TrackNTrace *Lifetime Edition*

written for MATLAB by Simon Christoph Stein and Jan Thiart, extended by Jan Christoph Thiele

TrackNTrace is a fast, easy-to-use MATLAB framework for single molecule localization, tracking and super-resolution applications. The purpose of this software is to facilitate development, distribution, and comparison of methods in the community by providing an easily extendable, plugin-based system and combining it with an easy-to-use graphical user interface (GUI). This GUI incorporates possibilities for quick inspection of localization and tracking results, giving direct feedback of the quality achieved with the chosen algorithms and parameter values, as well as possible errors, a feature neglected in most software packages available. The plugin system greatly simplifies adapting and tailoring methods towards any research problem's individual requirements. We provide a set of plugins implementing state-of-the-art methods together with the basic program, alongside tools for common post-processing steps such as STORM image generation, or drift correction. TrackNTrace should be useful to anyone who seeks to combine the speed of established software packages such as rapidSTORM or QuickPALM with the simplicity and direct modifiablity of MATLAB, especially when further post-processing is also done in MATLAB.

In general, TrackNTrace reads a movie file, corrects for camera artifacts if applicable and obtains a rough guess of all possible positions of bright spots in every image. These position candidates then serve as the basis for another routine which refines these candidates, obtaining position, amplitude and local background. Subsequently, these results are then returned to a particle tracking algorithm which tries to link particles close in time and space to form trajectories. Finally, a post-processing can be performed to determine additional parameters based on the tracks and potentially the movie file.

This manual will first provide all necessary steps for installation (section 1) and explain how to use the GUI (section 2).

Requirements

OS	Windows 7 64-bit or higher, Linux x86_64 (tested with Kubuntu 14.4)
MATLAB version	2017a or higher
Toolboxes	Image Processing, Statistics, Parallelization (optional)

Please also note: The detection and refinement step can currently only handle single-channel, 2D + t image stacks. Create a import plugin to load your experimental data accordingly or convert them to a corresponding Tif file.

Contents

1 Installation

TrackNTrace is available via a version-controlled git repository at https://github.com/scstein/TrackNTrace. The first step is not necessary if git is already installed on your system.

1.1 Windows

1. Download and install git which is available at http://git-scm.com.
2. Open a git bash in your MATLAB folder and clone the repository via the command
   `git clone https://github.com/scstein/TrackNTrace.git` .
3. Install the Visual Studio 2012 C++ Redistributable (x64). This can either be found in the TrackNTrace folder `external\vcredist_x64.exe` or downloaded from the Microsoft website here.
4. Open MATLAB, move to the TrackNTrace folder, and execute `RunTrackNTrace`.

1.2 Linux (tested with Kubuntu 14.4)

1. Install git it via your package manager (e.g. `sudo apt get install git`).
2. Open a terminal in your home folder and clone the repository via the command
   `git clone https://github.com/scstein/TrackNTrace.git` .
3. Open MATLAB, move to the TrackNTrace folder, and execute `RunTrackNTrace`.
4. Depending on your MATLAB version, TrackNTrace might run without the need for additional configuration.
   MATLAB in Linux comes with its own C++ standard library, which might be too old and not compatible with the shared libraries used by ceres. As MATLAB loads its own STL before the system libraries (by setting `LD_LIBRARY_PATH` to a MATLAB library directory) this will result in failures when the mex file (shared library) is called. If you encounter invalid mex files while executing the program or runtime linking errors try setting the `LD_PRELOAD` environment variable to the directory with your system libraries (where `libstdc++` and `libgfortran` are located; try either the `locate` command or `find/ -name` to find them) before starting MATLAB. If you still encounter problems, consider installing the ceres dependencies (see http://ceres-solver.org/building.html, Linux).

2 Overview

Before reading this manual a note: To make TrackNTrace easy to use, we want to emphasize that every UI element has a tooltip explanation, which pops out when resting the mouse on top of the element. Ideally these explanations should be enough to use the program/plugin efficiently.

Some of the general behavior of the TrackNTrace software can be altered by individual users by editing the `getDefaultOptions.m` file. For example, the default plugins on startup can be selected and it can be chosen if the program should use parallel processing or not. To see how to run TrackNTrace without the GUI check section 2.4.

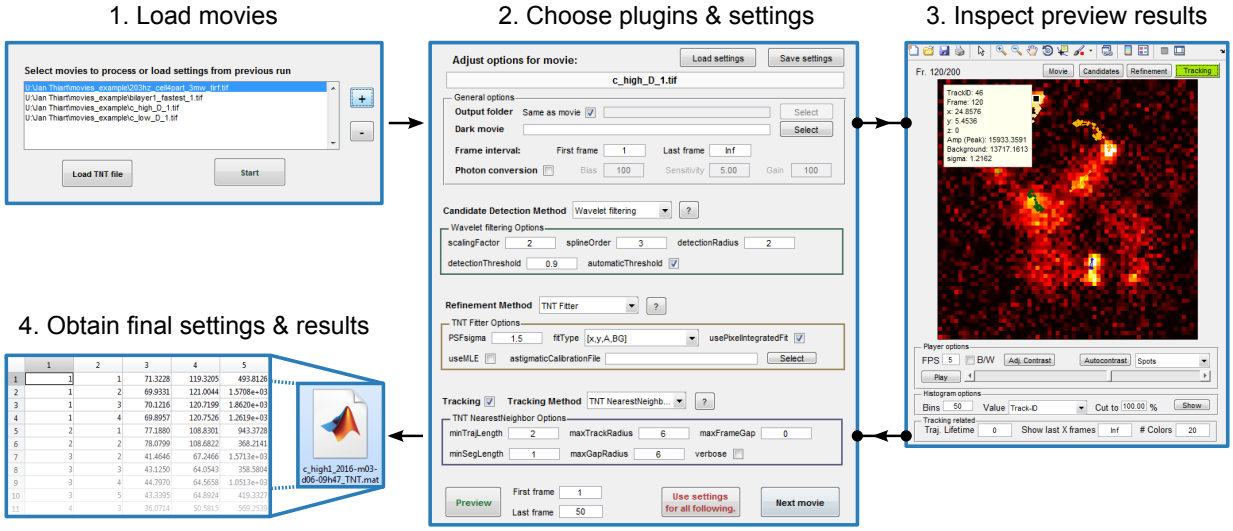

Figure 1: Program flow and user interface of TrackNTrace. First, a list of movies or a previously saved settings file is loaded before the main GUI is initialized. There, plugins for candidate detection, position refinement, and particle tracking are chosen and their settings adjusted for each movie. At any time during parameter tuning, a preview for an arbitrary part of the current movie can be computed and visualized. The visualizer is able to display the output from all stages (here shown in tracking mode). Selecting a candidate, localization, or track showcases the respective plugin-specific output (e.g. fitted parameter values). After parameter adjustment for all movies, the actual processing starts, saving each movie's output data along with the chosen settings in a single file.

2.1 Startup (Fig. 1.1)

When you first run `RunTrackNTrace.m`, the GUI as seen in figure 1.1 pops up. Here, you can select the relevant movie files to process. Movies are added via the "+" button, where multiple movies can be selected in one go using the "Shift" key. Likewise, the "-" button removes the selected entry from the list. Pressing Start invokes the main GUI (fig. 1.2). If "Load TNT file" is pressed, a TrackNTrace settings file (MATLAB's `*.mat` file format, files end on `_TNT`) can be selected, which loads all settings from a previous run, also starting the main GUI.

2.2 Main GUI (Fig. 1.2)

In the main GUI all processing settings are adjusted. On first startup it always shows the settings for the first movie in the list. The current settings can be saved to a file or previously saved settings loaded via the Save Settings and Load settings buttons on the top. Below the current file's name is displayed (hovering reveals full path). Next to the filename, a button opens the import options in case there are any (see section 5.1). The main part of the GUI is designed for five processing steps:

1. **Generating intensity image stack:** Measurements can be corrected for dark currents and camera artifacts. Algorithms based on quantitative analysis of photon signals (e.g. maximum-likelihood estimation, MLE) require subsequent conversion of analogue-digital-converter counts to photon numbers. If the import plugin supports TCSPC, a time gate can be defined.

2. **Detecting candidates:** Potential sources of signal above the background noise are identified in each frame to obtain rough estimates of emitter positions with pixel precision.

3. **Position refinement:** Each candidate's position estimate is refined with sub-pixel accuracy and additional information (background strength, brightness, dipole orientation, etc.) extracted. Commonly, this involves fitting a representation of the microscope's PSF to a subsection of the frame.

4. **Tracking:** Positions separated in time are connected frame-by-frame to form trajectories. High particle density, intersecting tracks, and re-appearing, previously lost emitters are the main obstacles to overcome during this stage.

5. **Post-processing:** Based on the tracks, additional properties can be determined. This step is used for lifetime fitting with the plugin fitLT.

The first part corresponds to the General options panel, where the following elements can be adjusted:

Output folder Folder where the TNT output file is saved. The filename is always derived from the movies name with an additional timestamp and the ending `_TNT`. For example: the output of a movie "`data.tif`" could become "`data_2016-m03-d14-13h15_TNT.mat`". If Same as movie is checked, the output will be right next to the movie. Otherwise press Select to open a file dialogue where you can select a folder (or simply type one into the field directly). Leaving the field empty saves in the current MATLAB location.

Dark movie Select a movie (*.tif only) taken with the exact same camera settings used in your experiment but with the shutter closed. This movie is used to correct for non-isotropic camera sensitivity, dead pixels and other artifacts according to [1].

Frame interval Select the first and the last movie frame for processing. Anything not in this interval will be discarded. If Last frame is higher than the movie size or set to Inf, the whole movie is processed, starting at First frame. With Binning multiple subsequent frames are merged by summing their intensity.

Photon conversion If checked, the movie frames are converted from ADC counts to photons before they are handed to the plugins for processing. The conversion formula is

$$I_{\texttt{phot}} = (I_{\texttt{ADC}} - \texttt{Bias}) \times \texttt{Sensitivity/Gain}. \tag{0.1}$$

A conversion of this type is only meaningful for EMCCD cameras. The values for `Bias`, `Sensitivity` and `Gain` should be available in the camera's performance sheet. For TCSPC based import plugins Photon conversion is automatically activated with `Bias` 0, `Sensitivity` 1 and `Gain` 1.

Timegating If checked, only photons arriving between the time bins Start and End are used to generate the intensity movie. The button Show TCSPC generates a preview. Only available if the import plugin supports TCSPC.

The lower part houses the plugins for steps 2-5, where the plugin for each step is selected using a popup menu. The [?] button besides these menus shows an explanation of the currently selected plugin. Tracking (step 4) and post-processing (step 5) can optionally be disabled/enabled, the other two steps are always carried out. That said, the `Use candidate data` plugin for step 3 simply copies the data from step 2 and converts it to the right output format. As each plugin's inputs are different, we will not explain the plugins here, but rely on their explanation and their tooltip.

At the bottom left part of the GUI the preview button is placed, which starts the visualizer (see below) for the frame interval specified by First frame and Last frame to the right of the button. This visualization is one of TrackNTrace's core features, making data analysis and parameter optimization much easier. The bottom right button either switches to the settings of the next movie in the list (displays "Next movie) or starts processing all movies (displays "START processing"). If all movies are alike, the "Use settings for all following" button applies the current movie's settings to all yet unadjusted movies in the list and starts the processing directly.

2.3 Preview/Visualizer (Fig. 1.3)

The visualizer shows the data acquired from steps 2 through 5 (if available). Aside from its use by `RunTrackNTrace.m`, the visualizer can also be started directly by executing `TNTvisualizer.m` (check the file for the different input options). This is useful for viewing data from previous runs or any kind of movie that is supported by an import plugin. In the top right the type of data to display can be selected. If a data point is selected with the mouse, all corresponding output parameters from the plugin that computed it are shown. The counter in the top left shows the current/maximum frame. A panel with player options is below the movie. If results are loaded, additional options to analyze and display the data are available at the bottom of the player in a tabbed GUI. If a raw movie file is loaded frame interval, timegating and options provided by the import plugin are shown. Controls for panning/zooming and toggling of datatip, colorbar, legend and scalebar are located in the toolbar at the top of the window.

Player options

FPS The frames per second (speed) the movie is played back with when Play is pressed.

FLIM If checked, the intensity is encoded as brightness (blend to black) and the lifetime according to the selected colormap.

invert If checked, the colormap is reversed.

Adj. contrast Shows a popup menu which allows manually changing the contrast.

Autocontrast The contrast is chosen automatically with a method chosen from the popup menu to the right. Similar to the popular `ImageJ` software, holding the Shift key during playback continously adjusts the contrast. Currently included methods are:
Spots: Emphasize highest 25% intensity values.
Min/Max: Contrast spans all values.
98% range: Cuts the lower and upper 1% of intensities.

γ Gamma correction of the intensities. $\gamma < 1$ helps to enhance lifetime contrast in darker regions.

Colormap Drop-down menu with colormaps. The colormaps heat, iso65, iso75 and rainbow are designed to be perceptionally uniform. The colormaps iso65 and iso75 are isoluminant. They are recommended for FLIM images to avoid cross-talk between lifetime and intensity.

Export Exports the current frame or movie according to the selected options. Press Ctrl and click to chose a frame range for stack export. Currently included options are:
TIFF (Stack): exports the movie (including marker) as RGB TIFF stack.
GIF (Stack): exports the movie (including marker) as GIF. Set FPS to control speed.
TIFF (Frame): exports the current frame (including marker) as RGB TIFF.
PNG (Frame): exports the current frame (including marker) as PNG.
TIFF (Stack, int): exports the raw intensity movie.
TIFF (Stack, tau): exports the raw lifetime movie.
New figure: copies the current frame to a new figure.
Please note that only TIFF (Stack, int) and TIFF (Stack, tau) preserve the original pixelsize.

Histogram

Value Parameter to histogram. What is available depends on the plugin that computed the current mode's data.

Bins Number of bins the histogram is computed for.

Cut to X % Cuts (100-X)/2 % of data from the lower and upper tails of the distribution before computing the histogram.

All frames If checked, the data from all frames is used, otherwise from the current frame.

Avg tracks If checked, the selected value is average for each track before calculating histogram.

Gauss fit If checked, a Gaussian is fitted to the histogram.

Histogram Computes and displays the histogram of the selected value.

2nd value Second parameter used for 2D and weighted histograms.

2D Hist. Computes and displays a 2D histogram of both selected values.

Weighted Hist. Computes and displays a histogram of the first selected value weighted by the second value.

Tracks (only affects Tracking mode)

Traj. Lifetime Number of frames a trajectory is visible after its detection.

Show last X frames Shows only the past X frames of each trajectory.

Colors Number of colors the trajectories are displayed in.

Filter

Expression Input field for a filtering expression. Standard Matlab syntax with support for vector operation, eg: `sigma<mean(sigma) & Amp_Peak_>2*Background`

Value List of available filter parameters.

Insert Inserts the currently selected value at the end of the filtering expression.

Apply Applies the filtering expression to the current, filtered dataset. On success the expression turns red, on error red.

Reset Resets the dataset of the current mode.

Show table Opens a table with the dataset of the current mode. By default the filtered dataset is shown. Ctrl + Click opens the unfiltered dataset and colors rows filtered out dark.

Export Exports the dataset of the current mode as csv or xlsx. By default the filtered dataset is used. Ctrl + Click exports the unfiltered one.

Reconstruct

Method Method of reconstruction. Currently included are:

Gaussian: Draws a pixel integrated, normalized Gaussian (σ=precision) at the position of each localization.

Weighted: 2D histogram of localization positions, each weighted with 1/precision.

Jitter: draws a point with a normal distributed random offset (σ=precision) for each localization. This is repeated n (default: 5) times.

Points: 2D histogram of localization positions.

Create Performs the selected reconstruction and displays it in a new window.

Value Property used for the mean reconstruction.

Mean Reconstructs an image where the number of localization is given in the first channel and the local average of the selected value in a second channel. Use the FLIM option of the TNTVisualizer to view the second channel.

Pixelsize Pixelsize of the movie in nm.

SuperRes Ratio between the pixelsize in the reconstructed image and the original movie.

Loc. precision Localization precision in nm. For NaN the localization precision is estimated for each localization based on a modified Mortensen formula. (See [2])

Avg tracks If checked, position and selected value are averaged for each track before the reconstruction.

Drift

Method Method for localization based drift correction. Currently included are:

RCC: redundant cross-correlation (see [3])

DCC: direct cross-correlation (see [3])

MCC: mean cross-correlation (see [3])

Calculate Calculates drift with the selected method based on the current, filtered dataset.

Export Exports the calculated drift as csv or xlsx.

Import Imports drift from a csv file. The file has to contain a header and the columns x, y and frame. If frames are missing the drift is interpolated.

Show Plots the calculated/imported drift and the currently applied drift in case it is different.

Apply Corrects the drift in the current dataset using the calculated/imported drift.

Resolution Resolution upscale of the intermediate images for drift correction. A higher resolution allows to detect smaller drift but requires more localization

Δr max Error threshold for recalculating the drift in px. (Only used by RCC)

Seg. length Number of frames in each segment. A lower number increases the time resolution but reduces the localizations per segment and therefore the spatial resolution.

Reset Removes the drift correction from the current dataset.

2.4 Headless mode

TrackNTrace can also run without interactive user input. This headless mode is useful for batch processing of many files or systematic variation of options. For this `RunTrackNTrace` is called with input arguments as shown in the code example:

```
moviefiles = {'movie_1.tif','movie_2.tif','movie_3.tif'};
% This template can be genreated using "Save settings" in the settings GUI
tnttemplate = load('template_TNT.mat');
% If required, the plugin options can be manipulated.
tnttemplate.candidateOptions.PSFsigma = 1.5;
% Convert template to a cell array.
tntargs = [fieldnames(tnttemplate) struct2cell(tnttemplate)]';
% Run TrackNTrace: All files are processed with the same options
tntfiles = RunTrackNTrace(moviefiles,tntargs{:},...
    'showStartupGUI',false,... Disables the Startup GUI (file selection)
    'showSettingsGUI',false... Disables the Settings GUI
    );
```

3 TrackNTrace output

All results obtained by TrackNTrace are saved in a `mat`-file in the format `'moviename_timestamp_TNT.mat'`. Here is a list of all variables it contains:

TrackNTrace data structures

Variable name	Description
`filename_movie`	String containing full path of movie file.
`metadata`	Struct returned by the import plugin containing information about the movie.
`dark_img`	2D double array containing correction image to be added to each movie frame.
`movieSize`	1×3 matrix saving the size of the movie [rowPixel, colPixel, nrFrames]
`firstFrame_lastFrame`	1×2 matrix [first frame, last frame] saving the first and last processed frame of the movie. This is important if only parts of a movie were read in and processed.
`globalOptions`	Struct of general options variables set by the GUI.
`candidateOptions`	Struct of candidate options parameters. Also contains the used plugins name and a parameter description for all output columns in `outParamDescription`.
`refinementOptions`	Struct of refinement options parameters. Also contains the used plugins name and a parameter description for all output columns in `outParamDescription`.
`trackingOptions`	Struct of tracking plugin parameters. Also contains the used plugins name and a parameter description for all output columns in `outParamDescription`.
`postprocOptions`	Struct of post-processing plugin parameters. Also contains the used plugins name and a parameter description for all output columns in `outParamDescription`.
`candidateData`	$n \times 1$ cell array of candidate positions where n is the number of analyzed frames. Each cell contains a $k \times p$ double array, where p is the number of model parameters and k is the maximum amount of particles in the respective frame. Each row represents a unique candidate fit and the column order is μ_x, μ_y. These columns are mandatory for the refinement to work. Plugins can output extra data.
`refinementData`	$n \times 1$ cell array of refined positions where n is the number of analyzed frames. Each cell contains a $k \times p$ double array, where p is the number of model parameters and k is the maximum amount of particles in the respective frame. Each row represents a unique fit and the column order according to the model PSF is μ_x, μ_y, μ_z, A, B (amplitude, background). These five columns are mandatory for most trackers to work correctly in step 4. Plugins can output extra data.
`trackingData`	2D double array, list of trajectories with columns [id,frame,μ_x,μ_y,mu_z] + additional columns. Every trajectory is given an id, starting at 1, after which the list is sorted. Frame number starts with 1.
`postprocData`	2D double array, list of trajectories. Columns like `trackingData` + additional columns.

4 How to write a TrackNTrace plugin

This section explains how plugins are added to the TrackNTrace framework, with a concise demonstration of all features shown in `plugins/pluginDemo.m`. A plugin is defined as a function in a single file named "`plugin_NAME.m`" inside the `plugins` subfolder of the TrackNTrace root directory.

4.1 Plugin header

The function inside the file returns a `TNTplugin` object. In the simplest case, the file content looks like this:

```
function [plugin] = plugin_NAME()
name = 'Some candidate plugin';
type = 1; %1 for candidate, 2 for refinement, 3 for tracking,
          %4 for post-processing, 5 for movie import
mainFunc = @arbitraryName_main;
outParamDescription = {'OutputVariableName1'; 'OutputVariableName2'};
plugin = TNTplugin(name, type, mainFunc, outParamDescription);
end

function [output] = arbitraryName_main(img, options, currentFrame)
% Actual algorithm implemented here
end
```

While the four parameters used in the constructor are mandatory for every plugin, there are additional parameters that can be set optionally after the object is constructed. Here is a complete list:

TNTplugin properties

Property	Description
`name`	Plugin name as displayed by the GUI.
`type`	Integer, type of plugin. 1: Candidate detection, 2: Refinement, 3: Tracking, 4: Post-processing, 5: Import
`mainFunc`	Handle to main function the plugin implementing the actual algorithm. For type 1 and 2 this is called by TrackNTrace in a loop for each individual frame of the input movie. The interface (output/input) of this function depends on the plugin type and must match the TrackNTrace specification (see below).
`outParamDescription`	Cell array of strings with description/name of all output parameters (columns) of the plugin
`info`	Description of the plugin itself. Should describe the method and the general way how to use it. Supports `sprintf` modifiers directly (e.g. `\n` for new line).
`initFunc`	Initialization function which is called once before the main is first executed.
`postFunc`	Post-processing function which is called after the main function is last executed.

`useParallelProcessing`	Boolean. If false, TrackNTrace does not parallelize this plugins main function on a frame-by-frame basis for type 1/2 plugins. Useful if the main function itself is parallelized (e.g. a compiled multithreaded mex file) or if global information is needed (e.g. access to multiple frames of the movie).

Parameters which can be adjusted by the GUI are defined by calling the `add_param(par_name, par_type, par_settings, par_tooltip )` function of the newly created plugin instance. Let's look at the "Radial-Symmetry" refinement plugin as an example:

```
function [plugin] = plugin_RadialSymmetry()
name = 'Radial symmetry';
type = 2;
mainFunc =  @refinePositions_radialSymmetry;
outParamDescription = {'x';'y';'z';'Amp (Peak)'; 'Background'; 'width'};

% Create the plugin
plugin = TNTplugin(name, type, mainFunc, outParamDescription);

% Description of plugin
plugin.info = ['Particle localization by radial symmetry centers. \n', ...
               'Algorithm published in Parthasarathy, NatMet 2012(9).'];
% Add parameters
plugin.add_param('PSFSigma',...
    'float',...
    {1.3,0,inf},...
    'PSF standard deviation in [pixel]. FWHM = 2*sqrt(2*log(2))*sigma.');
plugin.add_param('estimateWidth',...
    'bool',...
    true,...
    'Estimate particle width.');
end
```

As you can see, this plugin has two parameters of different types which will be visible in the GUI. Parameters are accessible for the plugin's main function via a struct (`candidateOptions`, `refinementOptions`, `trackingOptions`, `postprocOptions`) given as an input parameter. This means `PSFSigma` can be used inside the function as `options.PSFSigma`. Eventual whitespaces are replaced with underscores and dots within the parameter name are removed. Different parameters need different `par_settings` as the third input. For example, the `bool` type parameter takes only the default value, while `float` type parameters need `{defaultValue, lowerBound, upperBound}`. The complete list of parameter types and needed settings is:

par_type	par_settings	Example
'float'	Double variable, {defaultValue, lowerBound, upperBound}	{1.3,0,inf}
'int'	Integer variable, {defaultValue, lowerBound, upperBound}	{4,-10,10}
'bool'	Boolean variable, defaultValue	true
'string'	String variable, 'defaultValue'	'@exp'
'list'	Interactive list box, {'defaultEntry', 'Entry2',...}	{'x','[x,y]'}
'filechooser'	Interactive file chooser dialog {'defaultDir','fileEnding'}	{'C:/Sci/','csv'}

The fourth parameter of the `add_param` function is the tooltip shown when hovering over the parameter with the mouse (note: this supports `sprintf` modifiers, e.g. `\n` for new line).

To better structure the GUI users can call the `newRow()` function of a plugin instance between `add_param(...)` calls which starts a new row in the graphical representation of the plugin. Furthermore, headings or descriptions can be added with `add_text(text, horizontalAlignment, fontWeight, fontAngle, tooltip)`. The added text is automatically wrapped to fit into the plugin panel and always occupies its own rows (never shares rows with other parameters). Possible values for `horizontalAlignment` are: 'left', 'center', 'right'. Possible values for `fontWeight` are: 'normal', 'bold', 'light', 'demi'. Possible values for `fontAngle` are: 'normal', 'italic', 'oblique'.

4.2 Plugin body

A plugin must define a main function in `plugin.mainFunc`, which is called for every movie frame or only once in the case of tracking and post-processing. Unless explicitly forbidden, this process is parallelized using the parallelization toolbox. Movie import plugins have a slightly different syntax and are explained in section 4.5.
The main functions have a different interface for every plugin type shown below:

```
% Type 1, candidate detection
function [candidates_frame_i] = mainFuncName(img, candidateOptions, i)
%Find candidates in 'i'-th frame 'img' of the processed frame interval
   using the options struct 'candidateOptions', and return a 2D position
   array candidates_frame_i with at least two columns for x and y position
   , one row per candidate
end

% Type 2, fitting/refinement
function [fits_frame_i]= mainFuncName(img, candidates_frame_i,
   refinementOptions, i)
%Refine candidate positions candidates_frame_i in 'i'-th frame 'img' of
   the processed frame interval using the options struct refinementOptions
   , and return a 2D position array fits_frame_i with at least five
   columns for x,y and z position, an intensity and a background value,
```

```
    one row per fitted candidate
end

% Type 3, tracking
function [trackingData] = mainFuncName(refinementData, trackingOptions);
        %OR
function [trackingData, trackingOptions] = mainFuncName(refinementData,
    trackingOptions);
%Track the positions in the cell array refinementData (cells of 2D arrays
    fits_frame) using the options struct trackingOptions and return a 2D
    array trackingData which should have at least the columns [track_id,
    frame,x,y,z]. Optionally also return trackingOptions
end

% Type 4, post-processing
function [postprocData] = mainFuncName(trackingData, postprocOptions);
%OR
function [postprocData, postprocOptions] = mainFuncName(trackingData,
    postprocOptions);
%Post-processes the tracks using the options struct postprocOptions and
    return a 2D array postprocData which should have at least the columns [
    track_id,frame,x,y,z]. Optionally also return postprocOptions.
end
```

Note that all variable names are arbitrary, however, their usage/order, their output structure, and their number has to match the above specifications. If your plugin cannot provide some output, e.g. a z-position or a background value, use 0 instead.

If you already have a function `MyFunction.m` which, for example, finds candidates in an image and expects two parameters `p1` and `p2` which have been defined in the plugin header and saved in candidateOptions accordingly, adding the interface is very simple and could look like this:

```
function [candidate_xy] = wrapperForMyFunction(img, options, currentFrame)
candidates_xy = MyFunction(img,candidateOptions.p1,candidateOptions.p2);
end
```

4.3 Optional pre- and post-processing functions

The main functions are executed for every movie frame or all positions in the case of tracking. To avoid having to calculate derived parameters thousands of times althrough they are the same for all frames, or to perform post-processing steps which are only possible once the whole movie is processed, TrackNTrace can execute pre- and post-processing functions defined in `plugin.initFunc` and `plugin.postFunc`, if they are available. These functions are called just before or after the main function, respectively:

```
function [options] = initFuncName(options)

function [data, options] = postFuncName(data, options)
```

The `initFunc` can alter the `options` struct or add parameters to it before the main function is executed. This is useful for example for pre-computing masks for filter steps, creating lookup tables, or checking input parameter correctness. The `postFunc` function can be used to post-process `data` before it is saved and also add arbitrary information to `options`. Note that `data` (either `candidateData`, `refinementData`, `trackingData`, or `postprocData`) depends on the plugin type, see Sec. 3).

4.4 Global variables and parallelization

If needed, plugin functions can access external data not available to them via a function call by using global variables. Accessible variables are `globalOptions`, `candidateOptions`, `refinementOptions`, `trackingOptions`, `movie`, `filename_movie` and `imgCorrection` as defined in Sec. 3. In addition the boolean `parallelProcessingAvailable` is true if parallel processing is available to TrackNTrace. They could be used as such:

```
function [output] = fittingFun(img, options, currentFrame)
global globalOptions;
img_in_photons = globalOptions.photonConversion;

if img_in_photons
   output = PhotonFun(img,options);
else
   output = NoPhotonFun(img,options);
end

end
```

For example, the `TNT NearestNeighbor` plugin uses global access to `refinementOptions` to copy the variable `outParamDescription` of all `refinementData` columns that are not needed for the tracking.
Please note: The use of global variables inside type 1/2 plugin functions should be avoided, as functions with global variables can not be frame-by-frame parallelized by TNT, resulting in a possibly much slower execution time. Also note that, for memory reasons, the global `movie` variable gives access to the movie as returned from the import plugin (16-bit int without photon conversion for tif), regardless of the TrackNTrace settings. If your function both relies on having access to more than the current movie frame and needs a corrected and/or photon-converted movie, use the `correctMovie` function as in the following example:

```
global movie;
arbitaryMovieStack = movie{1}(:,:,1:4); % First 4 intensity frames
[correctedStack] = correctMovie(arbitraryMovieStack);
```

4.5 File import plugins

File import plugins are used as interface between TrackNTrace and the raw data. Any file-type specific processing should be done by the plugin. Import plugins are using the following fields.

TNTplugin properties for import plugins

Property	Description
`name`	Plugin name as displayed by the GUI.
`type`	Integer, type of plugin. 5 for file import
`mainFunc`	Handle to main function. (See syntax in code example below)
`info`	Struct with information about the plugin. The fields `Description` and `supportedFormats` are mandatory. Possible fields are: `Description` Description of plugin `supportedFormats` List of extensions, eg.: `{{'*.tif;*.tiff','TIF'}}` `hasFLIM` Boolean, `true` if the plugin can return FLIM images `hasTCSPC` Boolean, `true` if the plugin supports timegating and TCSPC extraction `getTCSPC` Function handle that returns the TCSPC of the first photons in the file. Syntax: `[tcspc,Resolution] = plugin.info.getTCSPC(file,maxPhotons)`. Used for the preview of the timegating. setFile Function handle that is called once a new file is selected. Syntax: `valid = plugin.info.setFile(file)`. Can return `false` to indicate incompatible files and/or set plugin options based on the file.
`postFunc`	If `hasTCPSC` is `true`, this specifies an accumulation function to retrieve the TCSPC of the localisations.
`initFunc`	Optional function handle to generate cache for `plugin.postFunc`.

TrackNTrace always uses the first import plugin matching with the given file extension and not returning `false` on calling `plugin.info.setFile`. See the implementation of plugin_loadPTU and plugin_fitLT for details on the syntax of `plugin.postFunc` and `plugin.initFunc`.
The movie is generated by TrackNTrace by calling `plugin.mainFunc` as in this example:

```
% Normal intensity movie:
FLIMflag = false; % plugins not supporting FLIM should ignore this input.
[movie,metadata] = plugin.mainFunc(importOptions,filename,...
                   [globalOptions.firstFrame, globalOptions.lastFrame],...
                   globalOptions.binFrame,FLIMflag);
% movie = {intensityMovie};

% FLIM movie with timegating
```

```
FLIMflag = true;
[movie,metadata] = plugin.mainFunc(importOptions,filename,...
                    [globalOptions.firstFrame, globalOptions.lastFrame],...
                    globalOptions.binFrame,FLIMflag,...
                    [globalOptions.tgStart globalOptions.tgEnd]);
% movie = {intensityMovie, lifetimeMovie};
```

The output `movie` should be a cell array containing the intensity movie as first element. The intensity movie is used for candidate detection and refinement should have the dimensions `[y,x,t]`. If not empty, the second cell is used by the TNTVisualizer in FLIM mode as lifetime movie. It should have the same dimensions as the intensity movie. Additional cells are ignored, but can be accessed by plugins using the global variable `movie`. The output `metadata` can be empty or a struct containing additional information about the movie. If present, the fields `pixelsize` and `pixelsize_unit` are used by the TNTVisualizer for scalebar and reconstruction.

5 Provided TrackNTrace plugins (selection)

Description of the input and output parameters of some of the provided plugins. All plugins provide a brief description and a explanation of their parameters in the tooltips.

5.1 File import plugins

TIFF (*.tif;*.tiff) Loads TIFF stacks. This plugin does not support color channels and provides no options.

PTU (PicoQuant Unified TTTR) single photon files (*.ptu) This plugin generates intensity and fluorescence lifetime movies for scanned measurements from the stream of photons saved in a PTU file. In a first step an intermediate index file is generated. During this step a shift between forward and reverse scans can be corrected automatically. The generated result can be cached along the raw data to save time when loading the same file again. This plugin supports timegating and TCSPC extraction.
This plugin provides following options:

alignBidirectional If enabled a shift from bidirectional scanning is corrected automatically by minimizing the difference between adjacent lines.

fastLT Method to estimate the lifetime for each pixel. Can be std, mean or median.

cacheMovie Number of versions to be cached. For 0 caching is disabled and the cache file deleted.

5.2 Candidate detection plugins

Cross correlation Candidate detection based on matching a Gaussian PSF template to the image using normalized cross-correlation.

A Gaussian PSF template is created using the PSFsigma value provided by the user. The whole image is cross-correlated with this image using Matlab's `normxcorr2` function and maximums (meaning: emitters) in this correlation image are detected through non-maximum suppression by image dilation using the search window size `2*round(PSFsigma*distanceFactor)+1`. Candidates at the border of the image cannot be detected efficiently.
This plugin provides following options:

PSFsigma Standard deviation of the PSF in pixels.
`sigma = FWHM/(2*sqrt(2*log(2)))` $\sim$ `0.21*lambda/NA` where `lambda` is the emission wavelength in pixels and `NA` is the numerical aperture of the objective.

CorrThreshold Correlation threshold for pixels to count as emitter candidates.
Must be between 0 and 1, a good range is 0.25 (many low quality candidates) to 0.4 (fewer, higher quality candidates).

distanceFactor The local maximum search window size is `2*round(PSFsigma*distanceFactor)+1`.
Lowering this value let's you detect candidates closer to each other but lowers candidate quality.

Wavelet filtering Candidate detection based on wavelet filtering with B-splines.
The algorithm performs a wavelet decomposition of the original image up to wavelet level 2. The result is an image where low-frequency components (i.e. emitters) of a shape and intensity distribution close to the B-spline kernel are highly elevated against high-frequency components such as noise. Maximums in the filtered image are then detected through local maximum suppression by image dilation. See Izeddin et al, Opt. Express 20(3),2012, doi: 10.1364/OE.20.002081 for details.
This plugin provides following options:

scalingFactor Wavelet scaling factor. This determines how fast the B-spline filter kernel falls of at the edges. Lower = greater falloff, which means sharper, more insular features. Values lower than 2 tend to find single pixels as candidates which is not recommended.

splineOrder B-spline order. Determines how smooth the filter falls off towards the edge. Higher = smoother falloff which correspond to larger, "flatter" structures.

detectionRadius Size of local maximum detection window. Should be roughly similar to B-spline order.

detectionThreshold Detection threshold. Higher means less detected candidates.

automaticThreshold If set to true, the standard deviation of the wavelet filtered image at wavelet level 1 times the `detectionThreshold` is taken as the absolute peak intensity threshold. In this case, setting `detectionThreshold` from 0.8 to 1.2 is recommended. Otherwise, `detectionThreshold` is directly compared against the wavelet image in which case values between 3 and 5 usually provide usable results.

Import CSV This plugin imports a list of localizations from a CSV file, eg. as exported from ThunderSTORM. Ensure that x, y and potentially sigma are in the unit pixels and the frame range matches that of the movie. To pass the data without modification to the tracking plugin, use the refinement plugin Use candidate data. Without the options sigma and otherValues only x and y are imported, as required for the TNT Fitter.
This plugin provides following options:

CSVfile Select a CSV file to import.

sigma If checked, the sigma of the PSF is imported

otherValues If checked, all additional columns present in the CSV are imported.

5.3 Refinement plugins

TNT Fitter Refine candidate positions by fitting a Gaussian PSF to candidate positions in C++.
The fitting code utilizes the ceres-solver library for optimization currently developed by Google (2015).
This plugin provides following options:

PSFsigma Standard deviation of the PSF in pixels.
`sigma = FWHM/(2*sqrt(2*log(2)))` $\sim$ `0.21*lambda/NA` where `lambda` is the emission wavelength in pixels and `NA` is the numerical aperture of the objective.

fitType Fit positions (xy), amplitude & background (A,BG), sigma (s, or sx & sy for elliptic Gaussian), and angle (for a rotated elliptic Gaussian).

usePixelIntegratedFit Use a pixel integrated Gaussian PSF for fitting which emulates the discrete pixel grid of a camera chip. This is automatically disabled when fitting a rotated Gaussian.

useMLE Use Maximum Likelihood Estimation in addition to Least-squares optimization. MLE fitting absolute requires photon conversion!

astigmaticCalibrationFile For astigmatic imaging, please select a calibration file created with the TNT z-calibration plugin. Leave empty otherwise!

5.4 Tracking plugins

TNT NearestNeighbor Track particles using simple and fast nearest neighbor tracking with gap-closing implemented in C++
Implementation uses the nanoflann library written by Marius Muja, David G. Lowe and Jose Luis Blanco. Nanoflann is under the BSD license.
This plugin provides following options:

minTrajLength Minimum length of trajectories in [frames].

maxTrackRadius Maximum allowed linking distance between two localizations in [pixels].

maxFrameGap Maximum allowed time gap between two segments used for gap closing in [frames]. When fluorophores blink or disappear, they obviuosly cannot be tracked anymore. If they reappear, however, tracked segments can be stitched back together. 0 = no gap closing.

minSegLength Minimum length of trajectory segments BEFORE gap closing in [frames].

maxGapRadius Maximum allowed linking distance between two segments uses for gap closing in [pixels].

outputLength Adds the number of localizations to each track

verbose Switch on to see tracking progress in command window.

5.5 Post-processing plugins

fitLT This plugin extracts the TCSPC from a definable area around each localization or track and determines the corresponding lifetime. It requires an import plugin that supports TCSPC extraction.
This plugin provides following options:

position Controls how the position of each tracks is processed. options are:

- preserve uses the positions provided.
- average averages the position over the length of the track.
- refit performs a Gaussian MLE fit on an sum image of all frames of the track.

maskRadius Radius of the mask for the TCSPC extraction in units of σ_{PSF}. For each frame, a constant mask size is used which based on the median σ_{PSF} in that frame.

sumTCSPC If enabled, the TCSPCs of all localization within one track are summed.

exportTCSPC If enabled, the TCSPCs are saved in the struct postprocOptions.

fitType Method for lifetime determination:

- fast lifetime estimates the lifetime using the std of the arrival times. (Very fast)
- monoexponential MLE performs an monoexponential MLE tailfit with a Nelder-Mead simplex search. Uses average DistFit to generate an initial guess. (Most accurate for monoexponential data)
- average DistFit generates 10 lifetime decays logarithmically spaced between minLT and maxLT, finds their amplitudes by a non-negative least-square optimization and returns the amplitude weighted lifetime average.
- maximum correlation generates 500 lifetime decays logarithmically spaced between minLT and maxLT, calculates their correlation with the TCSPC and returns the lifetime with highest correlation.

cutoff cutoff for tail fits with respect to the maximum of the TCSPC.

minAmp minimum number of photons in the TCSPC to perform a fit. Low number of photon TCSPCs are slower to fit and the results are less certain.

minLT lower bound for lifetime fitting.

maxLT upper bound for lifetime fitting.

attempts number of fit attempts. For each attempt the initial lifetime is slightly varied. (only used by monoexponential MLE)

tolerance relative termination tolerance of the fitting. (only used by monoexponential MLE)

The plugin adds (depending on the options) following columns to the output:

tau-fast standard deviation of the photon arrival times. (Equivalent to fitType = fast lifetime)

nphoton number of photons in the TCSPC.

lt-tau lifetime determined with the selected fitType.

lt-amp amplitude of the fit in total photons. (only for monoexponential MLE and average DistFit)

lt-bg background of the fit in total photons. (only for monoexponential MLE and average DistFit)

lt-chi2 reduced χ^2 of the fit (only for monoexponential MLE and average DistFit)

lt-corr correlation between lt-tau and the TCSPC (only for maximum correlation)

6 References

[1] Hirsch M, Wareham RJ, Martin-Fernandez ML, Hobson MP, Rolfe DJ: A Stochastic Model for Electron Multiplication Charge-Coupled Devices – From Theory to Practice. PLoS ONE 8(1), 2012.

[2] Rieger B, Stallinga S: The lateral and axial localization uncertainty in super-resolution light microscopy. ChemPhysChem, 15(4), 2014.

[3] Wang Y, Schnitzbauer J, Hu Z, Li X, Cheng Y, Huang Z-L, Huang B: Localization events-based sample drift correction for localization microscopy with redundant cross-correlation algorithm. Optics Express, 22(13), 2014.