aquatron_ptv_lowlight

April 15, 2019

1 Floating cork tracking with trackpy

This notebook is adapted from the trackpy walkthrough found here.

```
In [156]: from __future__ import division, unicode_literals, print_function  # for compatibili

          import matplotlib.pyplot as plt
          import numpy as np
          import pandas as pd
          from pandas import DataFrame, Series  # for convenience

          # packages for image manipulation and tracking (will need to be installed on local m
          import pims
          import trackpy as tp
```

Read in image sequence. The images should be preprocessed so the features to be tracked are light or dark relative to the background. Some preprocessing can be done within the trackpy environment, though I haven't really played around with this. The trackpy package is also capable of reading raw video, but this requires a bit more effort, and some extra packages.

```
In [157]: # frames = pims.ImageSequence(r'C:\Projects\Aquatron2019_April\data\interim\April11\
          frames = pims.ImageSequence(r'C:\Projects\Aquatron2019_April\data\interim\April11\ima
```

Sample image:

```
In [158]: testframe = frames[154]
```

Try locating features in a single image. The only required inputs are an image file, and a feature pixel diameter estimate (must be odd). If the features to be tracked are light/white, set invert=True. The trackpy tutorial suggests erring on the large side when estimating feature diameter. Try `tp.locate?` to see other optional inputs.

```
In [180]: d_cork = 11 # estimated diameter of tracked features
          f0 = tp.locate(testframe, d_cork, invert=False)
```

Trackpy produces a pandas dataframe that contains the data. Use the `head()` function to take a truncated look.

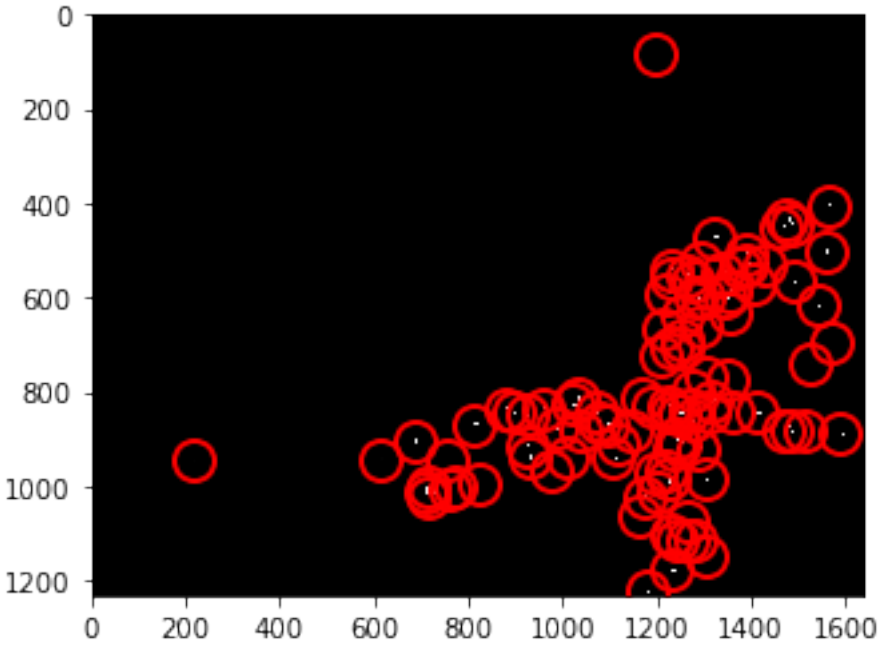
```
In [181]: f0.head() # shows the first few rows of data
```

Out[181]:

	y	x	mass	size	ecc	signal	raw_mass	ep	frame
0	83.322741	1196.672282	2061.600659	1.734591	0.164757	175.220270	3082.0	0.000231	154
3	404.659424	1563.332890	5339.482564	2.602177	0.112111	157.066819	11710.0	0.000061	154
4	434.074685	1475.420321	3635.425970	2.538886	0.330987	181.534514	6648.0	0.000107	154
5	445.759083	1483.814638	4475.220419	2.240205	0.088080	179.166673	8933.0	0.000080	154
6	447.972414	1465.078749	4921.163900	2.374922	0.119695	169.695307	10214.0	0.000070	154

Use the `annotate()` function to overlay the identified features on the image. It probably won't look great the first time around.

In [182]:
```
plt.figure()  # make a new figure
tp.annotate(f0, testframe)
```



Out[182]: `<matplotlib.axes._subplots.AxesSubplot at 0x2870dafffd0>`

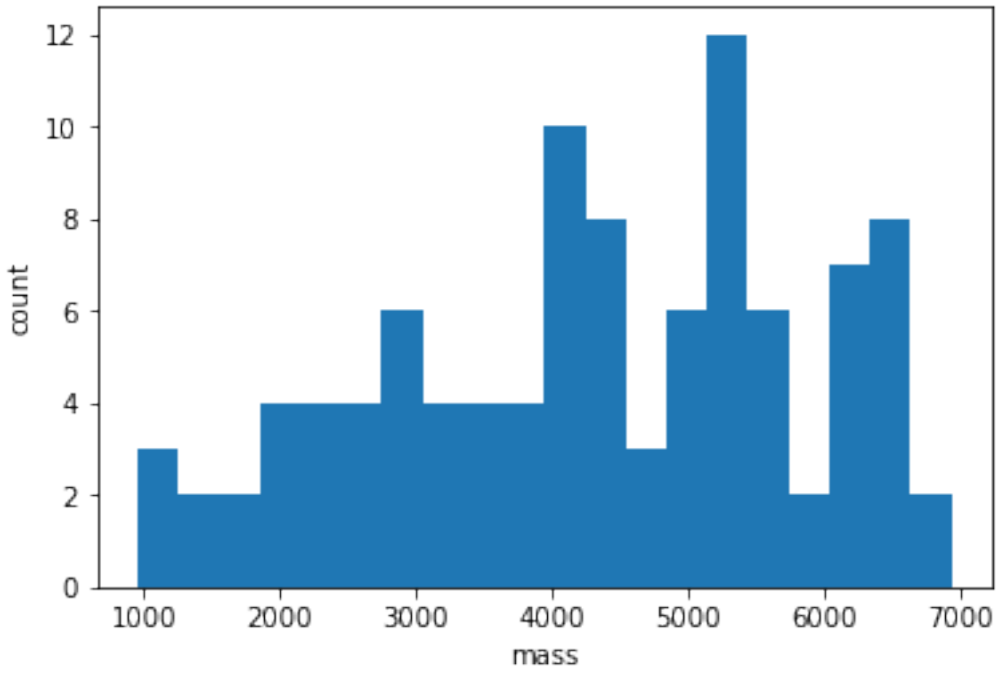
Plot a histogram of the "mass" of the features to be tracked. The mass is a measure of the total pixel intensity associated with each feature. Since I've already segmented out the corks into binary structures, the mass checking step is not all that useful. If I hadn't done this preprocessing step, the corks might be expected to have a characteristic mass, with a distribution that would be

identifiable in the histogram. One could then define a lower mass cutoff to eliminate features that aren't of interest.

```
In [183]: fig, ax = plt.subplots()
          ax.hist(f0['mass'], bins=20)

          # Optionally, label the axes.
          ax.set(xlabel='mass', ylabel='count')
```

```
Out[183]: [Text(0, 0.5, 'count'), Text(0.5, 0, 'mass')]
```

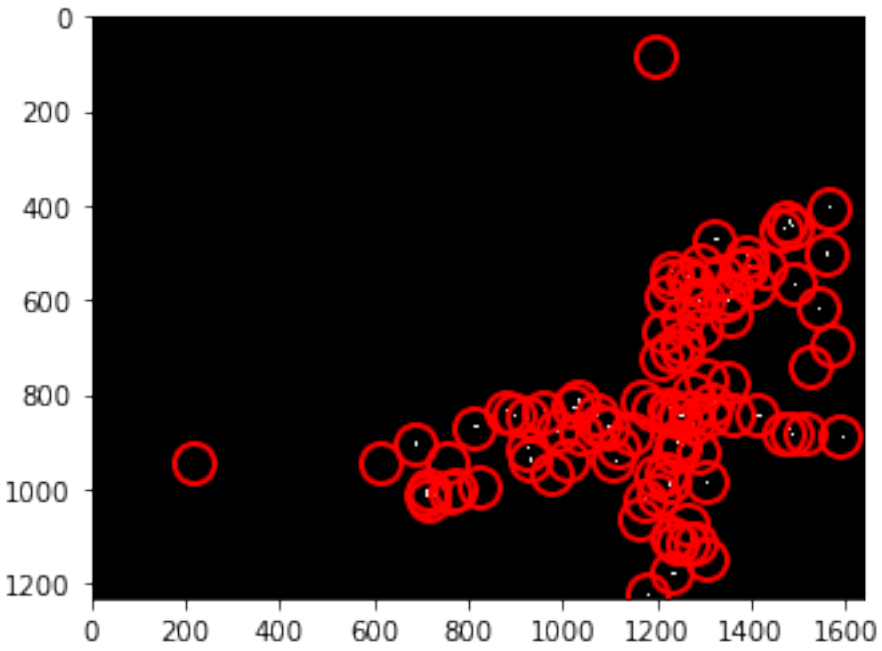


Here, I pick a cutoff of 0, since I'm interested in all the information that remains in the image.

```
In [184]: f0 = tp.locate(testframe, d_cork, invert=False, minmass=0)
```

Try overlaying data on image again. If any cutoff has been applied, the location estimates should be improved.

```
In [185]: plt.figure()
          tp.annotate(f0, testframe)
```



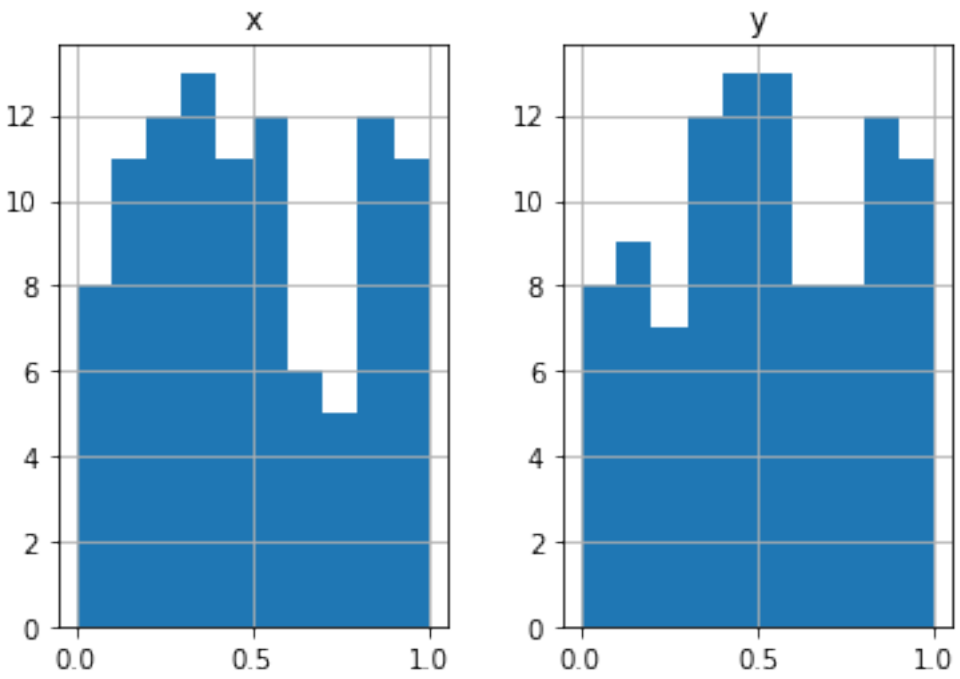

```
Out[185]: <matplotlib.axes._subplots.AxesSubplot at 0x2870dc1b470>
```

To check that the diameter of the kernel used is not too small, trackpy includes a plotting function for evaluating subpixel bias. If the kernel is too small, the x- and y-coordinates will be predominantly integer-valued. Look for a uniform (flat) distribution of decimal values in the x- and y-coordinate histograms. If the histograms display a "dip" in the middle, try using a larger value for feature diameter.

```
In [186]: plt.figure()
          tp.subpx_bias(f0)

Out[186]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000028711D06630>,
                  <matplotlib.axes._subplots.AxesSubplot object at 0x0000028711D4A048>]],
                dtype=object)

<Figure size 432x288 with 0 Axes>
```



Once satisfied with the quality of feature tracking, process the entire batch of images using the settings that were tuned earlier. This step might take a while if processing many images.

```
In [172]: f = tp.batch(frames[150:300], d_cork, minmass=0, invert=False)

Frame 299: 115 features
```

The `link_df` function takes a dataframe of feature locations, and returns a new dataframe with trajectory information. The only other required input is an integer pixel value defining how far a feature is allowed to travel between time steps. The `memory` option defines how many time steps a feature can be out of sight before being considered a different particle after reappearing.

```
In [173]: t = tp.link_df(f, 9, memory=5)

Frame 299: 115 trajectories present.
```

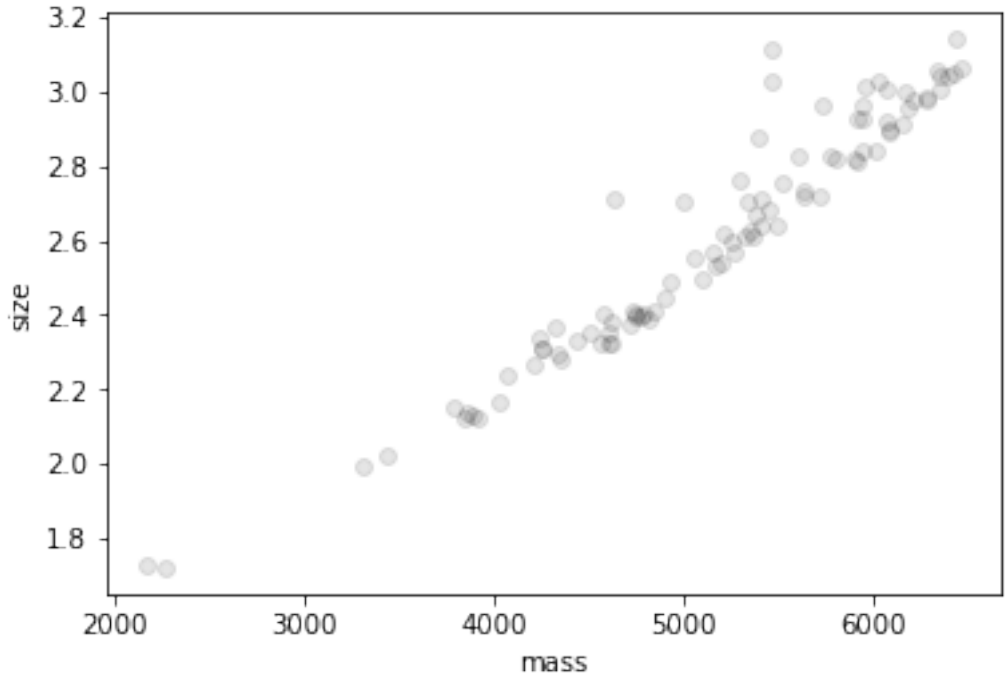
For this image set, the `filter_stubs` function is the most important noise-reducing step. It filters out trajectories that persist for fewer than the defined number of time steps (in this case, 50). This is effective for removing any remaining artefacts of light reflection from the water's surface.

```
In [175]: t1 = tp.filter_stubs(t, 50)
          # Compare the number of particles in the unfiltered and filtered data.
          print('Before:', t['particle'].nunique())
          print('After:', t1['particle'].nunique())
```

```
Before: 1765
After: 93
```

Filtering by appearance, e.g. in size-mass space, is also possible. This is generally a trial and error process. See the trackpy walkthrough for more discussion of appearance filtering.

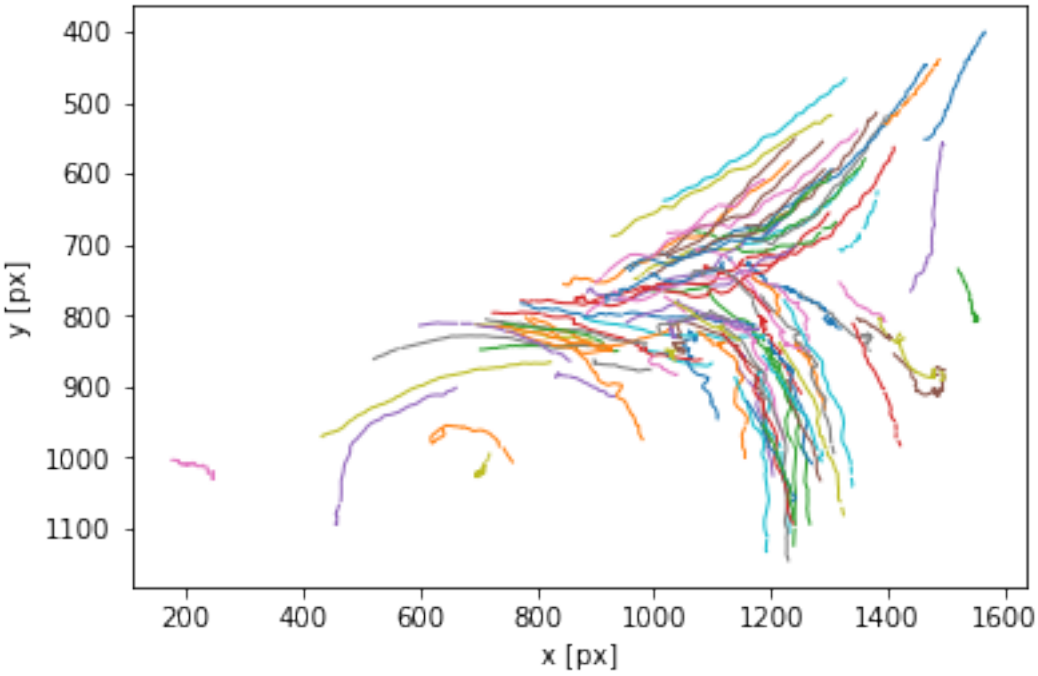
```
In [206]: plt.figure()
          tp.mass_size(t1.groupby('particle').mean())  # convenience function -- just plots si
```




```
Out[206]: <matplotlib.axes._subplots.AxesSubplot at 0x28714524d68>
```

If all the desired filtering of trajectory data is complete, we can plot the trajectories.

```
In [207]: plt.figure()
          tp.plot_traj(t1, colorby='particle')
```



Out[207]: <matplotlib.axes._subplots.AxesSubplot at 0x287147722e8>