Tracking Routine Manual

1 Getting started

1.1 How to run and install SPTAnalyser

1.1.1 Installation

This tracking routine uses ThunderSTORM to localize particles, a plugin for Fiji[1, 2]. The localizations can also be found with rapidSTORM[3]. Particles are connected to trajectories with swift[4].[A] Parameter determination for swift and further data analysis is realized in SPTAnalyser. SPTAnalyser also processes PALMTracer[B] data, a program that both localizes and tracks. The connections between the different softwares are shown in fig. 1.

Navigate to the directory with the SPTAnalyser *.whl file and execute listing 1 in an Anaconda prompt. The ThunderSTORM.jar plugin[C] must be placed in the plugins folder of Fiji. Install a h5-viewer.[D] Follow the swift installation guide (sec. 1.2).

Listing 1: Commands to install SPTAnalyser and dependencies in the Anaconda prompt.

```
conda create --name SPTAnalyser
conda activate SPTAnalyser
conda install pip pywin32
pip install jupyterlab
python -m pip install pyErmine
pip install SPTAnalyser-XXX-py3-none-any.whl
pip install jupyter_contrib_nbextensions
jupyter notebook
```

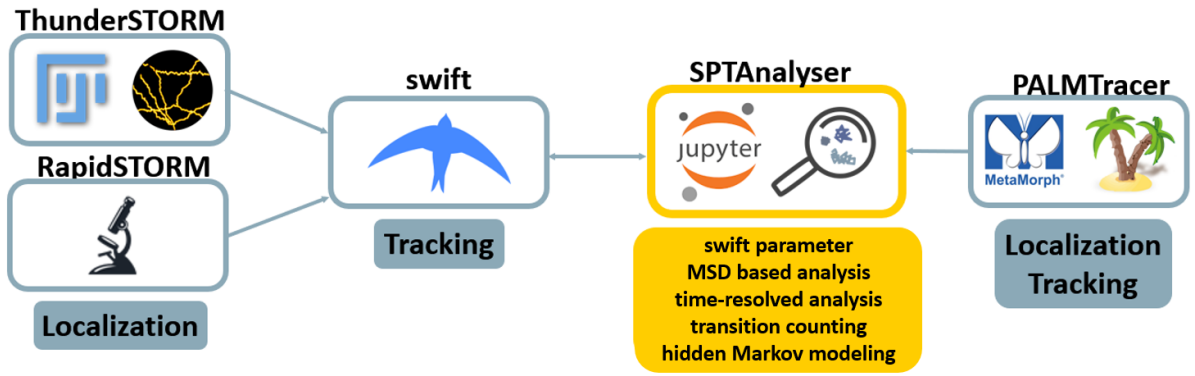

Fig. 1: Connections between the softwares. Swift is compatible with the localization softwares ThunderSTORM and rapidSTORM. SPTAnalyser is compatible with swift and PALMTracer trajectories.

[A] http://bit.ly/swifttracking
[B] https://docs.google.com/forms/d/e/1FAIpQLSdxkJXmoc5uRqoL8kzCSV2Rv90kEfRMIoaPuk--Bpm0Lttb0g/viewform
[C] https://github.com/zitmen/thunderstorm/releases/tag/dev-2016-09-10-b1
[D] https://www.hdfgroup.org/downloads/hdfview/

1.1.2 Execution

SPTAnalyser is either executed as scripts or in Jupyter notebooks. This section gives an overview of how to execute SPTAnalyser in both ways for users that are unfamiliar with these concepts.

1.1.2.1 How to run a script

Open the anaconda prompt, change the directory to the SPTAnalyser folder that contains the script and activate the environment. Execute the script by calling python first, followed by the script and config file name (listing 2). It is possible to run multiple instances of these scripts at once with multiple powershells, allowing different conditions to be analyzed in parallel.

Listing 2: Commands to run SPTAnalyser scripts.

```
cd C:\Documents\SPTAnalyser
conda activate SPTAnalyser
python script_name.py config_name.ini
```

1.1.2.2 How to run a notebook

Run the notebooks by following listing 3. Change to the directory of the notebooks (*.ipynb ending), activate the SPTAnalyser environment and start the notebooks - they will open in a browser. When running the notebooks for the first time, configure the notebook settings by clicking on the Nbextensions tab, enabling configuration and selecting "Collapsible Headings" and "Hide input all" (fig. **??**), to increase usability.

Listing 3: Commands to run SPTAnalyser notebooks.

```
cd C:\Documents\SPTAnalyser
conda activate SPTAnalyser
jupyter notebook
```

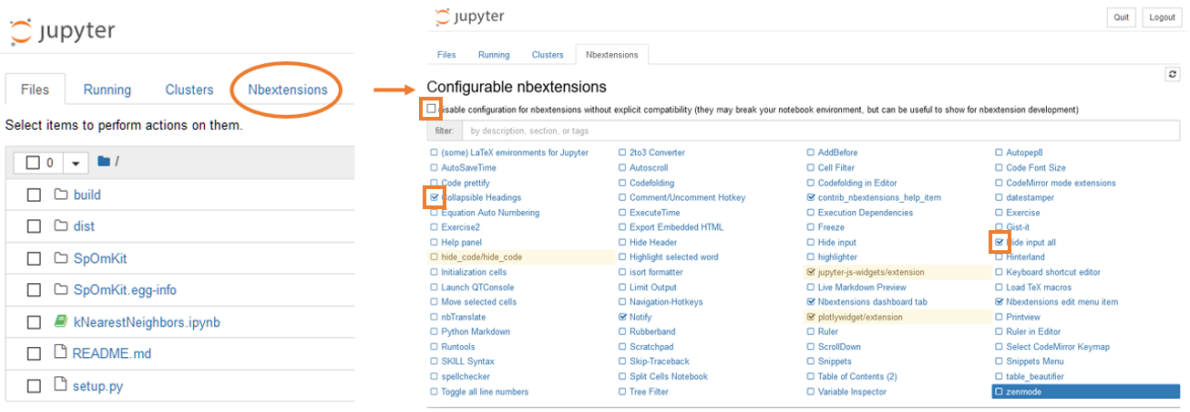

Fig. 2: Configure Jupyter Notebook settings. Enable "hide input all" to make code cells disappear. Enable "collapsible headings" to make sections disappear. Both settings will increase the overall usability by making the notebooks look cleaner.

In the files tab, select a notebook. A new tab opens where the notebook must be initialized by clicking on the double arrow (1). By clicking on "show codecell inputs" the code cells are hidden, which makes the interface more clean (2), and collapsing sections also helps to keep a better overview (3) (fig. 3).

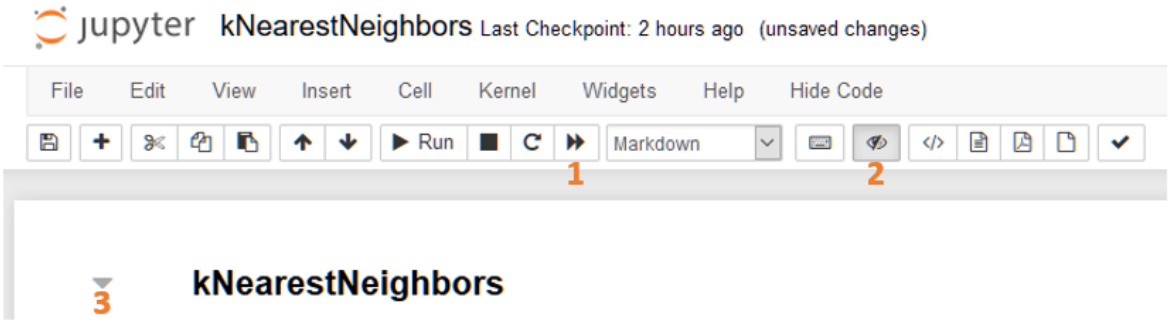

Fig. 3: Start and run a notebook.

It is possible to customize the default parameters of the widgets. Deactivate "show codecell inputs" to see the code. All adjustable settings are highlighted with comments and their values can be adjusted (fig. 4).

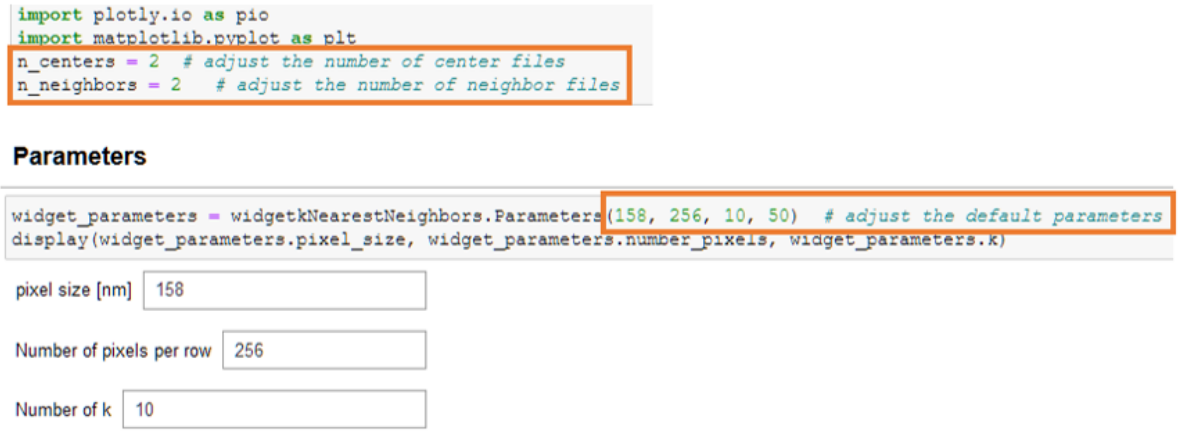

Fig. 4: Two examples of where to adjust the default parameters in the code. The places are marked with # comments.

1.2 Swift installation

Install the latest version of swift by joining the slack forum and download and execute the swift.exe from "versions" (`http://bit.ly/swifttracking`, pipeline was tested up to v0.4.3).

To use the scripts that automatically determine the swift parameters (swift_parameter.py and swift_parameter_converger.py), it is necessary to add swift as a path variable, which allows for the swift.exe to be executed from anywhere. To do this, first locate your swift installation and copy the path to the "/bin/" subfolder. Now open the control panel and navigate to "System and Security" → System → Advanced system settings (or just search for advanced system settings) and click on "environment variables" (fig. 5). The variable can either be added as a user variable (top) or as a system variable (bottom). Click on "path" and then on "edit". In the following window click on "new" and paste the copied swift_bin path. It might take a restart to get into effect.

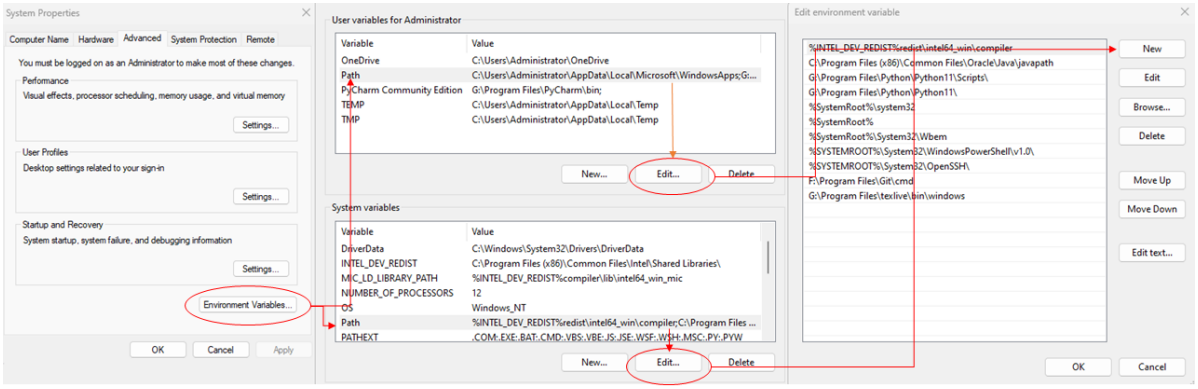

Fig. 5: How to add the swift_bin folder to "Path".

1.3 Recommended folder structure and file names

The following folder structure is recommended for the workflow using ThunderSTORM, swift, and SPTAnalyser (fig. 6). Individual folders contain data of each measured coverslip (CS) with background and cell measurements. Each cell folder (applies to background folder as well) contains a rois, tifs, and tracks folder. The roi folder consists of *.roi files, which mark the region of interest (ROI) in ThunderSTORM, and *.csv files, which contain the measured size of the ROI. The tifs folder contains recorded *.tif movies and transmitted light images *_dl.tif or *_tl.tif. The tracks folder will be filled up with localization files from ThunderSTORM and tracked files from swift during the analysis.
For the naming it is recommended to name every cell individually, for example date, condition, coverslip number, and cell number to be able to distinguish all targets for later analysis. For the time resolved analysis it is essential that the tif movies are named beginning with the date and that the coverslip number is between the second and third underscore (as shown in figure 6).

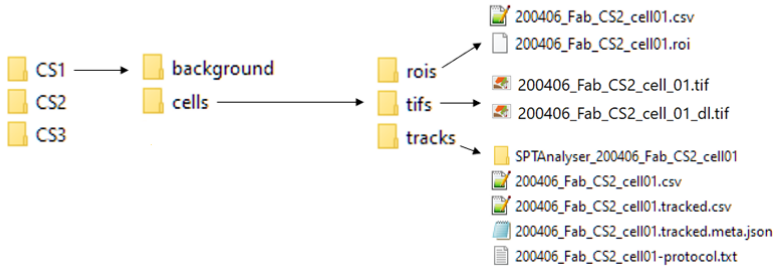

Fig. 6: Folder structure created in the course of the tracking routine.

1.3.1 Script to create folder structure

After measurement the *.tif files are distributed over multiple folders, by the measurement program Micromanager[5]. This folder structure is cleaned up with a script and the resulting structure (fig. 6) is necessary for further batch processing (and in generell recommended to keep an overview over the dataset). To use the script, the background measurement files must contain "background" in their names, the cell measurements "cell" and the transmitted light images "tl" or "dl".
Execute the adapt_folder_structure.py script + config file in Anaconda prompt to automatically create the structure in (fig. 7, listing 4). Define the paths to coverslip directories, multiple directories can be handled at once. Additionally, redundant names in the files can be removed, by defining remove_str = ... (empty = original file names are kept). Example: cell01_MMStack_Pos0.ome.tif will be renamed to cell01.tif.

```
[INPUT_DIRS]
# define the directories to coverslips
dir01 = D:\example_path\CS1

[PARAMETERS]
# define redundant file naming, XXX_MMStack_Pos0.ome.tif -> XXX.tif
remove str =  MMStack Pos0.ome.tif
```

Fig. 7: Config file for adapt_folder_structure.py script. Multiple directories are possible.

Listing 4: Execution of script in Anaconda prompt to create folder structure in fig. 6.

```
python adapt_folder_structure.py adapt_folder_structure_config.ini
```

1.4 Test data

The dataset folder in SPTAnalyser contains files of 60 HeLa cells marked with non-activating Fab ligand. This dataset can be used to execute TransitionCounts. To test the other parts of the analysis, please download the data from `https://www.ebi.ac.uk/biostudies/studies/S-BSST712`.

SPTAnalyser_dataset

- *.csv file = localization file from ThunderSTORM
- *-protocol.txt = information file with ThunderSTORM parameters
- *_tracked.csv file = tracked file from swift
- *.tracked.meta.json = information file with swift parameters
- *.h5 = SPTAnalyser file from trackStatistics

Required input files for the different analyses

- diffractionLimit.ipynb - localization file
- expDisplacement.ipynb - tracked file
- expNoiseRate.ipynb - localization file
- pBleach.ipynb - tracked file
- precision.ipynb - localization file
- trackAnalysis.ipynb - tracked file
- trackStatistics.ipynb - file from trackAnalysis
- transitionCounts.ipynb - tracked file and file from trackStatistics / trackAnalysis
- swift_parameter.py - coverslip pathway containing localization files
- swift_parameter_converger.py - coverslip pathway containing localization files
- Dmin_autocalculator.py - coverslip pathway containing localization and tracked files
- trackAnalysisStatistics_automizer.py - coverslip pathway containing localization and tracked files

2 ThunderSTORM

ThunderSTORM[1] is a localization software, available as a plugin for Fiji. The analysis requires transmitted light images and SPT movies of the target as well as information about the camera settings. Localized files are further processed in swift to create trajectories (sec. 3).

2.1 Mark regions of interest

Load the SPT movie and a transmitted light image of a cell into Fiji. Circle the region of interest with a selection tool (fig. 8). Select the transmitted light image window and open the ROI manager by pressing *t* on the keyboard. Select the SPT movie and click on the area in the ROI manager window to transfer the selected area to the movie. Click at *Measure* to calculate the area of the region in px and save this information as *.csv in the rois folder (choose same name as SPT movie, e.g. cell01.tif and cell01.csv). Save the region as *.roi (*More* → *Save...*), as the cell sizes will be needed at later points of the analysis.

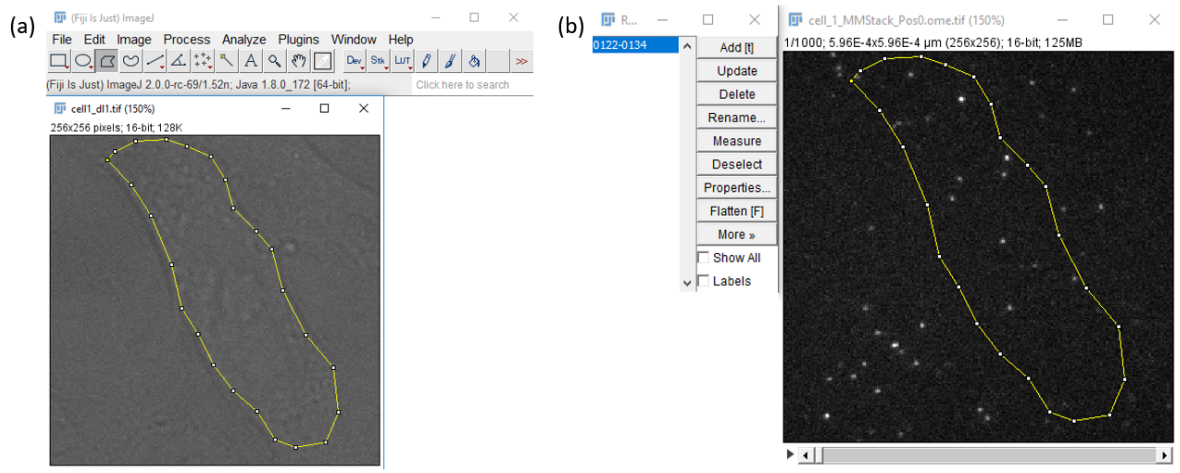

Fig. 8: (a) Selected region of a transmitted light cell image with the polygon selection tool. (b) Transferred region into a single particle tracking (SPT) movie.

2.2 Localization with ThunderSTORM

2.2.1 Analysis of one file

Run the analysis in Fiji with *Plugins* → *ThunderSTORM* → *Run analysis* (fig. 12). The following will briefly introduce the settings. For more information read the Thunder-

STORM user guide.[E,F]

Camera settings must be specified (fig. 11, *Plugins* → *ThunderSTORM* → *Camera setup*). The other recommended parameters can be extracted from fig. 12. The wavelet filter is chosen for feature enhancement. The localization of the molecules is done with the local maximum method in combination with 8-neighborhood-connectivity and integrated Gaussian fitting with maximum likelihood. Maximum likelihood fitting is computationally demanding, but yields the best results . Multi-emitter fitting analysis should be enabled, which distinguishes molecules which were detected as a single blob and further boosts the accuracy of results (fig. 9).

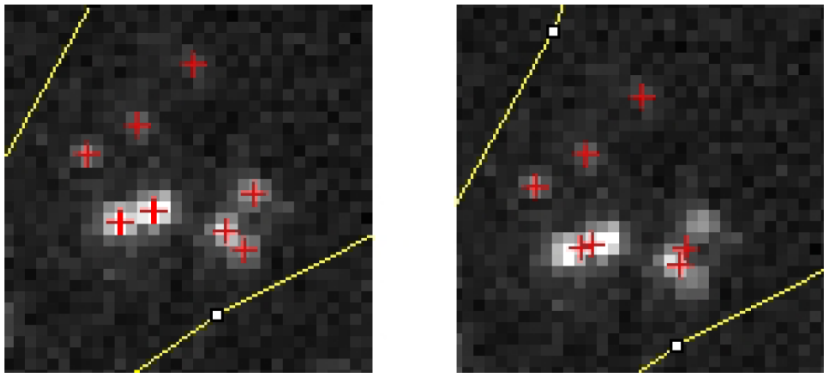

Fig. 9: (left) with multi-emitter fit (right) without multi-emitter fit

The intensity range of the multi-emitter fitting analysis **is the only parameter that must be adapted for new datasets**. Make sure to choose a realistic intensity range per molecule! A first impression are the intensity values of emitters in the raw SPT movie, which are displayed by hovering the cursor over a signal in Fiji. The final intensity range can be determined by localizing the movie with single emitter fit and taking the mean ± n times the standard deviation of the resulting log-transformed photon intensity distribution. Furthermore, check out the intensity values with multi-emitter fitting analysis activated. If the limit intensity range is chosen too small, an artificial intensity peak at high values is created (fig. 10). The maximum number of molecules can be set to 3. Choose a visualization method and finally run the analysis by clicking *Ok*.

[E] `https://www.researchgate.net/profile/Martin_Ovesny/publication/262638879_ThunderSTORM_Supplementary_Note_-_User%27s_Guide/links/0f3175384f21d24f56000000/ThunderSTORM-Supplementary-Note-Users-Guide.pdf`

[F] `http://www.neurocytolab.org/tscolumns/`

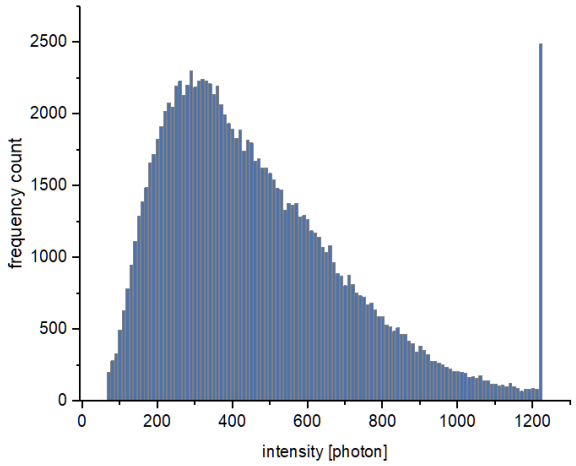

Fig. 10: Photonintensity extracted from ThunderSTORM column with multi-emitter fit. The intensity range was chosen too small and leads to an artificial peak at an intensity of 1220.

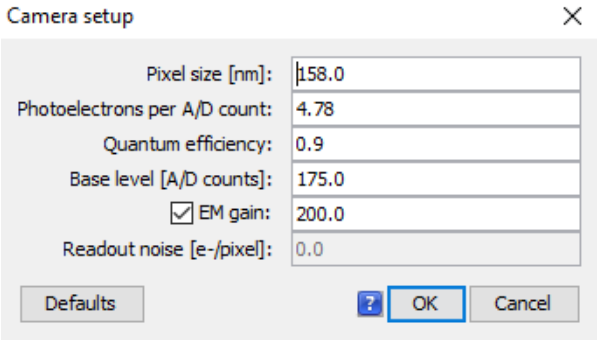

Fig. 11: Example settings for the N-STORM-Setup (Camera Andor iXon Ultra).

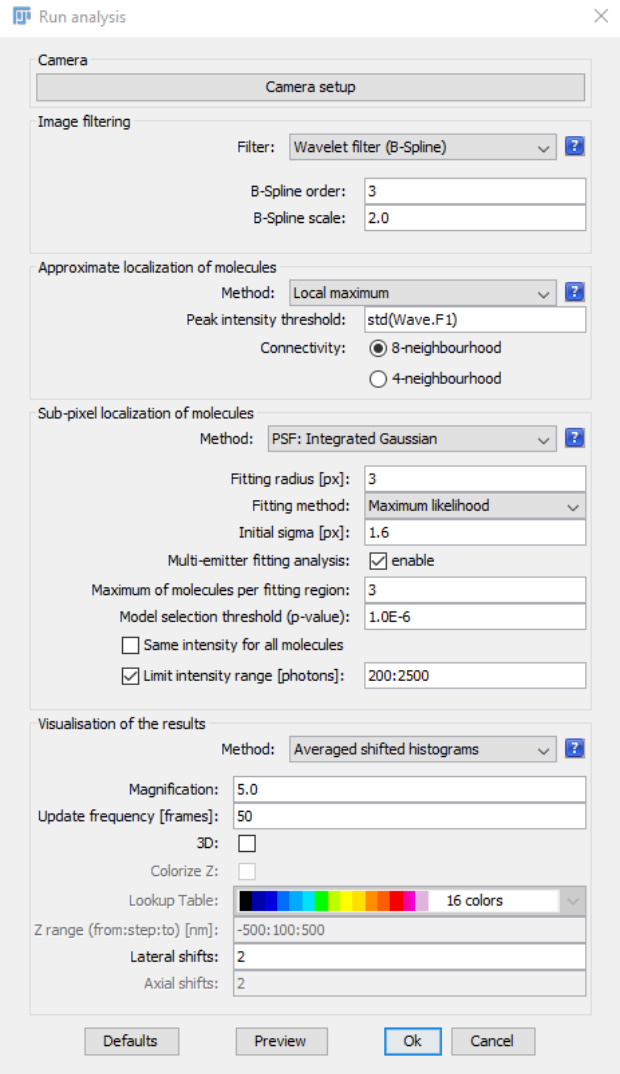

Fig. 12: Recommended ThunderSTORM settings.

The result is a list of localizations (fig. 13). An explanation of the columns can be found here `http://www.neurocytolab.org/tscolumns/`. Units must be in [nm] and [photons], and can be changed by right clicking on the column header. The multi-emitter fit leads to doubled localizations (=two localizations at exactly the same spot). They can be removed by applying the remove duplicates filter. Export the information as *.csv with all possible columns. A float precision of 3 is sufficient. Check "Save measurement protocol" to extract a metafile containing information about the settings (fig. 14).

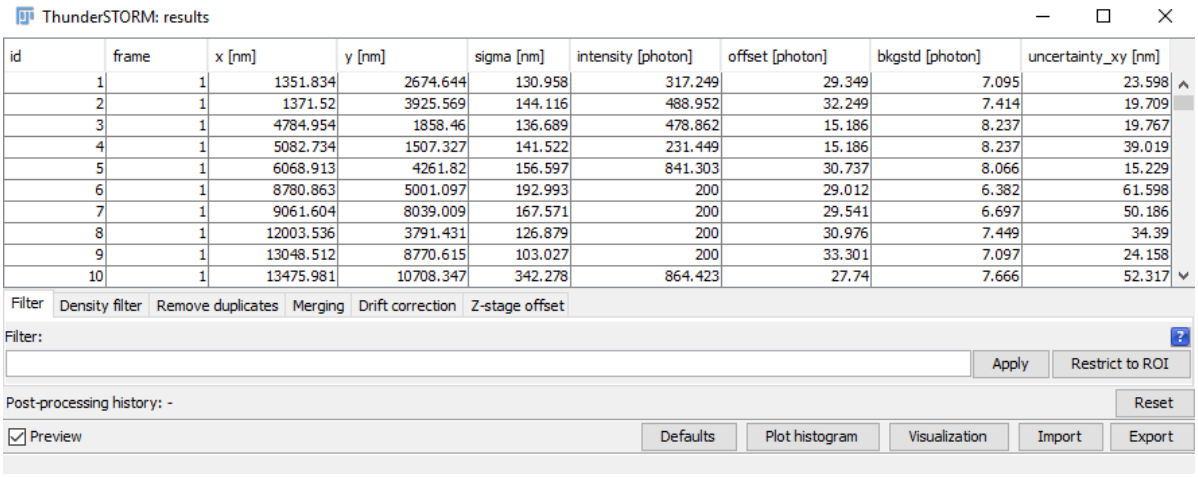

Fig. 13: ThunderSTORM results table with [nm] and [photon] as units.

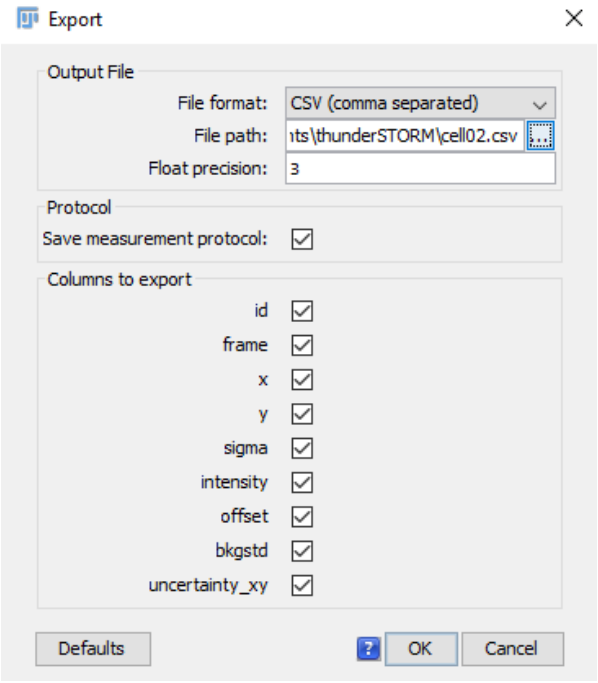

Fig. 14: Export the results as *.csv with all columns and also save the measurement protocol.

2.2.2 Batch processing of multiple files

Execute the localization analysis for multiple SPT movies automatically with a Fiji macro (listing 5). The macro runs the localization analysis, filters for duplicates, and saves the results.

The file paths of the SPT movies are defined by the target_cells array, the names of the SPT movies by the cell_tif_names array. The *.roi files are added to the target_rois array. The paths to save the results are added to the save_paths array. Make sure to write all paths in quotes and use double backslashes. Check for continuous indexing starting from 0 for all arrays and for matching instances per index (target_cell[0] must be consistent with target_roi[0] ...). **Camera settings cannot be changed via the macro and have to be set correctly before running the analysis.** Make sure to set the parameters correctly in the *run("Run analysis")* call, especially the limit intensity range.

Listing 5: Fiji macro to run and save the localization analysis in ThunderSTORM for multiple SPT movies.

```
requires("1.45s");
setOption("ExpandableArrays", true);

//Add the file path of the target objects
target_cells = newArray;
target_cells[0] = "C:\\Documents\\200406_Fab_CS2_cell_01.tif"
target_cells[1] = "C:\\Documents\\200406_Fab_CS2_cell_02.tif"

//Specify the tail of the path for each target object (example: "cell_name.tif")
cell_tif_names = newArray;
cell_tif_names[0] = "200406_Fab_CS2_cell_01.tif"
cell_tif_names[1] = "200406_Fab_CS2_cell_02.tif"

//Specify the paths to the *.roi files.
target_rois = newArray;
target_rois[0] = "C:\\Documents\\200406_Fab_CS2_cell_01.roi"
target_rois[1] = "C:\\Documents\\200406_Fab_CS2_cell_02.roi"

//Specify the paths where the analysis should be saved.
save_paths = newArray;
save_paths[0] = "C:\\Users\\pcoffice37\\Desktop\\200406_Fab_CS2_cell_01.csv"
save_paths[1] = "C:\\Users\\pcoffice37\\Desktop\\200406_Fab_CS2_cell_02.csv"

//Run the analysis loop
for (i=0; i<target_cells.length ;i++){
open(target_cells[i]);
roiManager("Open", target_rois[i]);
roiManager("Select", i);
run("Run analysis", "filter=[Wavelet filter (B-Spline)] scale=2.0 order=3
detector=[Local maximum] connectivity=8-neighbourhood threshold=std(Wave.F1)
estimator=[PSF: Integrated Gaussian] sigma=1.6 fitradius=3 method=[Maximum likelihood]
full_image_fitting=false mfaenabled=true keep_same_intensity=false nmax=3
fixed_intensity=true expected_intensity=200:2500 pvalue=1.0E-6
renderer=[Averaged shifted histograms] magnification=5.0 colorize=false
threed=false shifts=2 repaint=50");
run("Show results table", "action=duplicates distformula=uncertainty_xy");
run("Export results", "floatprecision=1 filepath=" + save_paths[i] +
```

```
" fileformat=[CSV (comma separated)] sigma=true intensity=true
chi2=true offset=true saveprotocol=true x=true y=true bkgstd=true
id=true uncertainty_xy=true frame=true");
selectWindow(cell_tif_names[i]);
close();
}
```

2.2.2.1 Script to create the ThunderSTORM Fiji macro

Automatic creation of the ThunderSTORM Fiji macro is possible by executing the python script write_TS_macro.py in an Anaconda prompt (fig. 15, listing 6). Define the paths to coverslip directories, the path to the background.roi file, define the intensity range for TS multi-emitter fit and a directory to save the created macro. For this process, the folder structure in sec 1.3 must be followed.

```
[INPUT_DIRS]
# define the directories to coverslips
dir01 = D:\example_path\CS1
dir02 = D:\example_path\CS2

[PARAMETERS]
# define the *.roi file that covers the camera chip area
background_roi = D:\example_path\background.roi
intensity_range = 60:2000

[SAVE_DIR]
# define the save directory for the written fiji macro
macro_directory = D:\example_path
```

Fig. 15: Config file for write_TS_macro.py script. Multiple directories are possible.

Listing 6: Execution of script in Anaconda prompt to write Fiji macro with config file.

```
python write_TS_macro.py write_TS_macro_config.ini
```

3 swift

swift[4] is a tracking software that is compatible with localization lists from ThunderSTORM and rapidSTORM. Tracked files can be further processed in SPTAnalyser (sec. 4). Swift has multiple data and performance parameters that have to be considered for each dataset. Data parameters can be partly determined with SPTAnalyser (sec. 4.1) and are essential for valid tracking results. Performance parameters can help to boost computational time for demanding datasets. More information on the parameters are found in the user manual of swift and in the swift_parameter_overview.pdf file.

3.1 Analysis for one file

Swift can be run with a graphical user interface (swiftgui.exe) and via the command line (swft.exe). Swiftgui has the advantage of visualizing the trajectories and files should be frequently analyzed here to visually check the tracking results. We will focus on the command line approach as it allows batch processing.
Navigate to swft.exe and start the command line from the folder (fig. 16). To process a file, run listing 7. If swift was added to the path variable, the working directory does not matter and the listing can be executed from any directory in the command line. The first command “swft.exe” is calling the tacking program. The second command is the file path to the localization file. It is mandatory to specify a camera integration time in ms and out_values as “all”. Further parameters can be set, e.g. the bleaching probability of 0.01. Press enter to start the algorithm. A *.tracked.csv file will be created.

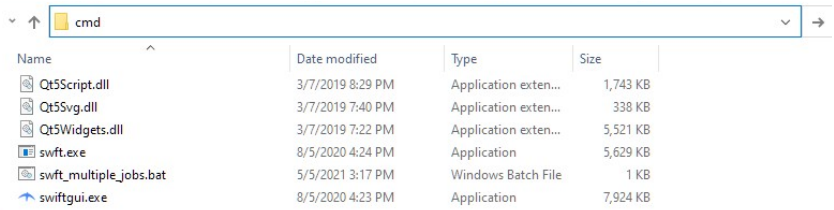

Fig. 16: Start the command line from the directory of the swft.exe by entering ”cmd”.

Listing 7: swift running in the command line to determine pre-analysis parameters.

```
C:\Documents\swift> swft.exe C:\Documents\Fab_CS2_cell01.csv
--precision "26.636" --p_bleach "0.01" --tau "20" --out_values "all"
```

3.2 Batch processing for multiple files

Multiple files are processed at once with a *.bat script. The file paths have to be defined followed by parameter specification (listing 8). The script is executed by double clicking it. An example bat file is contained in the SPTAnalyser repository (swft_multiple_jobs.bat)

Listing 8: Batch script to run the tracking analysis in swift multiple times with the final parameters.

```
@echo off
set list=C:\\Documents\\data\\Fab_CS2_cell01.csv
set list=%list%;C:\\Documents\\data\\Fab_CS2_cell02.csv
set list=%list%;C:\\Documents\\data\\Fab_CS2_cell03.csv
set list=%list%;C:\\Documents\\data\\Fab_CS2_cell04.csv
set list=%list%;C:\\Documents\\data\\Fab_CS2_cell05.csv

FOR %%A IN (%list%) DO (
swft.exe %%A --tau "20" --p_bleach "0.011"
--exp_displacement "56.222" --precision "26.636"
--out_values "all" --p_switch "0.01"
)

ECHO All swift jobs are done.
PAUSE
```

3.2.1 Script to create batch file

The batch file can be automatically created for all localized files in multiple directory with write_swift_batch.py (listing 9). In the config file, define the directories to the localized files and global parameters used for all files. For the parameters “precision” and “exp_noise_rate”, it is possible to set the parameters per directory (as in fig. 17) or individually per file by defining the path to the precision.txt and exp_noise_rate.txt files (SPTAnalyser outputs by running precision.ipynb and exp_noise_rate.ipynb, see chapter 4.1).

```
[INPUT_DIRS]
# define the directories to localized files
dir01 = D:\Documents\Chemie_phd\SPTAnalyser\SPT_Data\CS4\cells\tracks
dir02 = D:\Documents\Chemie_phd\SPTAnalyser\SPT_Data\CS5\cells\tracks

[PARAMETERS_GLOBAL]
# define global swift parameters
tau = 20
exp_displacement = 300
max_displacement = 2.5
max_displacement_pp = 3.5
p_bleach = 0.2
p_switch = 0.001
diffraction_limit = 150

[PRECISION_INDIVIDUAL]
# define precision values for each coverslip by defining a number, or for individual target by defining the path to *precision.txt from notebook
precision01 = 25
precision02 = 30

[EXP_NOISE_RATE_INDIVIDUAL]
# define precision values for each coverslip by defining a number, or for individual target by defining the path to *exp_noise_rate.txt from notebook
exp_noise_rate01 = 150
exp_noise_rate02 = 400

[SAVE_BATCH]
# define the path of batch file (.bat ending)
batch_path = D:\Documents\Chemie_phd\SPTAnalyser\swift\Swift_043\swft_multiple_jobs.bat
```

Fig. 17: Config file for write_swift_batch.py script.

Listing 9: Execution of script in Anaconda prompt to write fiji macro with config file.

```
python write_swift_batch.py write_swift_batch_config.ini
```

4 SPTAnalyser

SPTAnalyser is compatible with localization list from ThunderSTORM and rapidSTORM, and tracking files from swift and PALMTracer. SPTAnalyser offers following functionalities: tracking parameter estimation for swift (diffraction limit, expected displacement, expected noise rate, p bleach, precision), MSD-based diffusion analysis, time resolved analysis of the MSD-based values, transition counting of diffusion states within trajectories, and hidden Markov modeling of the trajectories.

4.1 swift parameters

4.1.1 Parameter explanations

precision

The parameter precision is the localization uncertainty in the x/y-plane in nm. The localization list contains a column with the localization uncertainties. To calculate a representative average, all values are ln-transformed to generate a normal distribution (fig. 18). From the ln-transformed values, the arithmetic mean is calculated, re-transformed and used as the parameter precision. It is recommended to calculate the precision for every target, as the localization uncertainty is highly influenced by the imaging plane. In swift, the parameter is relevant for detecting immobile particles as their apparent displacement is a result of the localization error.

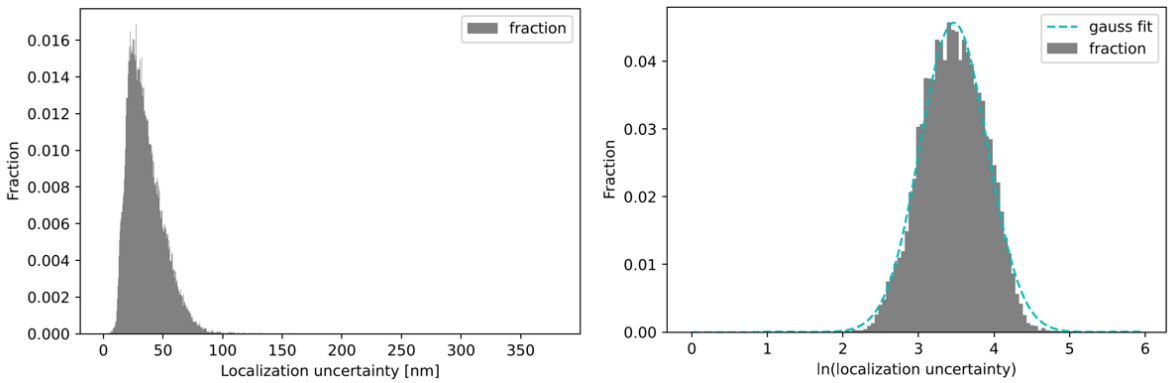

Fig. 18: (left) Histogram of localization uncertainties (right) histogram of logarithmized localization uncertainties. The average of logarithmized localization uncertainties is calculated, re-transformed, and used for the precision. The Gauss fit only assists in evaluating the histogram to be normally distributed in log-space.

exp_displacement

The expected displacement parameter is the expected undirected movement of a particle

per frame in x\y direction in nm. The expected displacement is estimated by calculating the average of the mean jump distances (mjd) weighted by the number of data points used to calculate the mean jump distance (mjd_n) of all particles (fig. 19). The average is still a good estimate when multiple subpopulations are present. Immobile particles should not be taken into account since swift version 0.4.0. We recommend to calculate an average value for all targets in a condition. Swift might prefer longer connections over shorter once for large expected displacement values.

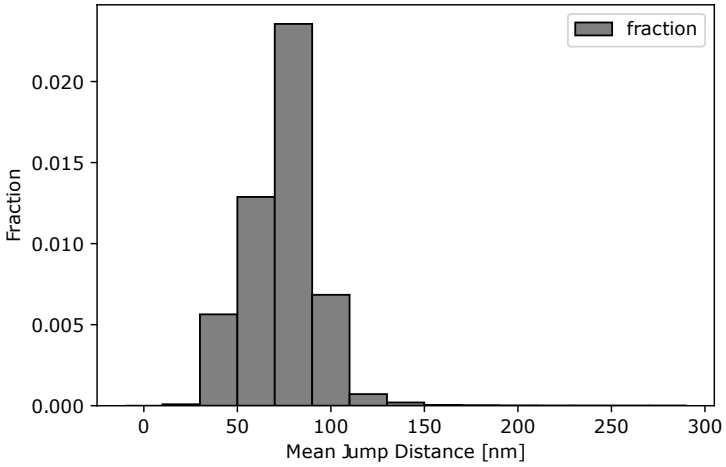

Fig. 19: Histogram of mean jump distances weighted by the lengths of trajectories.

diffraction_limit

The diffraction limit parameter is the minimum distance at which particles can still be distinguished during localization in nm, and is influenced by many variables such as the localization algorithm used. It is determined using a frame-wise nearest neighbor analysis. The minimum nearest neighbor distance of localizations per file is calculated. Manually average these values to get the final diffraction_limit. Multiple files should be analyzed and the minimum distances averaged to estimate this parameter. We recommend using the parameter for all targets in a condition or even for targets from multiple conditions. In swift, the parameter affects the probability of merging and splitting of trajectories, with high values promoting these occurrences.

p_bleach

p_bleach is the probability per particle and frame to bleach, $\in [0; 1]$. The number of trajectories that exist for a given time period are subtracted from the normalized sum of the total number of trajectories and plotted as a histogram (fig. 20). The decay rate is extracted with an exponential fit (eq. 1), with an initial a value (=1), an initial k value in

s^{-1} (=0.5), and the camera integration time Δt in s. The cumulative distribution function of an exponential distribution is used to determine p_bleach (eq. 2). It is possible to mask a number of values in the histogram, starting from time step = 0, because some data deviate from a monoexponential decay especially in the first time steps. The remaining data is normalized and fitted as usual. In swift, the parameter influences the willingness of making connections. The higher the value, the more reluctantly connections are made, as shorter life times of trajectories are expected. We recommend to use one value per condition.

$$f(t;k;a) = a \cdot e^{-\Delta t \cdot k} \tag{1}$$

$$F(\Delta t;k) = 1 - e^{-\Delta t \cdot k} \tag{2}$$

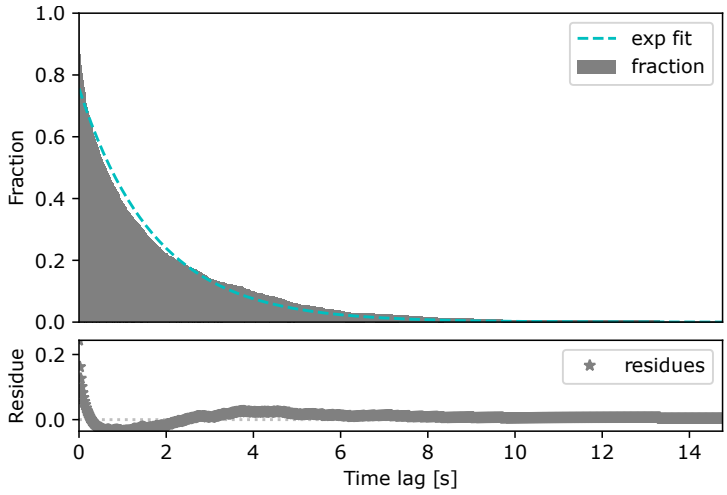

Fig. 20: The amount of trajectories existing after a time period is exponentially fitted and the decay rate used to calculate p_bleach.

exp_noise_rate

The expected noise rate estimates the percentage of false positives among all localizations in percent. The number of localizations per px^2 and frame is determined and averaged for the background measurements. The number of localizations per px^2 and frame is determined for the cell measurements. The expected noise rate is calculated by dividing the background average density through the individual cell density multiplyied by 100 (eq. 3). It is recommended to calculate the expected noise rate for each target as the parameter is influenced by localization density. In swift, the parameter is used in combination with the intensity of a localization to determine its individual likelihood to be noise. False positive points should not be connected into trajectories.

$$\frac{< densitybackground >}{densitycell_i} \cdot 100 \tag{3}$$

4.1.2 Calculation of the parameters with SPTAnalyser

The parameters can be determined with the notebooks, where they are either calculated per target or per coverslip, or with scripts, where multiple coverslips can be processed at once. It is recommended to use the notebooks to get familiar with the parameters, as graphs are plotted that can improve understanding. It is recommended to use the scripts for processing of large datasets.
For very dense datasets it might be helpful to use some not so dense instances for parameter estimation, as less tracking errors occur in less dense (simpler) datasets.
The parameters based on the localization files should be determined first (diffraction limit, expected noise rate, precision), followed by the parameters based on the tracking files (expected displacement and p bleach). There are also other important parameters to consider, for example p switch. Please go through the swift manual and check if parameter default values must be changed.

4.1.2.1 Determination of diffraction limit, expected noise rate, and precision

The parameters diffraction limit, expected noise rate, and precision can either be calculated via their respective notebooks or with the swift_parameter.py script by executing listing 10 from an Anaconda prompt. The directory in the Anaconda prompt must match the directory of the script. The coverslip directories must be defined in the config file and the folder structure from sec. 1.3 is required. The resulting parameters are saved in a folder called “swift_parmeters” per coverslip.
The swift_parameter.py furthermore generates *.LOG files, which contain information about the cell sizes (more on that in 4.2.2.2).

```
[SOFTWARE]
# choose software, either ThunderSTORM or rapidSTORM
software = ThunderSTORM

[INPUT_DIRS]
# directories of the coverslips
dir01 = D:\example_path\CS1

[CAMERA]
# camera parameters: pixel size in nm, camera integration time in s, pixel per row
pixel_size = 158
camera_integration_time = 0.02
pixel_per_row = 256
background_size = 65536
```

Fig. 21: Config file for the swift_parameter script.

Listing 10: Execution of script in Anaconda prompt to run the swift_parameter.py script.

```
python swift_parameter.py swift_parameter_config.ini
```

4.1.2.2 Manual convergence of p_bleach and exp_displacement

Set all known parameters and make initial estimates for exp_displacement and p_bleach. Perform the tracking analysis and determine exp_displacement and p_bleach in their notebooks. Repeat the tracking step with the recalculated values for exp_displacement and p_bleach. After a few iterations (≈3-5x) the values converge (fig. 22).

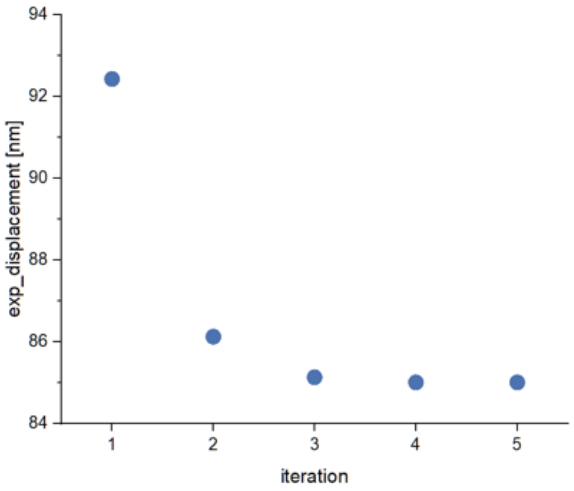

Fig. 22: The exp_displacement value converges after a few iterations of tracking and extracting the value with the expDisplacement.ipynb notebook.

4.1.2.3 Automated determination of p_bleach and exp_displacement

The script swift_parameter_converger.py automatically determines p_bleach and exp_displacement in an iterative fashion, where the files are tracked with the given p_bleach and exp_displacement values in the config file and the values recalculated based on the tracking results and used as new initial parameters for the next tracking round. The paths to the target coverslips must be defined as *dir01, dir02, ...* (folder structure from sec. 1.3 required). Furthermore, define the files that contain the parameter info of precision and expected noise rate. The iteration criteria are defined with max difference (=the maximal difference of the parameters between two runs) and with max iterations (how often the files are tracked sequentially). If one of the criteria is fulfilled, the algorithm stops. It is necessary to have added swift as a path variable (see section 1.2).

```
[SOFTWARE]
# choose software, either ThunderSTORM or rapidSTORM
software = ThunderSTORM

[INPUT_DIRS]
# list each coverslip
dir01 = D:\exampledir\CS1

# if the next two directories are not assigned, the files are expected in dir\P
[PRECISION_FILES]
# directory of the precision files, make sure the have the same key as the corr
dir01 = D:\exampledir\CS1\precisions.txt


[EXP_NOISE_RATE_FILES]
# directory of the exp_noise_rate files, make sure they are named the same as t
dir01 = D:\exampledir\CS1\exp_noise_rate.txt


[GLOBAL_PARAMETERS]
# define initial global swift parameters for the swift.bat. tau in ms, displace
tau = 20
exp_displacement = 120
max_displacement = 2.5
max_displacement_pp = 3.5
p_bleach = 0.05
p_switch = 0.01
diffraction_limit = 42

[P_BLEACH_PARAMETERS]
# set parameter for the pBleach fit that follows a·exp(-k·t). initial k in 1/s,
initial_k = 0.5
camera_integration_time = 0.02
number_of_mask_points = 0

[ITERATION_PARAMETERS]
# define the maximum allowed difference for the values to be considered converg
max_difference = 0.001
max_iterations = 10

[SAVE_DIRECTORY]
# where the swift.bat as well as the reports for the whole run are saved
save_path = D:\exampledir
```

Fig. 23: Config file for the swift_parameter _converger script.

Listing 11: Execution of script in Anaconda prompt to run the iterative determination of p_bleach and exp_displacement.

```
python swift_parameter_converger.py swift_parameter_converger_config.ini
```

4.2 Diffusion analysis with SPTAnalyser

The diffusion analysis is executed in two Jupyter Notebooks: TrackAnalysis and TrackStatistics. The first notebook calculates diffusion coefficients and determines between the diffusion type immobile, confined, free, and notype according to Michalet *et al.*[6], Rossier *et al.*[7] and Harwardt *et al.*[8] The results are saved in a h5 file. The calculations might take several minutes depending on the size of the dataset. The second notebook reloads the information from the created h5 files. Here, the data can be analyzed in detail. Global information on the tracking results is available as well as filtering according to different criteria.

4.2.1 Calculations of diffusion parameters

For a more detailed explanation check out the literature[6–8].

Diffusion coefficient

The diffusion coefficient for individual trajectories is obtained by fitting the first n points in the MSD plot with eq. 4, dof = degree of freedom, dof(2D diffusion) = 4.

$$MSD(\Delta t) = dof \cdot D \cdot \Delta t \tag{4}$$

Diffusion type immobile

Based on the dynamic localization precision, a D_{min} threshold is determined. All trajectories with a diffusion coefficient smaller than D_{min} are classified as immobile.
The dynamic localization precision is determined per cell with eq. 5 by averaging the y-intercepts and the diffusion coefficients of all trajectories within the cell. The third quartile of dynamic localization precisions is plugged into eq 6 to calculate D_{min}.

$$\sigma_{dyn} = \sqrt{\frac{\langle MSD(0)\rangle + \frac{dof}{3}\langle D\rangle \cdot dt}{dof}} \tag{5}$$

$$D_{min} = \frac{\sigma_{dyn}^2}{dof \cdot n \cdot dt} \tag{6}$$

Diffusion types confined and free

Particles that are not immobile are categorized as confined or free based on the analysis by Rossier *et al.*[7]. For each trajectory a fraction a of the MSD values are fitted with eq 7 and compared to a $\tau_{threshold}$ value. If $\tau < \tau_{threshold}$ the particle moves confined, if $\tau > \tau_{threshold}$ the particle moves freely. In some cases this fit does not converge and particles are labeled with no diffusion type. $\tau_{threshold}$ depends on the camera integration time dt, the fraction of MSD values a, the minimum length of trajectories l_{min}, and a factor 0.5.

$$MSD(\Delta t) = \frac{4}{3} r_c^2 \cdot (1 - e^{-\Delta t/\tau}) \quad \rightarrow \quad \tau = \frac{r_c^2}{3D_c} \tag{7}$$

$$\tau_{threshold} = dt \cdot a \cdot l_{min} \cdot 0.5 \tag{8}$$

Statistics Mean values and their SEMs are calculated per cell (eq. 9 and 10). A global value is calculated by averaging the mean values of the cells (eq. 11). The error of the global value is calculated by averaging the SEMs of the cells, according to Gaussian error propagation (eq. 12).

$$Y_i = \frac{1}{M} \sum_{j=1}^{M} y_j \tag{9}$$

$$\Delta Y_i = \frac{\sqrt{\frac{(m_j - \bar{m})^2}{M-1}}}{\sqrt{M}} \tag{10}$$

$$Y_{global} = \frac{1}{N} \sum_{i=1}^{N} Y_i \tag{11}$$

$$\Delta Y_{global} = \frac{1}{N} \sum_{i=1}^{N} \frac{\delta Y_{global}}{\delta Y_i} \cdot \Delta Y_i = \frac{1}{N} \sum_{i=1}^{N} \Delta Y_i \tag{12}$$

4.2.2 TrackAnalysis

TrackAnalysis handles tracked files from swift and PALMTracer. A directory from which all tracked files with fitting file ending are loaded has to be chosen (swift: *.tracked.csv, PALMTracer: *.trc).

PALMTracer creates multiple *.trc files and some of them do not contain tracking information. Either copy all *.trc files that contain the tracking information in one folder or use the option “mask words” that will ignore all files with the fitting file type but containing at least one of the mask words in the file name. Comma separate multiple

mask words. Typically, mask words only have to be considered for PALMTracer data.

For the analysis, several parameters must be set. Therefore, files from one measurement condition should be analyzed at once. Parameters are the pixel size in nm, the amount of pixels on the camera, the camera integration time in s, the cell sizes, the number of MSD values that are considered for the linear fit to determine the diffusion coefficient, the % of the MSD values that are used for the fit that distinguishes between confined and free trajectories, the degree of freedom (2D tracking = 4), the minimal detectable diffusion coefficient, and the minimal trajectory length (all trajectories shorter are ignored) have to be set.

A specialty of swift is the distinction of different diffusion modes within a trajectory called segments (fig. 24). It is recommended to execute the diffusion analysis with segments instead of complete trajectories. However, PALMTracer data does not differentiate different diffusion modes and can only be executed on the track level.

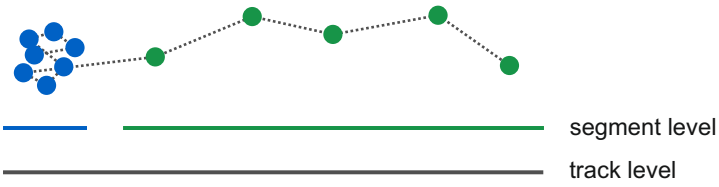

Fig. 24: Track and segment level of a trajectory. Segments represent different diffusion modes. The track level comprises the segments and represents the whole trajectory.

Get detailed information about a trajectory with “choose trajectory to plot”. “Plot global diffusion histogram and MSD plot” yields a diffusion histogram and a MSD plot averaged per diffusion state and cell. The diffusion coefficients are represented in a histogram with the normalized frequency plotted against the diffusion coefficient with logarithmic bin size (fig. 25, 26). Frequency counts of diffusion coefficients from different cells are normalized to their cell sizes.

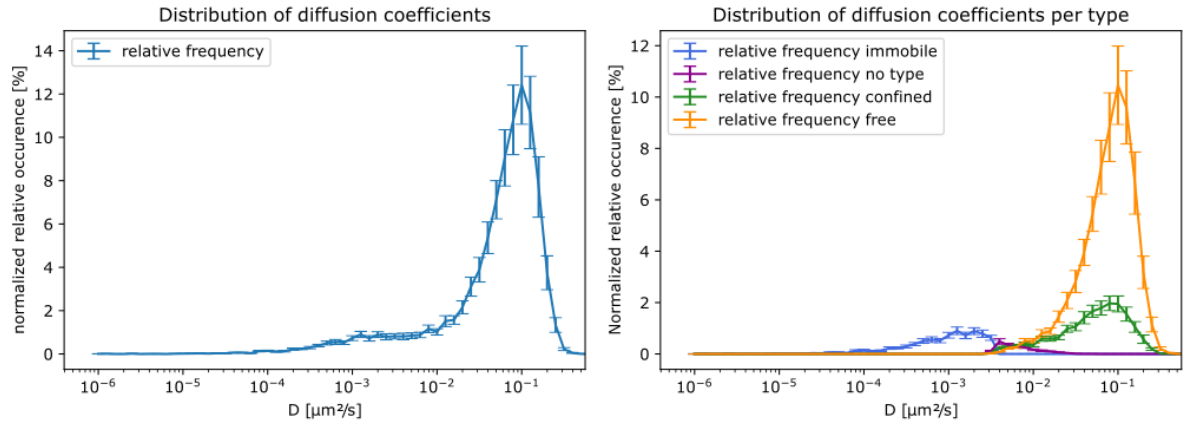

Fig. 25: Histograms of the diffusion coefficients from all analyzed files. The normalized occurrence in % is plotted against the diffusion coefficients in $\frac{\mu m^2}{s}$ with standard error of the means (SEM) and logarithmic bins.

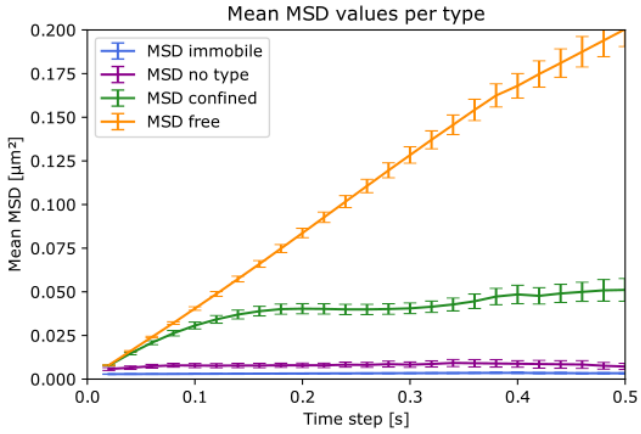

Fig. 26: Average MSD values per type and cell.

4.2.2.1 *.h5 file information

TrackAnalysis

"MSD" contains MSD Δt value pairs of all trajectories.

"diffusion" → "diffusionInfos" contains the trajectory id, diffusion coefficient and error (derived from linear fit uncertainty), y-intercept MSD(0), chi^2 (values vs linear fit) and trajectory length. "diffusion" → "diffusionPlots" contains the first n MSD Δt value pairs, the linear fit and the residues.

"rossier" → "rossierStatistics" contains the trajectory id, the diffusion type (0=False, 1=True) and the parameters from eq. 7 to distinguish between confined and free diffusion, r = confinement radius, $D_{confined}$ = confined diffusion coefficient, chi^2. "rossier" →

"rossierPlots" contains the the ath fraction of MSD Δt value pairs, the fit from eq. 7 and the residues.
"settings" contains the camera integration time dt, pixel size, cell size, $\tau_{\text{threshold}}$ from eq. 8, min trajectory length, fraction a of the MSD values fitted with eq. 7, degrees of freedom dof, dynamic diffusion error based on eq. 5 and if analysis is based on track or segment (fig. 24).
"statistics" → "statistics_4" references to the four diffusion types immobile, confined, free, and notype. This section contains the type % within the cell, the mean diffusion coefficients, trajectory lengths and respective SEMs per diffusion type.

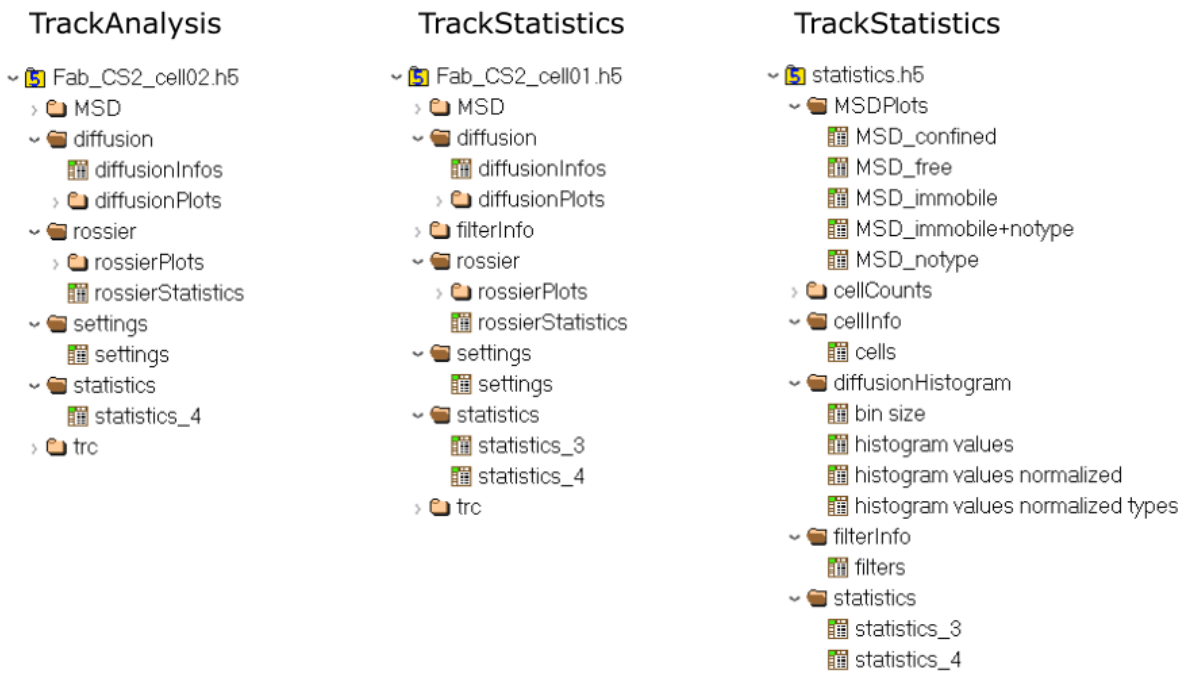

Fig. 27: TrackAnalysis saves a *.h5 file per cell. TrackStatistics saves a *.h5 file per cell containing the filtered trajectories and a statistics file that gives an overview of the filtered data of all loaded files in TrackStatistics.

4.2.2.2 Cell sizes

In order to normalize the frequency counts of the logarithmic diffusion coefficient histograms the cell sizes must be known. The measured cell sizes in sec. 2.1 are combined to a *.LOG file (fig. 28), where the first column defines the file names that must match the file names of the tracked files (with or without file ending, both is possible) and the second column the cell area in px^2. If tracked files could not be matched with names in the list of the cell areas, the size of the field of view is used as target size.
The *.LOG file can be created with the script write_cell_size_log.py + the config fi-

le write_cell_size_log_config.ini for multiple coverslips at once (fig. 29, listing 12). The swift_paramameter.py script from sec. 4.1.2.1 also creates the *.LOG file. PALMTracer automatically creates a file in the tracking routine.

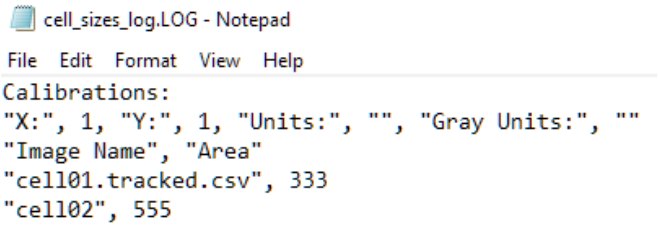

Fig. 28: Define cell sizes in pixels^2 in the PALMTracer .LOG file format.

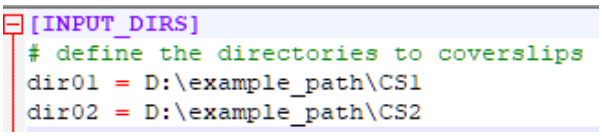

Fig. 29: Config file for write_cell_size_log.py script. Multiple directories are possible.

Listing 12: Execution of script in Anaconda prompt to write log file with cell sizes for SPTAnalyser.

```
python write_cell_size_log.py write_cell_size_log_config.ini
```

4.2.3 TrackStatistics

The *.h5 files from trackAnalysis are loaded in the trackStatistics Jupyter Notebook by choosing a directory (all *.h5 files within the folder and subfolders are loaded). Filtering options are available (fig. 30). Trajectories can be discarded by a minimal and maximal trajectory length, a minimal and maximal diffusion coefficient, and based on their diffusion type immobile, confined, free, and no successful type determination. The filtered dataset can be visualized similarly to TrackAnalysis and stored in *.h5 files along with some statistical informations. The diffusion types immobile and notype are treated separately and statistic information is calculated for four groups (immobile, confined, free, and notype). Additionally, the immobile and notype fraction are merged to one group and statistic information is calculated for three groups (immobile+notype, confined, and free).

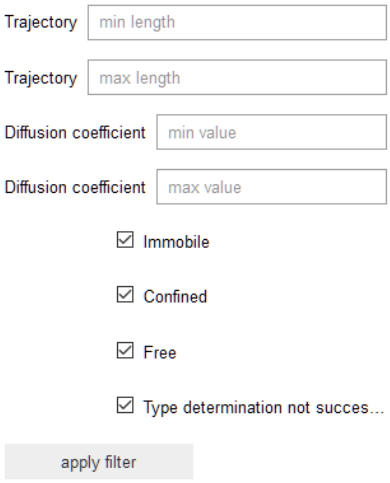

Fig. 30: Filter options in TrackStatistics.

4.2.3.1 Determination of D_{min}

A D_{min} value based on the filtered data is required to determine if a particle is immobile. All trajectories/segments with a diffusion coefficient less than D_{min} are labeled as immobile. D_{min} is derived from the dynamic localization error σ_{dyn} and can be easily computed by using the trackAnalysis and trackStatistics.

The localized files are loaded in trackAnalysis and analyzed without considering the default D_{min} value. The resulting *.h5 files are loaded into the TrackStatistics Jupyter Notebook and filtered. Based on the filtered dataset, σ_{dyn} is calculated per cell. The third quartile of σ_{dyn} results in a representative D_{min} value (eq. 13), where $dof = 4$ for 2D movement, n = number of points fitted in MSD to calculate the diffusion coefficient (typically 4), dt = camera integration time.

$$D_{min} = \frac{\sigma_{dyn}^2}{dof \cdot n \cdot dt} \tag{13}$$

Workflow: track files with default D_{min} value in trackAnalysis → filter files in trackStatistics → calculate D_{min} → track files with correct D_{min} value in trackAnalysis

4.2.3.2 Background correction

Background correction is optionally applied to the diffusion coefficient histogram. Background measurements are analyzed like normal measurements and h5 files created with trackAnalysis. In the trackStatistics Jupyter Notebook, background measurements can be loaded under “load background .h5 files”. Their counts in the logarithmic diffusi-

on coefficient histogram are subtracted from the counts of the measurements. Both a background-corrected diffusion histogram and a histogram without correction are saved.

4.2.3.3 *.h5 file information

In TrackStatistics the same file per cell like in TrackAnalysis is created. This file contains only the filtered trajectories and additional filter information in "filterInfo". "statistics" → "statistics_4" references to the four diffusion types immobile, confined, free and notype. In "statistics_3" all immobile and notype trajectories are considered as immobile.

Additionally, a statistics file is created that summarizes information of all *:h5 files that were loaded into TrackStatistics.
"MSDPlots" contains the average of all MSD values per type and cell. "cellCounts" contains the frequency counts / area of diffusion coefficients with logarithmic bin size per cell. "cellInfo" contains the names of the loaded *.h5 files and the cell sizes. "diffusionHistogram" contains the bin size in log-space, the mean histogram values and SEM (not normalized, normalized, distinguished per type and normalized). "filterInfo" summarizes the applied filters. "statistics" contains the averages of diffusion types, diffusion coefficients and trajectory lengths. → "statistics_4" references to the four diffusion types immobile, confined, free and notype. In "statistics_3" all immobile and notype trajectories are considered as immobile.

4.2.4 *.h5 file automated column extraction

Values regarding defined columns of a *.h5 file can be extracted with the extract_h5_files.py script for multiple files. Define a folder that contains the *.h5 files. Define the folder, file name, and column index of the target column, for example "rossier", "rossierStatistics", "8", to extract the confined radii. The confined radii will be saved in a *_values.csv file, each column is a target. An additinal file *_mean.csv is created that contains mean, standard deviation and standard error per target column. If the target column is a single value (for example "statistics", "statistics_3", "1", to target the percentage value of immobile and notype segments) the values are saved in a *_single_values.csv file with standard deviation and standard error. The parameter mask_files allows to define *.h5 file names that will be ignored. Define a folder and a file name to save results.

```
[INPUT_DIR]
# define the folder that contains the hdf5 files (results from trackAnalysis / trackStatistics)
dir = D:\example_folder

[PARAMETERS]
# define the folder file and column index (start counting from 1),
# mask files are h5 files with matching name in directory that will be ignored
folder_name = statistics
file_name = statistics_3
column_idx = 8
mask_files = [statistics, Target1]

[SAVE]
# define the directory and file name for result saving
save_dir = D:\example_folder
save_name = results
```

Fig. 31: Config file for extract_h5_files.py script.

4.2.5 Batch processing of trackAnalysis and trackStatistics

Multiple coverslips can be processed at once using the Dmin_autocalculator and trackAnalysisStatistics_automizer.py. The first script is used to calculate D_{min} and the second script executes trackAnalysis and trackStatistics.

```
# config file for Dmin_autocalculator.py. Determines the minimal
[SOFTWARE]
# choose software, either ThunderSTORM or rapidSTORM
software = ThunderSTORM

[INPUT_DIRS]
# path to the coverslips
dir01 = D:\exampledir

[MASK_WORDS]
# words to filter in your directory, e.g. BACKGROUND
mask = background

[BACKGROUND]
# whether to use background correction.
background = false

[BACKGROUND_DIRS]
# directories to the background, make sure they are in the same o
dir01 = ""

[CAMERA]
# camera parameters: pixel size in nm, camera integration time in
pixel_size = 158
camera_integration_time = 0.02
background_size = 65536

[DIFFUSION_TYPE_PARAM]
# parameters for the analysis of the diffusion types. min_detecta
number_of_points = 4
MSD_fit_area = 0.6
degree_of_freedom = 4
min_detectable_D = 0
min_track_length = 20

[ID_TYPE]
# whether to analyze by segments or whole tracks (see manual) val
id_type = seg id

[FILTER_PARAMETERS]
# parameters to filter the statistics by
min_trajectory_len = ""
max_trajectory_len = ""
min_D = 0
max_D = ""
filter_for_immobile = true
filter_for_confined = true
filter_for_free = true
filter_for_noType = true

[SAVE_DIR]
# where to create the trackAnalysis folder
save = D:\example\savedir
```

Fig. 32: Config file for the Dmin_autocalculator and trackAnalysis_automizer script, only the value for D_{min} needs to be changed between the scripts.

4.2.5.1 D_{min} autocalculator

The script Dmin_autocalculator.py creates the files sigma_dyn_per_cell.csv and D_min.csv,

the former holding the values of σ_{dyn} per cell, the latter containing the value of the third quartile of σ_{dyn} that was used to calculate D_{min}, as well as the value for D_{min}.

Listing 13: Execution of script in Anaconda prompt to run the determination of D_min .

```
python Dmin_autocalculator.py Dmin_autocalculator_config.ini
```

4.2.5.2 trackAnalysisStatistics automizer

With the value for D_{min} calculated by the previous script, this script runs trackAnalysis (execute MSD-based analysis) and trackStatistics (filter MSD-based analysis) and saves output from the latter. It is recommended to use it to filter for one parameter and use the trackStatistics jupyter notebook if the filter parameters should be changed. This is due to the fact that the script reruns the trackAnalysis with each execution which very time consuming compared to the trackStatistics notebook.

Listing 14: Execution of script in Anaconda prompt to run the batch analysis of data .

```
python trackAnalysisStatistics_automizer.py trackAnalysisStatistics_automizer_config.ini
```

4.2.6 Batch processing summary

- Create the required folder structure (sec. 1.3)
- Mark regions of interests (sec. 2.1)
- Run ThunderSTORM with Fiji macro (sec. 2.2.2)
- Determine the swift parameters precision, exp_noise_rate, and diffraction_limit with swift_parameter.py (sec. 4.1.2.1)
- Determine the swift parameters exp_displacement and p_bleach with swift_parameter_converger.py (sec. 4.1.2.3)
- Calculate D_{min} with Dmin_autocalculator.py (sec. 4.2.5.1)
- Execute the MSD-based analysis with trackAnalysisStatistics_automizer.py (sec. 4.2.5.2)

4.3 Time resolved analysis

This script calculates MSD-based analysis in a time resolved fashion to unravel time-dependent trends in SPT movies. It can calculate means and errors for either time or cell intervals across coverslips, sorts data into a new HDF5 file sorted by parameter, and plots it. Furthermore it runs statistical tests. This is done for diffusion types, the diffusion coefficients, confinement radii, trajectory lengths and number of trajectories based on the data obtained from executing TrackStatistics. Furthermore the script pools relevant data on single cells in the same HDF5 files so that further analysis is quicker. This is done for confinement radii, fractions, diffusion coefficients, number of segments, and segment lengths. The script creates a "TR_Analysis" folder in the save directory which contains two folders and an h5 file.

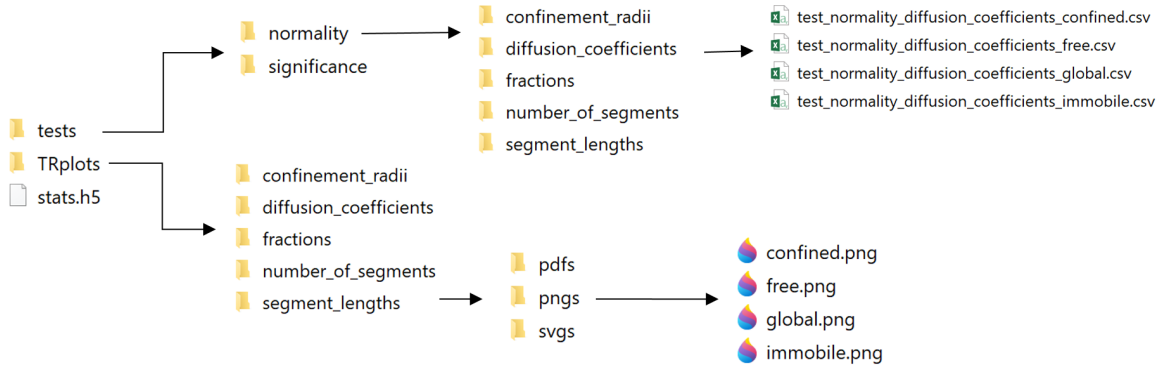

Fig. 33: Folder structure created by the timeResolvedAnalysis script

4.3.1 stats.h5

The stats.h5 file contains four folders: a bin folder, which contains the mean parameters of the bins of each coverslip, a metadata folder which contains a sheet of the data with the settings of the config file, at raw folder, which contains all the data that has been read in, with coverslips arranged next to each other in the same rows in each sheet, and a "stacked" folder, which contains all the data that has been read in which the coverslips arranged atop each other in the same columns.

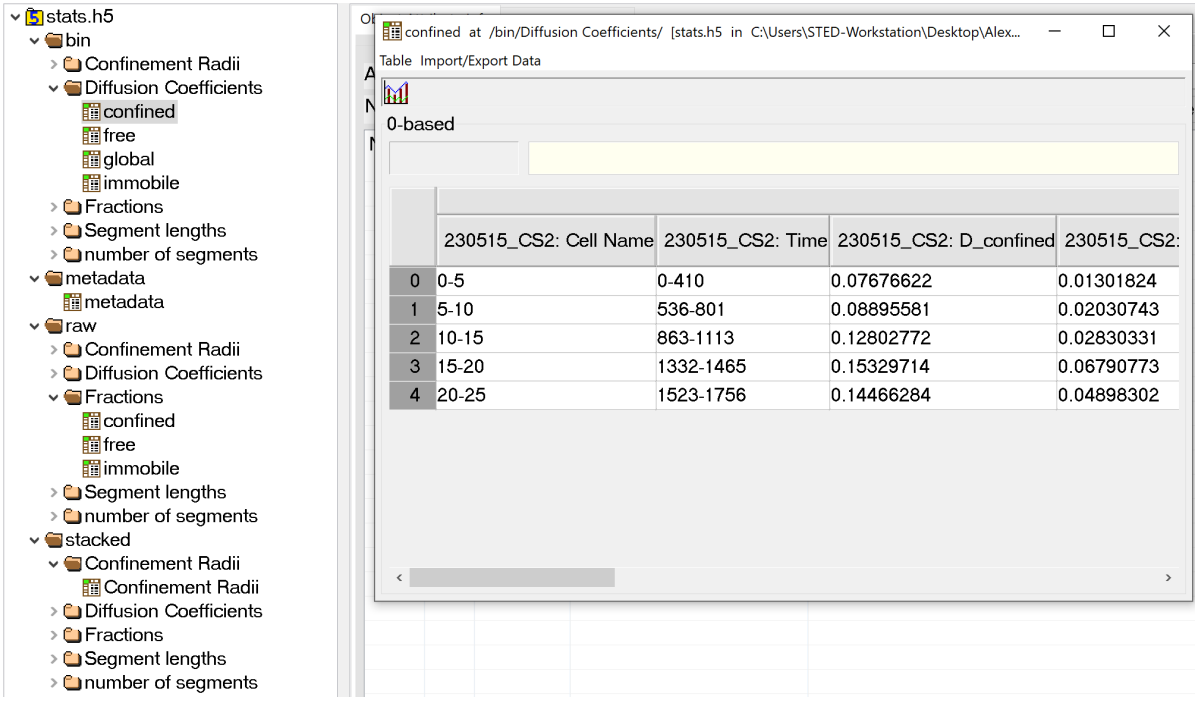

Fig. 34: Structure of the .h5 file, the binned diffusion coefficients of the confined fraction are opened on the right.

4.3.2 tests

tests contains .csv files with statistic tests done on all time bins across all coverslips. For normality tests, these are the Shapiro-Wilk and the Kolmogorov–Smirnov test and for the significance the paired t-test and the Wilcoxon-signed-rank test. Significance tests are only done from the second bin onwards and are compared to the first bin. Interval levels for the statistic can be set in the config file.

4.3.3 TRplots

This folder holds all plots the program generates. These are scatter plots, overlaid with bars that represent the mean over a timeframe for all given coverslip. It can either be plotted by time (use_timestamps = true) or by cell number (use_timestamps = false).

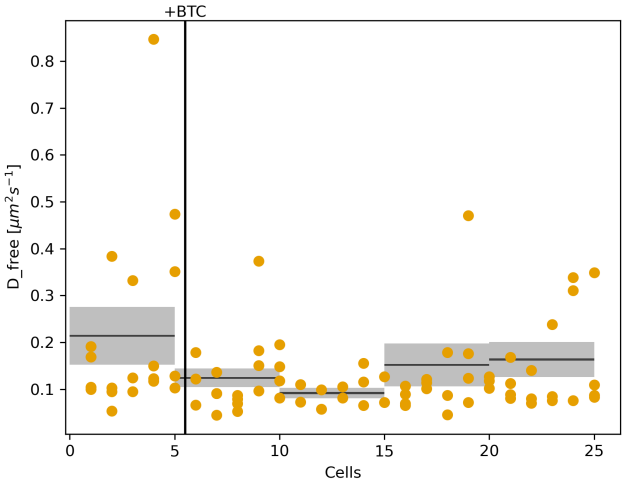

Fig. 35: Example of a plot generated by the timeResolvedAnalysis script. Used parameters: use_ timestamps = false, bin_size: cells = 5, ligand_index = 5, ligand_name = +BTC, error_type = SEM.

```
# creates means over bins of data by cell number or time, plots them
[GLOBAL_DIR]
# define the directories to the .h5 files. They MUST have their asso
# They must have been created by filtering for every parameter (glob
dir = D:\exampledir\filteredData

[USE_TIMESTAMPS]
#whether or not to use timestamps instead of cell numbers for binnin
use_timestamps = false

[CS_DIRS]
# directories of the analyzed coverslips
dir1 = D:\exampledir\CS1

[BIN_SIZE]
# size of the bins over which to calculate means. Use cells if use_t
cells = 5
minutes = 5
seconds = 0

[PLOT_SETTINGS]
# color for plots in hexcode, time before ligands were added (leave
dot_color = #E69F00
ligand_index = 3
ligand_name= +LIG
error_type = SD

[STAT_SETTINGS]
# interval values for the statistical tests
run_stats = true
alpha_norm = 0.05
alpha_sign_1 = 0.05
alpha_sign_2 = 0.01
alpha_sign_3 = 0.001

[SAVE_DIR]
# where to create the TR_Analysis folder. Note that the folder will
svdir = D:\examplesavedir
```

Fig. 36: Config file of the timeResolvedAnalysis script.

Listing 15: Execution of script in Anaconda prompt to run the time resolved analysis.

```
python timeResolvedAnalysis.py timeResolvedAnalysis_config.ini
```

4.4 Transition counts

In this notebook, the transitions of diffusion states within of segments within trajectories are counted. Required input files are matching (=same file name) *.h5 files form trackStatistics and *.tracked.csv files from swift (fig. 37).

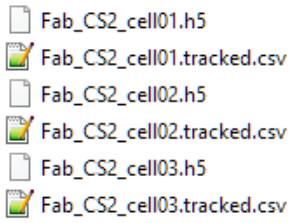

Fig. 37: Required input files for transition counting. Define the directory path containing matching files (=same filename) from trackStatistics (*.h5) and swift (*.tracked.csv).

For each single trajectory in which at least two segments were identified, the transition of the diffusion state between the segments is determined. For the three diffusion states of immobile (i), confined (c), and freely diffusing (f) particles, nine different transitions are distinguished: i-i, i-c, i-f, c-i, c-c, c-f, f-i, f-c, f-f. The number of diffusion state can be set to 4 to investigate no-type transitions. Unclassified segments that occurred between two segments can be neglected by defining a mask value. All segments with a length <= mask value in frames are skipped, and the transition between the segment before and after is counted. A mask value of 0 means no masking. A recommendation is to set the mask value to the classification threshold of diffusion states (e.g. if the diffusion type of segments with a minimal length of 20 frames is classified, mask value = 19). Results of transition counting are shown in fig. 38. Fig. 38 (A) shows the absolute counts per cell. In fig. 38 (B), transition counts were normalized per cell and summed to one to compare the occurrences of transition types. In fig. 38 (C), transition counts were normalized per diffusion state so that the counts of transition types proceeding from the same diffusion state sum up to one to compare the occurrences of diffusion states in adjacent segments. Further plots show the length of segments in frames, the number of segments per trajectory, and the number of counts without / with transitions that involve short, unclassified segments (fig. 39).

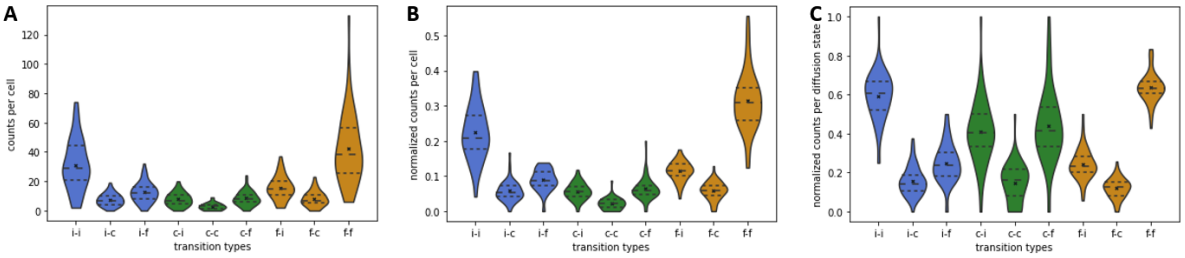

Fig. 38: Transiton counting results in (A) absolute counts, (B) counts normalized to one per cell, and (C) counts normalized to one per diffusion state.

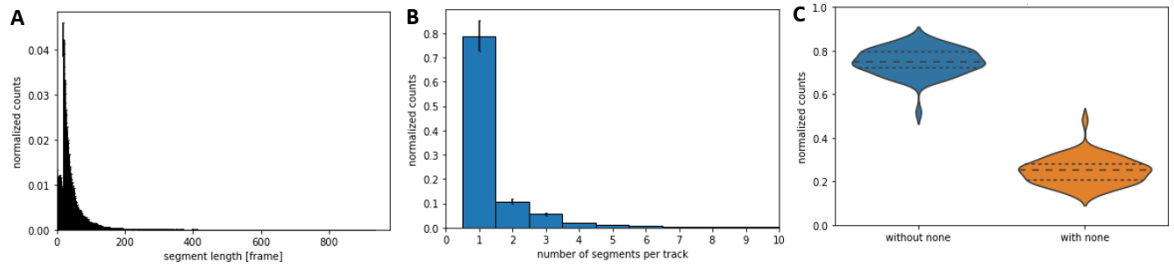

Fig. 39: (A) Lengths of segments within trajectories that contain at least one classified segment. (B) Number of segments per trajectory. (C) Relative occurence of transition counts without unclassified segments and with classified segments.

Plot axis ranges can be adjusted via the code cells (fig. 40). Significance tests of pairwise comparisons of the transition types are automatically saved and use Wilcoxon signed rank tests (SciPy). Levels of significance are classified as follows: $p > 0.05$ no significant difference (n.s.), $p < 0.05$ significant difference (*), $p < 0.01$ very significant difference (**), $p < 0.001$ highly significant difference (***).

```
# adjust axis min max limits of plots here, e.g. ylim=[0,1]
stats.vis_counts(counts="absolute", norm="absolute", ylim=[None, None])
stats.vis_counts(counts="absolute", norm="global", ylim=[None, None])
stats.vis_counts(counts="absolute", norm="split", ylim=[None, None])
stats.segment_lengths_plot(xlim=[0,None], ylim=[0, None])
stats.segments_per_trajectory_plot(xlim=[0,10], ylim=[0, None])
stats.transitions_wo_none_plot(ylim=[0,1])
```

Fig. 40: Show code cells and adjust the axis ranges via xlim and ylim parameters.

4.5 Hidden Markov Modeling

In this notebook, hidden Markov modeling[G] (HMM) is applied to cells to extract a number of diffusion states with diffusion coefficients, population probability and transition probabilities between the states. Required input files are tracked files from swift (*.tracked.csv), a model is trained per cell and final results are averaged over all cells. The notebook is divided into two sections, "Initial parameter estimation" to estimate the number of states and yield initial parameters and "Train hidden Markov model" to train a HMM on cells. HMM computation, especially with a high number of hidden states, takes a while.

4.5.1 Initial parameter estimation

Following parameters are required to train a HMM: number of hidden states and initial guesses about state weights, diffusion coefficients, and transition probabilities. Initial guesses can be extracted by other analysis such as MSD based (immobile, confined, free) or based on jump distance mixture modeling (eq. 14, 15), which is realized in this section.

$$p(r) = \sum_{i=1}^{n_{max}} \omega_i \cdot \frac{2r}{MSD_i} \cdot e^{-\frac{r^2}{MSD_i}} \tag{14}$$

$$MSD_i = 4D_i dt \tag{15}$$

The observation sequence in an HMM are jumps between two adjacent localizations, computed from all tracked files in the defined input directory. Within a defined range of number of states a jump mixture model is fitted to the jump distance distribution (jdd) per cell. Fig. 41 shows mixture models of 1 to 4 subpopulations with a kernel density function of the jdd. Each mixture model yields weights and diffusion coefficients that can be used as initial parameters for the HMM. In addition, information criteria (log likelihood - the higher the better; BIC, AIC, AICc - the lower the better) evaluate model quality and suggest a suitable number of states (fig. 42). Further hints about the number of states are elbow points (=drastic drops of information criteria values from one number of state to the other) and addition of redundant states (e.g. in fig. 41D state 3 and 4 are the same, indicating the use of 3 states). A high number of states requires a lot of computational time, 1-5 modes are suitable for a first impression. Use the averaged weights and diffusion coefficients for initial parameters for the HMM.

[G] `https://github.com/SMLMS/pyErmine`

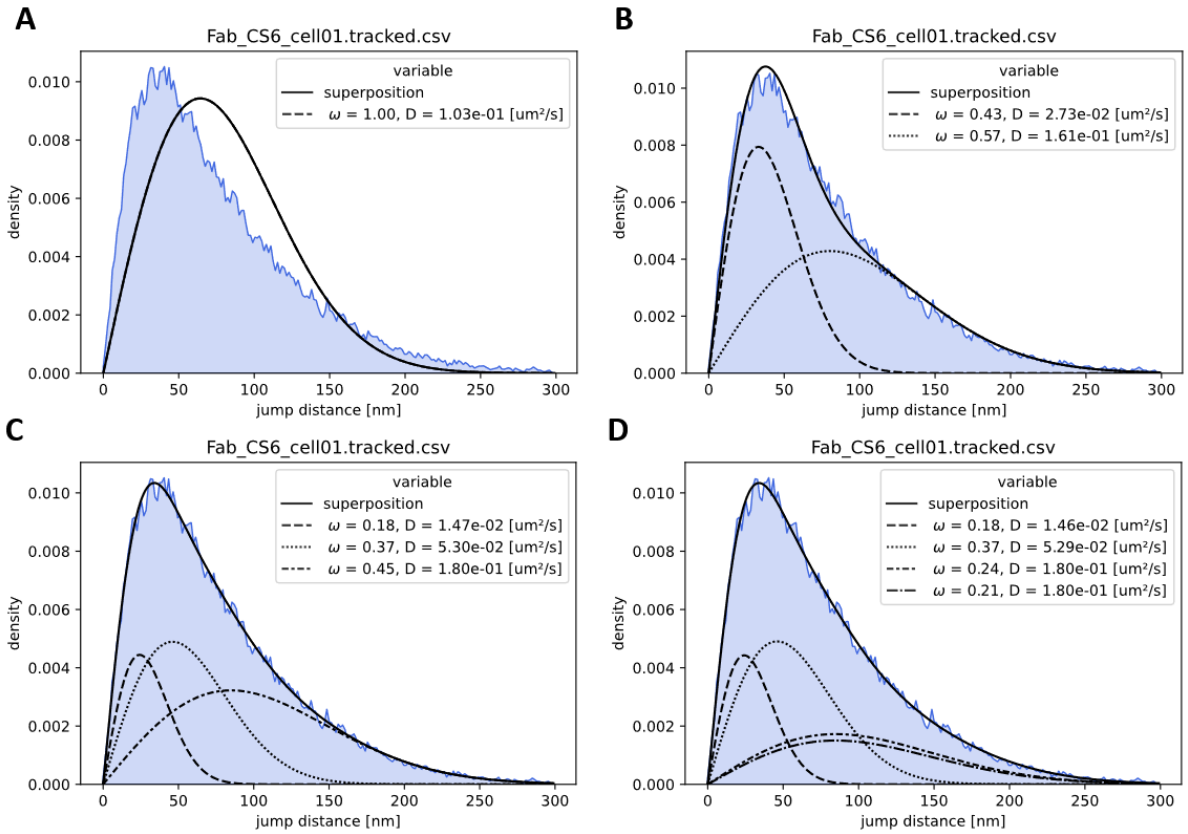

Fig. 41: Jump mixture model of n = 1 to 4 states of a cell.

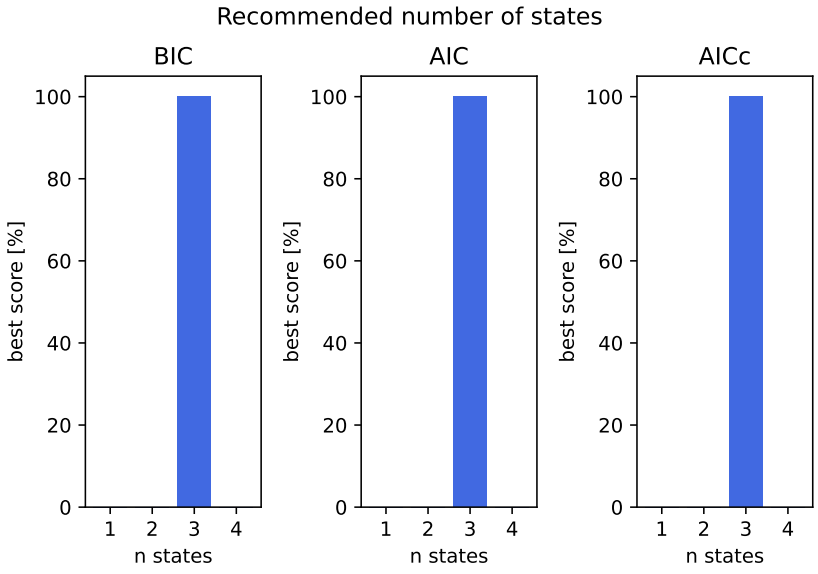

Fig. 42: The information criteria are calculated per cell and the number of states with the lowest values are counted per cell. In this case, all cells have the lowest information criteria values with n = 3. Additionally check out the values in detail, look for elbow points and redundant modes.

4.5.2 Train hidden Markov model

In this part of the notebook, an HMM is trained for all tracked cells within the defined input directory. The model describes the probability density distribution of jump widths and learns the inter-mode transition probabilities by analyzing the temporal sequence of jump distances. Results of a trained model are visualized in fig. 43. The jumps within trajectories are labeled with diffusion states using the Viterbi algorithm.

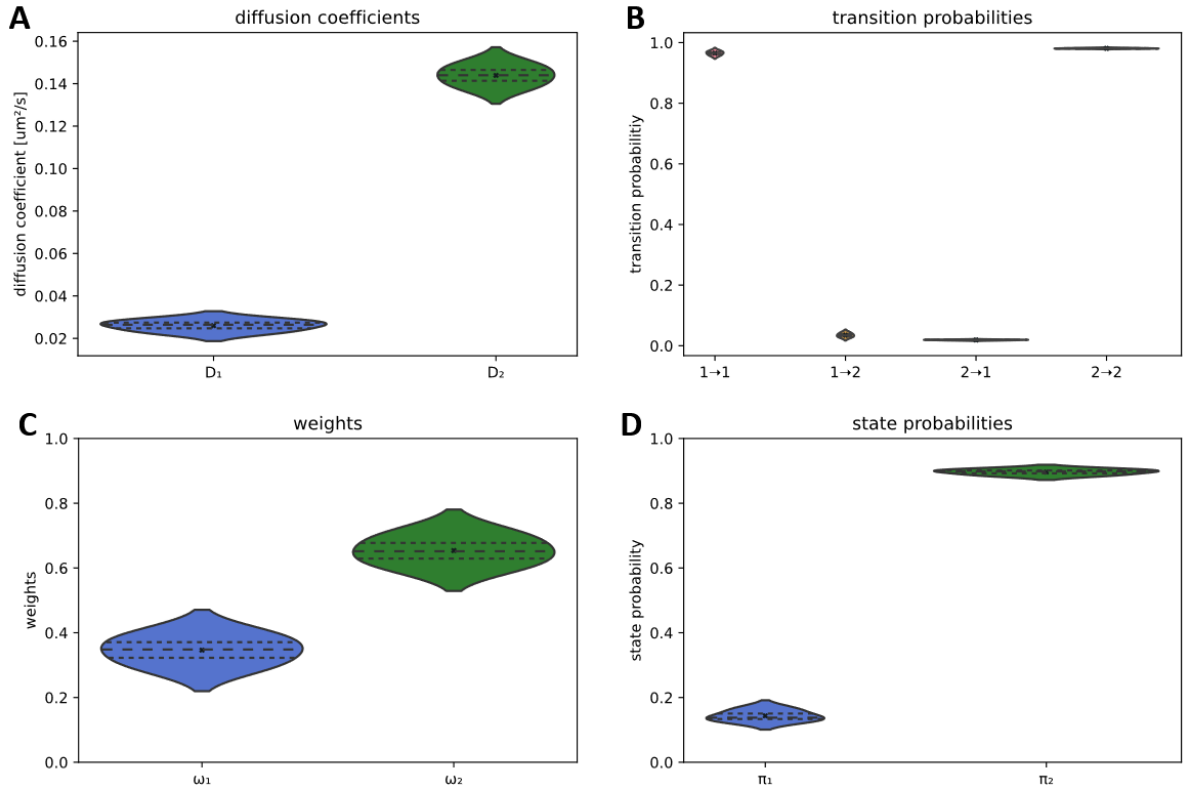

Fig. 43: Results of trained HMM (n=2) per cell. Per state the following parameters are determined: (A) diffusion coefficients, (B) transition probabilities, (C) weights based on jdd fit, and (D) state probabilities.

Different state probability estimations

There are three possibilities to estimate the state probabilities of the HMM states. The model directly yields the state probabilities π_i based on the initially given weights. However, fitting the jdd with fixed diffusion coefficients and state probabilties as weights does not fit the distribution. Therefore, a second option is to count the state labels of jumps given by the Viterbi algorithm. A third option is to train a jdd mixed model, optimizing only the weights and using the fixed diffusion coefficients from the hmm (fig. 44A).

Check for state immobility

The hidden Markov model yields apparent diffusion coefficients. The measured displacement of an immobile population would stem from the fact that the position of a fixed molecule was measured repeatedly with errors. An apparent displacement is the sum of true displacement and error (eq. 16). The true displacement is zero for an immobile population and the apparent mobility is the static measurement error (eq. 17). The error ϵ is calculated based on the immobile mode with NeNA. The value is compared to an immobility threshold, such as the localization uncertainty extracted from ThunderSTORM analysis. If NeNA from a mode is smaller than the immobility threshold, that mode can be called immobile.

$$MSD_a = MSD + 4\epsilon^2 \tag{16}$$

$$MSD_{a,i} = 0 + 4\epsilon^2 \quad \rightarrow \quad \epsilon = \sqrt{\frac{MSD}{4}} \tag{17}$$

Correct diffusion coefficients for errors

The apparent diffusion coefficients can be corrected for dynamic and static erros in this section. The dynamic error is introduced by the molecules moving between observed time periods. Eq. 18 calculates the corrected diffusion coefficients (ϵ is the static error, such as NeNA from an immobile mode or the localization uncertainty from measurements; σ equals the camera integration time dt).

$$D = \frac{MSD - 4\epsilon^2}{4dt - \frac{4}{3}\sigma} \tag{18}$$

The state population and transition probabilities are depicted in a state transition diagram, in which the circle areas scale with the state population and the arrow sizes scale logarithmically with the transition probabilties (fig. 44B). There are three options for the node size. The jdd fit weights are fixed diffusion coefficients from the trained HMM used in a jump distance mixture model that yields the weights. The state probabilities are directly yielded from the trained HMMs initial probabilities. The occurence are the counts in percent of labeled jumps of the observed sequences by the Viterbi algorithm.

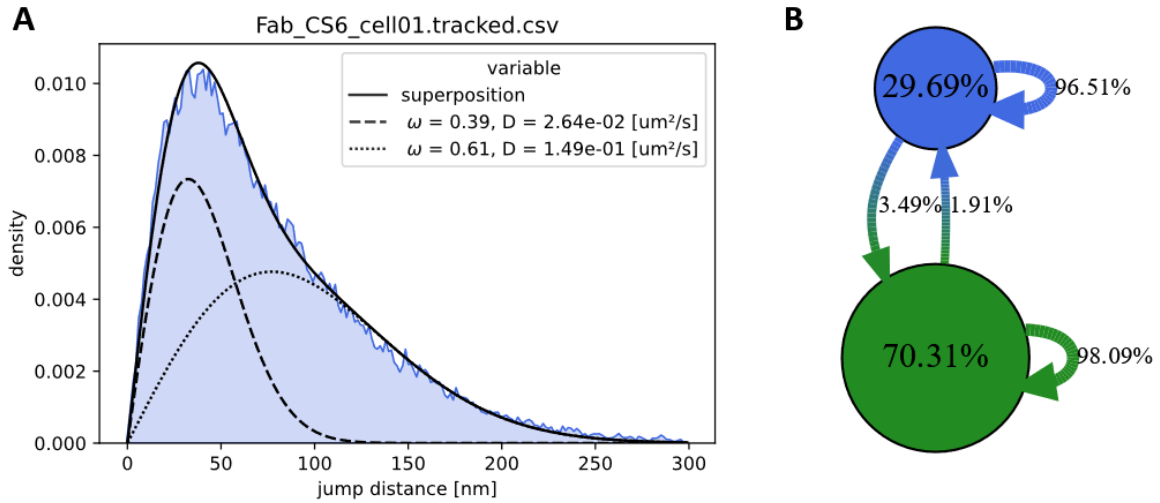

Fig. 44: (A) Jump distance distribution of a cell fit with a two state model. (B) Hidden Markov modeling of a pool of cells with two states.

5 References

1. Ovesny, M., Krizek, P., Borkovec, J., Svindrych, Z. & Hagen, G. M. ThunderSTORM: a comprehensive ImageJ plug-in for PALM and STORM data analysis and super-resolution imaging. *Bioinformatics* **30,** 2389–2390. doi:10.1093/bioinformatics/btu202 (2014).

2. Schindelin, J. *u. a.* Fiji: an open-source platform for biological-image analysis. *Nat. Methods* **9,** 676–682. doi:10.1038/nmeth.2019 (2012).

3. Wolter, S. *u. a.* rapidSTORM: accurate, fast open-source software for localization microscopy. *Nat. Methods* **9,** 1040–1041. doi:10.1038/nmeth.2224 (2012).

4. Endesfelder, M., Schießl, C., Turkowyd, B., Lechner, T. & Endesfelder, U. *manuscript in prep.*

5. Edelstein, A. D. *u. a.* Advanced methods of microscope control using μManager software. *J. Biol. Methods* **1,** e10. doi:10.14440/jbm.2014.36 (2014).

6. Michalet, X. Mean square displacement analysis of single-particle trajectories with localization error: Brownian motion in an isotropic medium. *Phys. Rev. E* **82,** 041914. doi:10.1103/PhysRevE.82.041914 (2010).

7. Rossier, O. *u. a.* Integrins β1 and β3 exhibit distinct dynamic nanoscale organizations inside focal adhesions. *Nat. Cell Biol.* **14,** 1057–1067. doi:10.1038/ncb2588 (2012).

8. Harwardt, M.-L. I. E. *u. a.* Membrane dynamics of resting and internalin B-bound MET receptor tyrosine kinase studied by single-molecule tracking. *FEBS Open Bio* **7,** 1422–1440. doi:10.1002/2211-5463.12285 (2017).