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 a fitting routine which refines these candidates, obtaining position, amplitude and local background. Finally, these fit 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). Some in-depth information about the operation of TrackNTrace is provided in section 4.

Requirements

OS	Windows 7 64-bit or higher, Linux (tested with Kubuntu 14.4)
Matlab version	2013a or higher
Toolboxes	Image Processing, Statistics

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

Contents

1 Installation

Installation procedure for Windows 7:

1. TrackNTrace is available via a version controlled git repository. Clone the TrackNTrace repository to some directory (e.g. your MATLAB directory). If you do not have git you can get it from `http://git-scm.com`.

2. For the included mex files (compiled C++ code) to work, you need to install the Visual Studio 2012 C++ 2012 Redistributable (x64). This can either be found in the folder `external\vcredist_x64.exe` or downloaded from the Microsoft website.

Installation procedure for Linux (tested with Kubuntu 14.4):

1. TrackNTrace is available via a version controlled git repository. Clone the TrackNTrace repository to some directory (e.g. your MATLAB directory). If you do not have git you can install it via your package manager (e.g. "sudo apt get install git").

2. 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 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.

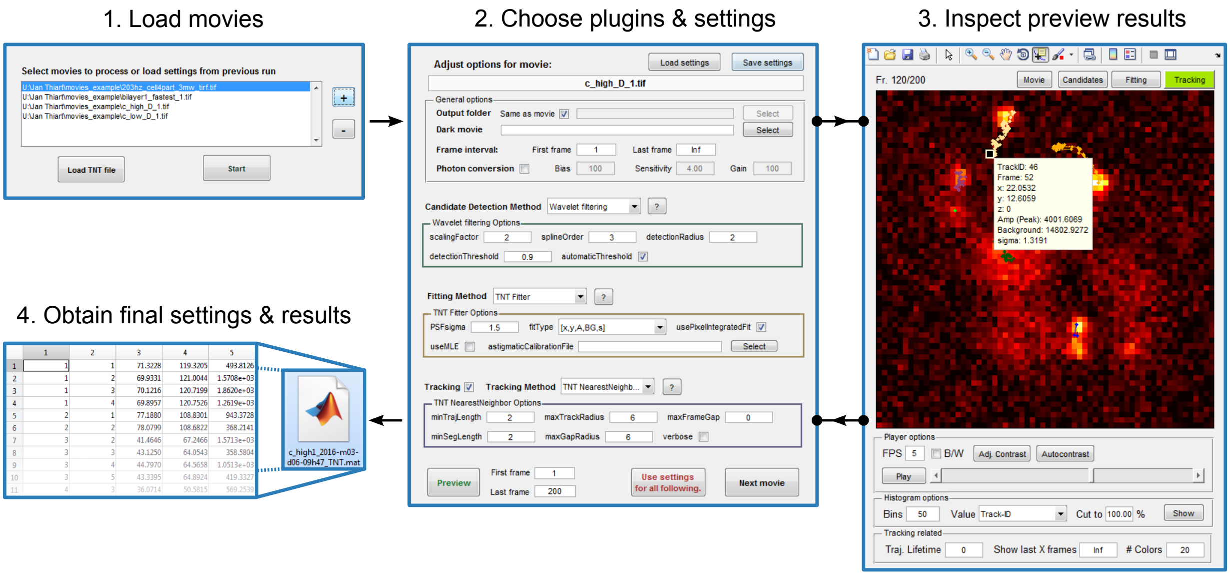

Figure 1: Program flow and user interface of TrackNTrace. First, a list of movies or a previously saved settings file is loaded before the main GUI is initialized. There, plugins for steps 2–4 are chosen and their settings adjusted for each movie. At any time during parameter tuning, a preview for an 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.

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 (Matlabs `*.mat` file format, files end on _TNT) can be selected, which loads all settings from a previous run, also starting the main GUI.

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 wtih 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 movie "`data.tif`" output 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

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

A conversion of this type is only meaningful for EMCCD cameras. The values for `Bias`, `Sensitivity` and `Gain` should be available in the cameras performance sheet.

The center part houses the plugins for steps 2-4, where the plugin for each step is selected using a popup menu. The ? button besides these menues 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 plugins 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.

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. Similar to the popular `imageJ` software, holding the "shift" key during playback continously adjusts the contrast.

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 data structures

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`	1x3 matrix saving the size of the movie [rowPixel, colPixel, nrFrames]
`firstFrame_lastFrame`	1x2 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 fitting options parameters. Also contains the used plugins name and a parameter description for all output columns in `outParamDescription`.
`fittingOptions`	Struct of fitting 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 fitted 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_yThese columns are mandatory for the fitting to work. Plugins can output extra data.
`fittingData`	$n \times 1$ cell array of fitted 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, 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,xpos,ypos,zpos] + additional columns. Every trajectory is given an id, starting at 1, after which the list is sorted. Frame number starts with 1.

4 How TrackNTrace works

After storing all options, reading in the movie and, if applicable, a closed shutter movie for correction, the fitting routine starts. First, the movie frame is cleaned up by adding a correction image obtained from the closed shutter movie as described in [1] and then converted to a photon count image if possible. Then, the respective candidate search function is called for the respective frame and the result is passed to the fit function. Depending on the settings, candidate search is carried out in each frame or only once for the first (or last) frame or an average image of the first few (or last few) frames. In the latter case, fit results from the frame before are passed to the fit function in the next iteration.
Two candidate search mechanism are possible: Normalized cross-correlation and intensity filtering. For cross-correlation, a Gaussian PSF mask is created and the `normxcorr2` function is called to calculate a correlation image of the original movie frame. This image is then processed with a local maximum filter using `imdilate`. Pixels with a correlation value higher than the user-provided threshold are accepted as spot candidates. In the intensity filtering step, the image is convolved with a normalized filter kernel consisting of a moving average window and a Gaussian bell curve of $\sigma = 1$ px against discretization noise before being processed with a local maximum filter. The kernel has a window size of $w = 2r+1$ where r is the spot radius given by the user. Next, a local background image [4] is calculated and all pixels have to pass two tests: The intensity must be among the top $q\%$ in the image and $1 - \mathrm{CDF}(\mu_b, \sigma_b) \leq p$ must hold, where CDF ist the normal distribution cumulative density function, and μ_b and σ_b are mean and standard deviation of the local background. p and q are user-provided thresholds.

A list of spot candidate positions is passed to a `Mex` file which handles all fitting procedures. First, a square quadrant I_{exp} around a candidate pixel is fit to a 2D Gaussian,

$$I_{\mathrm{theo}}(x,y) = A\exp\left(-\frac{(x-\mu_x)^2}{2\sigma^2} - \frac{(y-\mu_y)^2}{2\sigma^2}\right) + B \quad \text{with} \quad \min_{\theta} \sum_x \sum_y \left(I_{\mathrm{theo}} - I_{\mathrm{exp}}\right)^2$$

where $\theta = (A, \mu_x, \mu_y, B, \sigma)$. Fitting σ can be disabled by the user. A more accurate model can be obtained by taking into account the finite size of the camera chip pixel grid, where all signal photons hitting any point within a square pixel are accumulated:

$$\begin{aligned} I_{\mathrm{theo,px}} &= \int_{-1/2}^{+1/2} \mathrm{d}x \int_{-1/2}^{+1/2} \mathrm{d}y\, I_{\mathrm{theo}} \\ &= \frac{A\pi\sigma^2}{2} \cdot \left[\mathrm{erfc}\left(-\frac{x-\mu_x+1/2}{\sqrt{2}\sigma}\right) - \mathrm{erfc}\left(-\frac{x-\mu_x-1/2}{\sqrt{2}\sigma}\right)\right] \times \\ &\quad \left[\mathrm{erfc}\left(-\frac{y-\mu_y+1/2}{\sqrt{2}\sigma}\right) - \mathrm{erfc}\left(-\frac{y-\mu_y-1/2}{\sqrt{2}\sigma}\right)\right] + B \end{aligned}$$

Here, erfc is the complementary error function. Note that x- and y- dimension are decoupled, meaning that the two terms only have to be calculated once for one column and one line. Residual minimization is done by a Least-squares Levenberg-Marquardt algorithm. The result can be fitted again using Maximum Likelihood

Estimation which is proven to yield the best possible result [2]:

$$\min_{\theta}(-\log \mathcal{L}) = \min_{\theta}(I_{\text{theo}}(\theta) - I_{\text{exp}} \ln I_{\text{theo}}(\theta))$$

All optimization steps rely on the `ceres-solver` library, an open-source optimization library written in C++by Google Inc. We chose ceres for its very high performance, great customizability and the possibility of calculating all necessary first-order derivatives by using automatic differentiation which does not require user input and is therefore more robust. The Matlab implementation by Simon Christoph Stein is available on the Matlab file exchange where you can also find all necessary files and instructions for building TrackNTrace: `http://www.mathworks.com/matlabcentral/fileexchange/52417-fast-gaussian-point-spread-function-fitting--mex-`

After fitting is finished and if tracking is enabled, the result array is converted to a format suitable for the chosen tracker and passed to the tracking routine. Currently there are two options available: u-Track [3], and a custom, highly efficient nearest-neighbor tracker written in C++. Depending on the tracker, the time complexity of particle linking is at least quadratic in the number of particles, linear in the number of frames and quadratic or even exponential in the number of gap frames. In u-Track's case, splitting the movie into several parts can be much faster than processing all positions in a single batch. In such situations, particles within and directly after such a border or split frame are tracked again to try to re-link trajectory segments artificially cut in half by splitting the movie. Gap frames cannot be accounted for in this case, therefore, caution is advised when using larger gap values and splitting numbers together.
For every tracker, the end result is an array containing all trajectories with recorded frame number, and respective xy- position and amplitude. Every trajectory has a unique id to facilitate post-processing.
While all tracking algorithms are handled more or less the same, u-Track is an exception. u-Track can handle very difficult scenarios such as Brownian-directional motion-switching, particle merging and splitting and provides a large number of user-input variables for this. These options are all hidden in the `parseUtrackOptions.m` file and are only meant to be changed by an experienced user. Caution is advised.

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.

[2] Mortensen, KI, Churchman SL, Spudich, JA, Flyvbjerg H: Optimized localization analysis for single-molecule tracking and super-resolution microscopy. Nature Methods 7, 377 – 381, 2010.

[3] Jaqaman K et al: Robust single-particle tracking in live-cell time-lapse sequences. Nature Methods 5, 695 – 702, 2008.