TrackNTrace

written for MATLAB by Simon Christoph Stein and Jan Thiart

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. Finally, these results are then returned to a particle tracking algorithm which tries to link particles close in time and space to form trajectories.

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	2013a or higher
Toolboxes	Image Processing, Statistics, Parallelization (optional)

Please also note: TrackNTrace currently can only handle single-channel, 2D + t Tif image stacks. Convert your experimental data accordingly.

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 choosen if the program should use parallel processing or not.

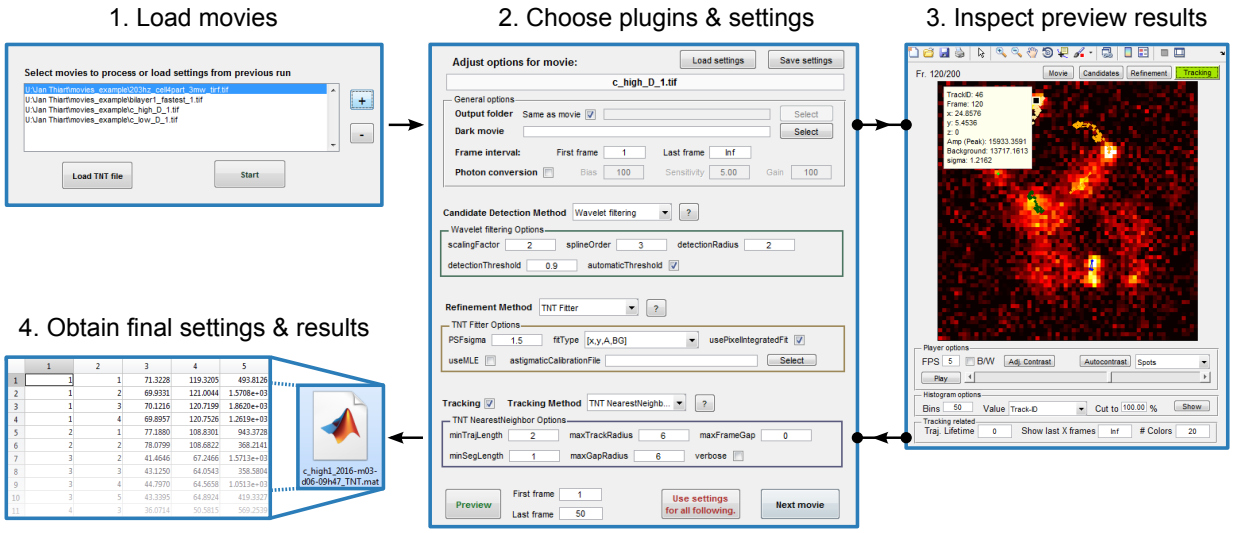

Figure 1: Program flow and user interface of TrackNTrace. First, a list of movies or a previously saved settings file is loaded before the main GUI is initialized. There, plugins for candidate detection, position refinement, and particle tracking are chosen and their settings adjusted for each movie. 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). The main part of the GUI is designed for four processing steps:

1. **Correcting raw data:** 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.

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.

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 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.

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

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

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.

The lower part houses the plugins for steps 2-4, 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) 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 4 (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 even just Tif movies. 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. Different modes can show different options at the bottom of the player.

Player options

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

B/W Black/white colormap for displaying the movie.

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.

Histogram options (unavailable in Movie mode)

Bins Number of bins the histogram is computed for.

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

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

Show Computes and displays the histogram.

Tracking related (only 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.

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.
`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`.
`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.

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
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
`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`) 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. Unless explicitly forbidden, this process is parallelized using the parallelization toolbox.
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
```

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`, or `trackingData`) 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 16-bit input movie without photon conversion, 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(:,:,1:4);

[correctedStack] = correctMovie(arbitraryMovieStack);
```

5 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.