trackpy Documentation

Release unknown

Trackpy Contributors

May 30, 2015
## Contents

1  Contents: 3
   1.1  trackpy Package 3

2  Indices and tables 15

Python Module Index 17
Release

date  May 30, 2015

The Crocker-Grier algorithm is a method of tracking features in a series of images from frame to frame. The core of the algorithm is to choose the frame-to-frame linking that globally minimizes the sum of the squared displacements.

Read more, including installation instructions and examples, on our [GitHub README page](https://github.com/soft-matter/trackpy).
1.1 trackpy Package

1.1.1 feature Module

A full-featured implementation of the Crocker-Grier algorithm for locating round features in images.

```python
import trackpy as tp
diameter = 5  # estimated size of features
tp.locate(image, diameter)
```

`locate` returns a DataFrame (a spreadsheet-like object) listing the position, mass (total brightness), size (radius-of-gyration of brightness), and eccentricity. It also lists the signal, a measure related the contrast, and ep for epsilon, the estimated uncertainty in the position of the feature.

`locate` prepares the image by performing a band pass using sensible defaults derived from the diameter you specified. You choose your settings or override this preprocessing all together; see the API documentation below.

Then, following the Crocker-Grier procedure, it locates all local maxima, filters very dim maxima away, and refines the remainder to subpixel accuracy by iteratively honing in on their center of brightness.

```
trackpy.locate(raw_image, diameter, minmass=100.0, maxsize=None, separation=None, noise_size=1,
smoothing_size=None, threshold=None, invert=False, percentile=64, topn=None, pre-process=True,
max_iterations=10, filter_before=True, filter_after=True, characterize=True, engine='auto')
```

Locate Gaussian-like blobs of a given approximate size.

Preprocess the image by performing a band pass and a threshold. Locate all peaks of brightness, characterize the neighborhoods of the peaks and take only those with given total brightnesss ("mass"). Finally, refine the positions of each peak.

**image** : image array (any dimensions)  **diameter** : feature size in px  **minmass** : minimum integrated brightness

Default is 100, but a good value is often much higher. This is a crucial parameter for elminating spurious features.

**maxsize** : maximum radius-of-gyration of brightness, default None  **separation** : feature separation, in pixels

Default is the feature diameter + 1.

**noise_size** [width of Gaussian blurring kernel, in pixels] Default is 1.

**smoothing_size** [size of boxcar smoothing, in pixels] Default is the same is feature separation.

**threshold** [Clip bandpass result below this value.] Default None, passed through to bandpass.
invert [Set to True if features are darker than background. False by default.]

percentile [Features must have a peak brighter than pixels in this percentile. This helps eliminate spurrious peaks.]

topn [Return only the N brightest features above minmass. If None (default), return all features above minmass.]

DataFrame([x, y, mass, size, ecc, signal]) where mass means total integrated brightness of the blob, size means the radius of gyration of its Gaussian-like profile, and ecc is its eccentricity (1 is circular).

preprocess : Set to False to turn off bandpass preprocessing. max_iterations : integer
   max number of loops to refine the center of mass, default 10

filter_before [boolean] Use minmass (and maxsize, if set) to eliminate spurrious features based on their estimated mass and size before refining position. True by default for performance.

filter_after [boolean] Use final characterizations of mass and size to eliminate spurrious features. True by default.


engine : {'auto', 'python', 'numba'}
batch : performs location on many images in batch

This is an implementation of the Crocker-Grier centroid-finding algorithm. [1]_

trackpy.batch(frames, diameter, minmass=100, maxsize=None, separation=None, noise_size=1,
   smoothing_size=None, threshold=None, invert=False, percentile=64, topn=None, preprocess=True,
   max_iterations=10, filter_before=True, filter_after=True, characterize=True,
   engine='auto', output=None, meta=True)

Locate Gaussian-like blobs of a given approximate size.

Preprocess the image by performing a band pass and a threshold. Locate all peaks of brightness, characterize the neighborhoods of the peaks and take only those with given total brightnesses (“mass”). Finally, refine the positions of each peak.

frames : list (or iterable) of images diameter : feature size in px minmass : minimum integrated brightness

   Default is 100, but a good value is often much higher. This is a crucial parameter for eliminating spurrious features.

maxsize : maximum radius-of-gyration of brightness, default None separation : feature separation, in pixels

   Default is the feature diameter + 1.

noise_size [width of Gaussian blurring kernel, in pixels] Default is 1.

smoothing_size [size of boxcar smoothing, in pixels] Default is the same as feature separation.

threshold [Clip bandpass result below this value.] Default None, passed through to bandpass.

invert [Set to True if features are darker than background. False by default.]

percentile [Features must have a peak brighter than pixels in this percentile. This helps eliminate spurrious peaks.]

topn [Return only the N brightest features above minmass.] If None (default), return all features above minmass.
DataFrame([x, y, mass, size, ecc, signal]) where mass means total integrated brightness of the blob, size means the radius of gyration of its Gaussian-like profile, and ecc is its eccentricity (1 is circular).

preprocess : Set to False to turn off bandpass preprocessing. max_iterations : integer
max number of loops to refine the center of mass, default 10

filter_before [boolean] Use minmass (and maxsize, if set) to eliminate spurrious features based on their estimated mass and size before refining position. True by default for performance.

filter_after [boolean] Use final characterizations of mass and size to eliminate spurrious features. True by default.


engine : {'auto', 'python', 'numba'} output : {None, trackpy.PandasHDFStore, SomeCustomClass}
If None, return all results as one big DataFrame. Otherwise, pass results from each frame, one at a time, to the write() method of whatever class is specified here.

meta [By default, a YAML (plain text) log file is saved in the current directory. You can specify a different filepath set False.]

locate : performs location on a single image
This is an implementation of the Crocker-Grier centroid-finding algorithm. [1]

These locate doesn’t do exactly what you want, you can dig into the lower- level functions and develop something of your own.

1.1.2 linking Module

Most users will rely only on link_df (for “link DataFrame”) which expects results the format given from locate and batch.

trackpy.link_df (features, search_range, memory=0, neighbor_strategy=u’KDTree’,
link_strategy=u’auto’, predictor=None, hash_size=None, box_size=None,
pos_columns=None, t_column=None, verify_integrity=True, retain_index=False)
Link features into trajectories, assigning a label to each trajectory.

features [DataFrame] Must include any number of column(s) for position and a column of frame numbers. By default, ‘x’ and ‘y’ are expected for position, and ‘frame’ is expected for frame number. See below for options to use custom column names.

search_range [integer] the maximum distance features can move between frames

memory [integer] the maximum number of frames during which a feature can vanish, then reappear nearby, and be considered the same particle. 0 by default.

neighbor_strategy [[‘KDTree’, ‘BTree’]] algorithm used to identify nearby features

link_strategy [[‘recursive’, ‘nonrecursive’, ‘numba’, ‘auto’]] algorithm used to resolve subnetworks of nearby particles ‘auto’ uses numba if available

predictor [function (optional)] Improve performance by guessing where a particle will be in the next frame. For examples of how this works, see the “predict” module.

trajectories [DataFrame] This is the input features DataFrame, now with a new column labeling each particle with an ID number. This is not a copy.
pos_columns [DataFrame column names (unlimited dimensions)] Default is ['x', 'y']

t_column [DataFrame column name] Default is 'frame'

hash_size [sequence] For 'BTree' mode only. Define the shape of the search region. If None (default), infer shape from range of data.

box_size [sequence] For 'BTree' mode only. Define the partition size to optimize performance. If None (default), the search_range is used, which is a reasonable guess for best performance.

verify_integrity [boolean] False by default, for fastest performance. Use True if you suspect a bug in linking.

retain_index [boolean] By default, the index is reset to be sequential. To keep the original index, set to True. Default is fine unless you devise a special use.

We continue to support link which expects trackpy.Point objects as a list of lists.

```
trackpy.link(levels, search_range, hash_generator, memory=0, track_cls=None, neighbor_strategy='BTree', link_strategy='recursive')
```

Link features into trajectories, assigning a label to each trajectory.

This function is deprecated and lacks some recently-added options, thought it is still accurate. Use link_df or link_iter.

levels [iterable of iterables containing Points objects] e.g., a list containing lists with the Points in each frame

search_range [integer] the maximum distance features can move between frames

hash_generator [a function that returns a HashTable] only used if neighbor_strategy is set to 'BTree' (default)

memory [integer] the maximum number of frames during which a feature can vanish, then reappear nearby, and be considered the same particle. 0 by default.

neighbor_strategy [{'BTree', 'KDTree'}] algorithm used to identify nearby features

link_strategy [{'recursive', 'nonrecursive', 'numba', 'auto'}] algorithm used to resolve subnetworks of nearby particles 'auto' uses numba if available

tracks : list of Track (or track_cls) objects

link_df, link_iter

class trackpy.Point

Base class for point (features) used in tracking. This class contains all of the general stuff for interacting with Track objects.

Note: To be used for tracking this class must be sub-classed to provide a distance() function. Child classes MUST call Point.__init__() (See PointND for example.)

class trackpy.PointND (t, pos)

Parameters

- t – a time-like variable.
- pos (iterable of length d) – position of feature

Version of Point for tracking in flat space with non-periodic boundary conditions.

class trackpy.Track (point=None)

Parameters point (Point) – The first feature in the track if not None.

Base class for objects to represent linked tracks. Includes logic for adding, removing features to the track. This can be sub-classed to provide additional track level computation as needed.

6 Chapter 1. Contents:
At the lowest level is `link_iter`, which can return results iteratively and process unlimited streams of data using a fixed amount of memory.

```python
trackpy.link_iter(levels, search_range, memory=0, neighbor_strategy=u'KDTree',
                  link_strategy=u'auto', hash_size=None, box_size=None, predictor=None,
                  track_cls=None, hash_generator=None)
```

Link features into trajectories, assigning a label to each trajectory.

- **levels** [iterable of iterables containing Points objects] e.g., a list containing lists with the Points in each frame
- **search_range** [integer] the maximum distance features can move between frames
- **memory** [integer] the maximum number of frames during which a feature can vanish, then reappear nearby, and be considered the same particle. 0 by default.
- **neighbor_strategy** [{'KDTree', 'BTree'}] algorithm used to identify nearby features
- **link_strategy** [{'recursive', 'nonrecursive', 'numba', 'auto'}] algorithm used to resolve subnetworks of nearby particles ‘auto’ uses numba if available
- **predictor** [function (optional)] Improve performance by guessing where a particle will be in the next frame. For examples of how this works, see the “predict” module.
- **labels** [list of integers] labeling the features in the given level
- **hash_size** [sequence] For ‘BTree’ mode only. Define the shape of the search region. (Higher-level wrappers of link infer this from the data.)
- **box_size** [sequence] For ‘BTree’ mode only. Define the partition size to optimize performance. If None (default), the search_range is used, which is a reasonable guess for best performance.
- **track_cls** [class (optional)] for special uses, you can specify a custom class that holds each Track
- **hash_generator** [function (optional)] a function that returns a HashTable, included for legacy support. Specifying hash_size and box_size (above) fully defined a HashTable.

**class** `trackpy.IndexedPointND` (t, pos, id)

**class** `trackpy.DummyTrack` (point)

Does not store points, thereby conserving memory.

The BTree link strategy uses a hash table that can be fully specified by keyword arguments, but you can also build one yourself.

**class** `trackpy.HashTable` (dims, box_size)

**Parameters**

- **dims** – the range of the data to be put in the hash table. 0<data[k]<dims[k]
- **box_size** – how big each box should be in data units. The same scale is used for all dimensions

Basic hash table to fast look up of particles in the region of a given particle

The KDTree link strategy uses a class that, at this point, is not exposed to the user and shouldn’t need to be, but here it is for completeness:

**class** `trackpy.TreeFinder` (points)
1.1.3 filtering Module

These are various tools for filtering trajectories generated by link_df, designed to help you eliminate spurrious ones.

```python
trackpy.filtering.bust_clusters(tracks, quantile=0.8, threshold=None)
```
Filter out trajectories with a mean particle size above a given quantile.

- **tracks** [DataFrame] must include columns named ‘particle’ and ‘size’
- **quantile** [number between 0 and 1] quantile of particle ‘size’ above which to cut off
- **threshold** [number] If specified, ignore quantile.

```python
trackpy.filtering.bust_ghosts(tracks, threshold=100)
```
Filter out trajectories with few points. They are often specious.

- **tracks** [DataFrame] must include columns named ‘frame’ and ‘particle’
- **threshold** [integer, default 100] minimum number of points to survive

```python
trackpy.filtering.filter(tracks, condition_func)
```
A workaround for a bug in pandas 0.12

- **tracks** [DataFrame] must include column named ‘particle’
- **condition_func** [function] The function is applied to each group of data. It must return True or False.

```python
Dataframe
```

```python
trackpy.filtering.filter_clusters(tracks, quantile=0.8, threshold=None)
```
Filter out trajectories with a mean particle size above a given quantile.

- **tracks** [DataFrame] must include columns named ‘particle’ and ‘size’
- **quantile** [number between 0 and 1] quantile of particle ‘size’ above which to cut off
- **threshold** [number] If specified, ignore quantile.

```python
trackpy.filtering.filter_stubs(tracks, threshold=100)
```
Filter out trajectories with few points. They are often specious.

- **tracks** [DataFrame] must include columns named ‘frame’ and ‘particle’
- **threshold** [integer, default 100] minimum number of points to survive

1.1.4 motion Module

These are various tools for characterizing trajectories generated by link_df.

```python
trackpy.motion.compute_drift(traj, smoothing=0)
```
Return the ensemble drift, \( x(t) \).

- **traj** : DataFrame of trajectories, including columns x, y, frame, and particle smoothing : integer
  Smooth the drift using a forward-looking rolling mean over this many frames.
drift : DataFrame([x, y], index=frame)

compute_drift(traj).plot() # Default smoothing usually smooths too much. compute_drift(traj, 0).plot() # not smoothed
compute_drift(traj, 15).plot() # Try various smoothing values.

drift = compute_drift(traj, 15) # Save good drift curves. corrected_traj = subtract_drift(traj, drift) # Apply them.

trackpy.motion.diagonal_size (single_trajectory, pos_columns=['x', 'y'], t_column='frame')
Measure the diagonal size of a trajectory.

single_trajectory : DataFrame containing a single trajectory pos_columns = ['x', 'y'] t_column = 'frame'

float : length of diagonal of rectangular box containing the trajectory

>>> diagonal_size(single_trajectory)

>>> many_trajectories.groupby('particle').agg(tp.diagonal_size)

>>> many_trajectories.groupby('particle').filter(lambda x: tp.diagonal_size(x) > 5)

trackpy.motion.direction_corr(t, frame1, frame2)
Compute the cosine between every pair of particles’ displacements.

t : DataFrame containing columns particle, frame, x, and y frame1 : frame number frame2 : frame number

DataFrame, indexed by particle, including dx, dy, and direction

trackpy.motion.emsd (traj, mpp, fps, max_lagtime=100, detail=False)
Compute the mean squared displacements of an ensemble of particles.

traj [DataFrame of trajectories of multiple particles, including] columns particle, frame, x, and y

mpp : microns per pixel fps : frames per second max_lagtime : intervals of frames out to which MSD is computed

Default: 100

detail [Set to True to include <x>, <y>, <x^2>, <y^2>. Returns] only <r^2> by default.

Series[msd, index=t] or, if detail=True, DataFrame([<x>, <y>, <x^2>, <y^2>, msd], index=t)

Input units are pixels and frames. Output units are microns and seconds.

trackpy.motion.imsd(traj, mpp, fps, max_lagtime=100, statistic='msd')
Compute the mean squared displacements of particles individually.

traj [DataFrame of trajectories of multiple particles, including] columns particle, frame, x, and y

mpp : microns per pixel fps : frames per second max_lagtime : intervals of frames out to which MSD is computed

Default: 100

statistic [{'msd', '<x>', '<y>', '<x^2>', '<y^2>'}, default is ‘msd’] The functions msd() and emsd() return all these as columns. For imsd() you have to pick one.

DataFrame([Probe 1 msd, Probe 2 msd, ...], index=t)

Input units are pixels and frames. Output units are microns and seconds.

trackpy.motion.is_typical (msds, frame=23, lower=0.1, upper=0.9)
Examine individual particle MSDs, distinguishing outliers from those in the central quantile.
**msds** [DataFrame like the output of imsd()]: Columns correspond to particles, indexed by lagtime measured in frames.

**frame** [integer frame number]: Compare MSDs at this lagtime. Default is 23 (1 second at 24 fps).

**lower** [float between 0 and 1, default 0.1]: Probes with MSD up to this quantile are deemed outliers.

**upper** [float between 0 and 1, default 0.9]: Probes with MSD above this quantile are deemed outliers.

Series of boolean values, indexed by particle number True = typical particle, False = outlier particle

\[ m = \text{tp.imsd}() \# \text{Index by particle ID, slice using boolean output from is_typical(), and then} \]
\[ \text{# restore the original index, frame number. typical_traj = traj.set_index('particle').ix[is_typical(m)].reset_index()} \]

\[ \text{typical_traj} = \text{traj.set_index('particle').ix[is_typical(m)].reset_index()}.set_index('frame', drop=False) \]

**trackpy.motion.min_rolling_theta_entropy(pos, window=24, bins=24)**
Compute the minimum Shannon entropy in any window.

**pos** : DataFrame with columns x and y, indexed by frame window : number of observations per window bins : number of equally-spaced bins in distribution. Default 24.

**float** : Shannon entropy

```python
>>> theta_entropy(t[t['particle'] == 3].set_index('frame'))
```

**trackpy.motion.msd(traj, mpp, fps, max_lagtime=100, detail=False)**
Compute the mean displacement and mean squared displacement of one trajectory over a range of time intervals.

**traj** : DataFrame with one trajectory, including columns frame, x, and y **mpp** : microns per pixel **fps** : frames per second **max_lagtime** : intervals of frames out to which MSD is computed

Default: 100

detail : See below. Default False.

DataFrame([<x>, <y>, <x^2>, <y^2>, msd], index=t)

If detail is True, the DataFrame also contains a column N, the estimated number of statistically independent measurements that comprise the result at each lagtime.

Input units are pixels and frames. Output units are microns and seconds.

**trackpy.motion.proximity(features, pos_columns=[u'x', u'y'])**

Find the distance to each feature’s nearest neighbor.

**features** : DataFrame **pos_columns** : list of column names

['x', 'y'] by default

**proximity** [DataFrame] distance to each particle’s nearest neighbor, indexed by particle if ‘particle’ column is present in input

Find the proximity of each particle to its nearest neighbor in every frame.

```python
>>> prox = t.groupby('frame').apply(proximity).reset_index()
```

```python
>>> avg_prox = prox.groupby('particle')['proximity'].mean()
```

And filter the trajectories...
trackpy Documentation, Release unknown

```python
>>> particle_nos = avg_prox[avg_prox > 20].index
>>> t_filtered = t[t['particle'].isin(particle_nos)]
```

trackpy.motion.shannon_entropy(x, bins)
Compute the Shannon entropy of the distribution of x.

trackpy.motion.subtract_drift(traj, drift=None)
Return a copy of particle trajectories with the overall drift subtracted out.

- traj : DataFrame of trajectories, including columns x, y, and frame drif : optional DataFrame([x, y], index=frame) like output of
  compute_drift(). If no drift is passed, drift is computed from traj.
- traj : a copy, having modified columns x and y

trackpy.motion.theta_entropy(pos, bins=24, plot=True)
Plot the distribution of directions and return its Shannon entropy.

- pos : DataFrame with columns x and y, indexed by frame bins : number of equally-spaced bins in distribution. Default 24. plot : plot direction histogram if True
- float : Shannon entropy

```python
>>> theta_entropy(t[t['particle'] == 3].set_index('frame'))
```

```python
>>> S = t.set_index('frame').groupby('particle').apply(tp.theta_entropy)
```

trackpy.motion.vanhove(pos, lagtime=23, mpp=1, ensemble=False, bins=24)
Compute the van Hove correlation function at given lagtime (frame span).

- pos : [DataFrame of x or (or!) y positions, one column per particle, indexed] by frame
- lagtime : [integer interval of frames] Compare the correlation function at this lagtime. Default is 23 (1 second at 24 fps).
- mpp : [microns per pixel, DEFAULT TO 1 because it is usually fine to use] pixels for this analysis
- ensemble : boolean, defaults False bins : integer or sequence
  Specify a number of equally spaced bins, or explicitly specify a sequence of bin edges. See np.histogram docs.

- vh : [If ensemble=True, a DataFrame with each particle’s van Hove correlation] function, indexed by displacement. If ensemble=False, a Series with the van Hove correlation function of the whole ensemble.

pos = traj.set_index(['frame', 'particle'])[x].unstack() # particles as columns vh = vanhove(pos)

trackpy.motion.velocity_corr(t, frame1, frame2)
Compute the velocity correlation between every pair of particles’ displacements.

- t : DataFrame containing columns particle, frame, x, and y frame1 : frame number frame2 : frame number
- DataFrame, indexed by particle, including dx, dy, and direction

1.1.5 plots Module

These functions generate handy plots.

trackpy.plots.annotate(*args, **kwargs)
Mark identified features with white circles.
centroids : DataFrame including columns x and y
image : image array (or string path to image file)
circle_size : Deprecated.

This will be removed in a future version of trackpy. Use plot_style={'markersize': ...} instead.

color [single matplotlib color or a list of multiple colors] default None
invert [If you give a filepath as the image, specify whether to invert] black and white. Default True.

ax : matplotlib axes object, defaults to current axes
split_category : string, parameter to use to split the data into sections

default None

split_thresh [single value or list of ints or floats to split ] particles into sections for plotting in multiple colors.
List items should be ordered by increasing value. default None

imshow_style [dictionary of keyword arguments passed through to] the Axes.imshow(...) command the displays the image

plot_style [dictionary of keyword arguments passed through to] the Axes.plot(...) command that marks the features

axes

trackpy.plots.make_axes(func)
A decorator for plotting functions. NORMALLY: Direct the plotting function to the current axes, gca().
When it's done, make the legend and show that plot. (Instant gratification!)

BUT: If the uses passes axes to plotting function, write on those axes and return them. The user has the option to draw a more complex plot in multiple steps.

trackpy.plots.make_fig(func)
See make_axes.

trackpy.plots.mass_ecc(*args, **kwargs)
Plot each particle’s mass versus eccentricity.

trackpy.plots.mass_size(*args, **kwargs)
Plot each particle’s mass versus size.

trackpy.plots.plot_displacements(*args, **kwargs)
Plot arrows showing particles displacements between two frames.
    t [DataFrame] trajectories, including columns ‘frame’ and ‘particle’
    frame1 [integer] frame number
    frame2 [integer] frame number
    scale [float] scale factor, if 1 (default) then arrow end is placed at particle destination; if any other number arrows are rescaled
    ax : matplotlib axes (optional)
Any other keyword arguments will pass through to matplotlib’s annotate.

trackpy.plots.plot_principal_axes(*args, **kwargs)
Plot bars with a length of 2 stddev along the principal axes.

This function is based on a solution by Joe Kington, posted on Stack Overflow at http://stackoverflow.com/questions/5869891/how-to-calculate-the-axis-of-orientation/5873296#5873296
trackpy.plot_traj(*args, **kwargs)
Plot traces of trajectories for each particle. Optionally superimpose it on a frame from the video.

traj : DataFrame including columns x and y
colorby : {'particle', 'frame'}
mpp : microns per pixel

If omitted, the labels will be labeled in units of pixels.

label : Set to True to write particle ID numbers next to trajectories.
superimpose : background image, default None

cmap : This is only used in colorby='frame' mode.

Default = mpl.cm.winter

ax : matplotlib axes object, defaults to current axes

t_column : DataFrame column name

Default is 'frame'

None

trackpy.plot.ptraj(*args, **kwargs)
Plot traces of trajectories for each particle. Optionally superimpose it on a frame from the video.

traj : DataFrame including columns x and y
colorby : {'particle', 'frame'}
mpp : microns per pixel

If omitted, the labels will be labeled in units of pixels.

label : Set to True to write particle ID numbers next to trajectories.
superimpose : background image, default None

cmap : This is only used in colorby='frame' mode.

Default = mpl.cm.winter

ax : matplotlib axes object, defaults to current axes

t_column : DataFrame column name

Default is 'frame'

None

trackpy.plot.subpx_bias(*args, **kwargs)
Histogram the fractional part of the x and y position.

If subpixel accuracy is good, this should be flat. If it depressed in the middle, try using a larger value for feature diameter.
CHAPTER 2

Indices and tables

- genindex
- modindex
- search
t

trackpy.__init__.3
trackpy.filtering.8
trackpy.motion.8
trackpy.plots.11
| A      | annotate() (in module trackpy.plots), 11 |
| B      | batch() (in module trackpy), 4          |
|        | bust_clusters() (in module trackpy.filtering), 8 |
|        | bust_ghosts() (in module trackpy.filtering), 8 |
| C      | compute_drift() (in module trackpy.motion), 8 |
|        | diagonal_size() (in module trackpy.motion), 9 |
|        | direction_corr() (in module trackpy.motion), 9 |
|        | DummyTrack (class in trackpy), 7         |
| E      | emsd() (in module trackpy.motion), 9      |
| F      | filter() (in module trackpy.filtering), 8 |
|        | filter_clusters() (in module trackpy.filtering), 8 |
|        | filter_stubs() (in module trackpy.filtering), 8 |
| H      | HashTable (class in trackpy), 7          |
| I      | imsd() (in module trackpy.motion), 9      |
|        | IndexedPointND (class in trackpy), 7      |
|        | is_typical() (in module trackpy.motion), 9 |
| L      | link() (in module trackpy), 6             |
|        | link_dil() (in module trackpy), 5         |
|        | link_iter() (in module trackpy), 7        |
|        | locate() (in module trackpy), 3           |
| M      | make_axes() (in module trackpy.plots), 12 |
|        | make_fig() (in module trackpy.plots), 12  |
|        | mass_ecc() (in module trackpy.plots), 12  |
|        | mass_size() (in module trackpy.plots), 12 |
|        | min_rolling_theta_entropy() (in module trackpy.motion), 10 |
|        | msd() (in module trackpy.motion), 10       |
| P      | plot_displacements() (in module trackpy.plots), 12 |
|        | plot_principal_axes() (in module trackpy.plots), 12 |
|        | plot_traj() (in module trackpy.plots), 12  |
|        | Point (class in trackpy), 6               |
|        | PointND (class in trackpy), 6              |
|        | proximity() (in module trackpy.motion), 10 |
|        | ptraj() (in module trackpy.plots), 13      |
| S      | shannon_entropy() (in module trackpy.motion), 11 |
|        | subpx_bias() (in module trackpy.plots), 13 |
|        | subtract_drift() (in module trackpy.motion), 11 |
| T      | theta_entropy() (in module trackpy.motion), 11 |
|        | Track (class in trackpy), 6               |
|        | trackpy.__init__ (module), 3              |
|        | trackpy.filtering (module), 8             |
|        | trackpy.motion (module), 8                |
|        | trackpy.plots (module), 11                |
|        | TreeFinder (class in trackpy), 7          |
| V      | vanhove() (in module trackpy.motion), 11   |
|        | velocity_corr() (in module trackpy.motion), 11 |