Documentation MATLAB to Python, RP Optical Tweezers Round 15

Stijn van der Lippe, Stein Glastra, Jeroen Sangers and Reinaart van Loon

We elaborate upon the issues we encountered in converting the MATLAB code of step 8 in GUI7_MDTIF to Python. Furthermore we elaborate upon some changes we made in the implementation of the code in Python and explain what some of the functions we added do.

Documentation step 8: QI tracker
The part of the code we were stuck on implementing is the interpolation of a non-rectangular grid. Several functions were tested in Python. Below we discuss why they do not give the results we want.

scipy.interpolate.griddata:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.griddata.html
This function has to perform a triangulation for every data point in the input array to find the nearest neighbour if the input array is irregular (as is the case for us). This happens every iteration. This is extremely slow and therefore this function is unusable. More details can be found here.

scipy.interpolate.RectBivariateSpline:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.RectBivariateSpline.html
This creates a class that takes in two 1D arrays of coordinates and a single 2D array of values at the coordinates.
Calling this class using *.ev()* allows the interpolation at data points using 1D arrays of x and y coordinates. Calling the class normally can't work since it needs an array of points of all (x,y) coordinates we want to evaluate at, in ascending order which is something we can't do since we can't sort our points in such a way that the (x,y) coordinates stay linked.

scipy.interpolate.interp2d:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.interp2d.html
This function won't work since it only works with 1D arrays for x and y and outputs a 2D array of interpolated values which would be easy to use if the grid was rectangular, but it isn't so it would take multiple nested for loops which will be slow and error prone. And even then we get an array of outputs that assumes a rectangular grid and would therefore be distorted.

scipy.ndimage.map_coordinates:
https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.map_coordinates.html
This function is promising but currently yields some strange results which we do not yet fully understand and can't really be tested until the previous code works. In our implementation, the function only returns integer values and we are unsure what the cause of this is.

Documentation subpix_step
In MATLAB an array [~,a] was created to get the value and index of the largest entry in the d-array, in Python this had to be substituted with *np.argmax()* to get only the index.

Documentation SymCenter
In MATLAB, NaNs in the input array were first located using a find function. Subsequently, they were padded using the value *nanmean* of the input array. In Python, this can be done in a single step using *np.where*. This was done by using the truth statement *np.isnan*. If the statement is true for an array element, the value of that element was replaced with *np.nanmean* of the input array.

Documentation get_eval_coords
A function that takes in a 2D array of coordinates and returns a flattened 1D array of these coordinates. It is used in the function *track_xy*.

Documentation rebuild_grid
A function that takes in a 1D array of points and a shape. The function then builds a 2D array with the same dimensions as shape. It is used in the function *track_xy*.

Documentation QI_tracker
A small error was found in the *__init__* function of *QI_Tracker*. In the variable assignment of *self.X0samplinggrid* and *self.Y0samplinggrid* both arrays were not transposed, while they should be. This was fixed by adding *.T* at the end of both.

removeNoise
We suspect the long duration of the calculation can be attributed to the time the noise removal takes. We tried to speed this process up by replacing an iteration function (*next*) with a numpy *np.where* function, which is not iterative.
The following line was replaced:
xIhalf = next(x for x, val in enumerate(ilstflat_t)
with
xIhalf = np.where(ilstflat_t>Ihalf)[0][0].
This had little influence on the speed of the process.

The following line was also replaced:
xmindist = next(x for x, val in enumerate(dist) if val == min(dist))
with
xmindist = np.argmin(dist)
This had a major impact on the speed, removeNoise now takes less time to run.

Trackpy
http://soft-matter.github.io/trackpy/v0.4.2/tutorial/walkthrough.html

Trackpy is a python package which is used for particle tracking. It proves to be easy in use and promises accurate results (article about accuracy: https://www.cell.com/biophysj/fulltext/S0006-3495(05)73136-2).
Using this Python function instead of the complicated QI-tracking would potentially not only make a more compact and structured code, it also showed some promising results in comparison with the MATLAB code. For the ‘Bead’ data set that we were given, the MATLAB code fails to track the bead. This is a result of large jumps in space of the bead in the image which cannot be followed by the QI-tracker. The trackpy function, however, seems to have no trouble following the bead whatever the size of the ‘jumps’.
Another advantage is that the trackpy function provides the error in position for each independent frame. Therefore, instead of guessing the error for the QI-tracker, exact error calculation can be performed.