Below are the main steps of FISIK and associated functions, as described in "FISIK: Framework for the Inference of in Situ Interaction Kinetics from single-molecule imaging data" by de Oliveira and Jaqaman, Biophysical Journal 2019. https://pubmed.ncbi.nlm.nih.gov/31443908/.

Software installation

The software has been tested on **Matlab R2018b** on **Linux 64-bit OS**. It includes a mex-file (maxWeightedMatching) that might need recompilation for other Operating Systems.

The software requires the following **Matlab toolboxes**:

- Statistics and Machine Learning Toolbox
- Control System Toolbox
- Robust Control Toolbox
- Optimization Toolbox
- Signal Processing Toolbox
- Symbolic Math Toolbox
- Image Processing Toolbox
- Datafeed Toolbox
- Bioinformatics Toolbox
- Computer Vision System Toolbox

The download package contains all functions (but not the Matlab toolboxes) necessary to run FISIK, except for those in the u-track package (for single particle tracking) and in the YALMIP package (for estimating the particle oligomerization state from its intensity and merging and splitting history). Please download those two packages separately:

- u-track: https://github.com/DanuserLab/u-track.
- YALMIP: https://yalmip.github.io/.

The descriptions below explain briefly the function(s) for each step of FISIK. For detailed information on the input and output arguments of these functions, please refer to the help section of each function.

FISIK steps and functions:

1. **Kinetic Monte Carlo simulations (Methods Sections 2 and 3):**

    a. To generate the trajectories and interactions of molecules freely diffusing in 2D, where all molecules have the same diffusion coefficient: **receptorAggregationSimpleOptimized.m**.

    b. To generate the trajectories and interactions of molecules freely diffusing in 2D, where each molecule has a different diffusion coefficient (as in Fig 6C): **receptorAggregationVaryDiffCoef.m**.

    c. To "label" a fraction of the simulated molecules and output their trajectories and interactions:

i. Without photobleaching:
genReceptorInfoLabeled.m.

ii. With photobleaching (as in Fig S4):
genReceptorInfoLabeledPhotobleaching.m.

Both functions take as input the output "receptorInfoAll" of the simulation functions listed in (a) and (b) above. Note that genReceptorInfoLabeled is called from within the simulation functions as well.

2. Oligomeric state estimation (Methods Section 4):

The procedure for estimating the labeled oligomeric state depends on the type of data (dynamic or static):

a. Dynamic trajectory data:

i. Probe simulations, with individual fluorophore intensity ~$N(1,0)$:
aggregStateFromCompIntensity.m

ii. Target simulations or output of u-track (more on u-track output in "Additional steps and functions related to images and u-track" section below), where intensity fluctuates over time:
aggregStateFromCompTracksMIQP.m

Both functions take as input the compound tracks of the labeled molecules. For tracks as directly output by the simulations (Step 1 above), the compound tracks are in the field "compTracks" inside the simulation output "receptorInfoLabeled". For tracks produced by u-track, the compound tracks are in the directory specified by the u-track package.

b. Static snapshot data (as in Fig 5A):

Here the oligomer distribution data are derived from the last time point of the simulation using **genAggregStateFromReceptorLabeledSuperRes.m.**

This function takes as input the output "receptorInfoAll" from the simulations (Step 1 above).

3. Intermediate statistics calculation (Methods Section 4):

The calculated intermediate statistics depend on the type of data (dynamic or static):

a. Dynamic trajectory data:

To calculate the labeled oligomer densities, off rates and on rates (note that on rates are calculated for completeness; they are not used as intermediate statistics, as explained in the manuscript):

clusterOnOffRatesAndDensity.m

This function takes as input the "defaultFormatTracks" field in the output of the dynamic data oligomeric state estimation functions (Step 2a above).

b. Static snapshot data (as in Fig 5A):

To calculate the labeled oligomer densities:
clusterDensityStatic.m

This function takes as input the output of the static data oligomeric state estimation function (Step 2b above).

4. Mahalanobis distance and p-value calculation (Methods Section 1):

To calculate the Mahalanobis distance and p-value between target and probe intermediate statistics, Steps 1-3 above must be repeated N times (e.g. N = 10 in our study) for the same parameters, but with different random number generator seeds, to generate multiple simulations and their intermediate statistics per probe or target.

Alternatively, for experimental target data (or in general target data obtained via u-track) with N repeats, Steps 2-3 are applied to each repeat to obtain all their intermediate statistics.

Given a set of intermediate statistics for the probe and another set for the target, the intermediate statistics are compared using the function:
calculationMahalonobisDistanceandPvalueDynamicStatic.m

This function takes as input a matrix of probe intermediate statistics and a matrix of target intermediate statistics, where number of columns = number of repeats. See function help for full details. It then internally calculates the intermediate statistics' mean and covariance matrix for Mahalanobis distance calculation.

This function is applicable to both dynamic and static data.

Note: In the manuscript, we show the results as p-value heatmaps. Those are generated using the matlab function **imagesc**.

Additional steps and functions related to images and u-track:

A. To generate synthetic single-molecule image series (Figure 7):

Use the function: **getImageStackFromTracksDirect_New.m**

This function takes as input the compound tracks output of the simulations (Step 1 above), i.e receptorInfoLabeled.compTracks.

The parameters we used to generate the synthetic images (see Videos S1-S4) are in Table S3 of the manuscript.

B. To "clean up" the u-track output:

a. To resolve simultaneous merges and splits (Note S1):
removeSimultaneousMergeSplit.m

This function takes as input the tracks as output by u-track.

b. To eliminate merges and splits often related to detection false positives:

This clean up step was not necessary for the u-track output employed in our study, but we supply these functions here nevertheless because they might be helpful for experimental data:

reformatSplitsMergesKeepLongerLivedSegment.m

removeSplitMergeArtifactsChronological.m

These two functions must be called one after the other. The input to the first function is the output of removeSimultaneousMergeSplit (Step (a) above), and the input to the second function is the output of the first.