User Manual Matlab Tracking Object

Pepijn Moerman
(Dated: April 18, 2018)

In short the code is operated as follows: 1) Open the code *particles.m*. 2) Create a tracking object by typing *MyObject=particles*. 3) Subsequently find and track particles in your movie by typing *MyObject=find_pos(MyObject)* and *MyObject=track_obj(MyObject)*. 4) Calculate parameters of your interest by running any of the other codes in *particles.m*. A more detailed description of the matlab object follows below.

Introduction

Many experiments in colloidal science involve bright-field of fluorescence microscopy to visualize the colloidal system under investigation. After this visualization a quantitative analysis is often looked for. This starts with finding the particle location in each frame and connecting the particle locations in between frames to form trajectories. After that manipulations of the particle trajectories allow one to measure useful parameters such as the Mean Squared Displacement, aggregation number, particle velocity in the case of active particles, or joint flexibility in the case of flexible clusters.

The aim of this manual is to guide you through a piece of Matlab code that first and foremost does particle tracking. On top of that it also contains a variety of functions that manipulate the obtained trajectories such that useful parameters can be calculated and graphs produced. You are encouraged to play with these functions to extract the information you look for, but this manual does not go into detail about them. A list of all functionalities of the matlab object will be provided below.

Installation

The concommittant matlab code works for all versions of Matlab after 2016b, so make sure your matlab is up to date. Then make sure all the codes that come with this object are placed in the same folder. In matlab, make that folder the current directory and you are ready to start.

Particle tracking

The input for this code is an 8-bit tiffstack of 2 dimensional images. If your raw data are of a different format (.avi for example), use a program like ImageJ (or Fiji) to convert to tiffstack. You can start tracking by typing

MyObject=particles;

This will open the current directory and ask you to click on the tiffstack you wish to analyze. Alternatively you can type

MyObject=particles(filename)

where filename is the full path to your tiffstack. Doing so will create an object of type *particles* with name *MyObject*. If you look into it you will find that the name of your tiffstack and the directory in which it is located, are saved in the top two elements of the object. Moreover the properties *scalebar* and *framerate* are 1. You can change this to the scalebar and framerate of your choosing by typing

MyObject=change_scale(MyObject,4);

MyObject=change_framerate(MyObject,5);

which will change the scale to 4 $\mu m/s$ and the framerate to 5 fps. The semicolon after the statement just represses output, but is not required. To find the positions of your particles, you can type

MyObject=find_pos(MyObject);

This will open call an interactive function to 1) optimize the parameters for particle finding and 2) find the particle positions in all frames of your tiffstack. The function will ask you estimates of particle size and low and high bandpassfilter values. It will then go through two image pre-processing steps (bandpassfilter and gaussian blur) and then use a gaussian blob finding algorighm to find the locations of your particles. All these steps use code by Crocker and Grier [1]. They also wrote an online tutorial on particle tracking that you are highly recommended to follow if this is new to you.

Provide the function with the numbers it asks for and it will return you a processed version of the first image in your stack after every number you feed it. Keep doing this until you are happy with the processing and keep in mind that the code looks for bright gaussian blobs on a dark background. After a while the code will show you red cirlces around the particles it found. Keep feeding it numbers until it finds all the particles and nothing but the particles. Then you will have optimized the parameters and you can start applying the code to all other frames. Examples of well and poorly chosen parameters are provided in Figure 1.

The code will automatically run through all the images and find the positions with the parameters you indicated to be optimal. In the end, it checks for pixel bias by plotting the residual pixel value. If tracking worked well, you expect your particles to have an equal probability to be at whole pixel values or in between them. If this is the case, the histogram shown in the end is flat (see Figure

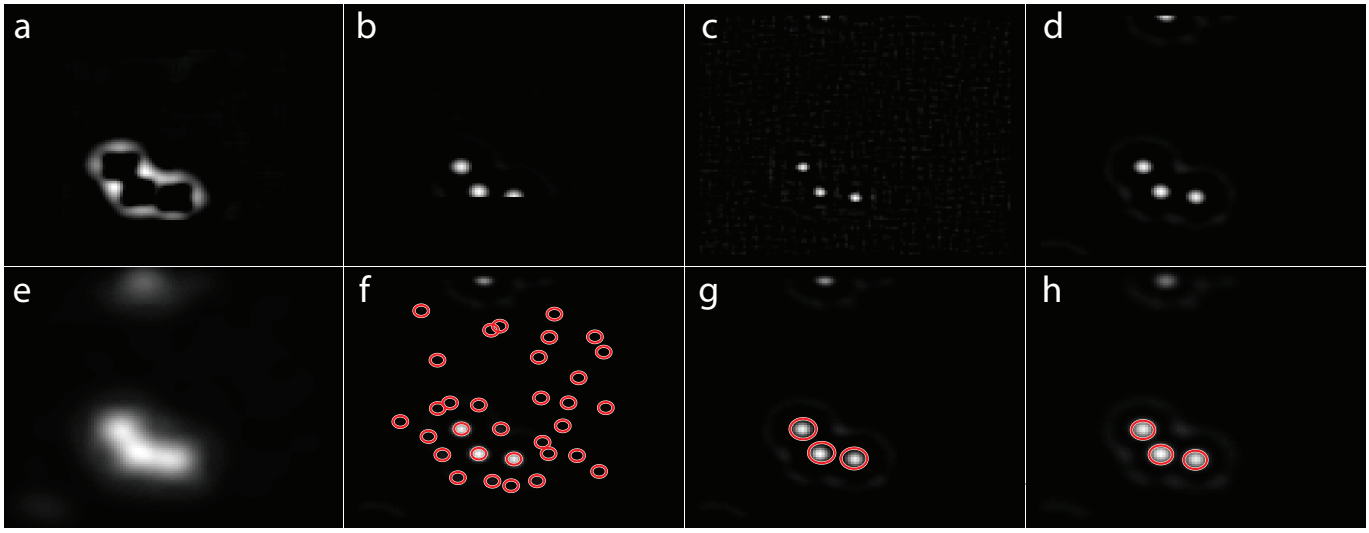

FIG. 1. Parameter optimization for particle finding. Examples shown are of a three-particle cluster of 2 μm Silica particles imaged with a brightfield image. a) Lowpass of bandpass chosen was chosen too large. The features of interest disappear. b) Highpass of bandpass was chosen too large. The border of image moves inward by the highpass length and overlaps with the features of interest. c) The highpass of bandpass was chosen too small. The features have a low intensity with respect to the noise. d) Lowpass and highpass in bandpass are chosen appropiately. e) Gaussian filter length was chosen too large and features overlap. f) Threshold for feature finding was chosen too small and many features are found that are not particles. g) Feature size was chosen too large and the centers of the particles are not found at the correct position. h) Feature size and threshold are chosen optimally.

2a). However, if you do have pixel bias, there is a dip in the histogram and particles are more likely found on whole pixels than in between them (see Figure 2b). This is unphysical and needs to be resolved. Try running the code again but with larger featuresize. This usually does the trick.

After running this piece of code, the properties *pos* and *posparam* will have filled up. The first contains a table with all particle positions in each frame. The second is a list of the parameters you used to find the particles. This list can come in handy if you want to track another movie of a similar system. Likely it will do well with similar parameters.

Next, you need to connect the particle positions into a trajectory. For this the tracking code by Crocker and Grier is used. You can call on that code by typing

MyObject=track_obj(MyObject);

This will again open an interactive function that asks you for some parameters. The max displacement is the maximum amount of pixels one particle can move in one frame before it is considered a different particle. Choose this high enough so that your positions can be connected into tracks, but low enough that the code does not mistake different particles for the same. The number of frames it can go missing asks for exactly that. If you fill in a 0 there, that means that every frame in which you did not find your particle will break the trajectory in two smaller pieces. The minimum number of frames for a track to be accepted allows you to give a minimum value. All tracks shorter than that value will be disregarded.

You can plot the trajectories you found using the function

plot_tracks(MyObject);

This will create an image with trajectories on top of the first image of your tiffstack. It will look something like Figure 3. You have now found the trajectories of your particles and can start calculating the parameters of your interest. But first it pays to make sure the trajectories you found are correct and there are a couple of codes in the matlab object to help you with that.

Checking and optimization

An important thing to check is that the trajectory of one particle is not split into multiple smaller trajectories, for example because you failed to find its position in one of the frames. Especially for longer movies this is a frequent issue. If such mistrack occurs, you can either try finding particle positions again with different parameters until you find particles in each frame, or you can try tracking again with different parameters. In Figure 3 particle 1 and 4 are actually the same particle, but the trajectory got split. If none of the previous tricks work, there is also a function that glues two smaller trajectories together into one big one:

MyObject=merge_tracks(obj,1,4);

This code merges trajectories of particle 1 and 4 into

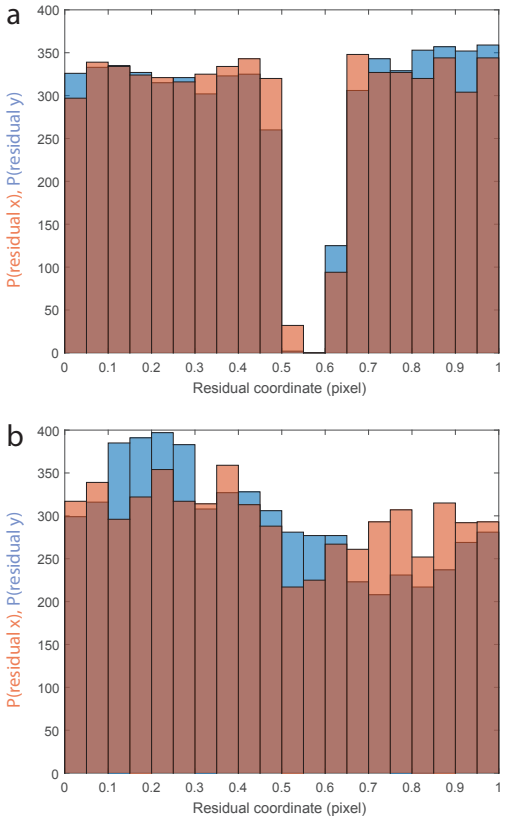

FIG. 2. Decimal number of particle position in x (orange) and y (blue). a) No particle centers are found in between two pixels, which is statistically unlikely. This indicates pixel bias. b) Particle centers are found at any distance from pixel centers, so no pixel bias is present.

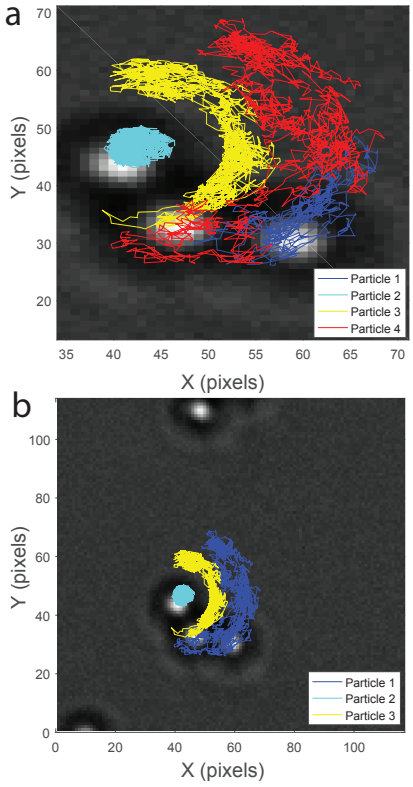

FIG. 3. Example the trajectories obtained after running the tracking code on a tiffstack of a trimer of 2 μm Silica particles. a) Trajectories before merging. Note that particle 1 and particle 4 are actually the same particle. b) Trajectories after merging. Note that particle 2 is stuck to the glass and does not move appreciably.

one trajectory. We obtain the trajectories in Figure 4. Another problem that frequently occurs is the tracking of dirt or stuck particles. You might want to remove these trajectories from the object before you continue with the analysis. This can be done by using the code

MyObject2=remove_stagnant_particles(MyObject)

It is adviced to rename the object to which you apply this code (for example to MyObject2), because removing particles is irreversible. Once you are content with the obtained trajectories, you can use this piece of code to calculate or plot a number of derived quantities

Derived quantities

Below follows a list of all codes that are available to the object *particles*. If you work with particle chains or droplets, additional code is available that specifically works for those types of particles. In that case use the objects *chains* or *droplets* instead. - obj=particles() Initialization, gets filename

- obj=add_pos(obj,pos) manually add particle position matrix

- obj=add_trace(obj,trace) manually add particle trajectory matrix

- obj=add_image_stack(obj,A) manually add image stack

- output=get_parameters(obj) returns parameters used for finding and tracking particles

- output=get_name(obj) returns the location of the imagestack connected to this object

- obj=read_parameters(obj) reads in parameters for particle finding and tracking. Uses format that comes out of get_parameters.

- obj=find_pos(obj) finds particle positions in each frame. Interactive.

- check_pixel_bias(obj) plots a graph that allows you to check for pixel bias

- output=get_positions(obj) returns a list of all particle positions

- obj=change_scale(obj,scale) saves a new scale to object in micrometer per pixel

- obj=change_framerate(obj) saves a new framerate to object in frames per second

- obj=track_obj(obj) Uses Crocker and Grier code to connect particle positions into trajectories

- output=get_trace(obj) returns a list of all paritcle trajectories

- plot_tracks(obj,tbegin,tend,ID) plots particle trajectories

- displacement_histogram(obj) plots displacement histograms

- obj=drift_correct(obj) corrects for constant drift in the sample

- obj=make_dist_matrix(obj) creates a table of all average distances between particles

- obj=make_av_dist_matrix(obj) creates a table of the average distances between all particles

- plot_trace_on_movie(obj) makes movie of particle motion with arrow of direction of motion on top

- output=find_abs_displacement(obj) returns the particles that have a larger displacement than 3 times over the average

- obj=complete_trace(obj) fills holes in trajectories with NaN

- obj=automatically_improve_trace(obj) uses find_abs_displacement to find likely mistracks, removes them and tracks again

- plot_tracked_image(obj,frame) plots image with vector of direction of motion on top

- obj=remove_point_trace(obj) manually removes one frame from the trajectories

- obj=merge_tracks(obj,ID1,ID2) Merges trajectories with label 1 and label 2 if they don't overlap

- obj=calc_MSD(obj) calculates the mean squared displacement

- plot_MSD(obj) plots the mean squared displacement vs time for each particle and the average

- output=find_stagnant_particles(obj) returns ID's of particles that move less than 3 times below the average

- obj=remove_stagnant_particles(obj) uses find_stagnant_particles to remove particles that are presumed stuck

- obj=find_reduced_trajectory(obj) calculates a 2D histogram of direction and magnitude of steps

- obj=plot_reduced_tracks(obj) plots the histogram calculated by find_reduced_trajectory

- obj=plot_average_reduced_track(obj) see two functions above: plotst the average histogram

- obj=plot_average_reduced_track_subplot(obj,dt) plots the histogram for different time intervals dt and places in subplot

- output=find_reduced_trajectory_temp(obj,dt) returns the reduced trajectory

- obj=correlate_motion(obj) uses reduced tracks to calculte Dtrans, Drot and v0(persistent speed)

- obj=crop_in_time(obj,tmin,tmax) removes all information outside of window tmin tmax

- obj=calc_speed_vs_time(obj,particlelist) for all particles in the list, calculates speed vs time

- obj=calc_speed_vs_time_neighbours(obj, ID) correlates speed of particle with ID with that of neighbours

- obj=find_neighbours(obj,cutoff) finds neighbours of all particles using distance matrix. All within cutoff are neighbours

- obj=find_neighbours_time(obj,cutoff) finds number of neighbours in each frame

- correlate_speed_numneighs(obj,cutoff,particlelist) correlates particle speed with number of neighbours

- output=corr_xy_motion(obj) finds average diffusion tensor for all particles in the object

Outlook

If you feel inspired by the concise and organized way in which matlab objects can store both code and analyzed data, there is plenty of work to be done to make this project more widely applicable. Currently no bond order parameters can be calculated using this code, but it would be extremely useful and relatively easy to implement codes that do this. The current code can deal with two-dimensional data, but expanding to three-dimensional particle positions obtained from confocal would be a useful next step. Also the tracking itself can be improved. The original Crocker and Grier method is fine for tracking of individual particles far from each other, but the positions at which particles are found is notoriously influenced by the proximity of close neighbours, making this technique unreliable for interaction measurements. To improve this, it would be useful to implement the recent software by Van der Wel et al. [2]. Last but not least, Matlab is not an open source program making the use of this code limited. Transfer to open source programs such as python would be a great advantage.

[1] J. C. Crocker and D. G. Grier, J. Colloid Interface Sci. **179**, 298 (1996).

[2] C. M. Van der Wel and D. J. Kraft, J. Phys. Condens. Mat. **29**, 44001 (2017).