ConfusionMetrics* attribute), 20
 cs (*vtool.ConfusionMetrics* attribute), 205
 csum() (in module *vtool*), 252
 csum() (in module *vtool.matching*), 123
 custom_sympy_attrs() (in module *vtool*), 252
 custom_sympy_attrs() (in module *vtool.symbolic*), 193
 cv (*vtool.confusion.ConfusionMetrics* attribute), 20
 cv (*vtool.ConfusionMetrics* attribute), 205

cvt_bbox_xywh_to_pt1pt2() (in module *vtool*), 253
 cvt_bbox_xywh_to_pt1pt2() (in module *vtool.geometry*), 51
 cvt_BGR2L() (in module *vtool*), 252
 cvt_BGR2L() (in module *vtool.image*), 66
 cvt_BGR2RGB() (in module *vtool*), 252
 cvt_BGR2RGB() (in module *vtool.image*), 66
 cyclic_distance() (in module *vtool*), 253
 cyclic_distance() (in module *vtool.distance*), 38

D

decompose_Z_to_invV_2x2() (in module *vtool*), 254
 decompose_Z_to_invV_2x2() (in module *vtool.keypoint*), 94
 decompose_Z_to_invV_mats2x2() (in module *vtool*), 254
 decompose_Z_to_invV_mats2x2() (in module *vtool.keypoint*), 94
 decompose_Z_to_RV_mats2x2() (in module *vtool*), 253
 decompose_Z_to_RV_mats2x2() (in module *vtool.keypoint*), 93
 decompose_Z_to_V_2x2() (in module *vtool*), 254
 decompose_Z_to_V_2x2() (in module *vtool.keypoint*), 94
 DEFAULT_DTYPE (in module *vtool*), 209
 demo_grabcut() (in module *vtool.segmentation*), 182
 demodata_match() (in module *vtool*), 254
 demodata_match() (in module *vtool.matching*), 123
 det_distance() (in module *vtool*), 254
 det_distance() (in module *vtool.distance*), 39
 det_ltri() (in module *vtool*), 254
 det_ltri() (in module *vtool.linalg*), 112
 detect_opencv_keypoints() (in module *vtool*), 254
 detect_opencv_keypoints() (in module *vtool.features*), 46
 dimkey_grammar() (*vtool.AnnotPairFeatInfo* method), 202
 dimkey_grammar() (*vtool.matching.AnnotPairFeatInfo* method), 117
 distance_to_lineseg() (in module *vtool*), 254
 distance_to_lineseg() (in module *vtool.geometry*), 51
 dot_ltri() (in module *vtool*), 254
 dot_ltri() (in module *vtool.linalg*), 112
 draw_border() (in module *vtool*), 254
 draw_border() (in module *vtool.geometry*), 51
 draw_kp_ori_steps() (in module *vtool*), 255
 draw_kp_ori_steps() (in module *vtool.patch*), 162
 draw_pair() (*vtool.inspect_matches.MatchInspector* method), 90

`draw_precision_recall_curve()` (in module `vtool`), 255
`draw_precision_recall_curve()` (in module `vtool.confusion`), 24
`draw_precision_recall_curve()` (`vtool.confusion.ConfusionMetrics` method), 20
`draw_precision_recall_curve()` (`vtool.ConfusionMetrics` method), 205
`draw_roc_curve()` (in module `vtool`), 255
`draw_roc_curve()` (in module `vtool.confusion`), 24
`draw_roc_curve()` (`vtool.confusion.ConfusionMetrics` method), 20
`draw_roc_curve()` (`vtool.ConfusionMetrics` method), 205
`draw_text()` (in module `vtool`), 256
`draw_text()` (in module `vtool.image`), 66
`draw_verts()` (in module `vtool`), 257
`draw_verts()` (in module `vtool.geometry`), 51
`draw_vsone()` (`vtool.inspect_matches.MatchInspector` method), 90
`dummy_img()` (in module `vtool`), 258
`dummy_img()` (in module `vtool.demodata`), 33
`dummy_seed()` (in module `vtool`), 258
`dummy_seed()` (in module `vtool.demodata`), 33

E

`edge_doubleclick()` (`vtool._grave.MultiMatchInspector` method), 1
`embed()` (`vtool.inspect_matches.MatchInspector` method), 90
`embed_channels()` (in module `vtool`), 258
`embed_channels()` (in module `vtool.image`), 68
`embed_in_square_image()` (in module `vtool`), 259
`embed_in_square_image()` (in module `vtool.image`), 68
`emd()` (in module `vtool`), 260
`emd()` (in module `vtool.distance`), 39
`empty_assign()` (in module `vtool`), 260
`empty_assign()` (in module `vtool.matching`), 123
`empty_neighbors()` (in module `vtool`), 260
`empty_neighbors()` (in module `vtool.matching`), 124
`ensure_3channel()` (in module `vtool`), 260
`ensure_3channel()` (in module `vtool.image`), 69
`ensure_4channel()` (in module `vtool`), 261
`ensure_4channel()` (in module `vtool.image`), 69
`ensure_alpha_channel()` (in module `vtool`), 261
`ensure_alpha_channel()` (in module `vtool.blend`), 5
`ensure_grayscale()` (in module `vtool`), 261
`ensure_grayscale()` (in module `vtool.blend`), 5
`ensure_metadata_dlen_sqrd()` (in module `vtool`), 261
`ensure_metadata_dlen_sqrd()` (in module `vtool.matching`), 124
`ensure_metadata_feats()` (in module `vtool`), 261
`ensure_metadata_feats()` (in module `vtool.matching`), 124
`ensure_metadata_flann()` (in module `vtool`), 261
`ensure_metadata_flann()` (in module `vtool.matching`), 124
`ensure_metadata_normxy()` (in module `vtool`), 261
`ensure_metadata_normxy()` (in module `vtool.matching`), 124
`ensure_metadata_vsone()` (in module `vtool`), 261
`ensure_metadata_vsone()` (in module `vtool.matching`), 124
`ensure_monotone_decreasing()` (in module `vtool`), 261
`ensure_monotone_decreasing()` (in module `vtool.util_math`), 195
`ensure_monotone_increasing()` (in module `vtool`), 262
`ensure_monotone_increasing()` (in module `vtool.util_math`), 195
`ensure_monotone_strictly_decreasing()` (in module `vtool`), 262
`ensure_monotone_strictly_decreasing()` (in module `vtool.util_math`), 196
`ensure_monotone_strictly_increasing()` (in module `vtool`), 263
`ensure_monotone_strictly_increasing()` (in module `vtool.util_math`), 196
`ensure_rng()` (in module `vtool`), 264
`ensure_rng()` (in module `vtool.other`), 141
`ensure_shape()` (in module `vtool`), 264
`ensure_shape()` (in module `vtool.numpy_utils`), 131
`ensure_shape()` (in module `vtool.other`), 141
`estimate_pdf()` (in module `vtool`), 265
`estimate_pdf()` (in module `vtool.score_normalization`), 177
`estimate_refined_transform()` (in module `vtool`), 265
`estimate_refined_transform()` (in module `vtool.spatial_verification`), 186
`evalprint()` (in module `vtool`), 265
`evalprint()` (in module `vtool.symbolic`), 193
`example_binary()` (in module `vtool`), 265
`example_binary()` (in module `vtool.clustering2`), 11
`execContextMenu()` (`vtool.inspect_matches.MatchInspector` method), 90
`execute_vsone()` (`vtool.inspect_matches.MatchInspector` method), 90
`expand_kpts()` (in module `vtool`), 265
`expand_kpts()` (in module `vtool.ellipse`), 42

[expand_scales\(\) \(in module vtool\), 265](#)
[expand_scales\(\) \(in module vtool.ellipse\), 42](#)
[expand_subscales\(\) \(in module vtool\), 265](#)
[expand_subscales\(\) \(in module vtool.ellipse\), 42](#)
[extent_from_bbox\(\) \(in module vtool\), 265](#)
[extent_from_bbox\(\) \(in module vtool.geometry\), 53](#)
[extent_from_verts\(\) \(in module vtool\), 266](#)
[extent_from_verts\(\) \(in module vtool.geometry\), 53](#)
[extract_chip_from_gpath\(\) \(in module vtool\), 266](#)
[extract_chip_from_gpath\(\) \(in module vtool.chip\), 8](#)
[extract_chip_from_gpath_into_square\(\) \(in module vtool\), 266](#)
[extract_chip_from_gpath_into_square\(\) \(in module vtool.chip\), 8](#)
[extract_chip_from_img\(\) \(in module vtool\), 266](#)
[extract_chip_from_img\(\) \(in module vtool.chip\), 8](#)
[extract_chip_into_square\(\) \(in module vtool\), 267](#)
[extract_chip_into_square\(\) \(in module vtool.chip\), 8](#)
[extract_feature_from_patch\(\) \(in module vtool\), 267](#)
[extract_feature_from_patch\(\) \(in module vtool.features\), 46](#)
[extract_features\(\) \(in module vtool\), 267](#)
[extract_features\(\) \(in module vtool.features\), 46](#)
[extrema_neighbors\(\) \(in module vtool\), 267](#)
[extrema_neighbors\(\) \(in module vtool.ellipse\), 42](#)

F

[feature\(\) \(vtool.AnnotPairFeatInfo method\), 203](#)
[feature\(\) \(vtool.matching.AnnotPairFeatInfo method\), 117](#)
[fill_holes\(\) \(in module vtool.segmentation\), 182](#)
[filter_dirty_items\(\) \(vtool.deprecated.ThumbnailCacheContext method\), 35](#)
[filterflags_valid_images\(\) \(in module vtool\), 267](#)
[filterflags_valid_images\(\) \(in module vtool.image\), 69](#)
[find\(\) \(vtool.AnnotPairFeatInfo method\), 203](#)
[find\(\) \(vtool.matching.AnnotPairFeatInfo method\), 117](#)
[find_best_undirected_edge_indexes\(\) \(in module vtool\), 268](#)
[find_best_undirected_edge_indexes\(\) \(in module vtool.other\), 142](#)
[find_clip_range\(\) \(in module vtool\), 269](#)

[find_clip_range\(\) \(in module vtool.score_normalization\), 177](#)
[find_dominant_kp_orientations\(\) \(in module vtool\), 269](#)
[find_dominant_kp_orientations\(\) \(in module vtool.patch\), 162](#)
[find_duplicate_items\(\) \(in module vtool\), 270](#)
[find_duplicate_items\(\) \(in module vtool.clustering2\), 11](#)
[find_elbow_point\(\) \(in module vtool\), 271](#)
[find_elbow_point\(\) \(in module vtool.other\), 142](#)
[find_first_true_indices\(\) \(in module vtool\), 271](#)
[find_first_true_indices\(\) \(in module vtool.other\), 143](#)
[find_k_true_indicies\(\) \(in module vtool\), 272](#)
[find_k_true_indicies\(\) \(in module vtool.other\), 144](#)
[find_kpts_direction\(\) \(in module vtool\), 272](#)
[find_kpts_direction\(\) \(in module vtool.patch\), 163](#)
[find_maxima\(\) \(in module vtool\), 272](#)
[find_maxima\(\) \(in module vtool.ellipse\), 42](#)
[find_maxima_with_neighbors\(\) \(in module vtool\), 272](#)
[find_maxima_with_neighbors\(\) \(in module vtool.ellipse\), 42](#)
[find_next_true_indices\(\) \(in module vtool\), 272](#)
[find_next_true_indices\(\) \(in module vtool.other\), 144](#)
[find_patch_dominant_orientations\(\) \(in module vtool\), 273](#)
[find_patch_dominant_orientations\(\) \(in module vtool.patch\), 163](#)
[find_pixel_value_index\(\) \(in module vtool\), 273](#)
[find_pixel_value_index\(\) \(in module vtool.image\), 70](#)
[first_show\(\) \(vtool.inspect_matches.MatchInspector method\), 90](#)
[fit\(\) \(vtool._grave.ScoreNormalizerUnsupervised method\), 1](#)
[fit\(\) \(vtool.confusion.ConfusionMetrics method\), 20](#)
[fit\(\) \(vtool.ConfusionMetrics method\), 206](#)
[fit\(\) \(vtool.score_normalization.ScoreNormalizer method\), 173](#)
[fit\(\) \(vtool.ScoreNormalizer method\), 216](#)
[fit_partitioned\(\) \(vtool.score_normalization.ScoreNormalizer method\), 173](#)
[fit_partitioned\(\) \(vtool.ScoreNormalizer method\), 217](#)
[flag_intersection\(\) \(in module vtool\), 274](#)

[flag_intersection\(\)](#) (in module *vtool.other*), 144
[flag_sym_slow\(\)](#) (in module *vtool*), 275
[flag_sym_slow\(\)](#) (in module *vtool.matching*), 124
[flag_symmetric_matches\(\)](#) (in module *vtool*), 275
[flag_symmetric_matches\(\)](#) (in module *vtool.matching*), 124
[flann_augment\(\)](#) (in module *vtool*), 275
[flann_augment\(\)](#) (in module *vtool.nearest_neighbors*), 127
[flann_cache\(\)](#) (in module *vtool*), 276
[flann_cache\(\)](#) (in module *vtool.nearest_neighbors*), 127
[FLANN_CLS](#) (in module *vtool._pyflann_backend*), 2
[flann_index_time_experiment\(\)](#) (in module *vtool*), 276
[flann_index_time_experiment\(\)](#) (in module *vtool.nearest_neighbors*), 127
[flatten_invV_mats_to_kpts\(\)](#) (in module *vtool*), 276
[flatten_invV_mats_to_kpts\(\)](#) (in module *vtool.keypoint*), 94
[flatten_scores\(\)](#) (in module *vtool*), 276
[flatten_scores\(\)](#) (in module *vtool.score_normalization*), 178
[fm](#) (*vtool.AssignTup* attribute), 204
[fm](#) (*vtool.matching.AssignTup* attribute), 118
[fn](#) (*vtool.confusion.ConfusionMetrics* attribute), 20
[fn](#) (*vtool.ConfusionMetrics* attribute), 206
[fnr](#) (*vtool.confusion.ConfusionMetrics* attribute), 20
[fnr](#) (*vtool.ConfusionMetrics* attribute), 206
[Font](#) (class in *vtool.fontdemo*), 47
[font_demo\(\)](#) (in module *vtool.fontdemo*), 48
[force_kpts_feasibility\(\)](#) (in module *vtool*), 277
[force_kpts_feasibility\(\)](#) (in module *vtool.demodata*), 33
[fourier_devtest\(\)](#) (in module *vtool.quality_classifier*), 172
[fp](#) (*vtool.confusion.ConfusionMetrics* attribute), 20
[fp](#) (*vtool.ConfusionMetrics* attribute), 206
[fpr](#) (*vtool.confusion.ConfusionMetrics* attribute), 21
[fpr](#) (*vtool.ConfusionMetrics* attribute), 206
[from_glyphslot\(\)](#) (*vtool.fontdemo.Glyph* static method), 47
[from_tp_and_tn_scores\(\)](#) (*vtool.confusion.ConfusionMetrics* class method), 21
[from_tp_and_tn_scores\(\)](#) (*vtool.ConfusionMetrics* class method), 206
[fromiter_nd\(\)](#) (in module *vtool*), 277, 278
[fromiter_nd\(\)](#) (in module *vtool.numpy_utils*), 131
[fromiter_nd\(\)](#) (in module *vtool.other*), 146
G
[gamma_adjust\(\)](#) (in module *vtool*), 280
[gamma_adjust\(\)](#) (in module *vtool.blend*), 5
[gauss2d_pdf\(\)](#) (in module *vtool*), 280
[gauss2d_pdf\(\)](#) (in module *vtool.linalg*), 112
[gauss_func1d\(\)](#) (in module *vtool*), 280
[gauss_func1d\(\)](#) (in module *vtool.util_math*), 198
[gauss_func1d_unnormalized\(\)](#) (in module *vtool*), 281
[gauss_func1d_unnormalized\(\)](#) (in module *vtool.util_math*), 198
[gauss_parzen_est\(\)](#) (in module *vtool*), 281
[gauss_parzen_est\(\)](#) (in module *vtool.util_math*), 199
[gaussian_average_patch\(\)](#) (in module *vtool*), 281
[gaussian_average_patch\(\)](#) (in module *vtool.patch*), 163
[gaussian_patch\(\)](#) (in module *vtool*), 282
[gaussian_patch\(\)](#) (in module *vtool.patch*), 164
[gaussian_weight_patch\(\)](#) (in module *vtool*), 282
[gaussian_weight_patch\(\)](#) (in module *vtool.patch*), 165
[GaussianBlurInplace\(\)](#) (in module *vtool*), 209
[GaussianBlurInplace\(\)](#) (in module *vtool.patch*), 160
[generate_to_patch_transforms\(\)](#) (in module *vtool*), 283
[generate_to_patch_transforms\(\)](#) (in module *vtool.patch*), 165
[get_accuracy\(\)](#) (*vtool.score_normalization.ScoreNormalizer* method), 173
[get_accuracy\(\)](#) (*vtool.ScoreNormalizer* method), 217
[get_affine_inliers\(\)](#) (in module *vtool*), 283
[get_affine_inliers\(\)](#) (in module *vtool.spatial_verification*), 186
[get_ave_precision\(\)](#) (*vtool.confusion.ConfusionMetrics* method), 21
[get_ave_precision\(\)](#) (*vtool.ConfusionMetrics* method), 206
[get_best_affine_inliers\(\)](#) (in module *vtool*), 284
[get_best_affine_inliers\(\)](#) (in module *vtool.spatial_verification*), 187
[get_best_affine_inliers_\(\)](#) (in module *vtool*), 284
[get_best_affine_inliers_\(\)](#) (in module *vtool.spatial_verification*), 187
[get_confusion_indicies\(\)](#) (*vtool.score_normalization.ScoreNormalizer* method), 173
[get_confusion_indicies\(\)](#) (*vtool.ScoreNormalizer* method), 217

```

get_correct_indices()
    (vtool.score_normalization.ScoreNormalizer
    method), 174
get_correct_indices() (vtool.ScoreNormalizer
    method), 217
get_coverage_grid_gridsearch_configs()
    (in module vtool.coverage_grid), 27
get_coverage_kpts_gridsearch_configs()
    (in module vtool.coverage_kpts), 29
get_covered_mask() (in module vtool), 284
get_covered_mask() (in module vtool.other), 146
get_crop_slices() (in module vtool), 284
get_crop_slices() (in module vtool.other), 146
get_cross_patch() (in module vtool), 284
get_cross_patch() (in module vtool.patch), 165
get_dummy_dpts() (in module vtool), 284
get_dummy_dpts() (in module vtool.demodata), 33
get_dummy_invV_mats() (in module vtool), 284
get_dummy_invV_mats() (in module
    vtool.demodata), 33
get_dummy_kpts() (in module vtool), 284
get_dummy_kpts() (in module vtool.demodata), 33
get_dummy_kpts_pair() (in module vtool), 284
get_dummy_kpts_pair() (in module
    vtool.demodata), 33
get_dummy_matching_kpts() (in module vtool),
    284
get_dummy_matching_kpts() (in module
    vtool.demodata), 33
get_dummy_xy() (in module vtool), 284
get_dummy_xy() (in module vtool.demodata), 33
get_error_indicies()
    (vtool.score_normalization.ScoreNormalizer
    method), 174
get_error_indicies() (vtool.ScoreNormalizer
    method), 217
get_even_point_sample() (in module vtool), 284
get_even_point_sample() (in module
    vtool.keypoint), 94
get_exif_dict() (in module vtool), 285
get_exif_dict() (in module vtool.exif), 43
get_exif_dict2() (in module vtool), 285
get_exif_dict2() (in module vtool.exif), 43
get_exif_tagids() (in module vtool), 285
get_exif_tagids() (in module vtool.exif), 43
get_exist() (in module vtool), 285
get_exist() (in module vtool.exif), 43
get_extract_features_default_params()
    (in module vtool), 285
get_extract_features_default_params()
    (in module vtool.features), 46
get_extramargin_measures() (in module vtool),
    285
get_extramargin_measures() (in module
    vtool.chip), 8
get_flann_cfgstr() (in module vtool), 286
get_flann_cfgstr() (in module
    vtool.nearest_neighbors), 128
get_flann_fpath() (in module vtool), 286
get_flann_fpath() (in module
    vtool.nearest_neighbors), 128
get_flann_params() (in module vtool), 286
get_flann_params() (in module
    vtool.nearest_neighbors), 128
get_flann_params_cfgstr() (in module vtool),
    287
get_flann_params_cfgstr() (in module
    vtool.nearest_neighbors), 129
get_fpr_at_recall()
    (vtool.confusion.ConfusionMetrics method), 21
get_fpr_at_recall() (vtool.ConfusionMetrics
    method), 206
get_gaussian_weight_patch() (in module
    vtool.coverage_kpts), 29
get_grid_kpts() (in module vtool), 287
get_grid_kpts() (in module vtool.keypoint), 94
get_histinfo_str() (in module vtool), 287
get_histinfo_str() (in module vtool.histogram),
    58
get_image_to_chip_transform() (in module
    vtool), 287
get_image_to_chip_transform() (in module
    vtool.chip), 9
get_index_at_metric()
    (vtool.confusion.ConfusionMetrics method), 21
get_index_at_metric() (vtool.ConfusionMetrics
    method), 206
get_infostr() (vtool.AnnotPairFeatInfo method),
    203
get_infostr() (vtool.matching.AnnotPairFeatInfo
    method), 117
get_invV_mats() (in module vtool), 291
get_invV_mats() (in module vtool.keypoint), 98
get_invV_mats2x2() (in module vtool), 291
get_invV_mats2x2() (in module vtool.keypoint), 98
get_invV_mats3x3() (in module vtool), 292
get_invV_mats3x3() (in module vtool.keypoint), 99
get_invVR_mats2x2() (in module vtool), 288
get_invVR_mats2x2() (in module vtool.keypoint),
    95
get_invVR_mats3x3() (in module vtool), 289
get_invVR_mats3x3() (in module vtool.keypoint),
    96
get_invVR_mats_oris() (in module vtool), 289
get_invVR_mats_oris() (in module
    vtool.keypoint), 96
get_invVR_mats_shape() (in module vtool), 290
get_invVR_mats_shape() (in module
    vtool.chip), 8

```

`vtool.keypoint)`, 97
`get_invVR_mats_sqrdscale()` (in module `vtool`), 290
`get_invVR_mats_sqrdscale()` (in module `vtool.keypoint`), 97
`get_invVR_mats_xys()` (in module `vtool`), 290
`get_invVR_mats_xys()` (in module `vtool.keypoint`), 97
`get_invVs()` (in module `vtool`), 292
`get_invVs()` (in module `vtool.keypoint`), 99
`get_kdtree_flann_params()` (in module `vtool`), 292
`get_kdtree_flann_params()` (in module `vtool.nearest_neighbors`), 129
`get_kpts_dlen_sqrdscale()` (in module `vtool`), 292
`get_kpts_dlen_sqrdscale()` (in module `vtool.keypoint`), 99
`get_kpts_dummy_img()` (in module `vtool`), 293
`get_kpts_dummy_img()` (in module `vtool.demodata`), 33
`get_kpts_eccentricity()` (in module `vtool`), 293
`get_kpts_eccentricity()` (in module `vtool.keypoint`), 100
`get_kpts_image_extent()` (in module `vtool`), 294
`get_kpts_image_extent()` (in module `vtool.keypoint`), 101
`get_kpts_strs()` (in module `vtool`), 294
`get_kpts_strs()` (in module `vtool.keypoint`), 101
`get_kpts_wh()` (in module `vtool`), 294
`get_kpts_wh()` (in module `vtool.keypoint`), 101
`get_lat_lon()` (in module `vtool`), 296
`get_lat_lon()` (in module `vtool.exif`), 43
`get_left_area()` (in module `vtool`), 296
`get_left_area()` (in module `vtool.score_normalization`), 178
`get_match_spatial_squared_error()` (in module `vtool`), 296
`get_match_spatial_squared_error()` (in module `vtool.keypoint`), 103
`get_metric_at_index()` (`vtool.confusion.ConfusionMetrics` method), 21
`get_metric_at_index()` (`vtool.ConfusionMetrics` method), 206
`get_metric_at_metric()` (`vtool.confusion.ConfusionMetrics` method), 21
`get_metric_at_metric()` (`vtool.ConfusionMetrics` method), 206
`get_metric_at_thresh()` (`vtool.confusion.ConfusionMetrics` method), 21
`get_metric_at_thresh()` (`vtool.ConfusionMetrics` method), 207
`get_no_symbol()` (in module `vtool`), 298
`get_no_symbol()` (in module `vtool.patch`), 165
`get_normalized_affine_inliers()` (in module `vtool`), 298
`get_normalized_affine_inliers()` (in module `vtool.spatial_verification`), 187
`get_num_channels()` (in module `vtool`), 298
`get_num_channels()` (in module `vtool.image`), 71
`get_ori_mats()` (in module `vtool`), 298
`get_ori_mats()` (in module `vtool.keypoint`), 104
`get_ori_strs()` (in module `vtool`), 298
`get_ori_strs()` (in module `vtool.keypoint`), 104
`get_orientation()` (in module `vtool`), 298
`get_orientation()` (in module `vtool.exif`), 44
`get_orientation_histogram()` (in module `vtool`), 299
`get_orientation_histogram()` (in module `vtool.patch`), 165
`get_orientation_str()` (in module `vtool`), 299
`get_orientation_str()` (in module `vtool.exif`), 44
`get_oris()` (in module `vtool`), 300
`get_oris()` (in module `vtool.keypoint`), 104
`get_partitioned_support()` (`vtool.score_normalization.ScoreNormalizer` method), 174
`get_partitioned_support()` (`vtool.ScoreNormalizer` method), 218
`get_pixel_dist()` (in module `vtool`), 300
`get_pixel_dist()` (in module `vtool.image`), 71
`get_pointset_extent_wh()` (in module `vtool`), 300
`get_pointset_extent_wh()` (in module `vtool.geometry`), 53
`get_pointset_extents()` (in module `vtool`), 300
`get_pointset_extents()` (in module `vtool.geometry`), 53
`get_prefix()` (`vtool.score_normalization.ScoreNormalizer` method), 174
`get_prefix()` (`vtool.ScoreNormalizer` method), 218
`get_recall_at_fpr()` (`vtool.confusion.ConfusionMetrics` method), 22
`get_recall_at_fpr()` (`vtool.ConfusionMetrics` method), 207
`get_right_area()` (in module `vtool`), 300
`get_right_area()` (in module `vtool.score_normalization`), 178
`get_round_scaled_dsize()` (in module `vtool`), 300
`get_round_scaled_dsize()` (in module `vtool.image`), 71
`get_RV_mats2x2()` (in module `vtool`), 283
`get_RV_mats2x2()` (in module `vtool.keypoint`), 94
`get_RV_mats3x3()` (in module `vtool`), 283
`get_RV_mats3x3()` (in module `vtool.keypoint`), 94
`get_scale_factor()` (in module `vtool`), 300
`get_scale_factor()` (in module `vtool.image`), 71
`get_scaled_size_with_dlen()` (in module

vtool), 300
get_scaled_size_with_dlen() (in module *vtool.chip*), 10
get_scales() (in module *vtool*), 300
get_scales() (in module *vtool.keypoint*), 104
get_shape_strs() (in module *vtool*), 300
get_shape_strs() (in module *vtool.keypoint*), 104
get_size() (in module *vtool*), 300
get_size() (in module *vtool.image*), 71
get_sqrd_scales() (in module *vtool*), 300
get_sqrd_scales() (in module *vtool.keypoint*), 104
get_star2_patch() (in module *vtool*), 301
get_star2_patch() (in module *vtool.patch*), 166
get_star_patch() (in module *vtool*), 301
get_star_patch() (in module *vtool.patch*), 166
get_stripe_patch() (in module *vtool*), 301
get_stripe_patch() (in module *vtool.patch*), 166
get_subbin_xy_neighbors() (in module *vtool.coverage_grid*), 27
get_support() (*vtool.score_normalization.ScoreNormalizer* method), 174
get_support() (*vtool.ScoreNormalizer* method), 218
get_test_patch() (in module *vtool*), 301
get_test_patch() (in module *vtool.patch*), 166
get_testdata_kpts() (in module *vtool*), 301
get_testdata_kpts() (in module *vtool.demodata*), 34
get_text_test_img() (in module *vtool.fontdemo*), 48
get_thresh_at_metric() (*vtool.confusion.ConfusionMetrics* method), 22
get_thresh_at_metric() (*vtool.ConfusionMetrics* method), 207
get_thresh_at_metric_max() (*vtool.confusion.ConfusionMetrics* method), 23
get_thresh_at_metric_max() (*vtool.ConfusionMetrics* method), 208
get_transforms_from_patch_image_kpts() (in module *vtool*), 302
get_transforms_from_patch_image_kpts() (in module *vtool.keypoint*), 105
get_uncovered_mask() (in module *vtool*), 302
get_uncovered_mask() (in module *vtool.other*), 146
get_undirected_edge_ids() (in module *vtool*), 303
get_undirected_edge_ids() (in module *vtool.other*), 147
get_uneven_point_sample() (in module *vtool*), 303
get_uneven_point_sample() (in module *vtool.keypoint*), 105
get_unixtime() (in module *vtool*), 304
get_unixtime() (in module *vtool.exif*), 45
get_unixtime_gps() (in module *vtool*), 304
get_unixtime_gps() (in module *vtool.exif*), 45
get_unwarped_patch() (in module *vtool*), 304
get_unwarped_patch() (in module *vtool.patch*), 166
get_unwarped_patches() (in module *vtool*), 304
get_unwarped_patches() (in module *vtool.patch*), 167
get_V_mats() (in module *vtool*), 283
get_V_mats() (in module *vtool.keypoint*), 94
get_warped_patch() (in module *vtool*), 304
get_warped_patch() (in module *vtool.patch*), 167
get_warped_patches() (in module *vtool*), 305
get_warped_patches() (in module *vtool.patch*), 167
get_xy_strs() (in module *vtool*), 306
get_xy_strs() (in module *vtool.keypoint*), 105
get_xys() (in module *vtool*), 306
get_xys() (in module *vtool.keypoint*), 106
get_Z_mats() (in module *vtool*), 283
get_Z_mats() (in module *vtool.keypoint*), 94
global_measure() (*vtool.AnnotPairFeatInfo* method), 203
global_measure() (*vtool.matching.AnnotPairFeatInfo* method), 117
Glyph (class in *vtool.fontdemo*), 47
glyph_for_character() (*vtool.fontdemo.Font* method), 47
grab_webcam_image() (in module *vtool*), 306
grab_webcam_image() (in module *vtool.other*), 148
grabcut() (in module *vtool.segmentation*), 182
grabcut2() (in module *vtool.segmentation*), 182
grabcut_fn() (in module *vtool.image_filters*), 88
gradient_fill() (in module *vtool*), 306
gradient_fill() (in module *vtool.patch*), 168
gradient_magnitude() (in module *vtool*), 307
gradient_magnitude() (in module *vtool.ellipse*), 42
greedy_setcover() (in module *vtool*), 307
greedy_setcover() (in module *vtool.other*), 148
gridsearch_addWeighted() (in module *vtool*), 307
gridsearch_addWeighted() (in module *vtool.blend*), 6
gridsearch_chipextract() (in module *vtool*), 307
gridsearch_chipextract() (in module *vtool.chip*), 10
gridsearch_coverage_grid() (in module *vtool.coverage_grid*), 27
gridsearch_coverage_grid_mask() (in module *vtool.coverage_grid*), 27
gridsearch_image_function() (in module *vtool*), 307

`gridsearch_image_function()` (in module `vtool.blend`), 6
`gridsearch_kpts_coverage_mask()` (in module `vtool.coverage_kpts`), 29
`group_consecutive()` (in module `vtool`), 307
`group_consecutive()` (in module `vtool.util_math`), 199
`group_counts()` (`vtool.AnnotPairFeatInfo` method), 203
`group_counts()` (`vtool.matching.AnnotPairFeatInfo` method), 117
`group_importance()` (`vtool.AnnotPairFeatInfo` method), 203
`group_importance()` (`vtool.matching.AnnotPairFeatInfo` method), 117
`group_indices()` (in module `vtool`), 308
`group_indices()` (in module `vtool.clustering2`), 12
`groupby()` (in module `vtool`), 310
`groupby()` (in module `vtool.clustering2`), 14
`groupby_dict()` (in module `vtool`), 310
`groupby_dict()` (in module `vtool.clustering2`), 14
`groupby_gen()` (in module `vtool`), 310
`groupby_gen()` (in module `vtool.clustering2`), 14
`groupedzip()` (in module `vtool`), 310
`groupedzip()` (in module `vtool.clustering2`), 14

H

`haversine()` (in module `vtool`), 311
`haversine()` (in module `vtool.distance`), 39
`height` (`vtool.fontdemo.Glyph` attribute), 47
`hist_argmaxima()` (in module `vtool`), 311
`hist_argmaxima()` (in module `vtool.histogram`), 58
`hist_argmaxima2()` (in module `vtool`), 312
`hist_argmaxima2()` (in module `vtool.histogram`), 58
`hist_edges_to_centers()` (in module `vtool`), 312
`hist_edges_to_centers()` (in module `vtool.histogram`), 59
`hist_isect()` (in module `vtool`), 312
`hist_isect()` (in module `vtool.distance`), 40
`histeq()` (`vtool.image_filters.IntensityPreproc` method), 88
`histeq_fn()` (in module `vtool.image_filters`), 88
`homogenous_circle_pts()` (in module `vtool`), 312
`homogenous_circle_pts()` (in module `vtool.ellipse`), 42

I

`iceil()` (in module `vtool`), 313
`iceil()` (in module `vtool.util_math`), 199
`imread()` (in module `vtool`), 313
`imread()` (in module `vtool.image`), 71
`imread_remote_s3()` (in module `vtool`), 315
`imread_remote_s3()` (in module `vtool.image`), 73
`imread_remote_url()` (in module `vtool`), 315
`imread_remote_url()` (in module `vtool.image`), 73
`imwrite()` (in module `vtool`), 315
`imwrite()` (in module `vtool.image`), 73
`imwrite_fallback()` (in module `vtool`), 315
`imwrite_fallback()` (in module `vtool.image`), 73
`inbounds()` (in module `vtool`), 315
`inbounds()` (in module `vtool.other`), 148
`INDEX_DTYPE` (in module `vtool`), 211
`index_partition()` (in module `vtool`), 316
`index_partition()` (in module `vtool.other`), 149
`index_to_boolmask()` (in module `vtool`), 316
`index_to_boolmask()` (in module `vtool.numpy_utils`), 133
`infer_vert()` (in module `vtool`), 317
`infer_vert()` (in module `vtool.image`), 73
`initialize()` (`vtool._grave.MultiMatchInspector` method), 1
`initialize()` (`vtool.inspect_matches.MatchInspector` method), 90
`inspect_pdfs()` (in module `vtool`), 317
`inspect_pdfs()` (in module `vtool.score_normalization`), 178
`IntensityPreproc` (class in `vtool.image_filters`), 88
`interact_roc_factory()` (in module `vtool`), 317
`interact_roc_factory()` (in module `vtool.confusion`), 25
`intern_warp_single_patch()` (in module `vtool`), 318
`intern_warp_single_patch()` (in module `vtool.patch`), 168
`interpolate_between()` (in module `vtool`), 319
`interpolate_between()` (in module `vtool.ellipse`), 43
`interpolate_maxima()` (in module `vtool`), 319
`interpolate_maxima()` (in module `vtool.ellipse`), 43
`interpolate_nans()` (in module `vtool`), 319
`interpolate_nans()` (in module `vtool.util_math`), 200
`interpolate_peaks()` (in module `vtool`), 319
`interpolate_peaks()` (in module `vtool.ellipse`), 43
`interpolate_peaks2()` (in module `vtool`), 319
`interpolate_peaks2()` (in module `vtool.ellipse`), 43
`interpolate_precision_recall()` (in module `vtool`), 319
`interpolate_precision_recall()` (in module `vtool.confusion`), 25
`interpolate_replbounds()` (in module `vtool`), 320
`interpolate_replbounds()` (in module `vtool.confusion`), 26
`interpolate_submaxima()` (in module `vtool`), 321

`interpolate_submaxima()` (in module `vtool.histogram`), 59
`interpolated_histogram()` (in module `vtool`), 322
`interpolated_histogram()` (in module `vtool.histogram`), 60
`intersect1d_reduce()` (in module `vtool`), 323
`intersect1d_reduce()` (in module `vtool.other`), 149
`intersect2d_flags()` (in module `vtool`), 323
`intersect2d_flags()` (in module `vtool.other`), 149
`intersect2d_indices()` (in module `vtool`), 323
`intersect2d_indices()` (in module `vtool.other`), 150
`intersect2d_numpy()` (in module `vtool`), 324
`intersect2d_numpy()` (in module `vtool.other`), 150
`intersect2d_structured_numpy()` (in module `vtool`), 325
`intersect2d_structured_numpy()` (in module `vtool.other`), 151
`inv_aliases` (`vtool.confusion.ConfusionMetrics` attribute), 23
`inv_aliases` (`vtool.ConfusionMetrics` attribute), 208
`inv_ltri()` (in module `vtool`), 325
`inv_ltri()` (in module `vtool.linalg`), 112
`inverse_normalize()` (`vtool.score_normalization.ScoreNormalizer` method), 174
`inverse_normalize()` (`vtool.ScoreNormalizer` method), 218
`invert_apply_grouping()` (in module `vtool`), 325
`invert_apply_grouping()` (in module `vtool.clustering2`), 15
`invert_apply_grouping2()` (in module `vtool`), 326
`invert_apply_grouping2()` (in module `vtool.clustering2`), 15
`invert_apply_grouping3()` (in module `vtool`), 326
`invert_apply_grouping3()` (in module `vtool.clustering2`), 15
`invert_invV_mats()` (in module `vtool`), 326
`invert_invV_mats()` (in module `vtool.keypoint`), 106
`inverted_sift_patch()` (in module `vtool`), 326
`inverted_sift_patch()` (in module `vtool.patch`), 169
`invertible_stack()` (in module `vtool`), 327
`invertible_stack()` (in module `vtool.nearest_neighbors`), 129
`invsum()` (in module `vtool`), 327
`invsum()` (in module `vtool.matching`), 125
`iround()` (in module `vtool`), 327
`iround()` (in module `vtool.util_math`), 200
`ishow()` (`vtool.matching.PairwiseMatch` method), 120
`ishow()` (`vtool.PairwiseMatch` method), 214
`iter_reduce_ufunc()` (in module `vtool`), 327, 328
`iter_reduce_ufunc()` (in module `vtool.numpy_utils`), 134
`iter_reduce_ufunc()` (in module `vtool.other`), 151

J

`jacc` (`vtool.confusion.ConfusionMetrics` attribute), 23
`jacc` (`vtool.ConfusionMetrics` attribute), 208
`jagged_group()` (in module `vtool`), 329
`jagged_group()` (in module `vtool.clustering2`), 15

K

`kerning_offset()` (`vtool.fontdemo.Font` method), 47
`kp_cpp_infostr()` (in module `vtool`), 329
`kp_cpp_infostr()` (in module `vtool.keypoint`), 106
`kpts_docrepr()` (in module `vtool`), 329
`kpts_docrepr()` (in module `vtool.keypoint`), 106
`KPTS_DTYPE` (in module `vtool`), 211
`kpts_matrices()` (in module `vtool`), 329
`kpts_matrices()` (in module `vtool.ellipse`), 43
`kpts_repr()` (in module `vtool`), 329
`kpts_repr()` (in module `vtool.keypoint`), 106

L

`L1()` (in module `vtool`), 211
`L1()` (in module `vtool.distance`), 36
`L2()` (in module `vtool`), 211
`L2()` (in module `vtool.distance`), 36
`L2_root_sift()` (in module `vtool`), 211
`L2_root_sift()` (in module `vtool.distance`), 36
`L2_sift()` (in module `vtool`), 211
`L2_sift()` (in module `vtool.distance`), 36
`L2_sift_sqrd()` (in module `vtool`), 211
`L2_sift_sqrd()` (in module `vtool.distance`), 36
`L2_sqrd()` (in module `vtool`), 212
`L2_sqrd()` (in module `vtool.distance`), 37
`lazy_test_annot()` (in module `vtool.inspect_matches`), 91
`learn_probabilities()` (`vtool._grave.ScoreNormalizerUnsupervised` method), 1
`learn_probabilities()` (`vtool.score_normalization.ScoreNormalizer` method), 174
`learn_probabilities()` (`vtool.ScoreNormalizer` method), 218
`learn_score_normalization()` (in module `vtool`), 329
`learn_score_normalization()` (in module `vtool.score_normalization`), 179

`learn_threshold()`
 (*vtool.score_normalization.ScoreNormalizer*
 method), 175
`learn_threshold()` (*vtool.ScoreNormalizer*
 method), 218
`learn_threshold2()`
 (*vtool.score_normalization.ScoreNormalizer*
 method), 175
`learn_threshold2()` (*vtool.ScoreNormalizer*
 method), 218
`linear_interpolation()` (*in module vtool*), 330
`linear_interpolation()` (*in module*
 vtool.histogram), 61
`list_compress_()` (*in module vtool*), 330
`list_compress_()` (*in module vtool.other*), 152
`list_take_()` (*in module vtool*), 330
`list_take_()` (*in module vtool.other*), 152
`loc_fmt` (*vtool.AnnotPairFeatInfo* attribute), 203
`loc_fmt` (*vtool.matching.AnnotPairFeatInfo* attribute),
 117
`local_measure()` (*vtool.AnnotPairFeatInfo* method),
 203
`local_measure()` (*vtool.matching.AnnotPairFeatInfo*
 method), 117
`local_rank()` (*vtool.AnnotPairFeatInfo* method), 203
`local_rank()` (*vtool.matching.AnnotPairFeatInfo*
 method), 117
`local_sorter()` (*vtool.AnnotPairFeatInfo* method),
 203
`local_sorter()` (*vtool.matching.AnnotPairFeatInfo*
 method), 117
`logistic_01()` (*in module vtool*), 330
`logistic_01()` (*in module vtool.util_math*), 200
`logit()` (*in module vtool*), 331
`logit()` (*in module vtool.util_math*), 201

M

`main()` (*in module vtool.__main__*), 1
`make_channels_comparable()` (*in module vtool*),
 331
`make_channels_comparable()` (*in module*
 vtool.image), 73
`make_dummy_fm()` (*in module vtool*), 331
`make_dummy_fm()` (*in module vtool.demodata*), 34
`make_exif_dict_human_readable()` (*in mod-*
 ule vtool), 331
`make_exif_dict_human_readable()` (*in mod-*
 ule vtool.exif), 45
`make_feature_vector()`
 (*vtool.matching.PairwiseMatch* *method*),
 120
`make_feature_vector()` (*vtool.PairwiseMatch*
 method), 214
`make_grid_coverage_mask()` (*in module*
 vtool.coverage_grid), 27
`make_heatmask()` (*in module vtool.coverage_kpts*),
 29
`make_kpts_coverage_mask()` (*in module*
 vtool.coverage_kpts), 29
`make_kpts_heatmask()` (*in module*
 vtool.coverage_kpts), 30
`make_match_interaction()` (*in module*
 vtool.inspect_matches), 91
`make_pairfeat_cfg()` (*vtool.AnnotPairFeatInfo*
 method), 203
`make_pairfeat_cfg()`
 (*vtool.matching.AnnotPairFeatInfo* *method*),
 117
`make_test_image_keypoints()` (*in module*
 vtool), 331
`make_test_image_keypoints()` (*in module*
 vtool.patch), 170
`make_video()` (*in module vtool*), 331
`make_video()` (*in module vtool.other*), 152
`make_video2()` (*in module vtool*), 331
`make_video2()` (*in module vtool.other*), 152
`make_white_transparent()` (*in module vtool*),
 331
`make_white_transparent()` (*in module*
 vtool.image), 74
`manta_matcher_filters()` (*in module*
 vtool.image_filters), 89
`mask_colored_img()` (*in module*
 vtool.segmentation), 182
`match_dist` (*vtool.AssignTup* attribute), 204
`match_dist` (*vtool.matching.AssignTup* attribute), 118
`match_inspect_graph()` (*in module vtool._grave*),
 1
`matched_vecs2()` (*vtool.matching.PairwiseMatch*
 method), 121
`matched_vecs2()` (*vtool.PairwiseMatch* *method*),
 215
`MatchingError`, 118, 212
`MatchInspector` (*class in vtool.inspect_matches*), 89
`maxima_neighbors()` (*in module vtool*), 332
`maxima_neighbors()` (*in module vtool.histogram*),
 61
`maximum_parabola_point()` (*in module vtool*),
 332
`maximum_parabola_point()` (*in module*
 vtool.histogram), 61
`maxwh()` (*vtool.chip.ScaleStrat* static method), 6
`maxwh()` (*vtool.ScaleStrat* static method), 215
`mcc` (*vtool.confusion.ConfusionMetrics* attribute), 23
`mcc` (*vtool.ConfusionMetrics* attribute), 208
`measure()` (*vtool.AnnotPairFeatInfo* method), 203
`measure()` (*vtool.matching.AnnotPairFeatInfo*

method), 118
 measure_type() (*vtool.AnnotPairFeatInfo method*), 203
 measure_type() (*vtool.matching.AnnotPairFeatInfo method*), 118
 median_abs_dev() (*in module vtool*), 332
 median_abs_dev() (*in module vtool.other*), 152
 medianblur() (*vtool.image_filters.IntensityPreproc method*), 88
 medianfilter_fn() (*in module vtool.image_filters*), 89
 minimizing_metrics
 (*vtool.confusion.ConfusionMetrics attribute*), 23
 minimizing_metrics (*vtool.ConfusionMetrics attribute*), 208
 mk (*vtool.confusion.ConfusionMetrics attribute*), 23
 mk (*vtool.ConfusionMetrics attribute*), 208
 montage() (*in module vtool*), 332
 montage() (*in module vtool.image*), 74
 mult_lists() (*in module vtool*), 333
 mult_lists() (*in module vtool.other*), 153
 multiaxis_reduce() (*in module vtool*), 333
 multiaxis_reduce() (*in module vtool.numpy_utils*), 134
 multigroup_lookup() (*in module vtool*), 333
 multigroup_lookup() (*in module vtool.other*), 153
 multigroup_lookup_naive() (*in module vtool*), 334
 multigroup_lookup_naive() (*in module vtool.other*), 153
 MultiMatchInspector (*class in vtool._grave*), 1

N

nan_to_num() (*in module vtool*), 334
 nan_to_num() (*in module vtool.confusion*), 27
 nearest_point() (*in module vtool*), 334, 335
 nearest_point() (*in module vtool.distance*), 40
 nearest_point() (*in module vtool.other*), 154
 nn() (*vtool.AnnoyWrapper method*), 204
 nn() (*vtool.clustering2.AnnoyWrapper method*), 10
 nn_index() (*vtool.AnnoyWrapper method*), 204
 nn_index() (*vtool.nearest_neighbors.AnnoyWrapper method*), 125
 non_decreasing() (*in module vtool*), 335
 non_decreasing() (*in module vtool.util_math*), 201
 non_increasing() (*in module vtool*), 335
 non_increasing() (*in module vtool.util_math*), 201
 nonunique_row_flags() (*in module vtool*), 335
 nonunique_row_flags() (*in module vtool.other*), 154
 nonunique_row_indexes() (*in module vtool*), 335
 nonunique_row_indexes() (*in module vtool.other*), 154

norm01() (*in module vtool*), 335
 norm01() (*in module vtool.other*), 154
 norm_dist (*vtool.AssignTup attribute*), 204
 norm_dist (*vtool.matching.AssignTup attribute*), 118
 norm_fx1 (*vtool.AssignTup attribute*), 204
 norm_fx1 (*vtool.matching.AssignTup attribute*), 118
 normalize() (*in module vtool*), 336
 normalize() (*in module vtool.linalg*), 112
 normalize_rows() (*in module vtool*), 336
 normalize_rows() (*in module vtool.linalg*), 113
 normalize_scores() (*in module vtool*), 337
 normalize_scores() (*in module vtool.score_normalization*), 180
 normalize_scores() (*vtool.score_normalization.ScoreNormalizer method*), 176
 normalize_scores() (*vtool.ScoreNormalizer method*), 219
 normalized_nearest_neighbors() (*in module vtool*), 337
 normalized_nearest_neighbors() (*in module vtool.matching*), 125

O

offset_kpts() (*in module vtool*), 337
 offset_kpts() (*in module vtool.keypoint*), 106
 on_cfg_changed() (*vtool.inspect_matches.MatchInspector method*), 91
 on_chip_cfg_changed() (*vtool.inspect_matches.MatchInspector method*), 91
 on_feat_cfg_changed() (*vtool.inspect_matches.MatchInspector method*), 91
 open_image_size() (*in module vtool*), 338
 open_image_size() (*in module vtool.image*), 75
 open_pil_image() (*in module vtool*), 339
 open_pil_image() (*in module vtool.image_shared*), 89
 or_lists() (*in module vtool*), 339
 or_lists() (*in module vtool.other*), 154
 ori_distance() (*in module vtool*), 339
 ori_distance() (*in module vtool.distance*), 40
 overlay_alpha_images() (*in module vtool*), 340
 overlay_alpha_images() (*in module vtool.blend*), 6
 pad_image() (*in module vtool*), 340
 pad_image() (*in module vtool.image*), 76
 pad_image_ondisk() (*in module vtool*), 340
 pad_image_ondisk() (*in module vtool.image*), 76
 pad_vstack() (*in module vtool*), 340
 pad_vstack() (*in module vtool.other*), 155

P

padded_resize() (in module vtool), 340
 padded_resize() (in module vtool.image), 76
 PairwiseMatch (class in vtool), 212
 PairwiseMatch (class in vtool.matching), 118
 paper_alias (vtool.confusion.ConfusionMetrics attribute), 23
 paper_alias (vtool.ConfusionMetrics attribute), 208
 paper_relations (vtool.confusion.ConfusionMetrics attribute), 23
 paper_relations (vtool.ConfusionMetrics attribute), 208
 parse_exif_unixtime() (in module vtool), 341
 parse_exif_unixtime() (in module vtool.exif), 45
 parse_exif_unixtime_gps() (in module vtool), 341
 parse_exif_unixtime_gps() (in module vtool.exif), 45
 partition_scores() (in module vtool), 341
 partition_scores() (in module vtool.score_normalization), 180
 patch_gaussian_weighted_average_intensities() (in module vtool), 342
 patch_gaussian_weighted_average_intensities() (in module vtool.patch), 170
 patch_gradient() (in module vtool), 342
 patch_gradient() (in module vtool.patch), 170
 patch_mag() (in module vtool), 342
 patch_mag() (in module vtool.patch), 170
 patch_ori() (in module vtool), 342
 patch_ori() (in module vtool.patch), 170
 pdist_argsort() (in module vtool), 342
 pdist_argsort() (in module vtool.distance), 41
 pdist_indicies() (in module vtool), 342
 pdist_indicies() (in module vtool.distance), 41
 perlin_noise() (in module vtool), 342
 perlin_noise() (in module vtool.image), 77
 perterb_kpts() (in module vtool), 343
 perterb_kpts() (in module vtool.demodata), 34
 perterbed_grid_kpts() (in module vtool), 343
 perterbed_grid_kpts() (in module vtool.demodata), 34
 plot_centroids() (in module vtool), 343
 plot_centroids() (in module vtool.clustering2), 15
 plot_metrics() (vtool.confusion.ConfusionMetrics method), 23
 plot_metrics() (vtool.ConfusionMetrics method), 208
 plot_postbayes_pdf() (in module vtool), 343
 plot_postbayes_pdf() (in module vtool.score_normalization), 181
 plot_prebayes_pdf() (in module vtool), 343
 plot_prebayes_pdf() (in module vtool.score_normalization), 181
 plot_vs() (vtool.confusion.ConfusionMetrics method), 23
 plot_vs() (vtool.ConfusionMetrics method), 208
 pn (vtool.confusion.ConfusionMetrics attribute), 23
 pn (vtool.ConfusionMetrics attribute), 208
 point_inside_bbox() (in module vtool), 343
 point_inside_bbox() (in module vtool.geometry), 53
 populate_edge_model() (vtool._grave.MultiMatchInspector method), 1
 pp (vtool.confusion.ConfusionMetrics attribute), 23
 pp (vtool.ConfusionMetrics attribute), 208
 predict() (vtool.score_normalization.ScoreNormalizer method), 176
 predict() (vtool.ScoreNormalizer method), 219
 preprocess() (vtool.image_filters.IntensityPreproc method), 88
 print_counts() (vtool.AnnotPairFeatInfo method), 203
 print_counts() (vtool.matching.AnnotPairFeatInfo method), 118
 print_image_checks() (in module vtool), 344
 print_image_checks() (in module vtool.image_shared), 89
 print_margins() (vtool.AnnotPairFeatInfo method), 203
 print_margins() (vtool.matching.AnnotPairFeatInfo method), 118
 printDBG() (in module vtool.segmentation), 182

Q

query_annoy() (vtool.AnnoyWrapper method), 204
 query_annoy() (vtool.clustering2.AnnoyWrapper method), 10

R

random_affine_args() (in module vtool), 344
 random_affine_args() (in module vtool.linalg), 113
 random_affine_transform() (in module vtool), 345
 random_affine_transform() (in module vtool.linalg), 114
 ratio_test_flags() (vtool.matching.PairwiseMatch method), 121
 ratio_test_flags() (vtool.PairwiseMatch method), 215
 read_all_exif_tags() (in module vtool), 345
 read_all_exif_tags() (in module vtool.exif), 45
 read_exif() (in module vtool), 345
 read_exif() (in module vtool.exif), 45
 read_exif_tags() (in module vtool), 345
 read_exif_tags() (in module vtool.exif), 45

read_one_exif_tag() (in module vtool), 345
 read_one_exif_tag() (in module vtool.exif), 45
 rebuild_partition() (in module vtool), 345
 rebuild_partition() (in module vtool.other), 155
 rectify_invV_mats_are_up() (in module vtool), 346
 rectify_invV_mats_are_up() (in module vtool.keypoint), 107
 rectify_to_float01() (in module vtool), 347
 rectify_to_float01() (in module vtool.image), 78
 rectify_to_square() (in module vtool), 347
 rectify_to_square() (in module vtool.image), 78
 rectify_to_uint8() (in module vtool), 347
 rectify_to_uint8() (in module vtool.image), 78
 refine_inliers() (in module vtool), 347
 refine_inliers() (in module vtool.spatial_verification), 187
 remove_homogenous_coordinate() (in module vtool), 347
 remove_homogenous_coordinate() (in module vtool.linalg), 114
 render_character() (vtool.fontdemo.Font method), 47
 render_text() (vtool.fontdemo.Font method), 47
 resize() (in module vtool), 348
 resize() (in module vtool.image), 78
 resize_image_by_scale() (in module vtool), 348
 resize_image_by_scale() (in module vtool.image), 78
 resize_img_and_bbox() (in module vtool.segmentation), 182
 resize_mask() (in module vtool), 348
 resize_mask() (in module vtool.image), 78
 resize_thumb() (in module vtool), 348
 resize_thumb() (in module vtool.image), 78
 resize_to_maxdims() (in module vtool), 349
 resize_to_maxdims() (in module vtool.image), 78
 resize_to_maxdims_ondisk() (in module vtool), 349
 resize_to_maxdims_ondisk() (in module vtool.image), 79
 resized_clamped_thumb_dims() (in module vtool), 350
 resized_clamped_thumb_dims() (in module vtool.image), 79
 resized_dims_and_ratio() (in module vtool), 350
 resized_dims_and_ratio() (in module vtool.image), 79
 RhombicuboctahedronDistanceDemo() (in module vtool._rhomb_dist), 2
 rn (vtool.confusion.ConfusionMetrics attribute), 23
 rn (vtool.ConfusionMetrics attribute), 208
 rotate_image() (in module vtool), 351
 rotate_image() (in module vtool.image), 81
 rotate_image_ondisk() (in module vtool), 351
 rotate_image_ondisk() (in module vtool.image), 81
 rotation_around_bbox_mat3x3() (in module vtool), 352
 rotation_around_bbox_mat3x3() (in module vtool.linalg), 115
 rotation_around_mat3x3() (in module vtool), 352
 rotation_around_mat3x3() (in module vtool.linalg), 115
 rotation_mat2x2() (in module vtool), 352
 rotation_mat2x2() (in module vtool.linalg), 115
 rotation_mat3x3() (in module vtool), 352
 rotation_mat3x3() (in module vtool.linalg), 115
 rowwise_operation() (in module vtool), 352
 rowwise_operation() (in module vtool.other), 155
 rp (vtool.confusion.ConfusionMetrics attribute), 23
 rp (vtool.ConfusionMetrics attribute), 208
 rrr() (vtool._grave.ScoreNormalizerUnsupervised method), 1
 rrr() (vtool.AnnotPairFeatInfo method), 203
 rrr() (vtool.matching.AnnotPairFeatInfo method), 118

S

safe_argmax() (in module vtool), 352
 safe_argmax() (in module vtool.other), 155
 safe_cat() (in module vtool), 352
 safe_cat() (in module vtool.other), 155
 safe_div() (in module vtool), 353
 safe_div() (in module vtool.other), 156
 safe_extreme() (in module vtool), 353
 safe_extreme() (in module vtool.other), 156
 safe_max() (in module vtool), 353
 safe_max() (in module vtool.other), 156
 safe_min() (in module vtool), 354
 safe_min() (in module vtool.other), 157
 safe_pdist() (in module vtool), 354
 safe_pdist() (in module vtool.distance), 41
 safe_vstack() (in module vtool), 354
 safe_vstack() (in module vtool.other), 157
 sample_ell_border_pts() (in module vtool), 354
 sample_ell_border_pts() (in module vtool.ellipse), 43
 sample_ell_border_vals() (in module vtool), 354
 sample_ell_border_vals() (in module vtool.ellipse), 43
 sample_uniform() (in module vtool), 354
 sample_uniform() (in module vtool.ellipse), 43
 save_dirty_thumbs_from_images() (vtool.deprecated.ThumbnailCacheContext

method), 36
 scale_around_mat3x3() (in module *vtool*), 354
 scale_around_mat3x3() (in module *vtool.linalg*), 115
 scale_bbox() (in module *vtool*), 354
 scale_bbox() (in module *vtool.geometry*), 54
 scale_extents() (in module *vtool*), 355
 scale_extents() (in module *vtool.geometry*), 54
 scale_mat3x3() (in module *vtool*), 355
 scale_mat3x3() (in module *vtool.linalg*), 115
 scaled_verts_from_bbox() (in module *vtool*), 355
 scaled_verts_from_bbox() (in module *vtool.geometry*), 54
 scaled_verts_from_bbox_gen() (in module *vtool*), 355
 scaled_verts_from_bbox_gen() (in module *vtool.geometry*), 54
 ScaleStrat (class in *vtool*), 215
 ScaleStrat (class in *vtool.chip*), 6
 ScoreNormalizer (class in *vtool*), 216
 ScoreNormalizer (class in *vtool.score_normalization*), 173
 ScoreNormalizerUnsupervised (class in *vtool.grave*), 1
 ScoreNormVisualizeClass (class in *vtool*), 216
 ScoreNormVisualizeClass (class in *vtool.score_normalization*), 173
 screenshot() (*vtool.inspect_matches.MatchInspector* method), 91
 segment() (in module *vtool.segmentation*), 182
 select_columns() (*vtool.AnnotPairFeatInfo* method), 203
 select_columns() (*vtool.matching.AnnotPairFeatInfo* method), 118
 set_match() (*vtool.inspect_matches.MatchInspector* method), 91
 shear() (in module *vtool*), 355
 shear() (in module *vtool.image*), 82
 shear_mat3x3() (in module *vtool*), 356
 shear_mat3x3() (in module *vtool.linalg*), 115
 show() (*vtool.matching.PairwiseMatch* method), 121
 show() (*vtool.PairwiseMatch* method), 215
 show_coverage_grid() (in module *vtool.coverage_grid*), 28
 show_coverage_map() (in module *vtool.coverage_kpts*), 31
 show_gaussian_patch() (in module *vtool*), 356
 show_gaussian_patch() (in module *vtool.patch*), 170
 show_hist_submaxima() (in module *vtool*), 356
 show_hist_submaxima() (in module *vtool.histogram*), 61
 show_matching_dict() (in module *vtool.inspect_matches*), 91
 show_mcc() (*vtool.confusion.ConfusionMetrics* method), 23
 show_mcc() (*vtool.ConfusionMetrics* method), 208
 show_ori_image() (in module *vtool*), 356
 show_ori_image() (in module *vtool.histogram*), 62
 show_ori_image_ondisk() (in module *vtool*), 356
 show_ori_image_ondisk() (in module *vtool.histogram*), 62
 show_patch_orientation_estimation() (in module *vtool*), 357
 show_patch_orientation_estimation() (in module *vtool.patch*), 170
 showEvent() (*vtool.inspect_matches.MatchInspector* method), 91
 signed_cyclic_distance() (in module *vtool*), 357
 signed_cyclic_distance() (in module *vtool.distance*), 41
 signed_ori_distance() (in module *vtool*), 357
 signed_ori_distance() (in module *vtool.distance*), 41
 significant_shape() (in module *vtool*), 357
 significant_shape() (in module *vtool.other*), 157
 sorted_indices_ranges() (in module *vtool*), 357
 sorted_indices_ranges() (in module *vtool.clustering2*), 15
 sparse_grid_coverage() (in module *vtool.coverage_grid*), 28
 spatially_verify_kpts() (in module *vtool*), 357
 spatially_verify_kpts() (in module *vtool.spatial_verification*), 188
 sqrd_error (*vtool.confusion.ConfusionMetrics* attribute), 23
 sqrd_error (*vtool.ConfusionMetrics* attribute), 208
 stack_image_list() (in module *vtool*), 359
 stack_image_list() (in module *vtool.image*), 82
 stack_image_list_special() (in module *vtool*), 359
 stack_image_list_special() (in module *vtool.image*), 83
 stack_image_recurse() (in module *vtool*), 360
 stack_image_recurse() (in module *vtool.image*), 83
 stack_images() (in module *vtool*), 361
 stack_images() (in module *vtool.image*), 84
 stack_multi_images() (in module *vtool*), 362
 stack_multi_images() (in module *vtool.image*), 85
 stack_multi_images2() (in module *vtool*), 362
 stack_multi_images2() (in module *vtool.image*), 85
 stack_square_images() (in module *vtool*), 363
 stack_square_images() (in module *vtool.image*),

86

`strictly_decreasing()` (in module *vtool*), 363

`strictly_decreasing()` (in module *vtool.util_math*), 201

`strictly_increasing()` (in module *vtool*), 363

`strictly_increasing()` (in module *vtool.util_math*), 201

`structure_rows()` (in module *vtool*), 363

`structure_rows()` (in module *vtool.other*), 157

`subbin_bounds()` (in module *vtool*), 364

`subbin_bounds()` (in module *vtool.histogram*), 62

`subpixel_values()` (in module *vtool*), 364

`subpixel_values()` (in module *vtool.image*), 86

`subscale_peaks()` (in module *vtool*), 364

`subscale_peaks()` (in module *vtool.ellipse*), 43

`sum_fmt` (*vtool.AnnotPairFeatInfo* attribute), 203

`sum_fmt` (*vtool.matching.AnnotPairFeatInfo* attribute), 118

`summary_binkey()` (*vtool.AnnotPairFeatInfo* method), 203

`summary_binkey()` (*vtool.matching.AnnotPairFeatInfo* method), 118

`summary_binval()` (*vtool.AnnotPairFeatInfo* method), 203

`summary_binval()` (*vtool.matching.AnnotPairFeatInfo* method), 118

`summary_measure()` (*vtool.AnnotPairFeatInfo* method), 203

`summary_measure()` (*vtool.matching.AnnotPairFeatInfo* method), 118

`summary_op()` (*vtool.AnnotPairFeatInfo* method), 204

`summary_op()` (*vtool.matching.AnnotPairFeatInfo* method), 118

`SV_DTYPE` (in module *vtool*), 215

`svd()` (in module *vtool*), 364

`svd()` (in module *vtool.linalg*), 115

`sver_flags()` (*vtool.matching.PairwiseMatch* method), 121

`sver_flags()` (*vtool.PairwiseMatch* method), 215

`symbolic_randcheck()` (in module *vtool*), 365

`symbolic_randcheck()` (in module *vtool.symbolic*), 193

`symmetric_correspondence()` (in module *vtool*), 365

`symmetric_correspondence()` (in module *vtool.matching*), 125

`sympy_latex_repr()` (in module *vtool*), 365

`sympy_latex_repr()` (in module *vtool.symbolic*), 193

`sympy_mat()` (in module *vtool*), 365

`sympy_mat()` (in module *vtool.symbolic*), 193

`sympy_numpy_repr()` (in module *vtool*), 365

`sympy_numpy_repr()` (in module *vtool.symbolic*),

193

T

`take()` (*vtool.matching.PairwiseMatch* method), 121

`take()` (*vtool.PairwiseMatch* method), 215

`take2()` (in module *vtool*), 365

`take2()` (in module *vtool.other*), 158

`take_col_per_row()` (in module *vtool*), 365

`take_col_per_row()` (in module *vtool.other*), 158

`TEMP_VEC_DTYPE` (in module *vtool*), 220

`test_affine_errors()` (in module *vtool*), 365

`test_affine_errors()` (in module *vtool.spatial_verification*), 189

`test_annoy()` (in module *vtool*), 365

`test_annoy()` (in module *vtool.nearest_neighbors*), 129

`test_average_contrast()` (in module *vtool.quality_classifier*), 173

`test_cv2_flann()` (in module *vtool*), 365

`test_cv2_flann()` (in module *vtool.nearest_neighbors*), 129

`test_homog_errors()` (in module *vtool*), 365

`test_homog_errors()` (in module *vtool.spatial_verification*), 189

`test_language_modulus()` (in module *vtool*), 367

`test_language_modulus()` (in module *vtool.util_math*), 201

`test_mser()` (in module *vtool*), 367

`test_mser()` (in module *vtool.features*), 47

`test_ondisk_find_patch_fpath_dominant_orientations` (in module *vtool*), 367

`test_ondisk_find_patch_fpath_dominant_orientations` (in module *vtool.patch*), 170

`test_score_normalization()` (in module *vtool*), 368

`test_score_normalization()` (in module *vtool.score_normalization*), 181

`test_show_gaussian_patches()` (in module *vtool*), 368

`test_show_gaussian_patches()` (in module *vtool.patch*), 171

`test_show_gaussian_patches2()` (in module *vtool*), 369

`test_show_gaussian_patches2()` (in module *vtool.patch*), 171

`testdata_annot_metadata()` (in module *vtool*), 369

`testdata_annot_metadata()` (in module *vtool.matching*), 125

`testdata_binary_scores()` (in module *vtool*), 369

`testdata_binary_scores()` (in module *vtool.demodata*), 34

`testdata_blend()` (in module *vtool*), 369

testdata_blend() (in module *vtool.blend*), 6
 testdata_coverage() (in module *vtool.coverage_kpts*), 31
 testdata_dummy_matches() (in module *vtool*), 369
 testdata_dummy_matches() (in module *vtool.demodata*), 34
 testdata_dummy_sift() (in module *vtool*), 370
 testdata_dummy_sift() (in module *vtool.demodata*), 34
 testdata_hist() (in module *vtool*), 370
 testdata_hist() (in module *vtool.distance*), 42
 testdata_imglist() (in module *vtool*), 370
 testdata_imglist() (in module *vtool.image*), 86
 testdata_matching_affine_inliers() (in module *vtool*), 370
 testdata_matching_affine_inliers() (in module *vtool.spatial_verification*), 192
 testdata_matching_affine_inliers_normalized() (in module *vtool*), 370
 testdata_matching_affine_inliers_normalized() (in module *vtool.spatial_verification*), 192
 testdata_nonmonotonic() (in module *vtool*), 370
 testdata_nonmonotonic() (in module *vtool.demodata*), 34
 testdata_patch() (in module *vtool*), 370
 testdata_patch() (in module *vtool.patch*), 172
 testdata_ratio_matches() (in module *vtool*), 370
 testdata_ratio_matches() (in module *vtool.demodata*), 35
 testdata_score_normalier() (in module *vtool*), 371
 testdata_score_normalier() (in module *vtool.score_normalization*), 182
 testdata_scores_labels() (in module *vtool*), 371
 testdata_scores_labels() (in module *vtool.confusion*), 27
 testdata_sift2() (in module *vtool*), 371
 testdata_sift2() (in module *vtool.distance*), 42
 testshow_extramargin_info() (in module *vtool*), 371
 testshow_extramargin_info() (in module *vtool.chip*), 10
 text_dimensions() (*vtool.fontdemo.Font* method), 47
 thresh (*vtool.confusion.ConfusionMetrics* attribute), 23
 thresh (*vtool.ConfusionMetrics* attribute), 209
 ThumbnailCacheContext (class in *vtool.deprecated*), 35
 tn (*vtool.confusion.ConfusionMetrics* attribute), 23
 tn (*vtool.ConfusionMetrics* attribute), 209
 tna (*vtool.confusion.ConfusionMetrics* attribute), 23
 tna (*vtool.ConfusionMetrics* attribute), 209
 tnr (*vtool.confusion.ConfusionMetrics* attribute), 23
 tnr (*vtool.ConfusionMetrics* attribute), 209
 to_undirected_edges() (in module *vtool*), 371
 to_undirected_edges() (in module *vtool.other*), 158
 tp (*vtool.confusion.ConfusionMetrics* attribute), 24
 tp (*vtool.ConfusionMetrics* attribute), 209
 tpa (*vtool.confusion.ConfusionMetrics* attribute), 24
 tpa (*vtool.ConfusionMetrics* attribute), 209
 tpr (*vtool.confusion.ConfusionMetrics* attribute), 24
 tpr (*vtool.ConfusionMetrics* attribute), 209
 transform_around() (in module *vtool*), 371
 transform_around() (in module *vtool.linalg*), 115
 TRANSFORM_DTYPE (in module *vtool*), 220
 transform_kpts() (in module *vtool*), 371
 transform_kpts() (in module *vtool.keypoint*), 108
 transform_kpts_to_imgspace() (in module *vtool*), 372
 transform_kpts_to_imgspace() (in module *vtool.keypoint*), 109
 transform_kpts_xys() (in module *vtool*), 372
 transform_kpts_xys() (in module *vtool.keypoint*), 109
 transform_points_with_homography() (in module *vtool*), 372
 transform_points_with_homography() (in module *vtool.linalg*), 115
 translation_mat3x3() (in module *vtool*), 372
 translation_mat3x3() (in module *vtool.linalg*), 116
 try_svd() (in module *vtool*), 372
 try_svd() (in module *vtool.spatial_verification*), 192
 trytake() (in module *vtool*), 373
 trytake() (in module *vtool.other*), 158
 tune_flann() (in module *vtool*), 373
 tune_flann() (in module *vtool.nearest_neighbors*), 129
 tune_flann2() (in module *vtool*), 374
 tune_flann2() (in module *vtool.clustering2*), 15

U

understanding_pseudomax_props() (in module *vtool*), 374
 understanding_pseudomax_props() (in module *vtool.distance*), 42
 uniform_sample_hypersphere() (in module *vtool*), 374
 uniform_sample_hypersphere() (in module *vtool.clustering2*), 15
 union_extents() (in module *vtool*), 375
 union_extents() (in module *vtool.geometry*), 54
 unique_row_indexes() (in module *vtool*), 375

unique_row_indexes() (in module
 vtool.numpy_utils), 135
 unique_rows() (in module vtool), 375
 unique_rows() (in module vtool.other), 158
 unnormalize_transform() (in module vtool), 375
 unnormalize_transform() (in module
 vtool.spatial_verification), 192
 unpack_mono_bitmap() (vtool.fontdemo.Glyph
 static method), 47
 unstructure_rows() (in module vtool), 375
 unstructure_rows() (in module vtool.other), 158
 unsupervised_multicut_labeling() (in mod-
 ule vtool), 375
 unsupervised_multicut_labeling() (in mod-
 ule vtool.clustering2), 16
 update() (vtool.inspect_matches.MatchInspector
 method), 91

V

verts_from_bbox() (in module vtool), 378
 verts_from_bbox() (in module vtool.geometry), 54
 verts_list_from_bboxes_list() (in module
 vtool), 378
 verts_list_from_bboxes_list() (in module
 vtool.geometry), 55
 visualize() (vtool._grave.ScoreNormalizerUnsupervised
 method), 1
 visualize() (vtool.score_normalization.ScoreNormalizer
 method), 176
 visualize() (vtool.ScoreNormalizer method), 219
 vtool (module), 201
 vtool.__main__ (module), 1
 vtool._grave (module), 1
 vtool._old_matching (module), 2
 vtool._pyflann_backend (module), 2
 vtool._rhomb_dist (module), 2
 vtool.blend (module), 2
 vtool.chip (module), 6
 vtool.clustering2 (module), 10
 vtool.confusion (module), 19
 vtool.coverage_grid (module), 27
 vtool.coverage_kpts (module), 29
 vtool.demodata (module), 33
 vtool.deprecated (module), 35
 vtool.distance (module), 36
 vtool.ellipse (module), 42
 vtool.exif (module), 43
 vtool.features (module), 46
 vtool.fontdemo (module), 47
 vtool.geometry (module), 48
 vtool.histogram (module), 55
 vtool.image (module), 64
 vtool.image_filters (module), 88
 vtool.image_shared (module), 89

vtool.inspect_matches (module), 89
 vtool.keypoint (module), 91
 vtool.linalg (module), 109
 vtool.matching (module), 116
 vtool.nearest_neighbors (module), 125
 vtool.numpy_utils (module), 130
 vtool.other (module), 135
 vtool.patch (module), 160
 vtool.quality_classifier (module), 172
 vtool.score_normalization (module), 173
 vtool.segmentation (module), 182
 vtool.spatial_verification (module), 183
 vtool.symbolic (module), 192
 vtool.trig (module), 193
 vtool.util_math (module), 194

W

warp_patch_onto_kpts() (in module
 vtool.coverage_kpts), 31
 warpAffine() (in module vtool), 378
 warpAffine() (in module vtool.image), 86
 warped_patch_generator() (in module
 vtool.coverage_kpts), 32
 warpHomog() (in module vtool), 379
 warpHomog() (in module vtool.image), 87
 weighted_average_scoring() (in module vtool),
 379
 weighted_average_scoring() (in module
 vtool.other), 158
 weighted_gaussian_falloff() (in module
 vtool.coverage_grid), 29
 weighted_geometric_mean() (in module vtool),
 380
 weighted_geometric_mean() (in module
 vtool.other), 159
 weighted_geometric_mean_unnormalized()
 (in module vtool), 381
 weighted_geometric_mean_unnormalized()
 (in module vtool.other), 159
 whiten_xy_points() (in module vtool), 381
 whiten_xy_points() (in module vtool.linalg), 116
 width (vtool.fontdemo.Glyph attribute), 47
 width() (vtool.chip.ScaleStrat static method), 7
 width() (vtool.ScaleStrat static method), 216
 wracc (vtool.confusion.ConfusionMetrics attribute), 24
 wracc (vtool.ConfusionMetrics attribute), 209
 wrap_histogram() (in module vtool), 381
 wrap_histogram() (in module vtool.histogram), 63
 wrapped_distance() (in module vtool), 381
 wrapped_distance() (in module vtool.distance), 42

Z

zipcat() (in module vtool), 382
 zipcat() (in module vtool.other), 160

`zipcompress()` (*in module vtool*), [382](#)
`zipcompress()` (*in module vtool.other*), [160](#)
`zipcompress_safe()` (*in module vtool*), [382](#)
`zipcompress_safe()` (*in module vtool.other*), [160](#)
`ziptake()` (*in module vtool*), [382](#)
`ziptake()` (*in module vtool.other*), [160](#)
`zstar_value()` (*in module vtool*), [382](#)
`zstar_value()` (*in module vtool.other*), [160](#)