u-track Documentation

Release 2.2.0

UT Southwestern - Danuser Lab

June 01, 2017

CHAPTER

ONE

GENERAL INFORMATION

1.1 License

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see http://www.gnu.org/licenses/.

1.2 Requirements

The program has been tested using the following configuration(s):

Operating systems: Redhat Linux (4.4.6-4) 64-bit, Windows 10 64-bit.

MATLAB version: 2013b and above

MATLAB toolboxes: to run ALL components of the software, the following MATLAB toolboxes MUST be installed:

- Curve Fitting Toolbox.
- Image Processing Toolbox,
- Statistics Toolbox,
- Optimization Toolbox,

1.3 Installation

1. Download the latest version of the u-track software from http://www.utsouthwestern.edu/labs/danuser/software/.
2. Extract all the files from the zip file that was downloaded.
3. Start MATLAB
4. Add the `software` directory containing the code to the MATLAB path using *Set Path -> Add with Subfolders...* or by clicking right on the folder and selecting *Add to Path -> Selected Folders and Subfolders*.
5. Run u-track either from the graphical user interface or the command line as described in the following sections.

1.4 Version changes

Modifications of u-track since the publication of the paper (Jaqaman et al., Nat. Methods 2008):

2.2.0 (December 2016)

- New parallel processing functionality for multi-movie datasets via GUI.
- Minor bug fixes related to MATLAB version updates.

2.1.3 (December 2014)

- Build fix: re-include the microtubule tracking functionalities missing in 2.1.2 in the distributed software.

2.1.2 (November 2014)

- Fix log4j initialization warning.
- Add support for MATLAB R2014b.
- Fix permission issues when saving microtubule track histograms. Thanks to Paul Thomas for reporting this issue.
- Improve performance of OMERO data selection and allow to browse data in read-only groups.
- Fix default analysis path when running u-track against OMERO images under Windows.
- Bug fix: restore help button in individual step settings interfaces.
- Add support for analysis of OMERO datasets

2.1.1 (March 2014)

- Allow analysis of images stored on an OMERO 5 server
- Add graphical user interfaces to the movie selection window to log in to an OMERO server and load images and datasets from this server.

2.1.0 (June 2013)

- Fix graphical interface bug in MATLAB R2013a and above. Thanks to Riccardo Felletti and Liam Holt for reporting this issue.
- Allow the analysis of movies stored on an OMERO image database. Analysis results are uploaded onto the OMERO server as a file attachment.

2.0.0 (March 2013)

- Include two new tracking applications as part of the u-track package: microtubule plus-end tracking (previously distributed as plusTipTracker) and nuclei tracking.
- Implement a third optional processing step to the analysis workflow: track analysis with two methods: motion analysis (see Jaqaman et al. Cell 2012) and microtubule plus-end classification (see Applegate et al. JSB 2011)

- Allow the preselection of the tracking application when running u-track for the first time on a movie. Detection, tracking and track analysis default parameters/methods/cost matrices are initialized with respect to the chosen tracking application.

21 December 2011

- Tracking code modifications to improve speed: The tracking code has been internally modified to enhance its speed; it runs faster and no longer slows down almost exponentially with increase movie length. No effect on input or output.
- New cost functions costMatRandomDirectedSwitchingMotionLink and costMatRandomDirectedSwitchingMotionCloseGaps: These new cost functions are similar to their predecessors with one added motion model option, namely moving in a directed manner in a certain direction without the possibility of immediate direction reversal as was the case before. These cost functions have 3 options for motion propagation (instead of 2):
  - linearMotion = 0: one motion model, namely random (Brownian) motion.
  - linearMotion = 1: two motion models, namely random motion and movement with constant velocity.
  - linearMotion = 2: two motion models, namely random motion and movement along a straight line but with the possibility of immediate direction reversal.

  What was linearMotion = 1 in the previous cost functions corresponds to what is linearMotion = 2 in the new cost functions. Everything else is the same.

  Examples: Motor-driven movement is best tracked with linearMotion = 1. One-dimensional diffusion is best tracked with linearMotion = 2. Diffusive movement without any drift or directionality is best tracked with linearMotion = 0.

1 April 2011

- Detection – using absolute background information: The detection code can take additional input arguments that supply it with images of "absolute background" and an alpha-value to compare local maxima to this absolute background.

  "Absolute background" is usually a cropped subpart from the original images, where the cropped area lies outside of the cell. Since this area tends to be quite dimmer than inside the cell, one can use a stricter alpha-value, for example 0.001, to compare local maxima to this background area. The use of a stricter alpha-value for comparison with background outside of the cell and a less strict alpha-value for comparison with local background inside the cell minimizes false positives outside of the cell AND false negative inside the cell, where the objects of interest are located.

  If this option is used, there must be an absolute background image for each original image.
- costMatLinearMotionCloseGaps2 – explicit definition of power for scaling search radius with time:

  In the previous version of costMatLinearMotionCloseGaps2, the user only defined "timeReachConfB" and "timeReachConfL." Given these parameters, the code internally scaled the search radius with the square root of time before "timeReachConfB" and "timeReachConfL," and with time^0.01 after.

  In the new version, the user defines these powers explicitly, to give more flexibility. Setting the new parameters brownScaling and linScaling to [0.5 0.01] is equivalent to the old settings.

25 April 2010

- Detection code bug fix: The previous version of the detection code was overestimating the position uncertainties by a factor of (PSF sigma)^2.

- Code modifications to handle larger datasets: Both the detection and tracking codes have been modified to better handle larger datasets. This has no effect on their input or output though.
- New cost functions costMatLinearMotionLink2 and costMatLinearMotionCloseGaps2:

  These new cost functions are similar to their predecessors, with a few "behind-the-scenes" changes.
- Distance cost scaling in costMatLinearMotionCloseGaps2: For gap closing, merging and splitting, the part of the cost based on distance is now scaled by the average frame-to-frame displacement of the track segments involved. This avoids punishing particles that are more mobile relative to those that are less mobile.
- Alternative costs in costMatLinearMotionCloseGaps2: Previously, the gap closing alternative cost was taken as the 90th percentile of the costs of all potential assignments. Furthermore, the merging and splitting alternative cost was determined on a case-by-case basis. In the new cost function, all alternative costs are assigned the same value, taken as the X percentile of the distribution of potential assignment costs, where X is calculated from the structure of the matrix of potential assignments (in particular, it takes into account the number of potential assignments each track segment has).
- Auxiliary (lower right) block costs in costMatLinearMotionLink2 and costMatLinearMotionCloseGaps2: Those were previously assigned to be the smallest costs in the cost matrix, the goal being that they should not influence the LAP outcome. But, in retrospect, having the lowest costs might favor them, thus influencing the LAP outcome. Thus, they have been modified to be equal to the alternative costs, so that they are truly neutral and do not influence the LAP outcome.

26 June 2009

- Diagnostics:
  1. Gap closing time window: If the additional field "gapCloseParam.diagnostics" is set to 1, the software will plot in the end of tracking a histogram of the closed gap lengths. This will help with assessing the quality of gap closing. Generally speaking, longer gaps should be less frequent than shorter gaps; thus, a gap length histogram with a plateau might be indicative of a too large gap closing time window.
  2. Maximum search radius: The frame-to-frame linking cost function has the additional input "parameters.diagnostics", through which the code will output histograms of frame-to-frame linking distances at the specified frames. For example, if parameters.diagnostics = [2 35], then the histogram of linking distances between frames 1 and 2 will be plotted, as well as the overall histogram of linking distances for frames 1->2, 2->3, ..., 34->35. The histograms can be plotted at any frame except for the first and last frame of a movie. To not plot, enter [ ].
- Gap length penalty: The gap closing cost function has the additional input "parameters.gapPenalty" to penalize longer gaps. If parameters.gapPenalty has the value p, then the penalty for a gap of length p will be p^n. Note that for p > 1 longer gaps are penalized, for p = 1 there is no gap length penalty, while for p < 1 longer gaps are favored.
- Resolution limit: The gap closing cost function has the additional input "parameters.resLimit", representing the resolution limit in pixels. The resolution limit is generally the Airy disk radius, but it could be smaller when iterative Gaussian mixture-model fitting is used for detection. The resolution limit is used to expand the merge/split search radius if it is found to be smaller than the resolution limit.

12 December 2008

- An additional Kalman filter function (default: kalmanReverseLinearMotion) to time-reverse Kalman filter information for the second and third rounds of frame-to-frame linking.

CHAPTER

TWO

GRAPHICAL USER INTERFACE

2.1 Getting started

1. From the MATLAB command prompt, launch the movie selector interface by typing:

```
>> movieSelectorGUI
```

   This command will bring up the movie selection panel (Fig. *Movie selection panel*). The left panel displays all of the movies to be processed. The buttons next to the list allow the user to modify the list by creating, opening, and removing movies. The right panel displays the available software packages that can be used to process the movies.

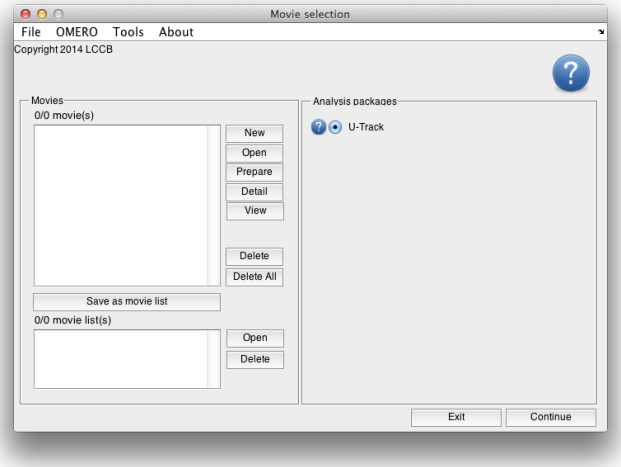

Fig. 2.1: Movie selection panel

2. Add a movie to the processing list by either creating new a new movie database from a image file on the filesystem (see Analysis of movies on the filesystem section below), loading an existing movie saved on disk (see *Movie loading* section below) or loading a movie stored onto an OMERO server (see Analysis of movies on OMERO). Repeat until all of the movies to be processed are listed in the left panel.

   A list of movie databases can be saved as a list using the *Save as Movie List* button. The resulting movie list is also saved as a MAT file on the disk. To load all of the movies in a list, follow the instructions in *Movie loading* and select the MAT file containing the movie list.

3. Select the package on the right panel and click *Continue* at the bottom right of the movie selection window.

   This will bring up the analysis panel.

2.2 Analysis of movies on the filesystem

There are two ways to create movie databases from files on the filesystem:

- from proprietary files using the Bio-Formats libary (see section *Movie creation using Bio-Formats* below).
- from series of TIFF files per channel (see section *Movie creation from series of TIFF files* below)

If a movie has already been analyzed, it cab be reloaded from disk (see section *Movie loading* below).

2.2.1 Movie creation using Bio-Formats

The easiest way to setup movies for analysis is to use the Bio-Formats library to read directly from image files of supported formats.

1. Before using Bio-Formats to read multiple movies or large movies, check the JVM memory settings used by MATLAB and increase them if necessary.
2. Create a new movie database by clicking on *New* from the *Movie selection panel*.

   This will bring up the movie edition interface (Fig. *Movie edition interface*).

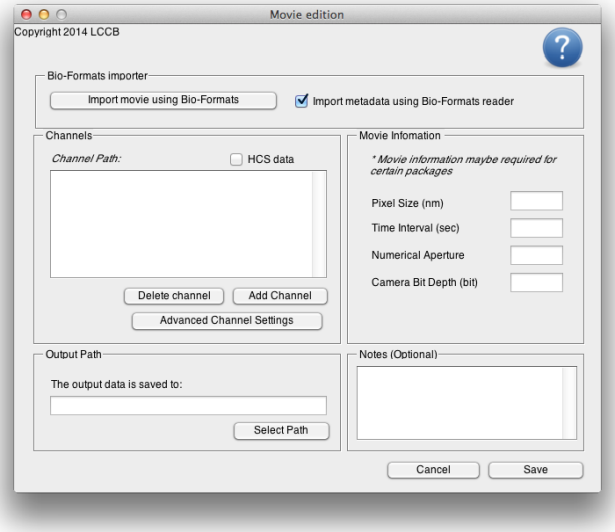

Fig. 2.2: Movie edition interface

3. Click on *Import movie using Bio-Formats* and select the file containing the movie. This will automatically read the metadata if present. It also allows the reading of image sequences from image stacks and proprietary image formats.

   For further information, click on the Help button in the upper right corner of the window, which will launch the online documentation for the software.

4. In the movie edition interface, fill in the pixel size (in nm), time interval (in sec), numerical aperture, and camera bit depth of the movie if these values have not been read from the image metadata.

   Once saved, these fields cannot be further modified. In case of an erroneous input, a new movie database must be generated by repeating all the steps above.

5. For each channel, click on *Advanced Channel Settings*. In the new window (figure *Channel edition interface*), fill out the emission wavelength (in nm) for each channel if this value have not been read from the image metadata.

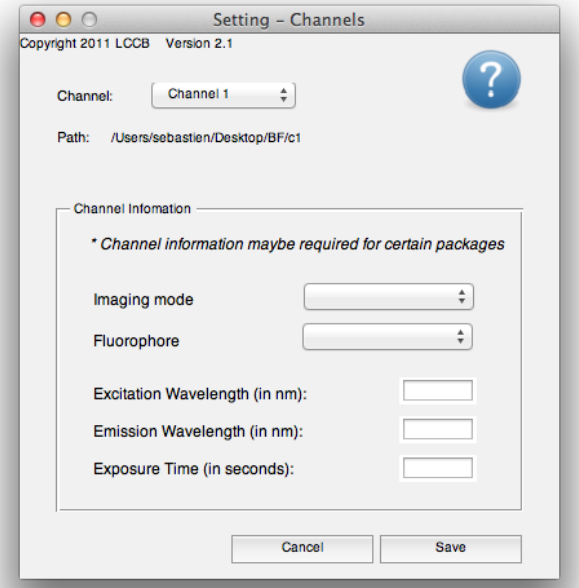

Fig. 2.3: Channel edition interface

6. Optionally, enter additional notes specific to the movie.
7. Click on *Save*.

2.2.2 Movie creation from series of TIFF files

Alternatively, movie databases can be directly created from series of TIFF files.

1. Prepare the image files for the database by storing each channel (wavelength) of each time-series in a separate directory (folder), with one file per frame (time point) of the movie. To indicate the time point for the software package, use the following filename convention: MyMovieXXX.tif.

   `MyMovie` is a placeholder for any string, including underscores and dashes that specifies the generic name of the movie. XXX is a placeholder for a numeric value identifying the time point of that particular frame, e.g., 007, 008, 009, etc.

2. In addition to organizing the raw image sequences, gather information on the camera (pixel size, acquisition time, bit depth), the objective lens (numerical aperture), and the fluorescent channels (emission wavelength) before launching the software.
3. Create a new movie database by clicking on *New* from the *Movie selection panel*. This will bring up the movie edition interface (Fig. *Movie edition interface*).
4. Click on *Add Channel*. Select the folder containing the images to be analyzed. Repeat the *Add Channel* operation for all of the channels of the movie that is to be analyzed.

5. Click on *Advanced Channel Settings*. In the new window (figure *Channel edition interface*), fill out the emission wavelength (in nm) for each channel.
6. In the movie edition interface, fill in the pixel size (in nm), time interval (in sec), numerical aperture, and camera bit depth of the movie.

   Once saved, these fields cannot be further modified. In case of an erroneous input, a new movie database must be generated by repeating all the steps above.
7. Optionally, enter additional notes specific to the movie.
8. In the output directory panel, click Select Path and choose a location on the disk where all results of the analysis should be saved.
9. Click on *Save*.

   This will open a pop-up window asking where to save the movie database. The operation will save a MAT file (MATLAB format) containing all the movie information including all results from the processing. This MAT file can be later reused for loading the movie database (see next section).

2.2.3 Movie loading

From the movie selection panel (Fig. *Movie selection panel*), load the new movie by clicking *Open*. Select the MAT file saved when creating the movie database.

If the movie database file has been relocated on the disk, the software will ask for relocation of all its components by comparing the new path of the movie database file to the old path.

2.3 Analysis of movies on OMERO

Since version 2.1, u-track is able to analyze movies stored on an OMERO server.

Below is a list of steps to load and analyze OMERO movies using u-track:

1. Download and install the OMERO.matlab toolbox corresponding to your OMERO server as described on the OMERO.matlab documentation page.
2. From the movie selection panel top-level menu, select *OMERO -> Log in*. This will bring up the OMERO log in interface (Fig. *OMERO log in interface*).
3. Enter the server name of the OMERO instance and the credentials of the user and select *Log in*. Once connected, select the appropriate group containing the images to analyze.
4. From the movie selection panel top-level menu, select *OMERO -> Load data*. This will bring up the OMERO data selection interface (Fig. *OMERO data selection interface*).
5. Navigate through the projects and datasets using the *Project* and *Dataset* pop-up menus. The listbox will update to display all the images contained in the selected Dataset.
6. Optionally, use the *User* pop-up menu to browse the images of another user in the group. Note that analyzing the images of another user is only possible in a read-annotate group.
7. Select the images to analyze and click *Load selected images*.

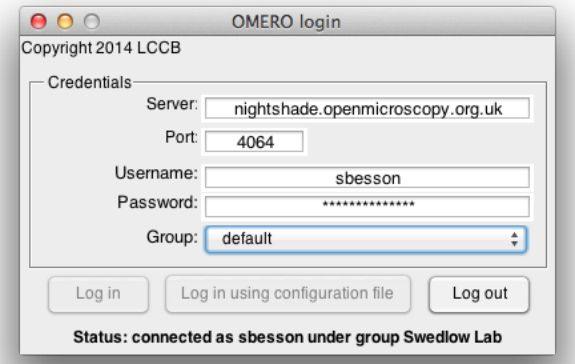

Fig. 2.4: OMERO log in interface

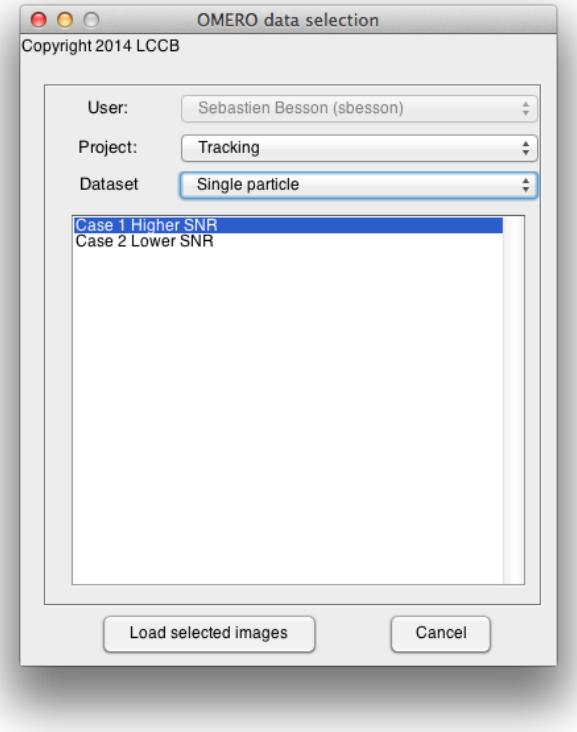

Fig. 2.5: OMERO data selection interface

2.4 Tracking

2.4.1 Overview

The tracking software can be launched by selecting the *Tracking* in the movie selection interface. If the tracking software has already been run on *all* the movies, the main analysis panel will show up. Else a dialog box will open (Figure *Tracking initialization dialog box*) asking the type of objects to be tracking.

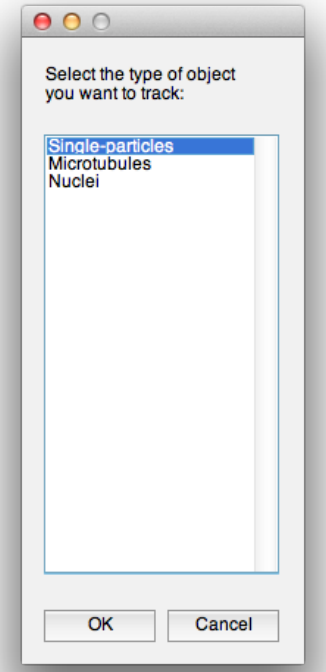

Fig. 2.6: Tracking initialization dialog box

Tracking is currently implemented for 3 kinds of applications: single-particles (2D), microtubules (2D) and nuclei (2D). Detection, tracking and track analysis default parameters will be determined based on the application choice.

2.4.2 Workflow

The u-track workflow currently consists of three consecutive analysis steps: detection, tracking and analysis. In accordance with this analysis work flow, check the components of following main panel of the u-track software (Fig. *Main panel of the u-track analysis software*):

1. A top panel that displays the current processed movie. If processing multiple movies, use the drop-down menu/arrows to switch between movies.
2. A sequence of processes that can be run either individually or in a complete flow.

   Each process needs to be set up before it can be run. To set up a process, click on its associated *Setting* button. Once set up, processes appear in bold characters.

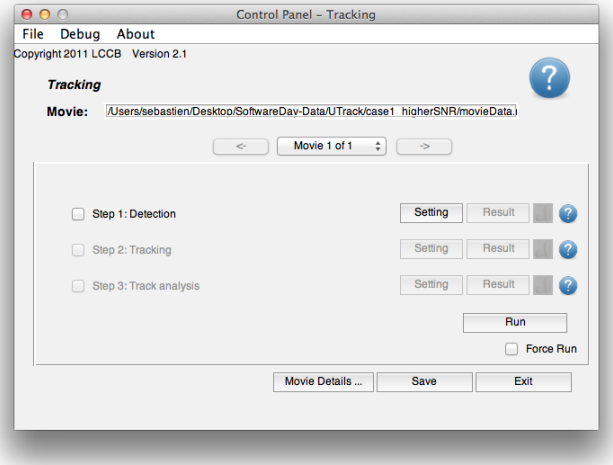

Fig. 2.7: Main panel of the u-track analysis software

3. A checkbox next to each process marks it as scheduled for processing.

   To run a process, check the respective box and click *Run*. Once successfully run, the output of the process can be visualized by clicking the *Result* button.

4. Icons next to each process indicate the current processing status. Only processes that have been run at least once successfully have an associated icon. To obtain more information, click on an icon.

5. A *Run* button. Only checked processes with non-green icons will be run when hitting *Run*. If a process that has been run before needs to be re-ran, check the *Force Run* box.

In the case of processing multiple movies through multiple steps, the processes that are to be run need to be individually set up and checked.

In each setting interface, check the *Apply to All Movies box* to set up processes in batch. Check *Apply Check/Uncheck to All Movies* to schedule processes to run for all movies. Finally, click *Run All Movies* to run scheduled processes of all movies in the list.

All interfaces of the u-track have a Help icon located in the top right corner of the window. Click on the icon to open the help document as a PDF file.

Each analysis step will be prepopulated with default values depending on the kind of tracking application the package was initialized with.

	Single particles	Microtubule plus-ends	Nuclei
Default detection method name	Gaussian Mixture-model fitting	Comet detection	Nuclei detection
Alternate detection method name	Point source detection	Anisotropic Gaussian detection	
Linking cost matrix	Brownian + Directed motion models	Microtubule plus-end dynamics	Brownian + Directed motion models
Gap closing cost matrix	Brownian + Directed motion models	Microtubule plus-end dynamics	Brownian + Directed motion models
Track analysis method	Motion analysis	Microtubule dynamics classification	Motion analysis

CHAPTER

THREE

COMMAND LINE

3.1 Detection

3.1.1 Output

The detection results will be stored in a MATLAB structure called `movieInfo`. If the movie consists of `N` frames, `movieInfo` will be an `N×1` structure array (i.e. containing one entry per frame). The structure `movieInfo` must contain at least 3 fields:

- `xCoord`: x-coordinates of particles
- `yCoord`: y-coordinates of particles
- `amp`: intensities of particles.

If the application is 3D, then there must be a 4th field:

- `zCoord`: z-coordinates of particles.

If there are `P` particles in frame `i`, each of `xCoord`, `yCoord`, `zCoord` (if applicable) and `amp` will be a `P×2` array (i.e. one row per particle). The first column contains the values (e.g. x-coordinates of particles). The second column defines the values' standard deviations (if unknown, put 0).

3.1.2 Script

We supply a code, `scriptDetectGeneral`, for the detection of diffraction-limited objects such as single molecules and small molecular aggregates that are below the resolution limit as presented in Supplementary Note 2 of the paper.

Note: This code corresponds to the `Gaussian Mixture-model fitting` detection method in the graphical interfaceNote and is not optimal for detecting objects with variable size above the resolution limit.

Note: The parameters and paths supplied in `scriptDetectGeneral` must be adjusted to the specific dataset and configuration of the software installation.

Importantly, the detection step is application specific. Thus, the detection and tracking codes in the software package are not linked and alternative detection codes can be used.

To run the detection code:

- Open the file `scriptDetectGeneral.m` from MATLAB command line.
- Define the required parameters (explained in `scriptDetectGeneral`):

  - Movie parameters.
  - Detection parameters.
  - Saving results: Indicate the name and location of the file where results should be saved.
  - Save the changes.

- Run `scriptDetectGeneral` from the MATLAB command line.

The output of scriptDetectGeneral, the variable `movieInfo` will be saved in the format specified above, and it will appear in the MATLAB workspace for use in the tracking code.

3.2 Tracking

3.2.1 Output

The tracking results will be stored in a MATLAB variable called `tracksFinal` If the tracker finds `N` tracks in the movie, `tracksFinal` will be an `N×1` structure array (i.e. every element corresponds to one track). These tracks are compound tracks. If, for example, two particles first move separately and then they merge, their tracks will appear together as one compound track entry in tracksFinal.

Every entry in `tracksFinal` (i.e. every compound track) contains 3 fields:

1. `tracksFeatIndxCG`: Connectivity matrix of particles between frames, after gap closing. Number of rows = Number of tracks merging with each other and splitting from each other (i.e. involved in compound track). Number of columns = Number of frames the compound track spans. Zeros indicate frames where particles do not exist, either because those frames are before the track starts or after it ends, or because of temporary particle disappearance.

2. `tracksCoordAmpCG`: The positions and amplitudes of the tracked particles, after gap closing. Number of rows = Number of tracks merging with each other and splitting from each other (i.e. involved in compound track). Number of columns = 8 × number of frames the compound track spans. For every frame, the matrix stores the particle's x-coordinate, y-coordinate, z-coordinate (0 if 2D),amplitude, x-coordinate standard deviation, y-coordinate standard deviation, z-coordinate standard deviation (0 if 2D) and amplitude standard deviation.

3. `seqOfEvents`: Matrix storing the sequence of events in a compound track (i.e. track start, track end, track splitting and track merging).

   Number of rows = number of events in a compound track (= 2 x number of tracks within the compound track). Number of columns = 4. In every row, the columns mean the following: 1st column indicates frame index where event happens. 2nd column indicates whether the event is the start or end of a track. 1 = start, 2 = end. 3rd column indicates the index of the track that starts or ends (The index is "local", within the compound track. It corresponds to the track row number in `tracksFeatIndxCG` and `tracksCoordAmpCG`). 4th column indicates whether a start is a true initiation or a split, and whether an end is a true termination or a merge. If the 4th column is NaN, then a start is an initiation and an end is a termination. If the 4th column is a number, then the start is a split and the end is a merge, where the track of interest splits from / merges with the track indicated by the number in the 4th column.

3.2.2 Script

After running the detection code, and with the results `movieInfo` in the MATLAB workspace (either directly from `scriptDetectGeneral` of by loading the file where the detection results were saved into the MATLAB workspace), run the tracking code `scriptTrackGeneral`:

- Open the file scriptTrackGeneral.m from the MATLAB command line.

- Define the required parameters (explained in `scriptTrackGeneral`, in Supplementary Note 9 and in the Version changes): - Names of cost functions and Kalman Filter functions. - General and cost function-specific tracking parameters. - Saving results: Indicate the name and location of the file where results should be saved.
- Save the changes.
- Run scriptTrackGeneral from the MATLAB command line.

Note: This code corresponds to the `Brownian + Directed motion models` cost matrices in the graphical interface.

Note: The parameters and paths supplied in `scriptTrackGeneral` must be adjusted to the specific dataset and configuration of the software installation.

The output `tracksFinal` will be saved in the format specified above and it will appear in the MATLAB workspace for use in the visualization code.

We also supply two functions that allow the conversion of the tracks from this structure format into a matrix format that might be easier for further track processing.

Warning: The matrix format can take a lot of memory, so be careful when using this conversion for large movies. Also, merging and splitting information will be lost with this conversion.

If tracking was done WITHOUT merging and splitting, call the command:

```
>> [trackedFeatureInfo, trackedFeatureIndx] = convStruct2MatNoMS(tracksFinal);
```

If tracking was done WITH merging and/or splitting, call the command:

```
>> [trackedFeatureInfo, trackedFeatureIndx, trackStartRow, numSegments] = convStruct2MatIgnoreMS(trac
```

Input:

- `tracksFinal`: This is the output of scriptTrackGeneral. This variable should exist in the MATLAB workspace after running scriptTrackGeneral.

Output:

- `trackedFeatureInfo`: This is the equivalent of tracksCoordAmpCG above, but for tracks together.
- `trackedFeatureIndx`: This is the equivalent of tracksFeatIndxCG above, but for all the tracks together.
- `trackStartRow`: An array indicating the row in the big matrices storing the information of the first segment of each compound track.
- `numSegments`: An array indicating the number of segments belonging to each compound track

3.3 Visualization

There are four visualization functions:

3.3.1 plotTracks2D

Statically plots the tracks generated by `scriptTrackGeneral`. The function can be called from the MATLAB command line as follows:

```
>> plotTracks2D(tracksFinal, timeRange, colorTime, [], indicateSE, newFigure, image, [], ask4sel, [],
```

It takes the following input:

- `tracksFinal`: This is the output of scriptTrackGeneral. This variable should exist in the MATLAB workspace after running scriptTrackGeneral.
- `timeRange`: Frame range to be plotted, e.g. [20 40] plots the tracks that exist between frames 20 and 40. To plot tracks in the whole movie, enter [].
- `colorTime`:
  1. `1` to draw tracks with time color-coding (tracks start in green, go through blue, end in red). Warning: This option makes the figure very big and slow if there are many tracks.
  2. `2` to cycle through 7 colors and give every track a random color.
  3. `3` to cycle through 23 colors and give every track a random color.
  4. `r`, `b`, `g`, etc. to plot all the tracks in the same color, which is the color indicated between quotes (r=red, b=blue, g=green, etc (in MATLAB, type `help plot` for line colors)).
- `indicateSE`: 1 to indicate track starts and ends with circles and squares, respectively; 0 not to.
- `newFigure`: 1 to plot tracks in a new figure window, 0 to plot them in an already open figure window (e.g. to overlay two sets of tracks).
- `image`: In order to overlay the tracks onto an image (e.g. the first image in a movie), supply the image (as a uint16 or double matrix which already exists in the MATLAB workspace). In this case, newFigure has to be 1.
- `ask4sel`: 1 to select tracks and get their information, 0 otherwise. If 1, there will be a question via the command line about selecting tracks. Answer "y" for yes or "n" for no. If "y," then you can click on points in the plot and get back information about the tracks. Use left-click, until the last point where a right-click is needed. After you get the information about the requested tracks, you can repeat this process until no longer desired. YOUR FINAL CHOICE MUST BE "n," OTHERWISE MATLAB MIGHT HANG UP.
- `minLength`: Minimum length (duration in frames) of a track to be included in plot.

If `plotTracks2D` is called with only the first input argument, i.e. one types the command ``plotTracks2D(tracksFinal)` in the MATLAB command line, the code's default is to plot all the tracks for the whole movie, with all the tracks colored black, showing track starts and ends, in a new figure window without an image underneath.

In the tracks, a dotted line = closed gap, a dashed line = merge, a dash-dotted line = split.

3.3.2 plotCompTrack

Plots one track at a time, showing coordinates and amplitude vs. time. The function can be called from the MATLAB command line as follows:

```
>> plotCompTrack(trackToBePlotted)
```

It takes the following input:

- `trackToBePlotted`: This is one of the tracks from the output of scriptTrackGeneral. For example, to plot track # 10, define trackToBePlotted = tracksFinal(10). This variable should exist in the MATLAB workspace after running scriptTrackGeneral.

In the tracks, a dotted line = closed gap, a dashed line = merge, a dash-dotted line = split.

3.3.3 overlayFeaturesMovie

Generates a Quicktime movie of the detected features overlaid on the analyzed images. The function can be called from the MATLAB command line as follows:

```
>> overlayFeaturesMovie(movieInfo, startend, saveMovie, movieName, [], showRaw, intensityScale, first
```

It takes the following input:

- `movieInfo`: This is the output of scriptDetectGeneral. This variable should exist in the MATLAB workspace after running scriptDetectGeneral.
- `startend`: Frame range to be plotted, e.g. [20 40] plots the detected features between frames 20 and 40. To plot all frames, enter [].
- `saveMovie`: 1 to save movie with overlaid features as a Quicktime movie; 0 otherwise.
- `movieName`: Name of movie, for example "detectionResults.mov", if saving is requested. Enter [] if movie is not to be saved.
- `showRaw`: 1 to show original movie to the left of the overlaid movie, 2 to show original movie to the top of the overlaid movie, 0 to only show overlaid movie.
- `intensityScale`: 0 to autoscale every image in the movie, 1 to have a fixed scale using intensity mean and std, 2 to have a fixed scale using minimum and maximum intensities.
- `firstImageFile`: Full name (including path) of first image file for overlaying. The file has to be the first image that has been analyzed even if not plotted. If file is not specified, i.e. if [] is used, user will be prompted to select the first image.
- `dir2saveMovie`: Directory where to save output movie. If not input, i.e. if [] is used, movie will be saved in directory where images are located by default.

3.3.4 overlayTracksMovieNew

Generates a movie of the tracks overlaid on the analyzed images. In the movie, detected objects will be indicated by circles, closed gaps by asterisks, merges by yellow diamonds and splits by green diamonds.

The function can be called from the MATLAB command line as follows:

```
>> overlayTracksMovieNew(tracksFinal, startend, dragtailLength, saveMovie, movieName, [], 0, highligh
```

It takes the following input:

- `tracksFinal`: This is the output of scriptTrackGeneral. This variable should exist in the MATLAB workspace after running scriptTrackGeneral.
- `startend`: Frame range to be plotted, e.g. [20 40] plots the detected features between frames 20 and 40. To plot all frames, enter [].
- `dragtailLength`: Length of track drag tail. To show tracks for the whole length of a movie, enter a value larger than the movie length.
- `saveMovie`: 1 to save movie with overlaid features as a Quicktime movie; 0 otherwise.
- `movieName`: Name of movie, for example "trackingResults.mov", if saving is requested. Enter [] if movie is not to be saved.
- `highlightES`: 1 to highlight object appearance and disappearance (appearance in green, disappearance in yellow); 0 otherwise.

- `showRaw`: 1 to show original movie to the left of the overlaid movie, 2 to show original movie to the top of the overlaid movie, 0 to only show overlaid movie.
- `imageRange`: Image region to make movie out of, in pixels.
- `classifyLft`: 1 to color-code objects based on their lifetime (white: objects that last throughout the whole movie, red: objects that appear AND disappear in the movie, purple: objects that appear OR disappear (but not both) in the movie); 0 otherwise.
- `intensityScale`: 0 to autoscale every image in the movie, 1 to have a fixed scale using intensity mean and std, 2 to have a fixed scale using minimum and maximum intensities.
- `colorTracks`: 1 to give the track dragtails different colors, instead of color-coding the object symbols; 0 otherwise.
- `firstImageFile`: Full name (including path) of first image file for overlaying. The file has to be the first image that has been analyzed even if not plotted. If file is not specified, i.e. if [] is used, user will be prompted to select the first image.
- `dir2saveMovie`: Directory where to save output movie. If not input, i.e. if [] is used, movie will be saved in directory where images are located by default.
- `minLength`: Minimum length (duration in frames) of a track to be included in movie.