1 Localization and tracking of single molecules (01. Tracking.ipynb)

1.1 Import and setup of required Python packages

```
[1]: %matplotlib widget

     import matplotlib.pyplot as plt
     plt.ioff()

     from smfret_analysis import Tracker, print_info
```

Press `Shift+Enter` to execute the cell.

1.2 Description of datasets

```
[2]: # Create a Tracker instance

     # If starting fresh, set load = False. If there is already saved data from a previous run,
     # load = True can be used to load it.
     load = False

     if not load:
         # Directory containing raw data
         data_dir = "../fret-analysis/sm-data"

         # Specify
         # - excitation sequence, where "c" means cell contours, "a" means acceptor excitation,
         #   and "d" means donor excitation
         # - data directory
         tr = Tracker("se + da * 300 + s", data_dir=data_dir)

         # Add datasets by giving
         # - an identifier
         # - a regular expression describing the file names relative to data_dir.
         #   Use forward slashes as path separators even on Windows.
         tr.add_dataset("DPPC control cells",
                        r"^DPPC_ctrl-J4/cells.*\.tif")
         tr.add_dataset("DPPC control no-cells",
                        r"^DPPC_ctrl-J4/no-cells.*\.tif")
         # If donor and acceptor emission channels are recorded by separate cameras, resulting
         # in separate video files, the second argument is a regular expression for the donor,
         # the third argument a regular expression for the acceptor emission. First donor file
         # will be matched with first acceptor file, second with second, etc., so make sure
         # they have the same sorting order.
         # tr.add_dataset("DPPC control cells",
         #                r"^DPPC_ctrl-J4/cells.*_donor\.tif",
         #                r"^DPPC_ctrl-J4/cells.*_acceptor\.tif")
         # tr.add_dataset("DPPC control no-cells",
         #                r"^DPPC_ctrl-J4/no-cells.*_donor\.tif",
         #                r"^DPPC_ctrl-J4/no-cells.*_acceptor\.tif")

         # Special datasets for registration etc. If donor and acceptor emission are recorded
         # in separate files, again use two regular expressions.
         tr.add_special_dataset("registration", r"beads/beads_.*\.tif")
         tr.add_special_dataset("donor-profile", "profile/green_.*\.tif")
         tr.add_special_dataset("acceptor-profile", "profile/red_.*\.tif")
         # tr.add_special_dataset("donor-only", r"^DO/no-cells.*\.tif")
         # tr.add_special_dataset("acceptor-only", r"^AO/no-cells.*\.tif")
     else:
         # Load
         tr = Tracker.load()
```

1. Setting `load = True` will load data previously saved using the `save()` method in the at the end of the notebook. Execute the cell (`Shift+Enter`) and go to any cell below to redo the analysis from there. For the initial analysis, use `load = False`.
2. Point `data_dir` to the directory containing your raw data. When using backslashes on Windows to separate folders, prefix the string with `r`, e.g. `r"C:\path\to\data"` to specify that the backslashes are to be taken literally.
3. Create a `Tracker` instance named `tr`. The first argument to the constructor call is a string describing the excitation sequence. Each letter corresponds to one frame. Use `d` for donor excitation, `a` for acceptor excitation, and `s` for segmentation image. Any other letters can also be used for costum frames, but are ignored by the analysis. `+` and `*` operators can be used to concatenate and repeat sequence parts. For instance, `"s + da * 300 + s"` describes a sequence of one segmentation

image excitation followed by 300 repeats of donor and acceptor alternating excitation, and another segmentation image excitation in the end. The sequence is automatically repeated as necessary, so `"da"` is equivalent to `"dadadadada"`.

4. Add datasets using `tr.add_dataset(…)` (multiple times if needed). The first argument is a string identifying the dataset. Choosing descriptive denomination is recommended, e.g. `"condition1 no-cells"`. The second (and potentially third) argument is a regular expression pattern (see, for instance, the documentation of Python's standard library's `re` module)[1] matching the names of raw data files which constitute the respective datasets. In regular expressions, a fullstop `.` serves as a wildcard for any character, and an asterisk `*` matches zero or more of the preceding character. Consequently, `.*` translates to "any number of any character." A literal fullstop is matched by `\.`. The patterns should be relative to `data_dir` and using forward slashes `/` to separate folders, even on Windows operating systems. For instance, to match all files which start with `no-cells` and have extension `tif` in a subfolder named `condition1`, use `"condition1/no-cells.*tif"`. In case donor and acceptor emission were recorded to the same files, the pattern should match these files. If they were written to separate files, the second argument describes the donor emission, and the third describes the acceptor emission.
5. Similarly, add datasets used for calculating correction coefficients using `tr.add_special_dataset(…)`. Here, the dataset identifiers can be either `"registration"`, `"donor-profile"`, `"acceptor-profile"`, `"donor-only"`, `"acceptor-only"`, or `"multi-state"` for fiducial markers, donor or acceptor emission using the densly labeled sample (can be the same files if the acceptor is photobleached), donor-only single-molecule sample for leakage correction, acceptor-only sample for cross-excitation correction, or a multi-state sample for excitation and detection efficiency correction.
6. Execute the cell.

1.3 Crop channels

[3]:
```
# Donor and acceptor emission channels were recorded side-by-side on the same
# camera. Use the UI to define where the channels are. To do so, load some
# exemplary files matching the reg. ex. passed to the method call.
tr.split_channels(r"beads/beads.*\.tif")
```

Select the emission channels' regions in the raw images. Only applicable if both donor and acceptor emission were recorded side by side, skip otherwise.

1. The argument to `tr.split_channels(…)` should be a regular expression matching representative files. For instance, to use the images of the fiducial markers, assuming they are TIFF files located in

a subfolder name `beads`, use `"beads/.*tif"`

2. Execute the cell, which will bring up UI elements to select the channels' regions in the image files.
3. Depending on the layout, press `split horizontally` or `split vertically` to split in the middle. The donor channel is marked by an orange rectangle, the acceptor channel by a yellow one. If you want to swap the channels, press the `swap channels` button
4. Fine-tuning can be achieved by manually entering the coordinates in the appropriate boxes or by dragging the rectangles. Note that you can only change the rectangle of channel that is currently selected via the tab bar on the top. For instance, to resize the acceptor rectangle, the acceptor tab needs to be activate by pressing the respective button.

1.4 Set parameters for localization of fiducial markers for image registration

[4]:
```
# Set localization options for fiducial markers, which are used below to
# calculate the coordinate transform between donor and acceptor channels.
tr.set_registration_loc_opts()
```

donor | acceptor

file: beads/beads_002_.tif   frame: 0
algorithm: 3D-DAOSTORM
radius: 1
threshold: 1500
model: 2d
find filter: Identity
Min. distance: 1
Size range
Show loc.

Figure 3

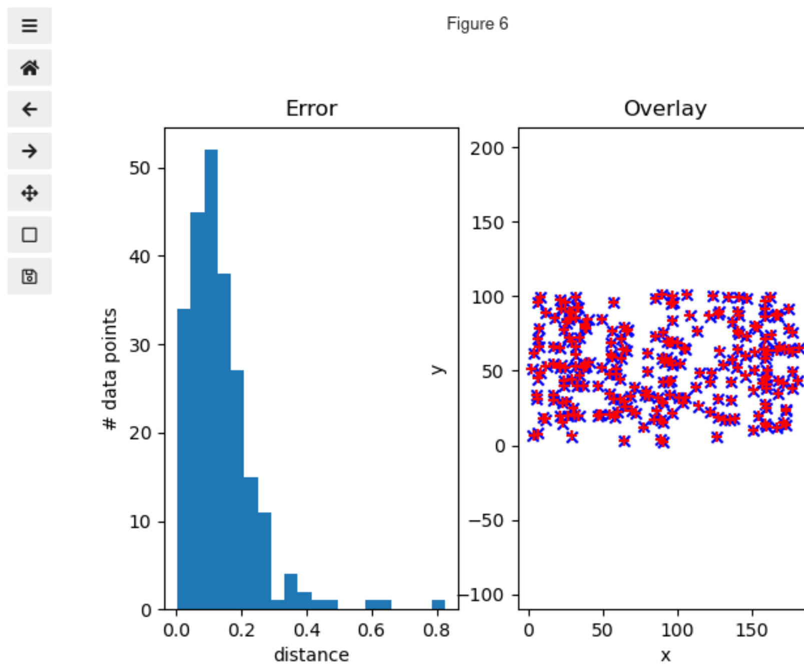
0 20 40 60 80 100
0 25 50 75 100 125 150 175

contrast 99 – 7331 Auto

1. Execute `tr.set_registration_loc_opts()`.
2. A widget will appear that allows for establishing the localization algorithm and parameters.
3. Browse images and adjust until satisfied with the outcome. Do so for both donor and acceptor channels.

1.5 Image registration

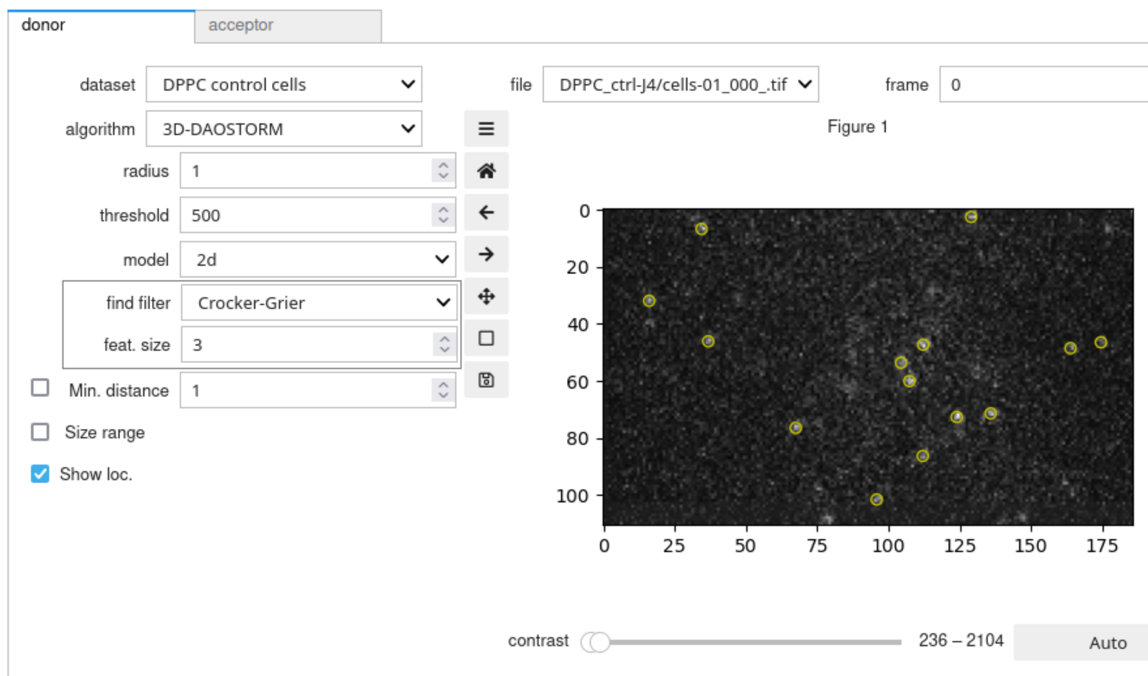
```
[5]: # Calculate the coordinate transform from bead localizations
tr.calc_registration()
```



Executing `tr.calc_registration()` will first localize all fiducial markers, then calculate the transformation mapping the donor channel coordinates onto the acceptor channel coordinates and vice versa. When finished, a plot is will show the result. In case this is not satisfactory, the algorithm can be tuned via optional parameters. Please refer to the documentation supplied with the code for details.

1.6 Set parameters for localization of FRET probes

```
[6]: tr.set_loc_opts()
```



1. Execute `tr.set_loc_opts()` to display a user interface that permits setting the localization algorithm parameters for detecting the FRET probes.
2. The `donor` tab displays the sum of donor and acceptor emission upon donor excitation. This allows for localizing the probes irrespectively of the FRET efficiency, since the total number of emitted photons remains approximately constant.
3. The `acceptor` tab shows the acceptor emission upon acceptor excitation.
4. Set the localization parameters for each channel individually.

1.7 Localize FRET probes

```
[*]: # Run localization algorithm. This can take a while.
tr.locate()
```

Locating DPPC_ctrl-J4/cells-01_012_.tif (2/25)

Execute `tr.locate()` to localize all FRET signals. This can take some time, depending on the data quality and available computing power.

1.8 Tracking

```
[*]: # Track the localizations.
tr.track(feat_radius=4, bg_frame=3, link_radius=5, link_mem=3, min_length=4, bg_estimator="mean")
```

Tracking DPPC_ctrl-J4/cells-01_032_.tif (5/25)

The following parameters for the tracking and fluorescence intensity measurement algorithms have to be set:

1. `feat_radius`: Radius (in pixels) of a disk large enough to contain the whole point spread function. This is used for measuring the fluorescence intensity—all pixels' intensities within the disk are added up and corrected for background—as well as for detecting whether point spread functions overlap, i.e. their distance is less than `2 * feat_radius + 1`, in which case the signal can be filtered out later.
2. `bg_frame`: A ring of `bg_frame` pixels around the disk described by `feat_radius` is used for local background determination. Pixels which are part of another probe's point spread function are automatically excluded.
3. `link_radius`: Maximally allowed distance for probe movement between consecutive frames in pixels. If chosen too low, localizations may not be linked properly. If too large, the algorithm can fail due to high complexity.
4. `link_mem`: Maximum gap size within a trajectory, i.e., number of consecutive frames where a signal can be absent. Set at least to 1 to permit tracks to continue after one constituent of the FRET pair has been bleached.
5. `bg_estimator`: A statistic to determine the local background from the pixel values selected via the `bg_frame`. `"mean"` is recommended, `"median"` is also an option.
6. For further options, consult the documentation supplied with the code.

Execute the cell to perform tracking and to determine fluorescence intensities. This can take some time, depending on the data quality and available computing power.

1.9 Extraction of auxiliary image data

```
[9]: # Get the images of the cells
tr.extract_segment_images()

# Get the laser profiles from additional recordings of bulk samples
tr.make_flatfield("donor", bg=200, smooth_sigma=10)
tr.make_flatfield("acceptor", bg=200, smooth_sigma=10)
```

1. `tr.extract_segment_images()` extracts images for segmentation (marked by the letter `s` in the excitation sequence) from image sequences.

2. `tr.make_flatfield(…)` determines the excitation light profiles across the field of view. Call once with `"donor"` as the first argument and once with `"acceptor"` as the first argument to do the computation for both channels. To compute the profile, the frame number selected via the `frame` argument is loaded from each image in the `donor-profile` or `acceptor-profile` special dataset. Subsequently, the pixel-by-pixel mean of these images is calculated, and the camera baseline given by the `bg` parameter is subtracted. The result is smoothed using a Gaussian blur with a σ of `smooth_sigma` pixels to smooth out sample impurities.
3. Execute the cell.

1.10 Save data

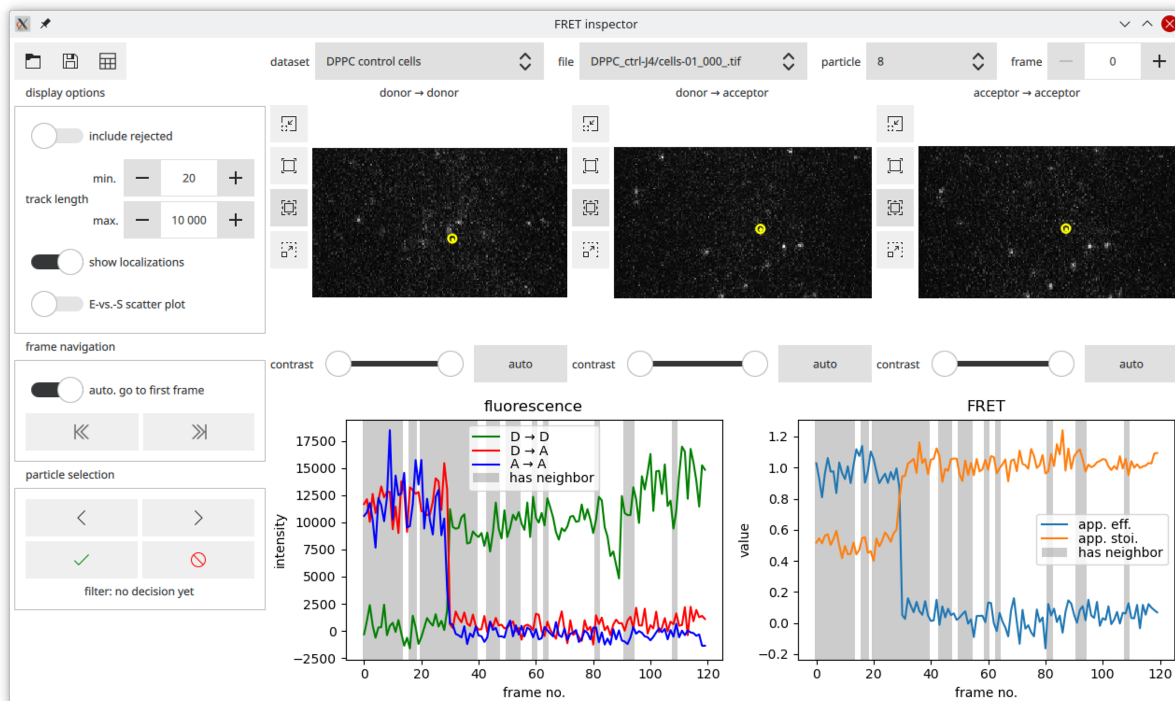
```
[10]: # Save data
      tr.save()
```

Executing `tr.save()` to save data to disk. This will create several files prefixed with `tracking`. To change the prefix, e.g. if there are multiple datasets in the same folder, pass a corresponding string as a parameter to `tr.save()`. Don't forget to pass the same prefix to `tr = Tracker.load()` in cell number 2 if rerunning parts of the localization and tracking process. The generated `.yaml` file can be openend in a text editor to inspect the parameters which were used. The `.h5` files contain single-molecule localization and tracking data. Contents may be loaded using the *pandas* Python library.

1.11 Inspection of data (optional)

Results can be inspected and modified at any point of the analysis. The tracker instance `tr` holds all data. Single-molecule tracking results can be obtained via the `tr.track_data` attribute. See the code documentation for information on the data structure.

2 Visualization of FRET trajectories (optional)



1. In an Anaconda prompt, execute

```
python -m smfret_analysis.inspector_gui
```

in the root directory. A new window will open, which may take a few seconds.

2. Press the `open` button (top left) button and select the `.yaml` file which was created using the tracking Jupyter notebook. Data will be loaded, which can take some seconds.
3. Top row dropdown menus permit selecting the track and frame number to display. Note that numbers start at 0. Frame numbers only count donor excitation frames. The nearest acceptor excitation frame is choosen automatically.
4. `display options` allow for choosing whether to display rejected tracks (see below), rejecting tracks based on their length, enabling and disabling the display of localizations and tracks on top of the raw images, and choosing whether to draw efficiency-vs.-stoichiometry scatter plots or time traces in the bottom right diagram.
5. Under `frame navigation` it is possible to toggle whether or not to go automatically to the first frame of a track after selecting it. The arrow buttons allow for navigating to the first and last frame of the currently selected track.
6. Using the `particle selection` buttons it is possible to proceed to the next or previous track and mark the current track either as manually selected or rejected.
7. If manual filtering was performed, use the `save` button (top left, second button) to save the selection.
8. Data can be exported to a spread sheet using the third button in the top-left corner. Note that this can takes several tens of seconds. Meanwhile, the application appears to be frozen.

3 Analysis and filtering of single-molecule data (`02. Analysis.ipynb`)

3.1 Import and setup of required Python packages

[1]:
```
%matplotlib widget

import matplotlib.pyplot as plt
plt.ioff()
import numpy as np

from smfret_analysis import Analyzer, DensityPlots, print_info
```

Press `Shift+Enter` to execute the cell.

3.2 Print information about used software versions

[2]:
```
print_info()
```

```
smFRET analysis software version 2.1
(git revision 2.2-11-gaa76315)
Output version 13
Using sdt-python version 16.1
```

Execute cell to print version of the FRET analysis software and underlying libraries.

3.3 Load data

[3]:
```
# Create an Analyzer instance. This will load the tracking data.
ana = Analyzer()
```

Execute cell to create an `Analyzer` instance named `ana`. This loads the tracking data from save files created by the tracking notebook. The `Analyzer()` call take can one optional argument: the file name prefix. Pass whatever was passed to the `save` function in the tracking notebook.

3.4 Choose datasets to visualize after each filtering step (optional)

```
[4]: # Show apparent E vs. S plots after filtering steps. Here we use the JupyterLab
     # Sidecar class (https://github.com/jupyter-widgets/jupyterlab-sidecar) for
     # display, but it would also possible to output to a normal notebook cell.

     # Which datasets to plot.
     plot_keys = []  # Use empty list to disable plotting.
     # plot_keys = [k for k in ana.analyzers if k.endswith(" cells")]  # Plot datasets with cells

     if plot_keys:
         from sidecar import Sidecar
         sc = Sidecar(title="Filtered plots")

         dp = DensityPlots(ana, plot_keys)
         with sc:
             display(dp)
```

Add the datasets' names (as specified by the first argument to `Tracker.add_dataset` calls in the tracking notebook) to the `plot_keys` list, e.g. `plot_keys = ["condition1 cells", "condition2 cells"]`. Leave the list empty the disable plots, i.e., `plot_keys = []`. Execute cell to pop up a new panel in the browser window which will be used for plotting. Note that plotting only works if the `sidecar` Python package is installed and set up (see main text, protocol step 1.7).

3.5 Visualize unfiltered datasets (optional)

```
[5]: # Plot initial data

     # Due to typically large amounts of data, this may take a few tens of seconds
     if plot_keys:
         ana.calc_fret_values(a_mass_interp="next")
         dp.dscatter("Initial")  # Scatter plot
         # dp.contourf("Initial")  # Contour plot
         # dp.colormesh("Initial")  # Continous color
```

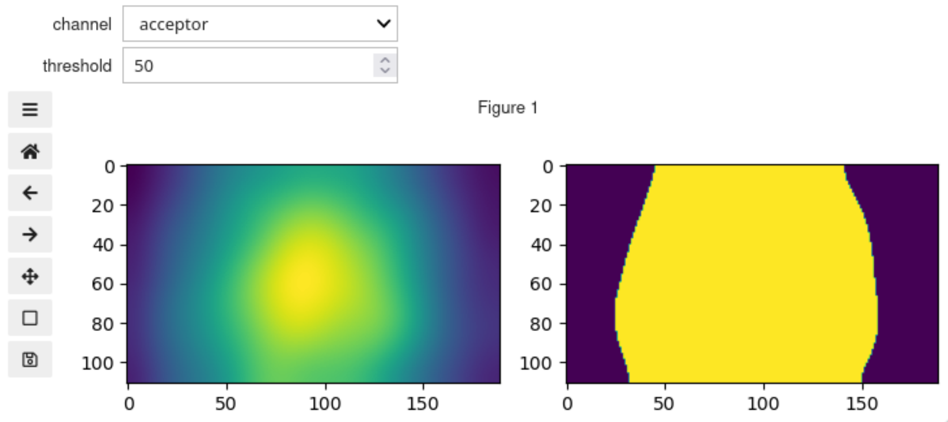
Execute cell to display apparent FRET efficiency vs. stoichiometry plots for the unfiltered datasets selected above. Plots are drawn in the side panel created in the previous step. The plot type depends on which function call is used:

- `scatter(…)` produces a scatter plot of all signals. Points are color-coded according to density in E–S space.
- `contour(…)` creates a contour plot, depeciting discrete levels of point densities in E–S space.
- `colormesh(…)` displays continous color map of densities in E–S space.

For large sets this may take several seconds. If no datasets were selected for plotting, this does nothing. All plotting functions take the plot title as an argument.

3.6 Select region of adaquate excitation intensity (optional)

```
[6]: # Display laser beam profile and find threshold of brightest xx %
     # to ensure good SNR
     ana.find_beam_shape_thresh()
```



After execution of `ana.find_beam_shape_thresh()`, a widget will display the donor or acceptor excitation light profile on the left panel. The right panel shows the regions with intensities above and below `threshold` (in percent of the maximum intensity), where `threshold` can be set using the respective input field. This serves to visualize the effect of the next filtering step.

3.7 Accept only adequately excited FRET probes (optional)

```
[7]: # Copy values from above to remove poorly illuminated datapoints
     # Manually copy the parameters from above here.
     ana.filter_beam_shape_region("donor", 50)

     if plot_keys:
         dp.dscatter("Beam shape")
```

`ana.filter_beam_shape_region(…)` accepts, upon execution, only signals that are within the region of sufficient excitation intensity to ensure an appropriate signal-to-noise ratio. The first argument specifies the channel (`"donor"` or `"acceptor"`), the second argument the minimum intensity as a percentage of the peak intensity. We typically filter based on donor excitation intensity and set a threshold of 45% to 50%. If `plot_keys` was set in step 3.4, the filtered data will be plotted.

3.8 Initial sanity checks (optional)

```
[8]: # Allow only tracks with with a high number (here, >75%) of datapoints where
     # PSFs don't overlap
     ana.query_particles("fret_has_neighbor == 0", min_rel=0.75)
     # For the remaining tracks, allow only those datapoints with non-overlapping
     # PSFs
     ana.query("fret_has_neighbor == 0")
     # Accept only tracks that are already visible in the beginning to record
     # full bleaching profile
     ana.present_at_start()

     if plot_keys:
         dp.dscatter("At start")
```

Execute the cell to perform the following:

- `ana.query_particles("fret_has_neighbor == 0", min_rel=0.75)` rejects all tracks that have a near neighbor in more that 25% of their datapoints, as these tracks would contain too many missing points.
- The call to `ana.query("fret_has_neighbor == 0")` removes datapoints with overlapping point spread functions and thus erroneous values from the remaining tracks. This essentially creates gaps in the trajectories, which are, however, gracefully handled in ensuing analysis steps.

- With `ana.present_at_start()`, only tracks which are already present in the first donor excitation frame are accepted to ensure that the full photobleaching sequence is captured. Note that this is only applicable if smFRET complexes are already present when starting the imaging process.

If `plot_keys` was set in step 3.4, the filtered data will be plotted.

3.9 Flatfield correction

```
[9]: # Apply flatfield correction in both channels
     ana.flatfield_correction()
```

Correct for inhomogeneous excitation intensities by executing `ana.flatfield_correction()`.

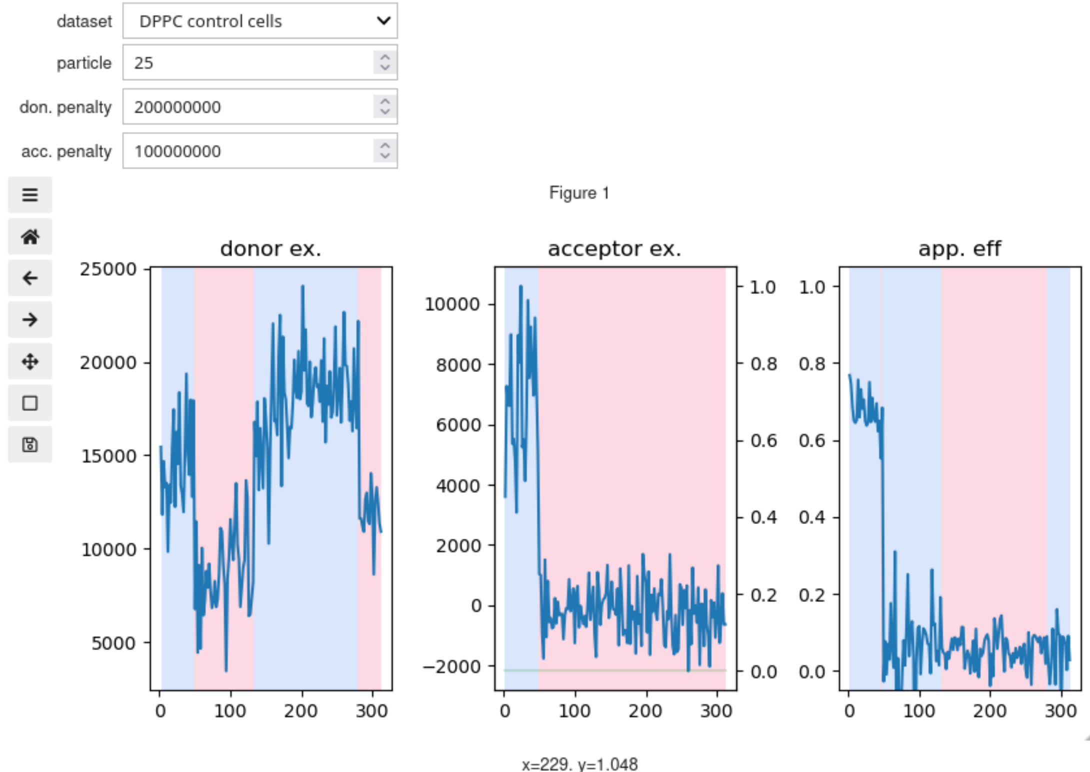
3.10 Calculate apparent FRET efficiencies and stoichiometries

```
[10]: # Calculate apparent FRET values, e.g., efficiency and stoichiometry
      ana.calc_fret_values(a_mass_interp="next")
```

Since acceptor excitation is performed after donor excitation, the acceptor emission upon exceptor excitation at the time of the donor excitation needs to be interpolated, which is specified by `a_mass_interp`. If `a_mass_interp=next`, the algorithm uses the next acceptor excitation frame. If `a_mass_interp=linear`, it performs a linear interpolation. See the code documentation for other interpolation methods. Execute the cell to calculate apparent FRET quantities.

3.11 Find changepoint detection parameters for analysis of bleaching steps

```
[11]: # Find thresholds for changepoint detection in intensity upon donor
      # excitation (d_mass) as well as upon acceptor excitation (a_mass)
      # Typically, one would find first the correct order of magnitude, then
      # fine-tune.
      ana.find_changepoint_options()
```



Analyze photobleaching steps in order to (a) identify monomeric probes and (b) accept only datapoints before a photobleaching event in each track. Bleaching steps are found via a changepoint detection alogrithm. Execute `ana.find_changepoint_options()` to display a widget that will allow you to select

a track to display, set the `penalty` changepoint detection parameter individually for the total intensity upon donor and acceptor excitation, and displays the respective time traces for the currently selected track. We suggest first browsing through the tracks until one is found that features one or more bleaching events. Subsequently adjust the `penalty` for the appropriate channel (total fluorescence intensity upon donor excitation, total fluorescence intensity upon acceptor excitation) so that detected changepoints as indicated by changing background color coincide with the bleaching events. To do so, first find the correct order of magnitude of the `penalty` (i.e., append zeros to make changepoint detection less sensitive, remove zeros to increase sensitivity), then fine-tune. Continue browsing time traces and refining the parameters until satisfactory results are obtained.

3.12 Perform changepoint detection for analysis of bleaching steps

[12]:
```
# Segment tracks using changepoint detection in the intensity time trace
# upon donor excitation and upon acceptor excitation.
# Manually copy the parameters from above here.
ana.mass_changepoints("donor", penalty=150000000)
ana.mass_changepoints("acceptor", penalty=100000000)
```

`ana.mass_changepoints(…)` performs changepoint detection on all time traces. It should be called once for donor and once for acceptor excitation. Specify channel name as the first argument and pass the respective `penalty` which was chosen in the previous stop. Execute the cell.

3.13 Filter tracks according to bleaching steps (optional)

[13]:
```
# Remove trajectories where acceptor does not bleach in a single step and
# donor shows partial bleaching.
# set_bleach_thresh sets maximum intensity which is considered bleached for
# donor and acceptor.
ana.set_bleach_thresh(1000, 500)
ana.filter_bleach_step("donor or acceptor")

if plot_keys:
    dp.dscatter("Bleaching steps")
```

Accept only tracks where there is no partial bleaching by executing `ana.filter_bleach_step(…)`. Additional restrictions can be specified using the `condition` parameter:

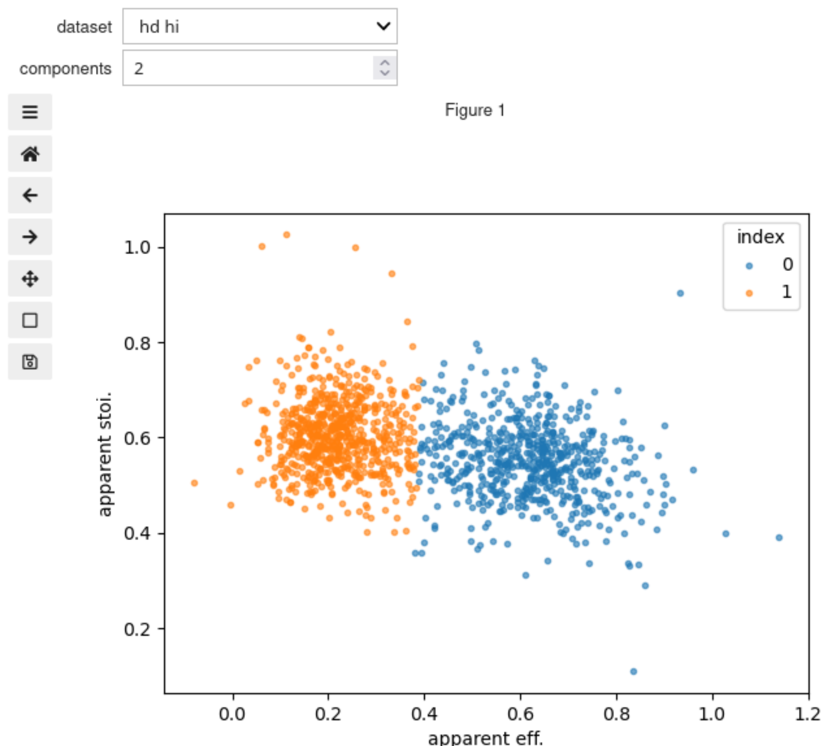
- If `condition="acceptor"`, the acceptor has to bleach in a single step before the donor.
- If `condition="donor"`, the donor has to bleach in a single step before the acceptor.
- If `condition="donor or acceptor"`, one of the fluorophores bleach in a single step.
- If `condition="no partial"`, no additional requirements are enforced.

Use `set_bleach_thresh(…)` to specify the maximum intensity up to which a signal is considered bleached in the donor (first argument) and acceptor (second argument) channel.

If `plot_keys` was set in step 3.4, the filtered data will be plotted in the side panel.

3.14 Identify subpopulations for computation of FRET correction factors

[14]:
```
# The excitation efficiency factor is computer from data with known 1:1
# stoichiomtry. The call below allows for selecting a dataset and for fitting
# Gaussian mixture models to select the right component in an E–S plot.
# Parameters should be passed to calc_excitation_eff() in the cell below.
ana.find_subpopulations()
```



Excitation and detection efficiency correction factors can be calculated from a sample with multiple discrete states which manifest themselves via their FRET efficiencies, such as Holliday junctions. In this case, the states have to be identified via a Gaussian mixture fit in the (apparent) efficiency-vs.-stoichiometry plot. If the detection efficiency is calculated from bleaching events, the excitation efficiency can by determined from a subpopulation featuring a $1:1$ donor-to-acceptor ratio, which also has to be identified via a Gaussian mixture fit in the efficiency–vs.–stoichiometry plot. Execute `ana.find_subpopulations()` to display a graphical interface which allows for setting the number of fit components as well as for identifying the subpopulations in the E_{app}–S_{app} plot to use for determination of correction factors.

3.15 Compute FRET correction factors

```
[15]: # Correction coefficients

# If determined in a separate experiment, set leakage and direct exc
# like below. Detection and excitation efficiency can be set similarly.
ana.set_leakage(0.07920904893381935)
ana.set_direct_excitation(0.04464781911255539)

# If donor-only and acceptor-only data is included in the datasets,
# the following can be used to calculate leakage and direct excitation
# coefficients
# ana.calc_leakage()
# ana.calc_direct_excitation()

# If you don't have donor-only and acceptor-only samples, leakage and direct
# excitation can can be calculated from those parts of traces that have one
# fluorophore bleached. This may not work so well for the direct excitation
# factor if the donor fluorophore is much more bleach-resistant than the
# acceptor. Setting print_summary=True will print the result as well as the
# number of datapoints used.
# ana.calc_leakage_from_bleached(print_summary=True)
# ana.calc_direct_excitation_from_bleached(print_summary=True)

# Calculate detection efficiency (gamma) factor from donor and acceptor
# intensity differences upon acceptor bleaching using the specified dataset.
ana.calc_detection_eff(min_part_len=5, how=np.nanmedian, dataset="DPPC control no-cells")
# Calculate excitation efficiency factor using the specified dataset and
# component from a Gaussian mixture model fit (see cell above for an
# interactive interface to find correct settings).
ana.calc_excitation_eff("DPPC control no-cells", n_components=1, component=0)

# Calculate detection efficiency (gamma) and excitation efficiency (delta)
# from a dataset featuring multiple discrete states
# This uses the dataset added via add_dataset("multi-state", …) in the tracking
# notebook
# ana.calc_detection_excitation_effs(n_components=3, components=[0, 2])
```

Correction factors can be set or calculated in various ways.

- To set them directly, for instance because they have been determined in a separate experiment, use `ana.set_leakage(…)`, `ana.set_direct_excitation(…)`, `ana.set_detection_eff(…)`, and `ana.set_excitation_eff(…)`, and pass the respective value as the sole parameter.
- If a sample lacking acceptor fluorophores was recorded, `ana.calc_leakage()` can be used. The corresponding donor-only sample had to be added before localization and tracking (see step 1.2).
- Leakage can also be calculated from regular samples' datapoints where the acceptor has been bleached by calling `ana.calc_leakage_from_bleached(…)`.
- Direct excitation can be computed using `ana.calc_direct_excitation()` by analyzing an acceptor-only sample (see also step 1.2) or via `ana.calc_direct_excitation_from_bleached(…)` by analyzing regular samples' datapoints featuring photobleached donors.
- If probes in a dataset feature high FRET efficiencies which remain approximately constant over time, acceptor bleaching can be analyzed to determine the correction factor for detection efficiencies using `ana.calc_detection_eff(…)`. This takes parameters `min_part_length`—the minimum number of datapoints before and after a bleaching step to be analyzed—, `how`–a function that computes a final factor from the individual tracks' values, typically `np.nanmedian` or `np.nanmean`, and `dataset`—the identifier of the dataset to use. Subsequently, a call to `ana.calc_excitation_eff(…)` determines the excitation efficiency correction. As arguments, pass the dataset identifier, the number of components (`n_components`) and the index of the component corresponding to the population with a stoichiometry of 0.5 (`component`) as determined in the widget from the previous step.
- If there are no datasets that allow for calculation of the detection efficiency correction from acceptor bleaching, a sample featuring multiple FRET efficiencies at stoichiometry 0.5 (such as isolated Holliday junctions, which fluctuate between two states) can be used for determination of detection and excitation efficiency corrections by means of the `ana.calc_detection_excitation_effs(…)` call. As arguments, pass the number of components (`n_components`) and a list of component indices to use for fitting (`components`) as determined in the widget from the previous step. If `dataset` is not specified, the dataset specified via `Tracker.add_special_dataset("multi-state, …)` is used.

Choose whatever is appropriate in your case to set the four correction factors and execute the cell.

3.16 Apply FRET correction factors

```
[16]: # Apply correction factors to calculate real FRET efficiencies and
      # stoichiometries
      ana.fret_correction()
```

Execute `ana.fret_correction()` to accurately determine the FRET efficiency and stoichiometry for each detected signal using the correction factors calculated in the previous step.

3.17 Select data before bleaching events

```
[17]: # Select only data before bleaching of a fluorophore, i.e., where the segment
      # number of both donor excitation and acceptor excitation intensity time traces
      # is 0. Select only data recorded during donor ("d") excitation.
      ana.query("fret_d_seg == 0 and fret_a_seg == 0 and fret_exc_type == 'd'")
      # Select only traces where > 75% of datapoints are within reasonable
      # stoichiometry limits
      ana.query_particles("0.35 <= fret_stoi <= 0.6", min_rel=0.75)
      # Remove data points where donor is gone (should not be necessary due to the
      # above)
      # ana.query("fret_d_mass > fret_d_mass.median() * 0.2")

      if plot_keys:
          dp.dscatter("Before bleaching, correct stoi")
```

Execute the cell for final sanity checks: `ana.query("fret_d_seg == 0 and fret_a_seg == 0 and fret_exc_type == 'd'")` only accepts signals upon donor excitation that appear before the first bleaching event in each track. `ana.query_particles("0.35 <= fret_stoi <= 0.6", min_rel=0.75)` accepts tracks where more than 75% of the datapoints have a stoichiometry between 0.35 and 0.6, which corresponds to a $1:1$ donor-to-acceptor ratio (optional).

If `plot_keys` was set in step 3.4, the filtered data will be plotted in the side panel.

3.18 Find parameters for image segmentation (optional)

```
[18]: # Display an UI to find thresholding parameters for cell images
      ana.find_segmentation_options()
```

If images for image segmentation were recorded, a thresholding algorithm can be used to automatically determine regions of interest. Execute `ana.find_segmentation_options()` to display a widget designed

for finding the appropriate parameters. The tabs allow for choosing the algorithm. In our experience, `adaptive` delivers the best results. Parameters for adaptive thresholding are the block size (`block_size`), σ of a Gaussian filter for smoothing (`smooth`), a constant offset for thresholding (`c`), and the thresholding method (`method`). The proper choice of parameter depends on the imaging conditions, e.g., how cells have been labeled, whether TIRF was used, the excitation intensity which was applied, etc.

3.19 Filter data with results from image segmentation (optional)

```
[19]: # Threshold cell images and select only data within cell-occupied areas
ana.apply_segmentation([k for k in ana.analyzers if not k.endswith("no-cells")],
                       "adaptive", block_size=65, c=-2, smooth=3, method="mean")

if plot_keys:
    dp.dscatter("Underneath cells")
```

To restrict certain datasets to datapoints from within regions identified by image segmentation, execute the `ana.apply_segmentation(…)` call. Its first parameter is a list of datasets which should be filtered, such as `["condition1 cells", "condition2 cells"]` or more generically `[k for k in ana.analyzers if "no-cells" not in k]` (all datasets whose names do not contain `"no-cells"`). The second parameter specifies the algorithm to use, e.g. `"adaptive"`. Subsequently, pass the parameters found in the previous step, e.g. `block_size=80, smooth=3, c=-2, method="mean"` as additional arguments to `ana.apply_segmentation(…)`.

If `plot_keys` was set in step 3.4, the filtered data will be plotted in the side panel.

3.20 Save data

```
[20]: # Save results to disk
ana.save()
```

Save results by executing `ana.save()`. All remarks concerning saving data after tracking (see step 1.10) apply here as well.

3.21 Inspection of data (optional)

Results can be inspected and modified at any point of the analysis. The `Analyzer` instance `ana` holds all data in the `ana.analyzers` Python dictionary, which maps dataset names to `sdt.fret.SmFRETAnalyzer` instances. See the *sdt-python* library documentation for details.

4 Plotting of results and further analysis (`03. Plot.ipynb`)

4.1 Import and setup of required Python packages

```
[1]: %matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

from smfret_analysis import print_info, Plotter
```

Press `Shift+Enter` to execute the cell.

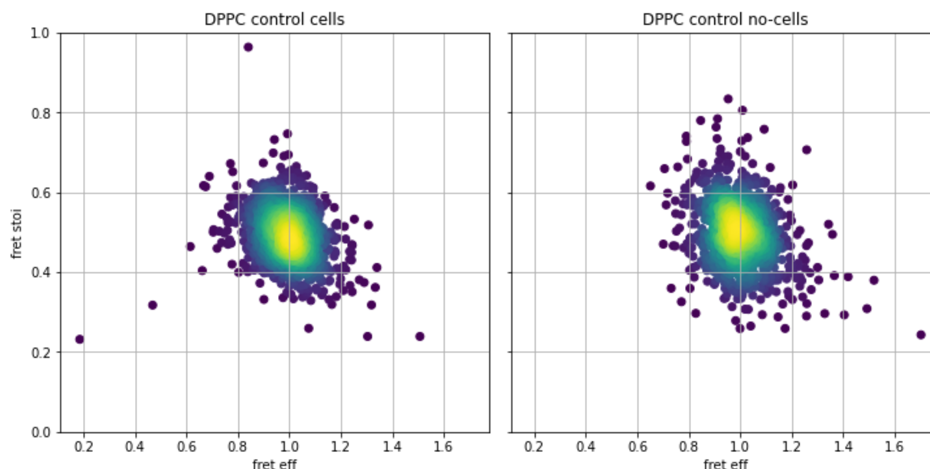
4.2 Load data

```
[2]: p = Plotter()
```

Execute cell to create a `Plotter` instance named `p` This loads the tracking data from save files created by the analysis notebook. The `Plotter()` call take can one optional argument: the file name prefix. Pass whatever was passed to the `save` function in the analysis notebook.

4.3 Generate FRET efficiency-vs.-stoichiometry scatter plots

```
[3]: fig, ax = p.scatter(frame=None, ylim=(0, 1));
     fig.savefig("scatter.pdf", bbox_inches="tight")
```

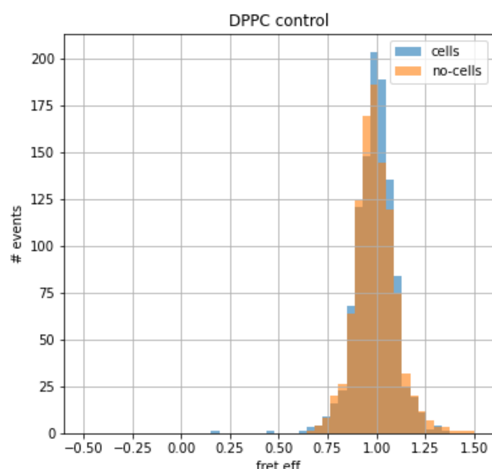


Execute `p.scatter(…)` to draw the plots plots. It is possible to pass `xlim` and `ylim` arguments which set lower and upper limits of x and y axes, respectively. For instance, use `xlim=(-0.5, 1.5)`. If not set, limits are chosen to fit all datapoints. Axis labels can be set via `xlabel` and `ylabel` parameters.

The call returns the *matplotlib* figure instance and array of axes, which can be further manipulated. For example, call `fig.savefig("scatter.pdf")` to save the plot to a file named `scatter.pdf`. `p.scatter(…)` calls `sdt.fret.smfret_scatter(…)`; see the documentation of the *sdt-python* library for additional parameters and details.

4.4 Generate FRET efficiency histograms

```
[4]: fig, ax = p.hist(group_re=(r"(.+) (cells|no-cells)", 1, 2),
                      hist_args={"alpha": 0.6, "density": False});
     fig.savefig("hist.pdf", bbox_inches="tight")
```



Execute `p.hist(…)` to draw the histograms. Arguments described in conjunction with the `p.scatter(…)` method can also be applied here. Furthermore it is possible to specify how to group datasets, i.e., put multiple datasets into the same plot. For instance, this allows to directly compare experimental results with the results from control experiments.

The `group_re` parameter consists of three entries: a regular expression with at least two groups, the index of the group which is used to identify datasets which should be in the same plot, and the index of the group which will be used to identify datasets within individual plots. Note that indices start with 1 in this case. As an example, consider datasets are named `"condition1 cells"`, `"condition1 no-cells"`, `"condition2 cells"`, and `"condition2 no-cells"`. Then, `group_re=[r"(condition.)`

`(cells|no-cells)", 1, 2]` will generate two plots, one for datasets starting with `condition1` and one for datasets starting with `condition2`, since index 1 (i.e., the group `(condition.)`) is specified to identify datasets for different plots. Each plot will contain one `cells` and one `no-cells` histogram, since index 2 (i.e., the group `(cells|no-cells)`) is to be used for datasets within the plots.

The call returns the *matplotlib* figure instance and array of axes, which can be further manipulated. For example, call `fig.savefig("hist.pdf")` to save the plot to a file named `hist.pdf`. `p.hist(…)` calls `sdt.fret.smfret_hist(…)`; see the documentation of the *sdt-python* library for additional parameters and details.

4.5 Further analysis

`p.track_data` is a Python dictionary mapping dataset names to tables (`pandas.DataFrames`) containing single-molecule data, one event per line. These can be exported to various file formats or further analyzed in the Jupyter notebook using, for instance, functionality provided by the *sdt-python* library.

References

1. Python Software Foundation Re — regular expression operations. at <https://docs.python.org/3/library/re.html> (2021).