Analysis code for smFRET: Documentation

Stephanie Johnson*
Department of Biochemistry and Biophysics,
University of California, San Francisco,
San Francisco, CA, USA

Contents

1 Introduction. **2**
1.1 Licensing and copyright information. 2

2 Getting started. **3**
2.1 General requirements. 3
2.2 Data format requirements. 3
2.3 Overview of the analysis process. 4

3 Running the analysis (and understanding what it does). **4**
3.1 Part 1A: Creating a channel map from fluorescent beads. 7
3.2 Part 2: Generate and examine FRET-versus-time traces. 12
3.3 Part 1B: Creating a channel map from paired DNAs. 16

4 Customization Options. **17**

5 Details and derivations. **18**
5.1 Channel mapping/transformations: General concepts 18
5.2 Affine versus polynomial maps. 19
5.2.1 Affine transformations. 19
5.2.2 Polynomial transformations. 19
5.2.3 Comparing affine and polynomial transformations of smFRET data. 22
5.3 Finding non-linear transformations. 22
5.3.1 Calculating the inverse transformation. 24
5.4 Channel mapping: Practical considerations. 24
5.4.1 How many spots are needed for a good mapping? 24
5.4.2 How much does the mapping vary hour-to-hour, day-to-day, or after removing the prism and replacing it? 25
5.4.3 Does the transformation differ significantly if it's calculated from beads versus real DNAs? 25

6 Acknowledgements. **26**

*Email: Stephanie.Johnson@ucsf.edu.

1 Introduction.

This is a software suite designed to calculate FRET-versus-time traces from a standard prism TIRF-based single-molecule FRET microscopy setup. The most current version of this suite can be found at https://github.com/stephlj/smFRETcode. It can be run as a stand-alone analysis suite, but it is written as modularly as possible in the hopes that it can be adapted fairly easily for other microscopy setups and acquisition software. The wrapper function that calls various portions of the analysis in proper order, called smFRET.m, is included but is highly tailored to our setup. Section 4 below discusses some possible options for customizing this suite to suit your particular needs.

The general outline of the analysis workflow is: (1) Calculate a map that correlates pixels in the acceptor channel image to pixels in the donor channel image, as these will never be perfectly aligned in an smFRET setup. Or, load an old one. (2) Find fluorescent spots in a movie or set of movies and allow the user to scroll through intensity-versus-time traces for each spot, and then save good traces for further analysis later.

The rest of this manual contains information about required Matlab toolboxes, a walkthrough of how to run an analysis session off-the-shelf, and details of how the analysis is done.

1.1 Licensing and copyright information.

2 Getting started.

2.1 General requirements.

Requires Matlab with the following toolboxes installed:

• Image Processing Toolbox

and will use the following if available:

• Curve Fitting Toolbox
• Optimization Toolbox

This code was written for analyzing prism-based TIRF images; it may not work well for objective-based setups (for example, you may have to alter or re-write the code to find spots, or do a more sophisticated background subtraction).

2.2 Data format requirements.

This software assumes the following about how the raw data is acquired and stored from the microscope:

• We acquire data with the open-source UCSF software Micro-Manager (http://www.micro-manager.org/), using its Multi-D acquisition tool. Micro-Manager stores the raw data as .tif files, in directories entitled *rootname_1, rootname_2, etc*, where "rootname" is defined by the user prior to acquisition, and a new folder with _1, _2 etc is created for each acquisition with the same rootname. Each of the resulting directories contains one tiff per frame from the acquired movie (if the "save as separate image files" option is selected in Micro-Manager), with filenames img_000000000_000.tif, img_000000001_000.tif, etc. In each movie's directory Micro-Manager also saves a metadata.txt file with information about frame rate, etc.

• The function LoadUManagerTifs assumes this file and directory structure; if you use something other than Micro-Manager to acquire data, you will have to rewrite or modify LoadUManagerTifs to load image files into Matlab for analysis. Similarly, if your acquisition software does not save a metadata.txt file, you will have to rewrite or modify GetInfoFromMetaData.m (or write a text file that mimics the parts of the metadata file structure that GetInfoFromMetaData.m accesses).

• Channel mapping (at least an initial one) is accomplished by means of images of fluorescent beads that are visible in both channels. It is assumed these beads are in directories that contain the word "Bead" (*e.g.*, "Beads_1", "Beads_2"). Although it is not assumed there are movies (rather than single frames) in these directories, I've found it's better to average 10-20 frames before finding bead positions.

• While you can set certain setup-specific parameters in smFRETsetup.m, such as whether the donor channel is the right half or left half (or top or bottom) of the image, you will have to write your own wrapper function for certain extensions such as images obtained on a three-color FRET setup.

2.3 Overview of the analysis process.

The workflow for analyzing smFRET data is, generally speaking, composed of five steps (which I will refer to as "modules"): (1) data pre-processing (which includes things like scaling each image to take advantage of the full dynamic range of the numeric type you're using); (2) finding a mapping between the donor and acceptor channels (which will never be perfectly aligned in any setup, so that a mathematical relationship between points in one channel to points in the other must be calculated, in order to identify FRET pairs); (3) spot finding (identifying which pixels contain fluorescence from real molecules, and which are background fluctuations); (4) calculating the intensity in each identified spot in each frame, and from these intensities, FRET-versus-time traces; and finally (5) displaying these traces for the user to comb through to pick out real results for further analysis. We have found that thinking about the analysis workflow in terms of these five modules has been helpful, but we note that modules are not necessarily performed in the order listed above; for example, we do a channel mapping (module 2) before data pre-processing (module 1), and this analysis suite also offers the opportunity to make a refined map (module 2) using FRET pairs that the user has looked through (module 5).

Details of how to run each module, and of what each module does to the data, are discussed in the next section. Figures 1 and 2 show call graphs that diagram the functions in this analysis suite that perform each module, with any required or optional Matlab toolboxes noted to the right of each function. Functions in colored boxes are "tools" that are called by multiple functions; pink are highly specific to our microscope setup, yellow should be more general. Blue asterisks indicate interactive sections requiring user input; red arrows indicate files saved to or loaded from disk (dashed lines indicate optional pathways based on user input).

3 Running the analysis (and understanding what it does).

Open the smFRETsetup.m file and change parameters as necessary.

For example, you will want to set defaultdatadir, defaultsavedir, *and* codedir. *Follow the instructions under Display Defaults to make sure figures will show up well on your screen. I recommend leaving most of the other parameters alone for now; most of them are explained below, and will make more sense once you've run the analysis once or twice.*

To start the analysis, enter

smFRET('*rootname*')

in the command line, where "rootname" is the name of the directory/directories you want to analyze, minus the "_<number>" that MicroManager adds.

Pass '1' as the second optional input to run in debug mode, which will provide some additional figures (detailed below).

You will be asked (in the command line) if you want to create a channel mapping using beads (Part 1A), load an old one (Part 2), or create a channel mapping from paired DNAs saved in Part 2 below (Part 1B).

We (and others, e.g., [2]), routinely image a slide with surface-immobilized fluorescent beads to calculate at least a preliminary channel map. This software does not support the creation of a map from real data de novo, *although there are workarounds that would allow it (for example, creating a*

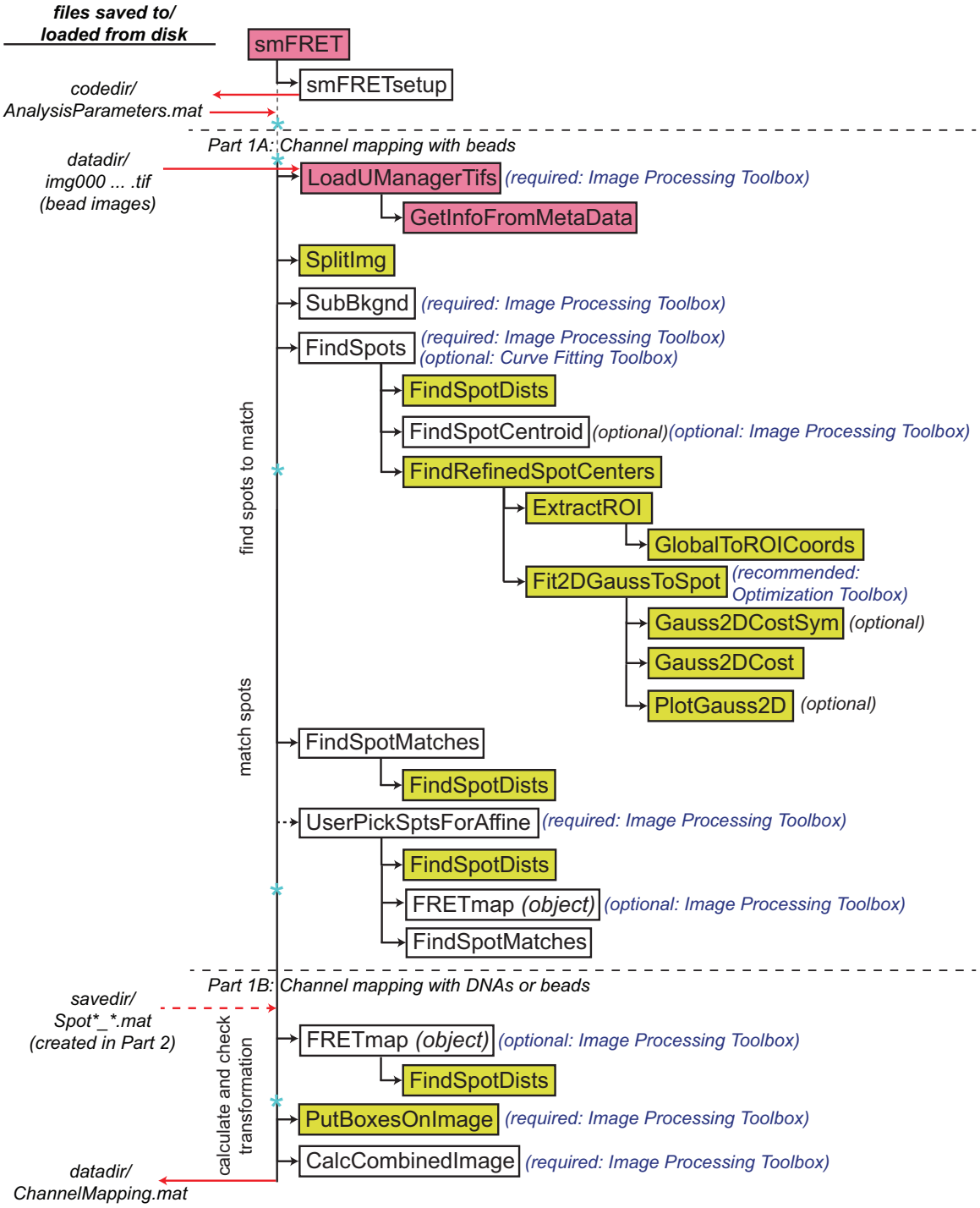

Figure 1: Call graph of the functions for calculating a channel mapping.

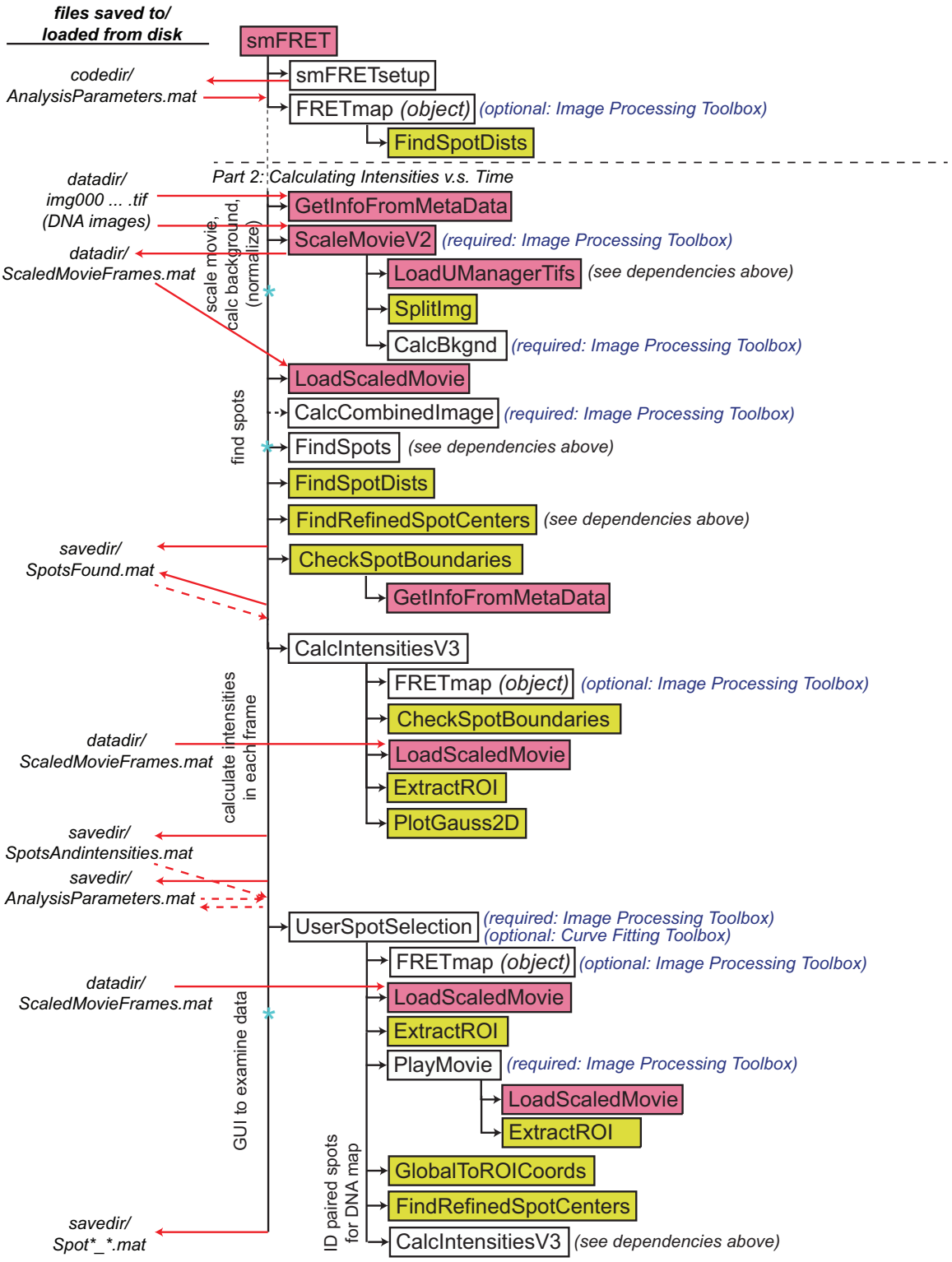

Figure 2: Call graph of the functions for analyzing real data.

FRETmap object that contains the identity transformation instead of a calculated affine or polynomial transformation, loading this map using the "L" option, then using the GUI to generate paired spots that could be re-loaded using the "D" option. However, without a bead-based map to get you close enough to see the real spot pairs in the GUI, your channels would have to be fairly well aligned to start with in order to make this work).

Note: any time it asks you to press "anything else", it will probably crash if you press the space bar. Choose a different anything else!

3.1 Part 1A: Creating a channel map from fluorescent beads.

(1) When prompted, navigate to the directory that contains the bead directories.

(2) If there is more than one directory with "Bead" in its title in the directory you choose, you will be given the option to use only some of the Bead data in creating a map, and to use the remaining bead data to quantify the fidelity of the mapping.

I usually collect data on 4-5 fields of view of our bead slide, use 2-3 of them to create a map, and use the remaining data to check the transformation. See Sec. 5.4 for some advice, based on my experience, on how many beads you need per field of view to get a good transformation, how much the transformation varies from day to day on our setup, etc.

Spot finding.

Spot finding is accomplished by finding local maxima and local minima across the image, and then calling real spots those maxima that are sufficiently larger than their local minima; the rest of the local maxima, that fall below the threshold, represent fluctuations in background. This method has the advantage of not setting an absolute intensity threshold for real spots that must apply across the entire image, thereby allowing the illumination to vary across the image. The Ha lab code does something roughly equivalent: they use a single absolute threshold value across the entire image, but they spot-find on a background subtracted image. (In the Ha lab code, the threshold is hard-coded into nxgn1_cm *(spot finding for beads) and* p_nxgn1_ffp *and ...*_brief *(spot finding for DNAs), in the parameter "std".) I prefer to keep spot-finding and background subtraction as separate procedures that can be modified independently without changes to one impacting the other.*

(3) Spot finding is done on one channel at a time, on an averaged image of the first FramesToAvg frames (where FramesToAvg is set in smFRETsetup; the Ha lab uses a hard-coded value of 10 frames). The spot-finding routine in FindSpots will attempt to identify an appropriate threshold value that distinguishes background fluctuations from real spots. The number of spots ("peaks") that, according to this threshold, are considered to be real signal are printed to the command line. A further refinement of all peaks found is then performed; the basic refinement that is always performed excludes spots too close to the edges or to another spot (where "too close" is defined as closer than DNASize or BeadSize pixels in smFRETsetup). The smFRET wrapper function passes an input pair ("Method", "Gauss") that does an additional refinement by fitting a 2D Gaussian to each peak and insisting that the variance is not too small (which is usually the case when a very bright single pixel got included as a "spot"), and the local background is not too high (which happens when two beads are right next to each other, but only one was found as a local maximum). The number of peaks kept after these refinements are then printed to the command line.

I don't think the Ha lab code excludes peaks too close to one another, so my code may find fewer

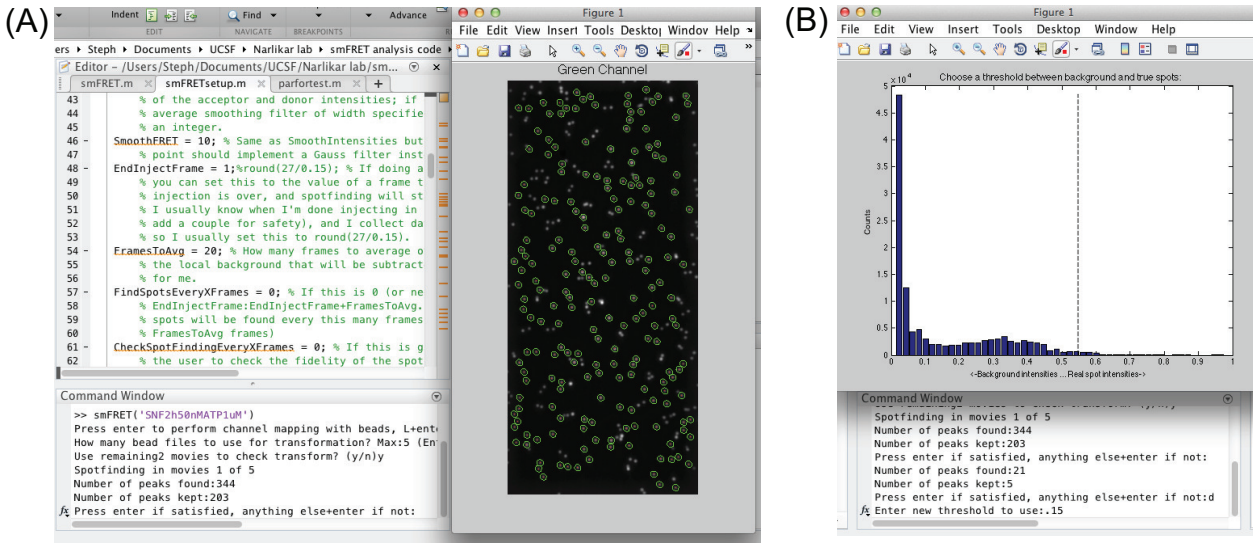

Figure 3: Screen shots of the spot finding routine. These are for spot finding on beads, but would be similar with real data. (A) The figure on the righthand side of the screen shows the donor channel half of the image from the camera, with green circles around each spot identified as a bead. (B) If the automatic threshold detection fails (as it obviously did here), you can set the threshold manually, using as a guide a histogram of the difference between each local maximum and its local minimum. With the beads it's easy to identify a good threshold–notice the two clearest peaks, one near zero that represents background, and one around 0.3 that represents the beads. The vertical dashed line indicates the current threshold value.

spots per field of view than you're used to if you use the Ha lab code, though hopefully a higher percentage of the spots found will be useful, especially for real data.

(4) Examine the image displayed, with green circles around spots found, and the information displayed at the command line. Press enter to accept the results, anything else+enter to choose a new threshold. A screenshot taken at this point in the analysis is shown in Fig. 3(A).

(5) If you rejected the results, a histogram of the differences between all local maxima and local minima will be displayed (as in Fig. 3(B)), with a vertical black line at the current threshold value. For the beads, there should be two clear peaks in the histogram, one near 0 (which represents fluctuations in background), and one which on our setup peaks around 0.2-0.3 (and maybe some small bumps at higher values that represent double beads, extra bright junk on the slide, *etc*). (These peaks won't be so clearly distinguishable with single dyes, but the same principle will apply there too.) Choose a new threshold value and press enter. Spot finding will be redone with this new threshold, after which you will again be given the option to accept the results or try another threshold.

NOTE(1): you must enter a threshold value before pressing enter, even if you've changed your mind and want to keep what it used the last time!

NOTE(2): The number of peaks found can be changed by changing the threshold (larger threshold will result in fewer peaks found); the number of peaks kept can be changed by changing the BeadSize parameter in smFRETsetup (or, for spot finding with real data, the DNASize parameter), and/or the cen_tolerance and VarTolerance parameters at the beginning of FindRefinedSpotCenters.m, and/or the bkgnd_tolerance parameter near the beginning of FindSpots.m.

(6) Continue spot-finding in the other channel, and repeat for all bead movies you are going to use to calculate the mapping and to check the mapping.

On our setup, the number of peaks found per field of view, with our bead slide, should be about 300-350, and with reasonable parameter settings the number of peaks kept should be about 175-200.

(7) For every Bead directory examined, after spots have been found in each channel, the number of spots paired will be printed to the command line.

If your microscope's channel alignment is pretty good, the number of spots paired should be about 80%-90% of the spots found. The pairing code uses a "greedy algorithm" to match spots, which essentially means it assumes for each spot that its match in the other channel isn't too far away, and is probably the closest one. Some mismatches at this point are ok; they will be removed before the final mapping is calculated (even a couple mis-pairings out of hundreds of true pairs can really mess up the fidelity of the mapping!).

If running in debug: after spots have been found in each channel, and before pairing is attempted, a figure will be displayed that shows the full image (both channels together) with white circles around each spot found. Execution will be paused until the user presses a key.

(8) If your channels aren't aligned well on the microscope, however, it may not get enough correct matches to get a good enough transformation to pick out the mismatches. In this case, the user will be asked to select three pairs of spots that are true matches (see Fig. 4(C)), from which an affine map will be calculated that will then be used to automatically match the rest of the spots. It's best to choose three spots that are well-spread across the image. (By the way, if you're wondering, why 3?—Three is the minimum number of points necessary to define an affine transformation. More details about affine transformations can be found in Sec. 5 below.)

This is actually what the Ha lab code does from the start, essentially: it asks the user to identify three pairs of spots, creates a rough affine map to match more spots, and then uses those additional spots to generate a polynomial transformation (which requires a minimum of 15 spots, for the polynomial used in our code, or minimally 16 spots for the polynomial used in the Ha lab IDL code). But, the more the better, because you will know how the polynomial mapping looks for more of the field of view.

If running in debug: after spots have been paired, a new figure of the whole image will be displayed, this time with only the spots paired circled in green. A second image that overlays the locations of the spots found in one channel on those found in the other, with a blue line connecting the ones it thinks are matches, will also be displayed, as shown in Fig. 4(A). I've found this second figure to be useful to visualize the alignment of the emission side optics on the microscope. Execution will again be paused until the user presses a button.

Calculating the transformation that maps from one channel to the other.

The mapping from one channel to another is accomplished by a mathematical transformation that relates a point in one channel to a point in the other channel. A best-fit transformation is calculated from paired sets of points, one set in each channel. As discussed in more detail below (see Sec. 5), we and others have found a 4th-degree polynomial to best describe the relationship between channels. The degree and kind of polynomial fitted, and the option to use affine instead of polynomial, are set in smFRETsetup. Both IDL and Matlab have built-in functions to fit polynomial or affine transfor-

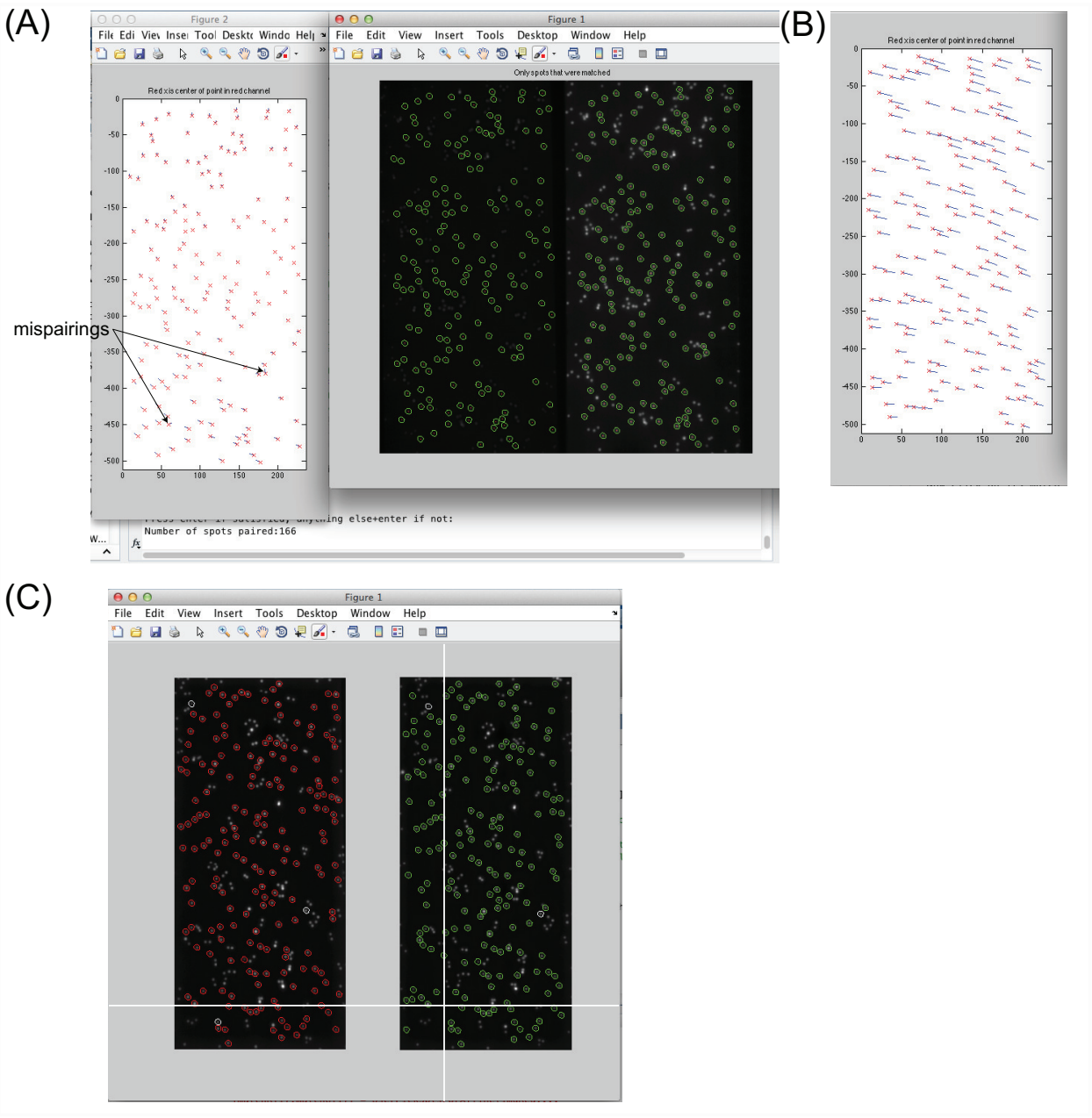

Figure 4: Screen shots of the debugging figures for the spot-matching routine, and of the manual pairing option for poorly aligned channels. (A) If running in debug mode, after spots have been found in each channel and then paired, a figure of the full image from the camera (left) with green circles around the paired spots will be displayed, and a plot of matched points in each channel overlaid on each other, with a blue line connected them (right). These figures are for a fairly good alignment on our microscope. Note the shape of the "vector field"—this gives a picture of the polynomial transformation that must be used to map between channels. Note also the mispairings—having only a couple mispairings, as here, is not a problem at this point; they will be removed before the final map is calculated, because they are obvious outliers. More than a couple mispairings, however, and it will not be clear enough what are outliers and what are real pairs for a map to be generated. (B) Same as the left figure in (A) but for a particularly bad alignment of the emission side optics. (C) For poor alignments like in (B), a greedy pairing algorithm will not work, since the correct spot to pair is not necessarily the closest one. In this case, the user will be shown a full image from the camera and will be asked to pick a spot in the red channel, then its pair in the green, then another spot in the red, until three pairs have been picked. (Specific instructions are printed to the command line.) Spots are circled in white after they are selected. White lines show the cross-hairs that Matlab displays for the user to pick points.

mations; the FRETPmap class object also includes hand-written code to do the same, though using Matlab's optimized (and professionally written) versions are always preferable when possible. (You will have to use my hand-written code if, for example, you want to fit a maximum degree 4 polynomial, which Matlab's built-in polynomial class does not support; see also Sec. 5.)

(9) After spots have been found and paired in all Bead directories, a preliminary mapping will be calculated, and then refined by the removal of obvious outliers, which are mis-paired or poorly localized spots. After every cycle of refinement, the remaining number of spots that are being used to calculate the mapping, and the residuals for the current map, will be printed to the command line. These residuals are the sum of the squared differences between mapped spot locations and real spot locations. The refinement process will insist that the residuals per spot drop below the ResidTolerance value in smFRETsetup; if it can't manage that, it will halt in the debugger and print some instructions to the command line about how to proceed.

Note that the residuals printed are summed over all spots; the ResidTolerance parameter is per spot, so a "ResidualsGtoRperSpot", which is the residuals divided by the total number of spots included in the transformation, is also printed to the command line, for comparison to the tolerance. Note also that because we usually calculate a polynomial transformation, which is not invertible (unlike affine), a separate transformation is computed for acceptor to donor channel, and donor to acceptor; therefore both sets of residuals ("ResidualsGtoR" and "ResidualsRtoG", where "R" stands for "red" and means "acceptor", and "G" means donor ("green")) are printed. There are also two sets of figures that display, one for one direction, one for the reverse. They're generally pretty similar (both the figures and the residuals), but it's good to check both.

(10) The final channel mapping will be saved (along with some additional information) in the same directory as the Bead files used to generate it. The map is contained in a FRETmap object. A number of figures (see Fig. 5) are then displayed so that you can check how well the transformation did. The most important of these are the histograms of distances between mapped spots and real spots. For the held-out data, if any (that is, bead data that wasn't used for the transformation), the vast majority of these distances should cluster between 0 and 0.5 pixels. There will always be some counts in the histogram at higher values—these represent mis-pairings. However, the mapping will only be useable if the difference between where it thinks a spot is and where it actually is, for true pairings, is less than about 0.5 pixels. For the non-held-out data, there should be no mismatches, and the differences between real spots and mapped spots should be mostly below 0.2 pixels. Lastly it will print a Mapping Tolerance to the command line. This is the maximum distance, rounded up to the nearest whole pixel, at which you can be confident a spot in one channel and a spot in the other are a pair. For a good channel mapping on our setup, the Mapping Tolerance should be 1 for beads (2 for single dyes).

NOTE (1): In order to (manually) load the FRETmap object back into Matlab, if you want to play around with it, you must have the directory containing the FRETmap.m class definition file in your known paths (or in your current directory).

NOTE (2): One of the sets of figures that are displayed at the end includes overlays of one channel over the other, using either an affine transformation (which does better) or polynomial (which does worse, even though when comparing fit residuals the polynomial does significantly better). This would be used to calculate a combined image, if you wanted to spot-find using a combined image to more easily catch mid-FRET spots. For reasons that I'm still not clear on, it (very obviously) does not work with Matlab's built-in overlay functionality ... so for the moment, at least on our setup, using a combined image to find spots is not an option. You should be able to find even mid-FRET spots, however, by

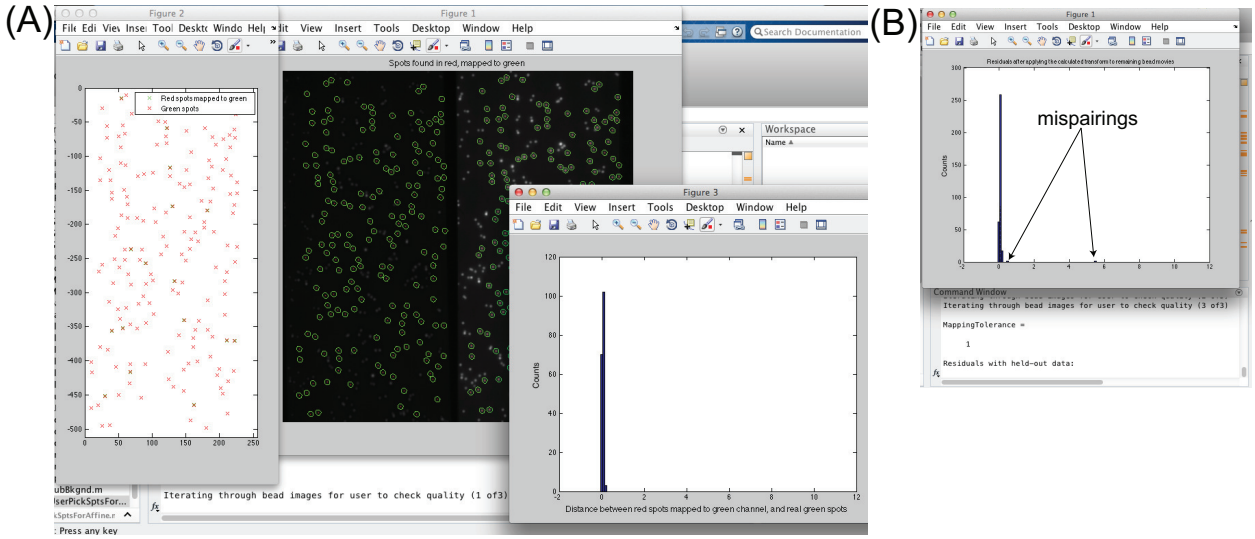

Figure 5: Screen shot of the figures that are displayed for the user to check the channel mapping. (A) For each Bead movie that was used to calculate the mapping, two sets of figures will be displayed, with a pause in between. The first set contains 5 figures that allow the user to check the mapping fidelity from donor to acceptor channel, three of which are shown here: an image of the full view of the camera, with the spots used to calculate the mapping circled in green; a scatter plot of the locations of the centers of all the acceptor-channel spots, and of where the mapping thinks their centers should be; a histogram of the distances between the real acceptor-channel spots and where the mapping thinks they should be; and overlays of the donor channel onto the acceptor, using either an affine or a polynomial transformation (not shown). While the histogram is generally the most useful of these figures, the scatter plot and image can be useful for picking out where the mapping is failing, if it is. Note that there should be no counts above about 1 pixel in the histogram. The second set of figures is the same as the first three listed here, but for the inverse (acceptor to donor) transformation (and in fact the images shown are for the inverse transformation). (B) If the user chose to keep some bead images as held-out data on which to check the transformation (that is, spot-find and pair in some images, but don't use this information to calculate the transformation), after the previous sets of figures have been displayed for all bead images used to calculate the transformation, a histogram will be displayed showing the distances between mapped acceptor spots and real acceptor spots. In this case, there may be some residuals at higher pixel values, which should be mis-pairings in the held-out data. (I've found that if the residuals look fine for the data used to generate the map, it's not necessary to check the residuals for the inverse transformation as well, so only the residuals for donor-to-acceptor are displayed.)

changing the threshold in the spot-finding routine, and/or by spot-finding every 100 or so frames.

3.2 Part 2: Generate and examine FRET-versus-time traces.

(1) After you've calculated a map, or if you chose to load an old one, navigate (when prompted) to the directory that has the data directory/directories you want to analyze, whose names should start with the "rootname" entered as the argument to smFRET.

Scaling and (optionally) normalizing images, and calculating background.

Data movies must be scaled in order to take advantage of the full dynamic range available, which for Matlab's Image Processing Toolbox is optimally [0 1]. The Ha lab acquisition code scales to minimum and maximum intensity values set before acquisition begins; the UCSF mrc to pma converter does what we do here, which is to scale to the minimum and maximum intensity values across the entire movie. That is, the largest intensity in the entire movie will be set to a value of 1, the smallest intensity will be set to 0, and all other intensities will be linearly scaled between these two values. The data is

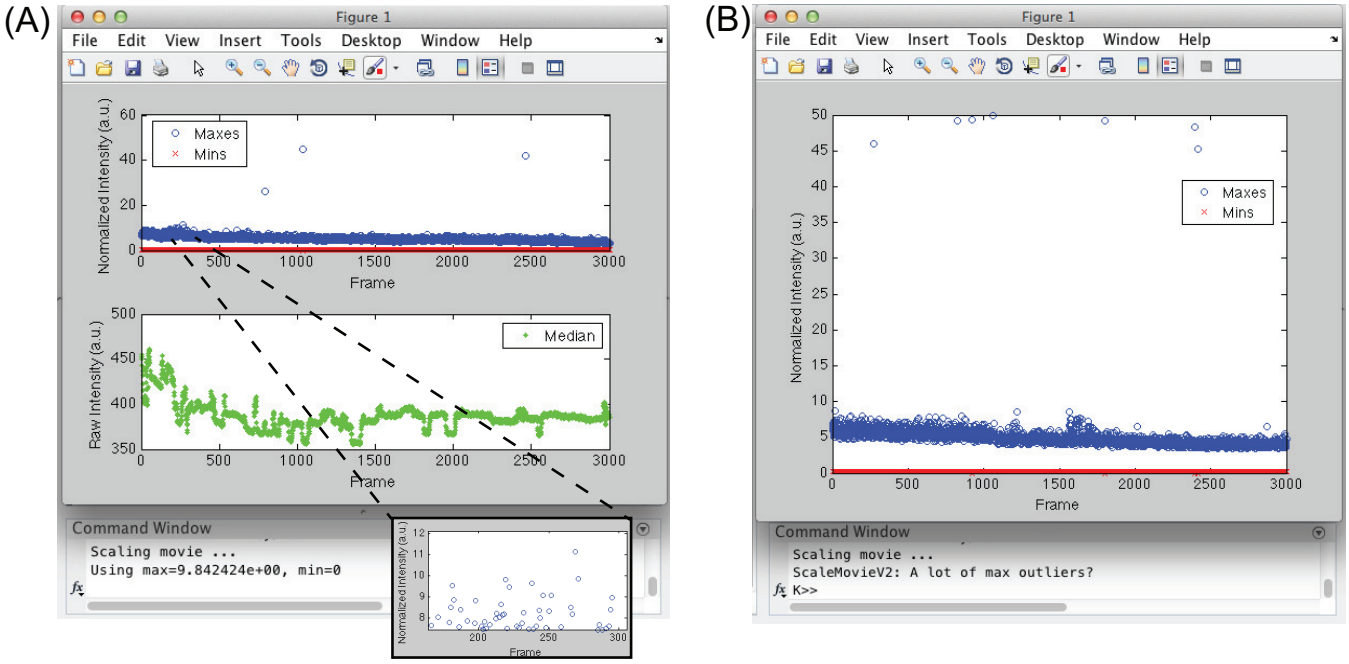

Figure 6: Screenshots for the movie scaling (data pre-processing) steps. (A) For every frame in a movie, the minimum (top panel, red), maximum (top panel, blue) and median (bottom panel, green) intensity values are calculated and displayed. Obvious outliers in intensity maxima (corresponding to one very bright pixel in some frames) are excluded before the maximum intensity that will be set to 1 is computed. The inset shows a zoom of the maxima around 250 frames, which includes the maximum that will be used for the scaling. The maximum and minimum values to be used for scaling are printed to the command line and execution is paused so that the user can confirm the values are reasonable. The bottom panel shows the roughly 7% median intensity value fluctuations that we observe in most data sets and that we believe are due to laser intensity fluctuations. If these are not removed, these fluctuations will show up in all intensity-versus-time traces for individual spots and obscure real signal. If the NormImage parameter in smFRETsetup is set to 1, each frame will be normalized to (specifically, divided by) its median value, removing the median intensity fluctuations. (B) In some cases, for example if more than 5 max intensity outliers are removed, execution will pause and ask the user to confirm that the scaling algorithm is truly removing outliers or if it is starting to remove real maxima. If the maxima plot looks like it does in (B), where there are more than 5 clear outliers, enter "dbcont" in the command line to override the checkpoint. (You may have to do that several times, as here, because there are quite a few outliers in the maximum intensities and the scaling algorithm will pause after each one to check that it's still only removing outliers.)

also converted to double precision at this point, again for optimal interface with Matlab's toolboxes. Additionally, we've had problems with laser intensity fluctuations; therefore if NormImage is set to 1 in smFRETsetup, all pixels in each frame will be normalized to the median intensity value in each frame to remove these fluctuations. Lastly, a background image is calculated for each frame and saved to disk, for use later in calculating the intensity of a spot. Our background calculation procedure is essentially the same as the Ha lab's, except we calculate one for every frame instead of one for the whole movie from an average of the first 10 frames, again to allow for a time-varying background due to laser intensity fluctuations.

(2) We often have large outliers in maximum intensities—single frames where one pixel got very, very bright for no good reason. These outliers are removed before the maximum value to set to 1 is determined. ScaleMovie will display the maximum, minimum and median intensity values for the

movie as a function of frame, and print the values it's going to set to 0 (min across the movie) and 1 (max across the movie) to the command line (see Fig. 6(A)). Then it will pause and wait for you to press any button to accept and proceed. If it has to remove a lot of outliers to find a good maximum to scale to, it will stop in the debugger and ask for user input. If everything looks fine anyway, enter "dbcont" into the command line to proceed.

NOTE: This is currently the longest step of analysis—on my laptop this takes ∼15-20 minutes for a data set with ∼3000 frames. It's mostly limited by the rate Matlab can load from disk, so anything that will reduce load times (like copying data to your internal drive, if you have the space) will make it faster.

Spotfinding.

(3) Spotfinding proceeds as described for the beads in the previous section, with the option of the following modifications that are controlled by parameters in smFRETsetup:

(a) If instead of only finding spots in an averaged image of the first FramesToAvg frames, you want to find spots throughout the movie, set the FindSpotsEveryXFrames parameter to a value greater than 0. After an initial round of asking the user to set a good threshold for real spots, spot finding will proceed with this same threshold until a multiple of CheckSpotFindingEveryXFrames (see next point) is reached.

(b) I highly recommend setting the CheckSpotFindingEveryXFrames parameter to a value equal to FindSpotsEveryXFrames or to 5*FindSpotsEveryXFrames, or something like that. At least with my data, the optimal threshold varies enough over, say, 100 frames that if you just keep using the threshold for the first FramesToAvg frames, you run the risk of a significant number of false positives (background fluctuations that get counted as real spots) or missing real spots because the threshold was too high or too low. Working on improving the automatic threshold detection ... *NOTE: If you're running in debug mode, every CheckSpotFindingEveryXFrames, it will display a figure showing the new spots that have been found since the last round, in red, and all old spots in green, for each channel.*

(4) At the end of the spot finding session, all spots found in both channels will be saved to the directory you chose to save data in. Subsequent to that, spots found in the donor channel will be converted to the frame of reference of the acceptor channel, using the channel map that you just calculated or that you loaded.

NOTE: Any time after this point, you can quit out of smFRET, rerun smFRET with the same input argument (i.e. the same rootname), load a different channel map, and then load the spots found in both channels and re-convert to the acceptor channel frame of reference, by selecting the "Load previously found spots?" option (NOT the "Load previously found spots and their intensities?" option). Spot intensities will then be re-calculated. (Since spot-finding is independent of a channel map and happens for each channel separately, you don't have to find spots again just because you want to change where their pair in the other channel should be.)

Calculating intensities-per-time for each spot.

Nominally, calculating the intensity of a spot in each frame is simple—the intensity is just the sum of the intensities of all the pixels defined as part of the spot. However, there are several refinements that are usually done before this sum is calculated: (1) A background value is subtracted before the intensities are summed. Ideally, after a fluorophore has bleached, the summed intensity should drop to approximately zero, if the background was effectively subtracted. (2) A weighted average instead of a straight average is performed by some groups (including the Ha lab). If the IntensityWeightGauss

option is 1 in smFRETsetup, which I highly recommend, each spot's intensity will be weighted, in each frame, with a 2D gaussian centered on the spot and having the amplitude you set in GaussWeightAmp. This has the effect of smoothing out some of the fluctuations in a spot's intensity due to noise. The Ha lab uses an amplitude of 2, which just stretches the results out along the intensity axis. Also, you can either weight with a Gaussian whose width is fixed to the values set in FixSpotVar; or you can weight each spot by a Gaussian whose width was calculated when the spot was originally found. The Ha lab fixes the spot variances; I've found it doesn't make a huge difference. Using spot-specific variances can be nice if part of the field of view has distortions that result in more ellipsoid, rather than round, spots. (Note that if you want ellipsoid spots, the UseSymGauss option must be 0.)

(5) Raw (that is, unsmoothed, if smFRETsetup has smoothing options selected) intensities for donor and acceptor channels will be calculated and the results will be saved to the save directory, as a "SpotsAndIntensities" file. Note that at this point, FRET values are not calculated, nor are any corrections to the intensities (like alpha) or modifications like smoothing, so any relevant options in smFRETsetup (alpha, gamma, SmoothFRET and SmoothIntensities) can be changed by changing smFRETsetup and rerunning.

Any time after this point, you can quit out of smFRET and re-load traces into the GUI, without recalculating their raw intensities, by selecting the "Load previously found spots and their intensities?" option. Use this option if, for example, you want to change the smoothing values in smFRETsetup and then rerun the GUI. You will be asked if you want to load the GUI with old parameters; if you've changed smFRETsetup, select "n".

(6) A GUI will appear with a figure showing intensity-versus-time and FRET-versus-time traces, and a separate figure showing (initially) an average of the first FramesToAvg frames. Note that the displayed intensities and FRET values are smoothed and have the correction factors gamma and alpha applied (see smFRETsetup for definitions of gamma and alpha); that is, they are NOT the raw intensities (though these raw intensities are saved along with modified intensities—see below). Instructions for GUI options print to the command line. Most options should be straightforward, with the possible exceptions of:

"b" (background adjustment): If the background subtraction routine didn't remove all the background (*i.e.*, intensity values are significantly nonzero after photobleaching), select this option, then click on two parts of the intensity-versus-time or FRET-versus-time trace over which you want to define a new background value. The average donor and acceptor intensities between these two points will be subtracted from every point.

"r" (reset background): remove any additional background subtraction you added with the "b" option.

"z" (zoom): Click on two points in the intensity-versus-time or FRET-versus-time traces to zoom in on that area.

"u" (unzoom): undo any zoom. NOTE that this will define how much of each trace is saved—see below.

"f" (show a particular frame), "m" (play movie between two time points), "a" (show average around a particular frame): In the case of "f" or "a", click on a point in the intensity-versus-time or FRET-versus-time traces that you want displayed in Fig. 1. Note that if you go forward or backwards a spot, Fig. 1 will reset to an average of the first FramesToAvg frames. "m" will leave Fig. 1 showing the last frame loaded (but again Fig. 1 will reset if you go forwards or backwards).

"l" (re-Locate spot): If the channel mapping didn't find a spot pair well in one channel, you can

click on where you think the true spot center is, and smFRET will attempt to re-find the spot in that area, and then re-calculate the intensities using that new spot center.

NOTE: You don't need to press enter after making a selection—for example, to go forward one spot, just press ".", no enter needed. Any time you enter something into the command line, however, you do need to press enter—for example, if you select "g" to go to a particular spot, you will be prompted to enter the spot number in the command line, after which you will *need to press enter. You need to make all selections with Fig. 2 current—if you don't, you'll notice you'll just type stuff into the command line and nothing will happen ...*

(7) What is saved to disk when you select the "s" option in the GUI: When you select "s" for a spot, a new file will be created in the save directory called "Spot<movienum>_<spotnum>.mat" (or if you select "s" for a spot for which a file has already been created, you will overwrite the previous file with a new one). This file contains:

(a) vectors of raw donor and acceptor channel intensities for each FRAME ("rawRedI", "rawGrI");

(b) vectors of unsmoothed, but with alpha applied for the acceptor intensities and gamma applied for FRET values, donor and acceptor intensities and FRET values for each frame ("unsmoothedRedI", "unsmoothedGrI", "unsmoothedFRET");

(c) vectors of donor and acceptor intensities and FRET values with any smoothing applied—these will be identical to the unsmoothed versions if no smoothing was applied;

(d) The frame rate at which data were acquired ("fps", which is actually frames per MILLISECOND)

(e) The (x, y) coordinates of the center of the spot in the donor channel and the acceptor channel, in the coordinate systems of their respective channels (note these coordinates will depend on the value of PxlsToExclude in smFRETsetup) ("Rspot","Gspot"), and the variance used for the Gaussian weighted sum, which is assumed to be the same for both channels ("variance").

In addition to this Spot file, an eps and a Matlab .fig file of whatever is in Fig. 2 will also be saved, with the same filename. NOTE: Only what is in Fig. 2 will be saved—meaning if you zoomed in on part of the trace, the Spot file will only contain the intensities and FRET values for the zoomed region. You can choose to cut out parts of traces this way (for example, you can choose not to keep everything after both fluorophores have bleached).

3.3 Part 1B: Creating a channel map from paired DNAs.

You can make a map using true paired DNAs (that you've confirmed are true FRET pairs by their intensity traces, and that you made sure, using the GUI, were well-localized) by running smFRET with any input, selecting the "D" option, and navigating to a save directory that contains the "Spot<>.mat" files saved with the "s" option from the GUI. As with the beads, a transformation will be calculated, the residuals refined as much as possible, and then the user will be asked to check the results. Works the same as described above for beads, although most of the figures that are generated for bead maps will not be generated here.

4 Customization Options.

- FindSpots:
- CalcBkgnd and SubBkgnd:

5 Details and derivations.

5.1 Channel mapping/transformations: General concepts

The essence of finding a map that relates locations in the donor channel to locations in the acceptor channel, and vice versa, is finding a transformation in the mathematical sense. We will be particularly concerned with massaging the desired channel maps into *linear* transformations, because linear transformations can be described by matrices, and we can bring the power of linear algebra to bear on them. In particular, for a set of points (think spots—beads or dyes—in one of the channels) $\vec{x} = \{\vec{x_1}, \vec{x_2}, ..., \vec{x_n}\}$, where the Cartesian coordinates of each point are given by $\vec{x_i} = (x_{i1}, x_{i2})$, we can write a linear transformation to the other channel in terms of a matrix $\mathbf{A}$, such that

$$\vec{y} = \mathbf{A}\vec{x}, \tag{1}$$

where $\vec{y}$ are the Cartesian coordinates of the points in the other channel. We will use the same convention for $\vec{x}$ and $\vec{y}$ as in the code, namely, that a single spot's coordinates will be a column vector, and so a set of spots will be a 2xn matrix, where n is the number of spots. Then we know that in the simple example of Eqn. (1) above, $\mathbf{A}$ will be a 2x2 matrix, though this will not be the case for all of the transformations we describe below.

The goal, of course, is to find $\mathbf{A}$, which will be the map describing how to find a spot's location in the other channel, if we know where it is in the first channel. More precisely, what we want is a *best fit* value for $\mathbf{A}$. That is, we want to find the $\mathbf{A}$ that minimizes the error of our estimate of $\vec{y}$, given $\vec{x}$; *i.e.* we want to find $\mathbf{A}^*$ such that

$$\mathbf{A}^* = \underset{\mathbf{A}}{\operatorname{argmin}} \|\vec{y} - \mathbf{A}\vec{x}\|_2^2. \tag{2}$$

According to Eqn. (2), $\mathbf{A}^*$ will be the $\mathbf{A}$ that minimizes the difference between the true positions of the points represented by $\vec{y}$, and where we estimate them to be based on applying matrix $\mathbf{A}$ to $\vec{x}$. (The notation $\|...\|_2$ just means the "norm-2" (Euclidean norm or Euclidean distance) of whatever is between the $\|$'s; in 2-space, you will recall that for two points (x_1, x_2) and (y_1, y_2), their Euclidean distance is $\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$.)

How do we find $\mathbf{A}^*$? We use the **normal equations** as conditions of optimality. The normal equations state that $\vec{y} - \mathbf{A}^*\vec{x}$ need to be orthogonal to the rowspace of $\vec{x}$ (err ... or the rowspace of Ax?), in order for $\mathbf{A}^*$ to fulfill Eqn. (2). By the definition of orthogonality, this means that

$$(\vec{y} - \mathbf{A}^*\vec{x}) \cdot \vec{x}^T = 0. \tag{3}$$

If it's been a while since you've thought about rowspaces and the like, here's another, more intuitive way to think about Eqn. 3: The rowspace of $\vec{x}$ is the plane? surface? described by treating each row of $\vec{x}$ as a point in 3-space. (Um ... given my points are 2xn, should I be talking about the columnspace of x ... ? Getting stuck here, Matt, help!!)

anyway ...

Now what we need to do is solve Eqn. (3) for $\mathbf{A}^*$. Rearranging Eqn. (3) and moving the term containing $\mathbf{A}^*$ to the other side, we have

$$\vec{y}\vec{x}^T = \mathbf{A}^*\vec{x}\vec{x}^T. \tag{4}$$

Remember these are matrices, so we cannot "divide" both sides by $\vec{x}\vec{x}^T$, but we can multiply both sides of the equation, on the righthand side, by the inverse of $\vec{x}\vec{x}^T$. Is $\vec{x}\vec{x}^T$ invertible? Apparently you

can show that it is (which I'm not going to do here, but you can Wikipedia linear least squares and/or look up the normal equations in a linear algebra textbook if you'd like to know more). So we have our final solution that

$$\vec{y}\vec{x}^T(\vec{x}\vec{x}^T)^{-1} = \mathbf{A}^*. \tag{5}$$

In summary, to find the best-fit transformation that takes a set of spots $\vec{x}$ in one channel to a set of spots $\vec{y}$ in another channel, we calculate the lefthand side of Eqn. 5. In practice, Matlab has a smarter way of estimating the "Moore-Penrose pseudoinverse", which is what $(\vec{x}\vec{x}^T)^{-1}$ is, which is represented by a slash. So in the channel mapping code of this repository, you will find $\mathbf{A}^*$ computed as $\vec{y}\vec{x}^T/(\vec{x}\vec{x}^T)$ rather than as written in Eqn. 5.

5.2 Affine versus polynomial maps.

Before we describe how to find $\mathbf{A}$, or more precisely, how to fit it using a set of control points whose locations we know in both channels, we first give some intuition about what the different transformations implemented in this code do to a set of spots.

5.2.1 Affine transformations.

Affine transformations allow for stretching/shrinking, rotation, shear, translation, and reflection (which isn't really applicable to TIRF microscopy). An affine transformation can be represented as

$$\vec{y} = \mathbf{M}\vec{x} + \vec{b} \tag{6}$$

Fig. 7 shows how the elements of $\mathbf{M}$ and $\vec{b}$ accomplish the various changes possible with affine maps. As we will see below, an affine transformation describes donor-acceptor pairs of spots in smFRET fairly well, though a polynomial transformation tends to do slightly better (in our hands as well as in others'—see, for example, [1, 2]).[1]

5.2.2 Polynomial transformations.

Polynomial transformations represent a more general and less well-defined class of transformations than affine transformations, and in fact, as we will see, even the terms used to describe them in the IDL versus Matlab documentation differ in significant ways.

A polynomial transformation is one in which each element of the output vector, y_{j1} and y_{j2}, is determined by a polynomial of the input vector elements, x_{i1} and x_{i2}. The polynomials for y_{j1} and y_{j2} can be different. An example of a very simple polynomial transformation is

$$\begin{aligned} y_1 &= 3 + x_1 + 2x_2 \\ y_2 &= 1 + 5x_1 + 3x_2 \end{aligned} \tag{7}$$

Each term (3, x_1, $2x_2$, *etc*) is called a **monomial**. Note that this is actually a linear system of equations, and therefore is in fact an affine transformation—it can be written in matrix form as in Eqn. (6) above.

To be a polynomial transformation that is not affine, the **degree** of one or both of the polynomials that define the transformation must be greater than 1. Degree is defined in two ways: the **maximum**

[1] We note here that the Ha lab IDL code first generates an affine map, by asking the user to pick three true pairs of spots (three being the minimum number of points needed to define an affine transformation), and uses this to automatically match more spots with which to generate a polynomial map.

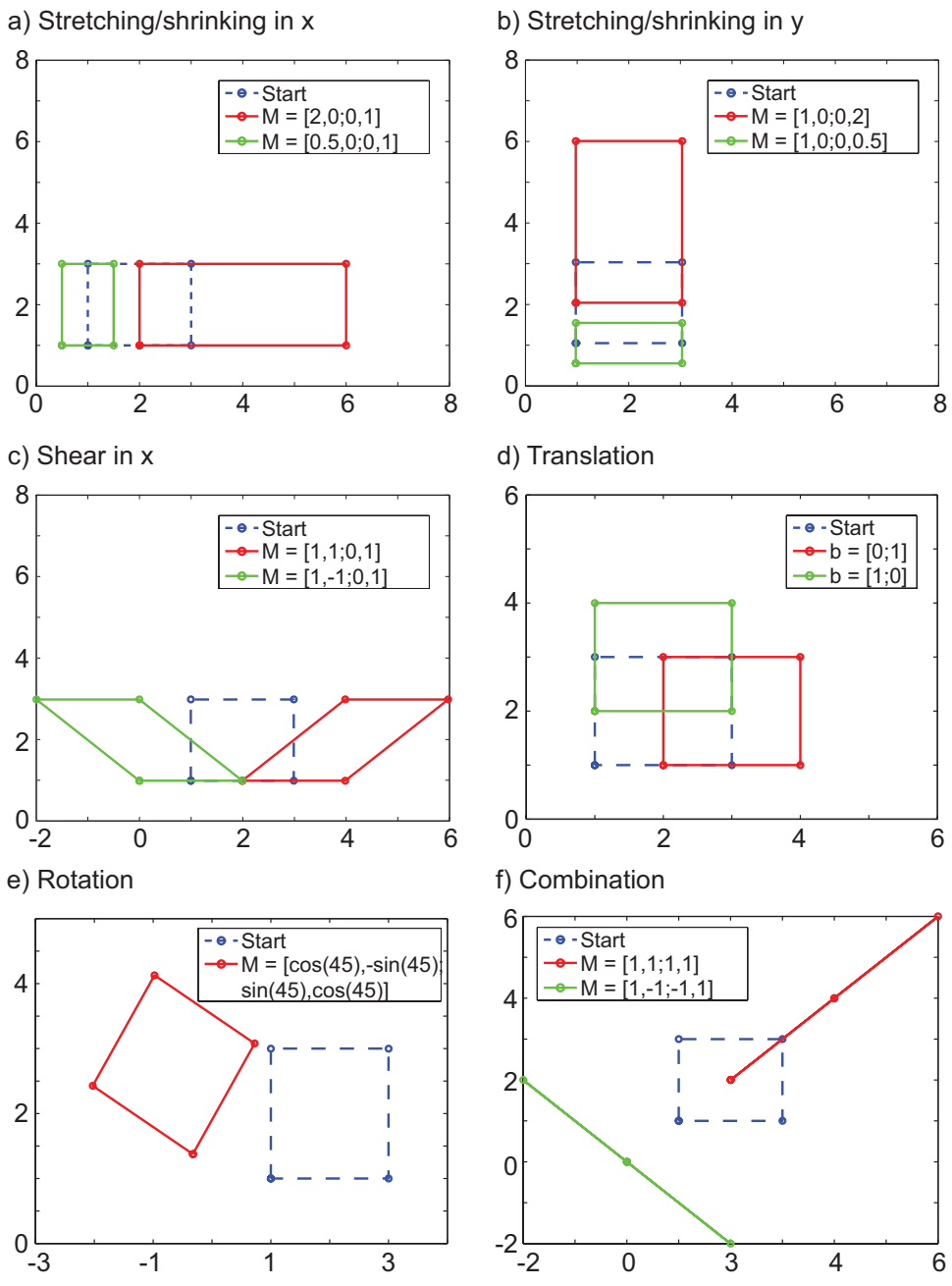

Figure 7: **Examples of affine transformations.** In all panels, the blue box outlined in dashed lines represents $\vec{x}$, the starting set of points. The legends show the choice of $\mathbf{M}$ or $\vec{b}$ that gives the transformed set of points $\vec{y}$, shown in green or red. Where not listed, $\mathbf{M} = [1, 0; 0, 1]$; and $\vec{b} = [0,0]$.

degree is the highest power of a single variable that appears in the polynomial; for example, in the polynomial $3 + x_1x_2 + x_1^2x_2$, the maximum degree is 2. The **total degree** is the highest power of combinations of variables that appears; for example, in the polynomial $3 + x_1x_2 + x_1^2x_2$, the total degree is 3.

This distinction becomes important when we look at the polynomial transformations that can be fit to sets of control points using IDL's POLYWARP function versus Matlab's fitgeotrans function. IDL defines the degree of the polynomial to be fit as the *maximum degree*, whereas Matlab defines the degree of the polynomial to be fit as the *total degree*. The Ha lab IDL code uses POLYWARP to fit a "third-degree" polynomial to control points to find the channel mapping; Matlab's fitgeotrans only implements total degree 2, 3, or 4 polynomials, which means we cannot use Matlab's built-in functionality to fit the same kind of polynomial as in the Ha lab code. To be specific, POLYWARP with a degree 3 polynomial allows terms like $x_1^3x_1^2$, because the maximum degree for that polynomial is 3; but because its total degree is 5, this monomial would not be included in a degree-4 polynomial fit (or a degree-3 fit) by Matlab's fitgeotrans.

We will examine below how these different polynomial transformations affect a set of points, but first we specifically define polynomial transformations according to IDL's POLYWARP and Matlab's fitgeotrans. POLYWARP finds parameter vectors $\vec{K_x}$, $\vec{K_y}$ such that, for a starting set of points $\left[\begin{smallmatrix} x_1 \\ x_2 \end{smallmatrix}\right]$ and an ending set of points $\left[\begin{smallmatrix} y_1 \\ y_2 \end{smallmatrix}\right]$,

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \sum_{i,j=0}^{D} K_{x_{i,j}} x_1^i x_2^j \\ \sum_{i,j=0}^{D} K_{y_{i,j}} x_1^i x_2^j \end{bmatrix}, \tag{8}$$

where D is the degree of the polynomial fit (set by the user). Note that for a maximum degree of 3, the highest-order term possible here is total degree 6, namely $x_1^3x_2^3$. More explicitly, we can write out Eqn. (8) as

$$y_1 = K_{x00} + K_{x10}x_1 + K_{x01}x_2 + K_{x11}x_1x_2 + K_{x20}x_1^2 + ... \tag{9}$$

$$y_2 = K_{y00} + K_{y10}x_1 + K_{y01}x_2 + K_{y11}x_1x_2 + K_{y20}x_1^2 + ... \tag{10}$$

Although POLYWARP returns $\vec{K_x}, \vec{K_y}$ as vectors, we can think of them as matrices, with the rows corresponding to the coefficients for each monomial in x_1 (x^0(=constant),x,x^2, ...) and the columns corresponding to the coefficients of each monomial in x_2. That is, for a degree 3 polynomial fit, we can arrange the elements of $\vec{K_x}$ as

	$\mathbf{x_2^0}$	$\mathbf{x_2^1}$	$\mathbf{x_2^2}$	$\mathbf{x_2^3}$
$\mathbf{x_1^0}$	K_{x00}	K_{x01}	K_{x02}	K_{x03}
$\mathbf{x_1^1}$	K_{x10}	K_{x11}	K_{x12}	K_{x13}
$\mathbf{x_1^2}$	K_{x20}	K_{x21}	K_{x22}	K_{x23}
$\mathbf{x_1^3}$	K_{x30}	K_{x31}	K_{x32}	K_{x33}

(11)

and similarly for $\vec{K_y}$. Note that there will therefore be 32 coefficients (16 for K_x, 16 for K_y) for a degree 3 polynomial fit in IDL.

On the other hand, Matlab's documentation (see images.geotrans.PolynomialTransformation2D class) tells us that a degree 2 polynomial fit returns a $\vec{K_x}$ vector of length 6 for a 2-degree fit, length 10 for degree 3, and length 15 for degree 4. This means that a third degree fit in Matlab will have only 20 parameters, rather than the 32 of IDL. Even the degree 4 fit will have only 30 parameters; though this will result in a fit of a comparable number of parameters to the IDL degree-3 fit, the particular

monomials that are allowed in that fit will be different. In particular, a third-degree fit in Matlab will return a $\vec{K_x}$ vector that would correspond to the matrix

$$\begin{array}{c|cccc} & \mathbf{x_2^0} & \mathbf{x_2^1} & \mathbf{x_2^2} & \mathbf{x_2^3} \\ \mathbf{x_1^0} & K_{x_{00}} & K_{x_{01}} & K_{x_{02}} & K_{x_{03}} \\ \mathbf{x_1^1} & K_{x_{10}} & K_{x_{11}} & K_{x_{12}} & - \\ \mathbf{x_1^2} & K_{x_{20}} & K_{x_{21}} & - & - \\ \mathbf{x_1^3} & K_{x_{30}} & - & - & - \end{array} \tag{12}$$

where we have indicated monomials that are not included in the fit by a dash.

How much does the degree of the polynomial, and the allowed monomials, affect the goodness of fit for pairs of smFRET spots? Is it simply the number of parameters that is important—do we just need 30 parameters to capture all the degrees of freedom of the microscope image—or the kinds of monomials that matter most? This is the question we take up in the next section.

5.2.3 Comparing affine and polynomial transformations of smFRET data.

Going to have two figures in this section: one of a grid to which we apply various transformations, and the other of real channel data under affine vs. various polynomial transformations.

Sum of squared residuals for: Affine = 42.5; Poly,Max3 = 1.45; Poly,Tot3 = 1.62; Poly,Tot4 = 1.39 ¡- for 200 beads. For a different mapping: Affine = 162; Poly,Max3=5, Poly,Tot3 = 5.7,Poly,Tot4 = 4.9 ¡-for 900 beads

We conclude that: (1) The complexities of the distortions between channels require more than the motions allowed by an affine transformation; therefore a polynomial transformation is required. (2) Since a polynomial transformation of total degree 4, with 30 parameters, does slightly better than a polynomial transformation of maximum degree 3 (and total degree 6), with 32 parameters, we conclude that it is not simply the addition of more parameters that improves a polynomial fit, but rather the increase in the complexity of the transformation by the addition of higher-order terms (x^4 rather than just x^3). Moreover, allowing additional cross-terms (for example, $x_1^3 x_2^2$) also does not seem to improve the ability of the polynomial transformation to fit the channel mapping points, which may be why Matlab's built-in polynomial transformation function, which is part of the image processing toolbox, focuses on total degree not maximum degree.

5.3 Finding non-linear transformations.

In Sec. 5.1 we described how to find the best-fit *linear* transformation, given a set of control points whose positions we knew with reasonable accuracy in both channels. Comparing Eqns. (3) and (6), it is not immediately obvious how to use the normal equations to find an affine transformation, and it is even less clear how to find a polynomial transformation.

What we need to do is to embed the affine and polynomial transformations in a linear space, such that they take the form of Eqn. (1). It is actually relatively straightforward to do this for an affine transformation. Redefine $\vec{x}$, $\vec{y}$, $\mathbf{M}$, and $\vec{b}$ as $\vec{x}'$, $\vec{y}'$, and $\mathbf{A}$ such that

$$\vec{x}' = \begin{bmatrix} \vec{x} \\ 1 \end{bmatrix}, \tag{13}$$

$$\vec{y}' = \begin{bmatrix} \vec{y} \\ 1 \end{bmatrix}, \tag{14}$$

and

$$\mathbf{A} = \begin{bmatrix} \mathbf{M} & \vec{b} \\ 0 & 1 \end{bmatrix}. \tag{15}$$

(All we're doing is adding a row of one's to $\vec{x}$ and $\vec{y}$, and combining $\mathbf{M}$ and $\vec{b}$ into a new matrix along with a new row of two 0's and one 1. Note that for an affine transformation in 2-space, $\mathbf{M}$ is 2x2 and $\vec{b}$ is 2x1—see also Fig. 7.) It should be clear that now $\vec{y}' = \mathbf{A}\vec{x}'$ is a linear transformation, whose solution for a best-fit $\mathbf{A}$ is given in Eqn. (5), and that it is equivalent to the affine transformation of Eqn. (6).[2] (If that second point is not immediately clear, do the matrix multiplication for $\mathbf{A}\vec{x}'$ and see that the first row of the resulting matrix is $\mathbf{M}\vec{x} + \vec{b}$.)

We can do a similar thing to embed a polynomial transformation in a linear space, with some slight modifications. Instead of the redefinitions above, we will instead define a new variable $\vec{\tilde{x}}$ such that

$$\vec{\tilde{x}} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1 x_2 \\ x_1^2 \\ \cdots \end{bmatrix}, \tag{16}$$

up to/including as many monomials as we want to include in our transformation, and where as a reminder (x_1, x_2) are the Cartesian coordinates of each point in the starting vector $\vec{x}$. Note then that if our transformation included, say, 20 monomials, and we have n points in $\vec{x}$, $\vec{\tilde{x}}$ would be a 20xn matrix. We define a new variable $\vec{\tilde{y}}$ similarly. Then our transformation matrix $\widetilde{\mathbf{A}}$ will contain the coefficients of the polynomials that relate the elements of the input vector (x_1, x_2) to those of the output vector (y_1, y_2).

Take, for example, the polynomial transformation

$$\begin{aligned} y_1 &= 3 + x_1 + 2x_1x_2 + 4x_2^2 \\ y_2 &= 1 + 5x_1x_2 + 3x_1x_2^2 \end{aligned} \tag{17}$$

We would then have

$$\vec{\tilde{x}} = \begin{bmatrix} 1 \\ x_1 \\ x_1 x_2 \\ x_2^2 \\ x_1 x_2^2 \end{bmatrix} \tag{18}$$

and

$$\widetilde{\mathbf{A}} = \begin{bmatrix} 3 & 1 & 2 & 4 & 0 \\ 1 & 0 & 5 & 0 & 3 \end{bmatrix}. \tag{19}$$

As with the embedding of the affine transformation in a linear space above, it should be clear from Eqns. (18) and (19) that $\vec{\tilde{y}} = \widetilde{\mathbf{A}}\vec{\tilde{x}}$ is linear, and that $\widetilde{\mathbf{A}}\vec{\tilde{x}}$ represents the polynomial transformation given by Eqn. (17) above.

[2] Note that Matlab's built-in affine transformation calculation function (cp2tform or fitgeotrans, depending on your version of Matlab) uses a different convention than I use here. Matlab computes the transformation: transpose($\vec{y}'$) = transpose($\vec{x}'$)$\mathbf{A}_{\text{Matlab}}$, such that $\mathbf{A}$ = transpose($\mathbf{A}_{\text{Matlab}}^{-1}$), where $\mathbf{A}$ is the transformation matrix in my notation.

5.3.1 Calculating the inverse transformation.

In the preceding sections we have focused on the question of finding a transformation that maps the points in one channel, denoted by $\vec{x}$, to the other channel, denoted $\vec{y}$. However, for smFRET we need to be able to go both directions. Here we take up the question of how to calculate the transformation from $\vec{y}$ to $\vec{x}$.

For affine transformations this is straightforward: given an affine transformation $\vec{y} = \mathbf{M}\vec{x} + \vec{b}$, the inverse mapping is simply (according to the rules of linear algebra) $\vec{x} = \mathbf{M}^{-1}(\vec{y} - \vec{b})$. This will work as long as $\mathbf{M}$ is invertible, which we know to be the case because ... well at the very least it's square ...

For polynomial transformations, on the other hand, we cannot say that $\vec{\tilde{x}} = \widetilde{\mathbf{A}}^{-1}\vec{\tilde{y}}$; at the very least, it is unlikely that $\widetilde{\mathbf{A}}$ will be square. In this case, we have to calculate the reverse transformation explicitly?

5.4 Channel mapping: Practical considerations.

This section considers several questions relating to the practice of creating and using channel mappings. Obviously these are most likely highly system specific. However the most important practical consideration for generating an accurate mapping is more universal, and that is to get correctly matched red- and green-channel spot pairs. The Ha lab IDL code does this by asking the user to hand-pick three pairs, which are then used to generate an affine mapping, which is good enough to find the spot-pair matchings that are then used to calculate a polynomial transformation. The code described here instead uses a greedy algorithm (described above) to find spot pairs, calculates a polynomial transformation with them, and then excludes spot-pairs that were not well-fit by the calculated transformation and recalculates a new polynomial transformation. In my experience the excluded pairs are always mismatches that shouldn't have been included in the transformation calculation anyway. It is important to note, however, that even a handful of mismatches (*e.g.* 5-10 mismatched pairs out of almost 1000 pairs total) significantly reduce the accuracy of the calculated transformation.

All of the transformations mentioned in this section are total degree 4 polynomial transformations calculated using Matlab's built-in fitgeotrans functionality.

5.4.1 How many spots are needed for a good mapping?

Consider a set of four bead movies of different areas on a bead slide, taken without removing the prism or the sample. When the first movie, with about 170 beads, is used to calculate the transformation, and this transformation is used to estimate the locations of the beads in the other three movies (total of about 600 beads), it is able to do so to within half a pixel or less. When instead the first two movies are used to calculate the transformation (about 350 beads total), that transformation estimates the locations of the beads in the other two movies to within 0.3 pixels or fewer. When the first movie's beads are subsampled and used to calculated polynomial transformations, these transformations are able to estimate the locations of beads in the other movies to within about 1 pixel or less, even with as few as 50 beads selected out of the first movie's 170 beads.

In conclusion, then, only one movie's worth of spots (100-200 spots) should be sufficient in order to calculate a good transformation, and indeed the Ha lab IDL code accepts only one movie with which to perform the mapping. However, in the interest of good scientific paranoia I usually take 2-3 movies, to ensure at least one will have a good distribution of beads with which to calculate the transformation, and usually use all of the movies to calculate the transformation, to ensure that I get a good estimate of the channel mapping in all areas of the field of view. I also sometimes check the transformation I

calculate on a given day by using 2 or 3 movies to calculate a transformation, and check how well it can find the beads in the other movie(s) (and this functionality is built into this analysis suite).

5.4.2 How much does the mapping vary hour-to-hour, day-to-day, or after removing the prism and replacing it?

As a starting point, consider two sets of mapping data that I collected one right after the other, without removing the prism or the sample. Both consisted of 5 bead movies taken at different areas of the slide, for a total of about 900 beads used for the mapping; however, the residuals for the first mapping set were 12, whereas for the second they were about 5, perhaps because I improved the focus and alignment of the TIRF spot between the two data sets. Regardless, using either data set to estimate where the beads should be in the other data set resulted in an error in the estimated bead position of less than 1 pixel. That is, applying a transformation to a new set of beads in the green channel finds the beads in the red channel to within 1 pixel of where they actually are, and vice versa for the red channel.

We next consider how much the mapping varies over a day. For this experiment I calculated a transformation based on five bead movies (total of about 850 beads; the "before" sample), removed the prism and the bead sample, took data on a different sample for an afternoon, and at the end of the afternoon replaced the bead sample and then collected 5 additional bead movies (the "after" sample). When I used the "before" transformation to estimate where the beads in the "after" sample should be, all of the "after" sample beads were found to within 1.1 pixels or less. The reverse was similar (using a transformation calculated from the "after" sample estimated the beads in the "before" sample to within 1 pixel or less. Note that this also means the removal and replacement of the prism does not affect the calculated transformation, eliminating one possible source of error in using a bead slide to calculating a transformation for use with a DNA sample.

Finally we consider how much the mapping varies day-to-day or week-to-week. A map created from data one month before a second test data set was able to estimate the locations of the beads in the test data set to within 0.6 pixels or less; a map created about two weeks prior to the same test data set was able to estimate the locations of the test beads to within 1 pixel or less; a second set of mapping- and test-data sets taken about two weeks apart produced an estimation error of 1.4 pixels or less. (The coefficients of the different days' polynomial transformations were similar but not identical.) Out of an abundance of caution, I prefer to calculate a map at the start of each day regardless, as this is a good way to make sure everything about the setup is working well, and it is important to be able to locate spots in each channel to within a pixel of where they actually are. But, the calculated transformation should not change much day-to-day.

5.4.3 Does the transformation differ significantly if it's calculated from beads versus real DNAs?

YES ARGGGGHHH

6 Acknowledgements.

I am extremely grateful to Matt Johnson and Luke Breuer for extensive help developing this software. This project would have been dead in the water without the significant intellectual and time contributions that they made! I am also indebted to TJ Ha's lab at UIUC, which has made what I consider to be "industry-standard" smFRET code freely available on their website. In this repository I make frequent allusions to the Ha lab code, noting where my code differs from theirs and where I have followed their lead. Thanks to the UCSF SMUG crew (Margaret Elvekrog, Thomas Noriega, Megan Mayerle, Sarah Ledoux, Daniel Elnatan) and the Narlikar lab, especially John Leonard, for helpful discussions about code as well as experimental development, and to Sarah Ledoux for the DNAs used to collect the data included in the Sample FRET Data repository. Finally I am grateful for financial support through NIH grants to my PI, Dr. Geeta Narlikar, and a Leukemia and Lymphoma Society fellowship.

References

[1] C. Joo and T. Ha. Single-molecule FRET with total internal reflection microscopy. In *Single-molecule techniques: A laboratory manual*, pages 3–36. Cold Spring Harbor Laboratory Press, 2007.

[2] S. Deindl and X. Zhuang. Monitoring conformational dynamics with single-molecule fluorescence energy transfer: applications in nucleosome remodeling. *Methods Enzymol*, 513:59–86, 2012.