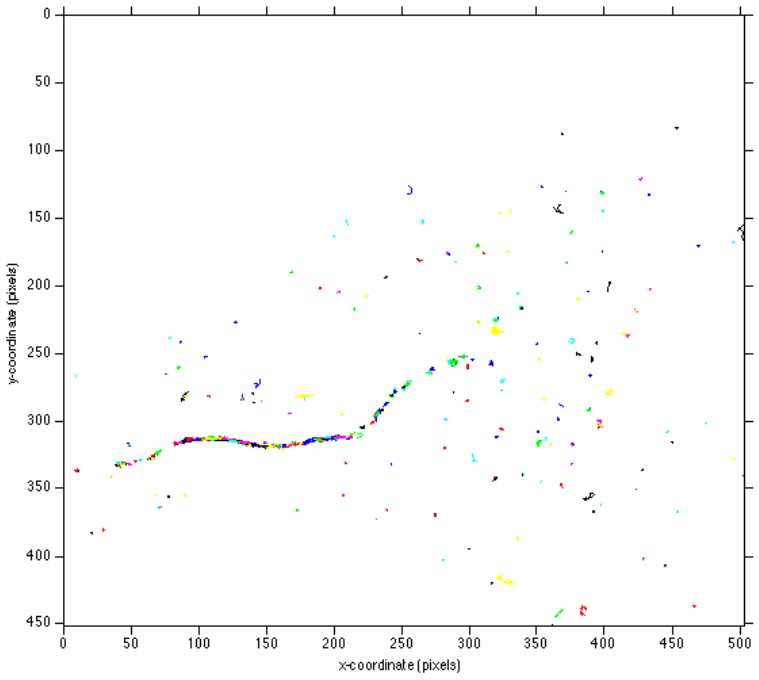
Kristen's uTrack Guide for tracking Nav1.6-dendra2

Updated: 5/22/13

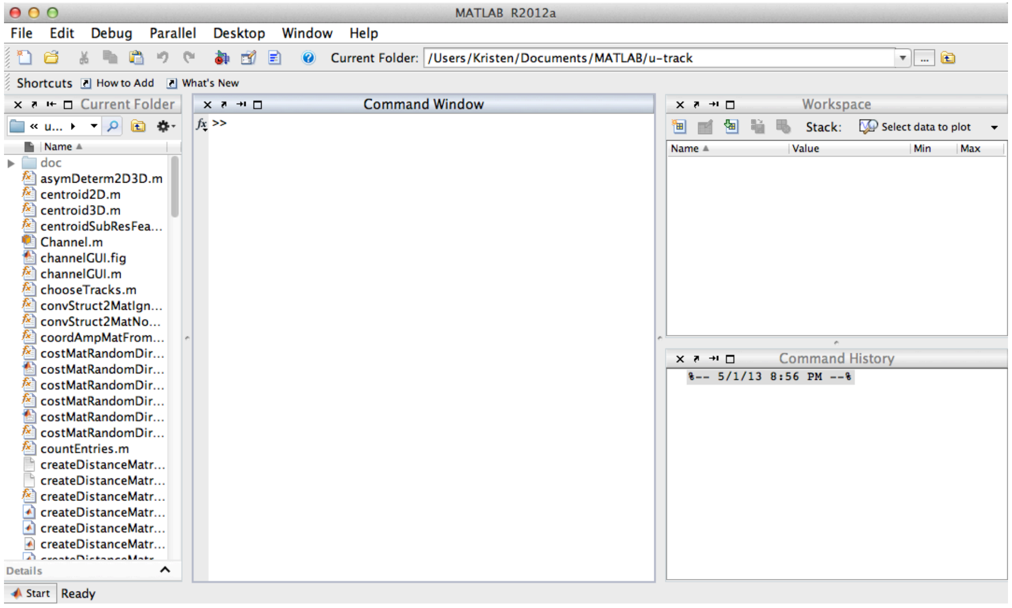
Files described are from the older version of u-track



Contents:

Setting up the workspace

1. Matlab will need to be (and should already be) installed on the computer of interest. Visit http://www.mathworks.com/ if you need to download the program or need help.
2. The uTrack files need to be downloaded (http://lccb.hms.harvard.edu/software.html for the latest version). This should also already be on the computer. These should be placed in your MATLAB folder
3. Need to have all of your folders in the path. To see what your current path settings are, go to File->Set Path. A list of all the paths should be displayed. Look to make sure all the folders with anything the program needs to access are in the path. Use "Add Path.." to add any missing folders.
4. Open Matlab. Go to the u-track folder. The window should look something like this:



Current Folder: /Users/Kristen/Documents/MATLAB/u-track

```
doc
asymDeterm2D3D.m
centroid2D.m
centroid3D.m
centroidSubResFea...
Channel.m
channelGUI.fig
channelGUI.m
chooseTracks.m
convStruct2MatIgn...
convStruct2MatNo...
coordAmpMatFrom...
costMatRandomDir...
costMatRandomDir...
costMatRandomDir...
costMatRandomDir...
costMatRandomDir...
costMatRandomDir...
countEntries.m
createDistanceMatr...
createDistanceMatr...
createDistanceMatr...
createDistanceMatr...
createDistanceMatr...
createDistanceMatr...
```

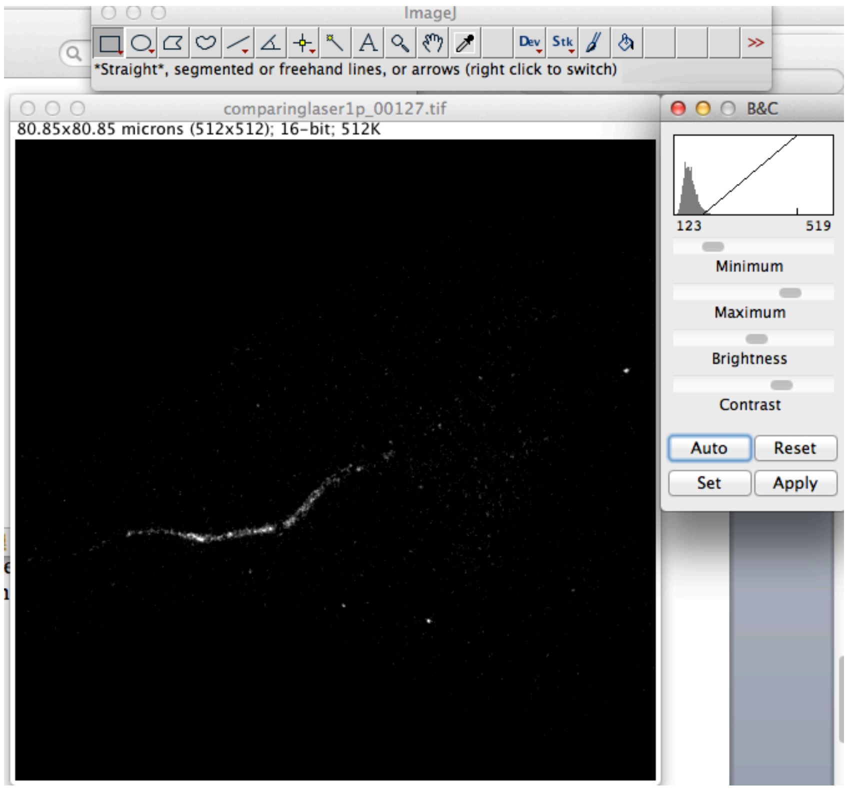
```
>>
```

```
%-- 5/1/13 8:56 PM --%
```

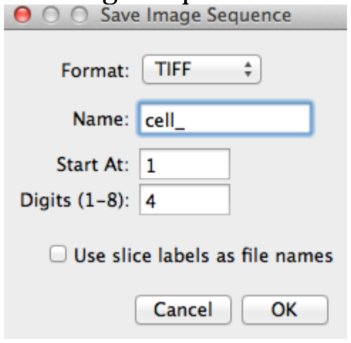
   a. If you are in the correct folder, you'll find the list of u-track specific functions on the far left side (Current Folder). Double clicking on any of these should result in opening of the script file in the Matlab Editor.
   b. The Command Window is where you will type commands to call programs to be used for analyzing movies.
   c. The Workspace window stores all the output variables for easy access by you and the program.
   d. Command History is useful in case you wish to easily re-run a script you've already run before without having to retype it in. This can be done by double-clicking the command in the window. If you don't want to do this, just exit out of the window.

Images

1. ImageJ should already be installed on the computer. If you need to download it - http://rsbweb.nih.gov/ij/download.html
2. The Bioformats plugin should also be on the computer, but if you need it you can download it here - http://loci.wisc.edu/bio-formats/imagej The website has instructions.
3. Now you can import the movies taken on the TIRF by going to ImageJ and selecting Plugins->LOCI->Bio-Formats Importer. Then simply find your .ND2 movie and open it. Another window will pop up asking for settings, the one that is most useful is checking "Virtual Stack" to save memory. Sometimes this didn't allow me to save changes to the image sequence though.
4. Images should be edited so that the background noise is reduced and the particles in the movie are still easily visible. This can be done in ImageJ by playing around with certain image properties. For example, what I have been doing is the following:
    a. Adjust brightness/contrast (Image->Adjust->Brightness/Contrast). This brings up the B&C window below with a histogram of intensity values for your image. Play with the settings until you have something you are happier with. It's best to adjust these settings in order (Min, Max, Brightness, then Contrast) because it will often undo changes if you go backwards at all.
    b. Subtract a certain value of intensity & below. Anything above the value is kept in the image. Find a dimmer, background pixel with the mouse and the "value = #" will appear in the toolbar window. Insert this number into the window (Process->Math->Subtract). Check "Preview" to see how it looks in the current image and adjust as needed.
    c. Gaussian Blur (Process->Filters->Gaussian Blur..). Use this to smooth the pixels with a desired pixel radius. This will get rid of a lot of noise since the blur averages the pixels within that radius. I've often used a sigma (radius) value of 0.6-0.7, but this can also be previewed so adjust as needed.


ImageJ
Straight, segmented or freehand lines, or arrows (right click to switch)
comparinglaser1p_00127.tif
80.85x80.85 microns (512x512); 16-bit; 512K
B&C
123
519
Minimum
Maximum
Brightness
Contrast
Auto
Reset
Set
Apply

5. The image sequence should be saved as a set of .tiff files. To save the set of images go to File->Save As..->Image Sequence.


Save Image Sequence

Format: TIFF

Name: cell_

Start At: 1

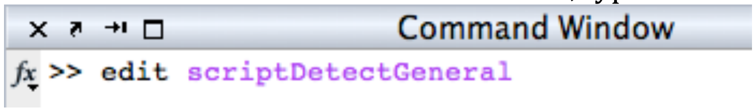
Digits (1-8): 4

☐ Use slice labels as file names

Save the files as "TIFF", name them whatever you would like. It's often best to use the format: **NAME_** so that the number sequence follows the underscore. I like to start my images at 1 so that the program doesn't skip the first image when it is reading zeroes (just in case). The number for Digits should be the number of digits in your last frame. For example, if you have 10,000 frames, Digits = 5. This allows the sequence to remain in the appropriate order as well as allowing the uTrack software to read the image files in the correct sequence.

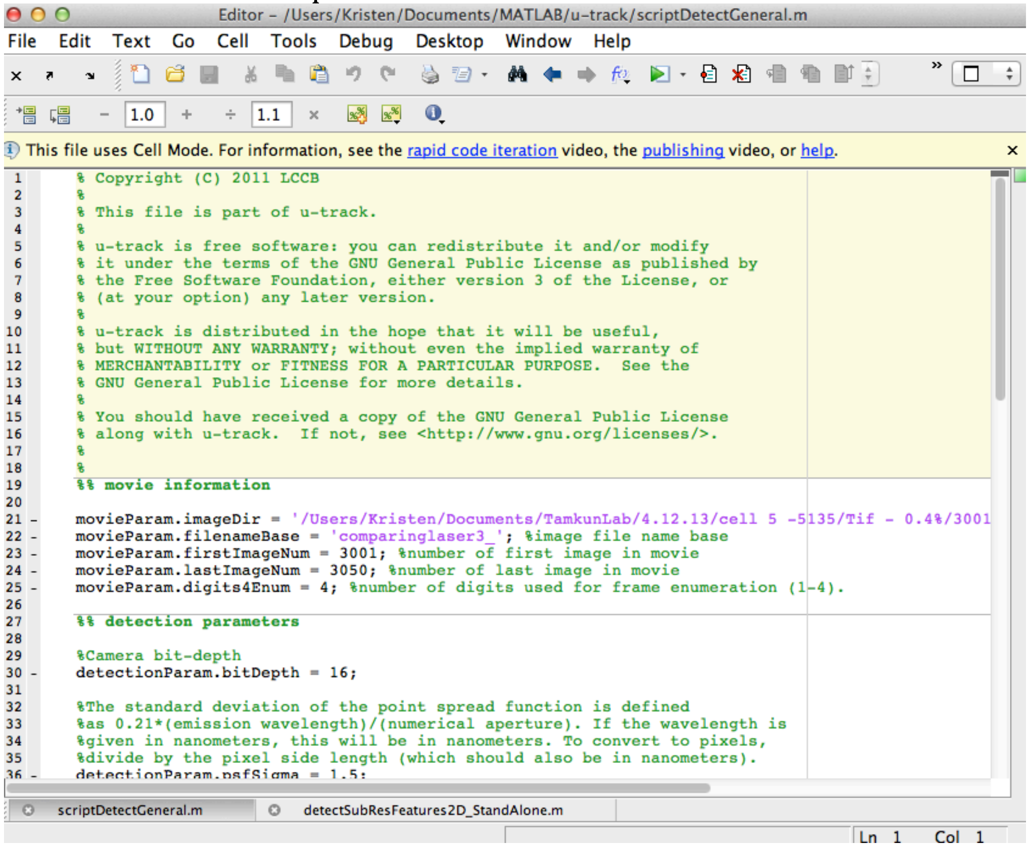
Adjusting Detection Parameters

1. In order to adjust how uTrack detects the particles in your images, you'll need to open the script itself. In the command window of Matlab, type in



```
>> edit scriptDetectGeneral
```

and hit enter. This should open the file in the Editor. The window will look like this:


Editor – /Users/Kristen/Documents/MATLAB/u-track/scriptDetectGeneral.m
File Edit Text Go Cell Tools Debug Desktop Window Help
1.0
1.1
scriptDetectGeneral.m
detectSubResFeatures2D_StandAlone.m
Ln 1 Col 1

This file uses Cell Mode. For information, see the rapid code iteration video, the publishing video, or help.

```
% Copyright (C) 2011 LCCB
%
% This file is part of u-track.
%
% u-track is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% u-track is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with u-track.  If not, see <http://www.gnu.org/licenses/>.
%
%
%% movie information

movieParam.imageDir = '/Users/Kristen/Documents/TamkunLab/4.12.13/cell 5 -5135/Tif - 0.4%/3001
movieParam.filenameBase = 'comparinglaser3_'; %image file name base
movieParam.firstImageNum = 3001; %number of first image in movie
movieParam.lastImageNum = 3050; %number of last image in movie
movieParam.digits4Enum = 4; %number of digits used for frame enumeration (1-4).

%% detection parameters

%Camera bit-depth
detectionParam.bitDepth = 16;

%The standard deviation of the point spread function is defined
%as 0.21*(emission wavelength)/(numerical aperture). If the wavelength is
%given in nanometers, this will be in nanometers. To convert to pixels,
%divide by the pixel side length (which should also be in nanometers).
detectionParam.psfSigma = 1.5;
```

As a brief overview of how the editor is set up for scripts…

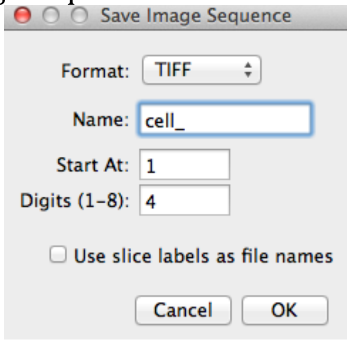
- Green text follows a "%" indicating a comment made by the programmer. This often has useful information about how the program works and how to use it.
- Pink/Purple text indicates anything between ' ' that allows the program to either output some kind of text or read file names for example. When working in uTrack using this notation ('insert text here') will be for typing in the directory where files are located/to be saved and what the file names are to be read/saved.
- Black text is code. Don't touch anything that isn't a number in this file. If you do, the code won't work.
- You can run the code straight from the editor by clicking this button ▶. The file will also automatically save if you didn't before pressing the button.

- At the bottom of the window is a list of the script files you have open in Matlab's editor. To switch to a different script file, you don't have to close the editor, just open the script, and click on the name at the bottom to open it (if it isn't already). This makes it easy to switch between the detection and tracking scripts. Click on the "x" to close any of the scripts.
- If you are typing and something that was a color other than black becomes black, you are probably missing either the apostrophe at the beginning/end or a % sign. Be sure that these all look correct or you will get an error.

2. Editing the script. For this and subsequent editing-related notes, I've simply copy+pastied the text instead of taking a screen shot. This will be in sequential order. **My notes are BELOW the script excerpts**

```
%% movie information

movieParam.imageDir = '/Users/Kristen/Documents/TamkunLab/4.12.13/cell
5 -5135/Tif - 0.4%/3001-6012/'; %directory where images are
movieParam.filenameBase = 'comparinglaser3_'; %image file name base
movieParam.firstImageNum = 3001; %number of first image in movie
movieParam.lastImageNum = 6012; %number of last image in movie
movieParam.digits4Enum = 4; %number of digits used for frame
enumeration (1-4).
```

This set of parameters is simple and so the program can find the files it needs to analyze. Figure out where you saved your image sequence and type it into the line where I currently have my last set's directory. The next line requires the image name that you saved when using ImageJ. In the case below I saved it as 'cell_', so that's what I would insert for the file name base. Enter the first and last frame number in the folder of images. And then the digits is the same number you used for "Digits" when saving the image sequence.


Save Image Sequence

Format: TIFF

Name: cell_

Start At: 1

Digits (1-8): 4

☐ Use slice labels as file names

```
%% detection parameters

%Camera bit-depth
detectionParam.bitDepth = 16;
```

This should stay the same for each movie. I haven't seen any reason to change it as it should be the same for each movie (depends on the TIRF I believe).

```
%The standard deviation of the point spread function is defined
%as 0.21*(emission wavelength)/(numerical aperture). If the wavelength
is
%given in nanometers, this will be in nanometers. To convert to pixels,
%divide by the pixel side length (which should also be in nanometers).
detectionParam.psfSigma = 1.5;
```

This should also remain the same in this case as the emission wavelength will be the same for each movie using dendra2. However if you need to calculate it, the formula is in the comments. I just used 1.5 as this is what has been previously used in the lab. (0.21) * [(553 nm) / (numerical aperture)] * (number of pixels per 1 nm).

```
%Number of frames before and after a frame for time averaging
%For no time averaging, set to 0
detectionParam.integWindow = 3;
```

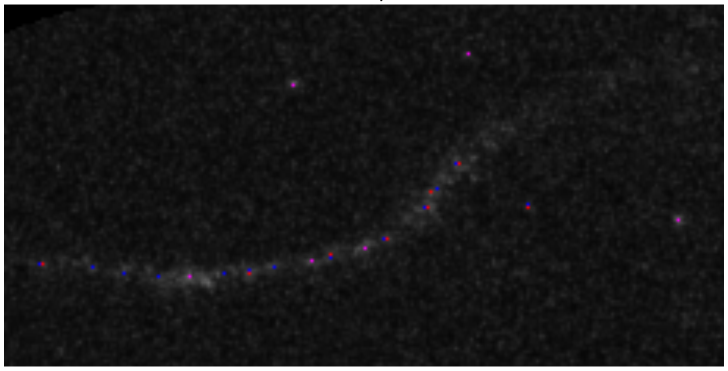
This is used for time-averaging of the frames. The number represents the number of frames to be used in averaging around each image. If there is little to no movement of your detected particles, this number can be large. For our purposes, since the channels are moving around, we want to use a smaller number. Changing this number will change the signal-to-noise ratio. It blends the given number of frames around each image it is looking for local maxima so that random noise will be eliminated (background). Anything that stays in the same general area over the frames is considered a detected local maximum and can move on to the next step.

The larger the number, the stricter the criteria are for being considered a particle. These local maxima will go through further model fitting before being deemed a detection. In the example on the next page, ignore the different colors since they indicate a later step. For now, look at the total number of colored points between 2 and 3 frames. There isn't much variation between the two for the axon, which makes sense because there isn't much movement there. Where this may become important is in the soma because there is more movement overall.

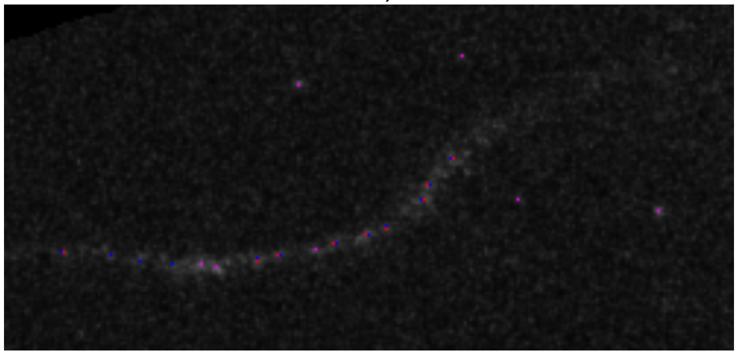
If you want more particles detected overall, it's probably best to let this be below 3, since it'll average that number of frames before and after the frame it's analyzing. Anything from 0-3 should work, but this will depend on your movie and how happy you are with the number of detected particles or tracked particles.

Examples of changing this parameter (on the same frame):

3 frames, axon:



2 frames, axon:



3 frames, soma:

2 frames, soma:

```
%Alpha-value for initial detection of local maxima
detectionParam.alphaLocMax = 0.0001;
```

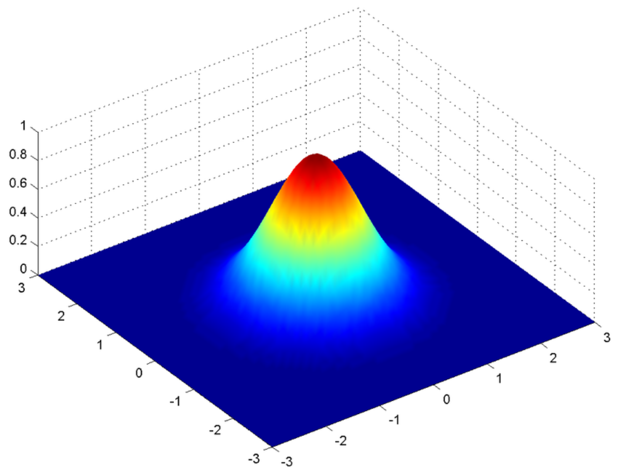
For this, the alpha value is used to compare the detected local maxima with the background to see if it is significantly brighter. The smaller the number, the stricter it is. This will also help eliminate noise in the detection process. So far a value of 0.0001 has been good enough for detecting particles and filtering out the potential noise. If we go to a larger value, more objects will be detected to move onto the Mixture Model Fitting below, but there is a higher chance of seeing noise as a detected maximum. For lower intensity movies (like using the 405nm laser at 0.3% power) it may be worthwhile to increase this number, since fewer particles are detected over all, but 0.0001 has been fine for longer movies. For example, in the images above, there are pink/red/blue pixels indicating detections. The pink/red ones are the ones that pass the test in the end. The blue ones are not considered detected particles. Lowering alpha will likely decrease the number of pink/red ones. Increasing alpha will likely increase the number of blue ones.

```
%Maximum number of iterations for PSF sigma estimation for detected
local
%maxima
%To use the input sigma without modification, set to 0
detectionParam.numSigmaIter = 10;
```

The number of times the program will estimate the Point Spread Function (PSF) sigma for the local maxima. It takes the sigma value you entered before (in this case I've been using 1.5), then it will use that and go through iterations to estimate a more accurate PSF sigma for each movie. In the movies I have analyzed it has generally been around 0.8-1.4. 10 iterations is a good number for this, any more likely won't give you a more accurate fit (likely the change in number will come in the 1/1000th place or so after 10 iterations). This is good to do so the program knows how much intensity spread there is for each spot we consider a channel. It helps it detect the channels better since we are lowering our PSF below the "limit" of the microscope by using TIRF + PALM, but it won't be exactly the same for every movie.

```
%1 to attempt to fit more than 1 kernel in a local maximum, 0 to fit
only 1
%kernel per local maximum
%If psfSigma is < 1 pixel, set doMMF to 0, not 1. There is no point
%in attempting to fit additional kernels in one local maximum under
such
%low spatial resolution
detectionParam.doMMF = 0;
```

This is where the program decides how many Gaussian kernels it will attempt to fit to each detected local maxima. Essentially, a Gaussian is a function that has a large peak centered around some point and the curve spreads. It looks like this for a 2D fit.



The peak (red) corresponds to the brightest pixel in our detected maximum. The spread of the intensity varies for each particle, but should be fairly similar. The program will attempt to fit something like this for each detected maximum and if it isn't able to, the detected maximum won't be considered a particle. If you want a more accurate fit, set this to 1 and the program will attempt to fit multiple kernels to each detected maximum. This will make the run-time longer, but may be worth it for some movies if you're concerned about the particles that were detected. If this is on (1), then uTrack may even be able to more easily distinguish two particles that are really close to each other using the alpha values for statistical tests below. It tells you that if your PSF sigma is <1, then to set it =0. In some cases, the PSF sigma becomes a value <1. If you run the detection once with doMMF=0, check the PSF sigma value (in the workspace window, it'll have the value), then if it is >1, run it again, you can compare the two and see if the fits improve.

Here is an example of detection with the doMMF = 1 / 0

```
%Alpha-values for statistical tests in mixture-model fitting step
detectionParam.testAlpha =
struct('alphaR',0.001,'alphaA',0.001,'alphaD',0.001,'alphaF',0);
```

Here are another set of alpha values used to determine how good the fit of the Gaussian is. The R=Residual, A=Amplitude, D=Distance. Here is a description of each copied from one of the PDF files included with the uTrack package. F=Final isn't included in the test, so always keep it at 0.

Residuals: Alpha-value to decide whether the addition of a Gaussian significantly improves the model fit to the image at a local maximum. Needed only if "Do Iterative Gaussian Mixture-Model Fitting" is checked.
Amplitude: Alpha-value for testing whether the amplitude of the fitted Gaussian (reflecting object intensity) is significantly different from zero, given the uncertainty in the amplitude and noise fluctuations.
Distance: Alpha-value for testing whether the distance between two fitted Gaussian centers (reflecting the underlying object positions) is significantly different from zero, given the uncertainty in the positions.

Again, like the alpha value earlier in the detection code, it's more strict criteria wise if the value is smaller. AlphaR is only required if doMMF=1 in the previous line of code and is used to determine if the extra Gaussian fits are better than the first one. If you do a quick check of your detections (say, limit the detection to 10 frames and turn on the visualization below) and you don't see enough detected particles (pink/red pixels = those that pass the model fitting) then these would be parameters to play with. Increasing them will give you more particles because the criteria are less strict. If your movie doesn't have clearly Gaussian looking spots, it's probably better to go with less strict parameters.

```
%1 to visualize detection results, frame by frame, 0 otherwise. Use 1
```

```
only
%for small movies. In the resulting images, blue dots indicate local
%maxima, red dots indicate local maxima surviving the mixture-model
fitting
%step, pink dots indicate where red dots overlap with blue dots
detectionParam.visual = 0;
```

This will give you the images from your movie with a pixel overlaid on each detected local maximum, like in the images earlier in this section if it is set to 1. You want to keep this at 0 when doing your main round of detection because otherwise you'll have to wait for it to load all of the images as well as close them all when it's finished. If you want to look at how the detection works with your parameters, it's best to change the first and last image (from the very first section of this script) to a smaller, more manageable number. I think that doing this before running the full detection (changing first/last frame back to the original numbers)

```
%absolute background information and parameters...
%(for not using this section, simply comment it out, as supplied by
default)
% background.imageDir = ???; %directory where background images are, in
the same format as movieParam.imageDir
% background.filenameBase = ???; %background image file name base.
NOTE: There must be a background image for each image to be analyzed
% background.alphaLocMaxAbs = 0.001; %alpha-value for comparison of
local maxima to absolute background
% detectionParam.background = background;
```

You would only need to use these if you have an image of what the background looks like compared to the neuron, which wouldn't be the case here since there is just random noise in the background and specks that sometimes go away anyway. If you decide to use it, just delete the % in front of each command and fill it out. I've not touched this and don't think you'll need to either.

```
%% save results

saveResults.dir = '/Users/Kristen/Documents/TamkunLab/5.2.13/Cell 1 -
5100/Tif - 0.4%/'; %directory where to save input and output
saveResults.filename = 'Cell1_Detection_test.mat'; %name of file where
input and output are saved
```

This is just for changing the directory where you store the results. Each time you run a detection, you'll need to make sure it's saved in the appropriate folder and with a name that will help you determine which run this is. For instance, I used the word "test" in mine so I could find an image for my presentation and this guide without rewriting the original saved file. If you keep the name the same, each time the script runs, it will overwrite that save. So if you are just playing with parameters, using something like "test" in the name is useful. If you wanted to save the different results due to the different parameters, you want to save each file separately.

Adjusting the Tracking Parameters

1. First open the script in the Matlab editor by typing "edit scriptTrackGeneral" into the command window or by double-clicking on the script itself in the current folder window. It should open up the same way that the detection script did.
2. In order for the tracking to work, you need to have the results from the detection open. If you run the tracking script immediate after the detection process has finished, it should be opened already. If for some reason, the tracking needs to be done at a separate time then you'll need to find where you saved the results for the detection. In the example script I posted, my detection file was saved as "Cell1_Detection_test.mat", so if I find this file and open it, all the data will show up in the workspace so that the tracking script can actually find it and put it all together. This is also useful if you are playing with seeing how different detection parameters affect tracking in the end.
3. Editing the script works the same as for the detection script. My comments are again, below the actual script excerpts.

```
%% Cost functions

%Frame-to-frame linking
costMatrices(1).funcName = 'costMatRandomDirectedSwitchingMotionLink';

%Gap closing, merging and splitting
costMatrices(2).funcName =
'costMatRandomDirectedSwitchingMotionCloseGaps';

%----------------------------------------------------------------------
----

%% Kalman filter functions

%Memory reservation
kalmanFunctions.reserveMem = 'kalmanResMemLM';

%Filter initialization
kalmanFunctions.initialize = 'kalmanInitLinearMotion';

%Gain calculation based on linking history
kalmanFunctions.calcGain = 'kalmanGainLinearMotion';

%Time reversal for second and third rounds of linking
kalmanFunctions.timeReverse = 'kalmanReverseLinearMotion';

%----------------------------------------------------------------------
----
```

These are not to be touched, they are here for the program to get the right functions (written in other matlab scripts).

```
%% General tracking parameters

%Gap closing time window
gapCloseParam.timeWindow = 2;
```

This number indicates how many frames the detected particle is allowed to disappear for and still be part of that track. Since the program will look at each frame for detections and uses a prediction-based algorithm to make trajectories, it takes into account frames prior to and after the one it's analyzing. This is useful if you're looking at cases where the fluorescence dims for some reason, but comes back into the field of view. If you were confident they are the same particle, longer closing times would work well. In this case, however, it's probably better for it to be smaller since it's not obvious to determine if the disappearance of a particle isn't due to bleaching, but rather moving away from the surface quickly then coming back. Perhaps in a couple of frames, the detected local maximum doesn't fit a nice Gaussian kernel to it, so the program will then separate the trajectories because there are >2 frames that didn't have a Gaussian fit to it. This would possibly become a problem if the rate of bleaching and photoactivation were very fast and there were many molecules appearing and disappearing in that short time frame. In such a case, as one molecule disappears, another may show up within 2 frames and be connected to that track even though they were separate molecules. 2-3 has been a pretty good number so far, but in watching the movies you can probably see if these events are occurring frequently enough (or at all) to justify lowering this to 1 or even 0 (which in my opinion, 0 would be too strict because of the possibility that the molecule just didn't have a good Gaussian kernel fit for 1 frame). Increasing this number may give you more trajectories and longer trajectories, but how trustworthy this is in our particular set up is iffy at best. Though this may be good to increase when looking at quantum dots that "blink" for some average number of frames.

```
%Flag for merging and splitting
gapCloseParam.mergeSplit = 1;
```

This is just in case you wish to keep merging and splitting events as detected by the program. When watching the tracks overlaid on movies, these events will show up as a cyan/blue * or a green/yellow diamond depending on the parameters set for the overlay (discussed later). The idea is that the program will look to merge or split track segments if they seem to overlap or a track of one particle turns into a track for two. This can be a problem if two particles run into each other, overlap to the extent that they aren't resolved enough to distinguish by the program. Then the program will take one of those tracks and connect it to the merged particles' tracks/motion. If they then move apart, the track will split into two separate tracks. It will also try to determine if the merged track is part of one of the split tracks and connect those tracks if so. Sometimes this happens even when the program is accurately following a particle and the merging/splitting can allow you to have tracks that don't get chopped up just because it moves close to another particle. If you have this on, it's useful to look at the movie. If you are getting a lot of these events within your movie, it may be useful to not allow merging and splitting. This will give you shorter track segments and potentially multiple short tracks for one particle, but to eliminate the possibility of inaccurate long tracks I would turn it off. In the case of lower laser power we have fewer particles close enough to create many large blobs for the merging/splitting to be completely inaccurate, so I've also left it on in the past and it has been fine from what I can tell. To turn it off, set it =0, on=1.

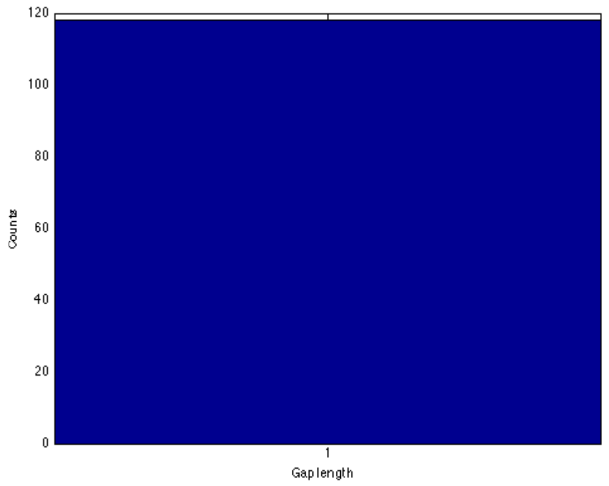
```
%Minimum track segment length used in the gap closing, merging and
%splitting step
gapCloseParam.minTrackLen = 15;
```

This determines the minimum number of frames a particle must be a part of a track for the program to connect it to other track segments in the following step for gap closing (explained below). Essentially, if you keep this around 10-15 frames, we won't get a lot of very short segments being connected together, being merged together or being split from each other. This helps us prevent two spots passing near each other and becoming one track, then splitting apart if each segment is <15frames in this case.

```
%Time window diagnostics: 1 to plot a histogram of gap lengths in
%the end of tracking, 0 or empty otherwise
gapCloseParam.diagnostics = 0;
```

Setting this equal to 1 will turn on Gap Length Histograms. When the program finishes tracking, figures (bar graphs) will pop up showing you the distribution of gap lengths in that movie. This is a way to very roughly see if you are getting some track jumping or two particles being part of the same track. This is because the gap length is the number of frames that the particle is allowed to disappear for and still be part of the track. Essentially, it takes the end of one track and attempts to fit it to the beginning of another track if they seem to be the same particle. This is useful, again, in the case where there are a couple of frames without a good Gaussian fit for that particle, but maybe within those couple of frames, it still moves around. It's bad for cases where a particle bleaches at the same time as one photoactivates within a close distance because then those trajectories could be connected even though they are clearly separate particles. If there are a lot of bins (number of events with a certain feature) in the larger gap lengths, then there are a lot of these tracks being joined together with a gap length of whatever your maximum gap length is. Which may or may not be accurate.

The histograms look like this:

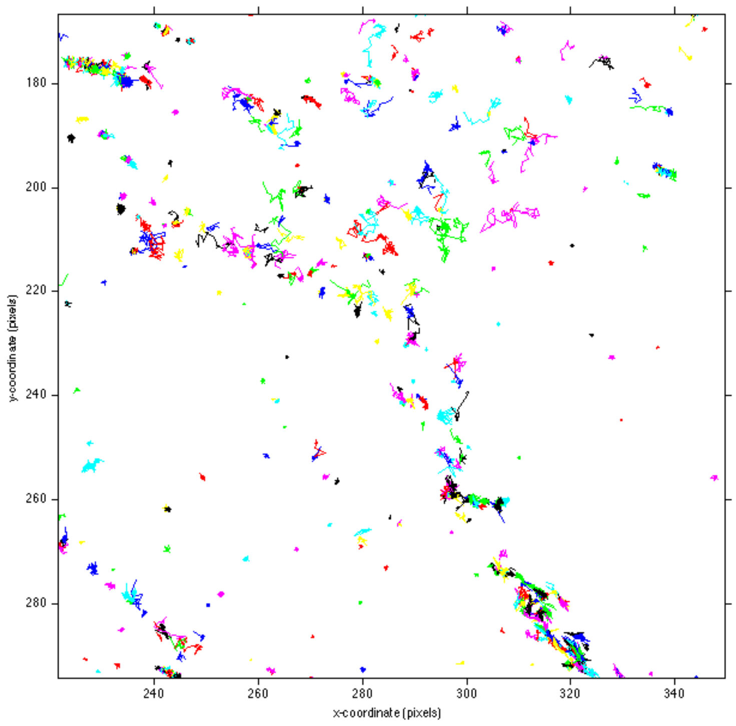


Since I've set my gap length = 2, there are a limited number of possible bins. However, from this plot it looks like I've got about 120 counts of 1 frame disappearances, and no 2 frame disappearances in my tracks. I'd say that's pretty good overall, even if we did get some 2 frame counts.

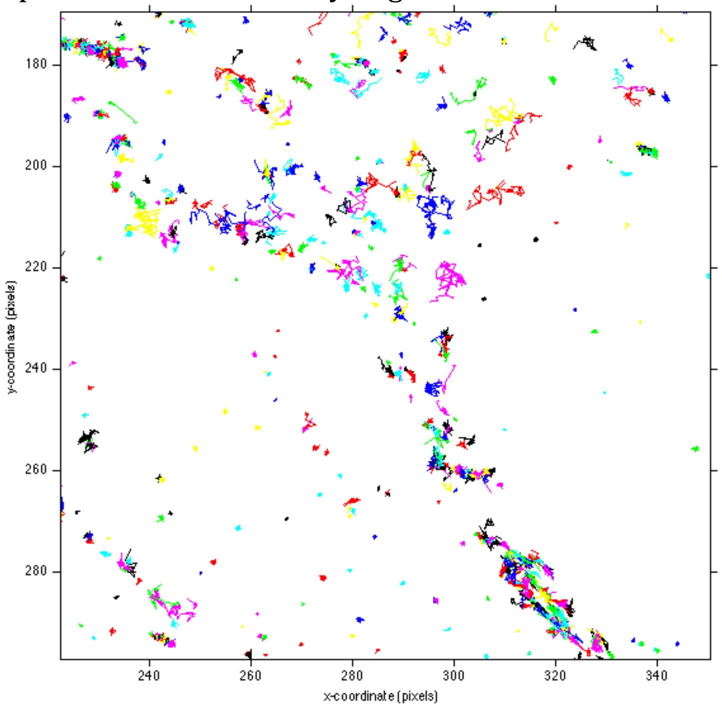
```
%-----------------------------------------------------------------------
----

%% Cost function specific parameters: Frame-to-frame linking

%Flag for linear motion
parameters.linearMotion = 0;
```

This is used to tell the program whether or not it should be looking for linear motion or random motion. To have it on, set it =1, this will then tell the program to try to use a prediction-based algorithm for linear motion to predict where the particle is going. This will allow the program to effectively find trajectories when there is clear directed motion. If it's off, the algorithm will be better for looking at randomized motion. I think either one has been fine for movies I've looked at. Here's an example with it on and off:

Linear Motion OFF (=0), Random motion is assumed



Linear Motion ON (=1), linear motion is looked for using a prediction algorithm to guess where the particles are most likely to go.



Ignore the weird drift in the tracks, that's discussed later. This is just for an example. There doesn't seem to be much difference in the way the tracks look between the two.

```
%Search radius lower limit
parameters.minSearchRadius = 2;
```

The minimum search radius doesn't mean that the program will only connect particles that are within a circle of radius >2 or = 2pixels (in this case). It will still look for detected particles within a radius from 0 to the number you enter. However, the initial search for the particle in the next frame will start with a circle with a radius that is the *average* of your minimum and maximum search radius. So, if you want to try to fix jumping, it may be useful to lower this to 1 or 0.5 or something smaller so that the first time it looks for a link, it searches a very short distance and moves up to the maximum search radius and back down to 0. If it finds something initially, the chances of it keeping them together in a track is higher, or so I like to believe, I'm not actually positive on that one. I would think that if it were to link two particles in two frames, it would keep the first one it found and if it didn't find any, then it would search the entire search radius until it found something. This is also a way to try to reduce track jumping. Setting it =1 may even be better.

```
%Search radius upper limit
parameters.maxSearchRadius = 3;
```

This is one of the main parameters I played with to attempt to fix track jumping in the AIS. Essentially it will not look for detected particles that are outside a circle of radius = 3 around the detected particle it is attempting to link to others in the surrounding frames. Initially what was being used was a search radius of 5. I think a search radius of 3 does the job pretty well

```
%Standard deviation multiplication factor
parameters.brownStdMult = 3;
```

This is the multiplication factor for determining the search radius from the displacement standard deviation. I don't think it's crucial to change this for our purposes.

```
%Flag for using local density in search radius estimation
parameters.useLocalDensity = 1;
```

I leave this on (=1), so that it takes into account the local density of the spots when estimating the search radius. In theory, I would expect this to be important in preventing track jumping when there are many nearby spots for adjusting the search radius, but it wasn't crucial in getting rid of the jumping.

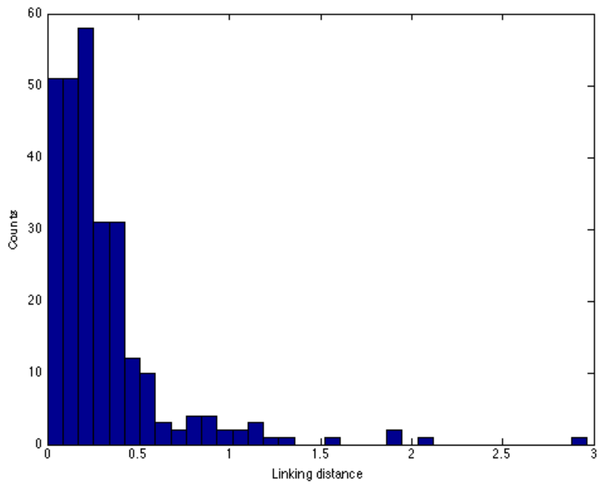
```
%Number of past frames used in nearest neighbor calculation
parameters.nnWindow = gapCloseParam.timeWindow;
```

This defines the number of frames that will be used in the "nearest neighbor calculation". I assume that the program takes into account the surrounding particles when looking to link particles together in a track. However, sometimes those particles will disappear so it can't use that as a correlation for the other particle being the same. This basically lets it use past frames to determine what is happening. The default it uses here is the number of frames you gave it for the gap closing time window (in this script we have it =2). However, you don't have to keep it that way and can define a different number for it to use.

```
%Optional input for diagnostics: To plot the histogram of linking
distances
%up to certain frames. For example, if parameters.diagnostics = [2 35],
%then the histogram of linking distance between frames 1 and 2 will be
%plotted, as well as the overall histogram of linking distance for
frames
%1->2, 2->3, ..., 34->35. The histogram can be plotted at any frame
except
%for the first and last frame of a movie.
%To not plot, enter 0 or empty
parameters.diagnostics = 0;
```

Generally I have this off. If you suspect track jumping may occur, looking at the histograms of linking distances for whichever range of frames [1st, last] would be a quick way of seeing if there is any. If you turn this on, just do it for a few frames (since it creates one histogram for each set of linked frames). If you're getting track jumping the histogram will have a lot of bins in a larger distance, if not, then the distribution should decay from 0.

Example:



However, this histogram is a snapshot for one frame's linking distances, so I generally like to look over a few frames and see if there's any distinct pattern different from this. Say, a lot of bins in the "3" column over multiple frames might indicate that for some reason, there are many tracks being linked a distance of 3 pixels, which is actually quite a bit in our movies.

```
%Store parameters for function call
costMatrices(1).parameters = parameters;
clear parameters

%-----------------------------------------------------------------------
----
```

```
%% Cost function specific parameters: Gap closing, merging and
splitting

%Same parameters as for the frame-to-frame linking cost function
parameters.linearMotion = costMatrices(1).parameters.linearMotion;
parameters.useLocalDensity =
costMatrices(1).parameters.useLocalDensity;
parameters.minSearchRadius =
costMatrices(1).parameters.minSearchRadius;
parameters.maxSearchRadius =
costMatrices(1).parameters.maxSearchRadius;
parameters.brownStdMult =
costMatrices(1).parameters.brownStdMult*ones(gapCloseParam.timeWindow,1
);
parameters.nnWindow = costMatrices(1).parameters.nnWindow;
```

These are just setting the parameters for this section equal to some from the earlier portion of the script. You could change these to actual numbers if you wanted, but I don't see a reason for you to at this point. Could be interesting to play around with at some point though.

```
%Formula for scaling the Brownian search radius with time.
parameters.brownScaling = [0.5 0.01]; %power for scaling the Brownian
search radius with time, before and after timeReachConfB (next
parameter).
```

This determines how the search radius grows over time after it's initial search. Another reason I believe that the tracks were jumpy previously is that the initial search radius was r=(2+5)/2 = 3.5, then the search radius grew with some rate to 5, then does the 0->2 search. So it finds the spot next to it first, connects the tracks and moves on before looking within the 2 pixel radius (which is where they seem to stick often in the AIS. By changing the initial search radius to (3+2)/2=2.5 or (3+1)/2=2, this helps it not pick a different particle first. So these two numbers will determine how quickly the search radius grows based on the time scale below…

```
parameters.timeReachConfB = 4; %before timeReachConfB, the search
radius grows with time with the power in brownScaling(1); after
timeReachConfB it grows with the power in brownScaling(2).
```

This determines the number of frames to include in the growing search radius. For 4 frames it will grow with power, r^0.5 and after 4 frames, it'll grow by r^0.01 in our case. So it grows more slowly after 4 frames.

```
%Amplitude ratio lower and upper limits
parameters.ampRatioLimit = [0.7 4];
```

Sets the maximum ratio of intensity difference between connected spots. So, say your channel moves and dims a bit at the same time, the program would only correlate that to being the same spot if the ratio of intensities match what you give it. These numbers would depend on how much you think the intensities change on average for a movie. In theory, they shouldn't change very much since we're somewhat strict about what gets detected as a particle in the first place. So if it dims so much that it doesn't get considered a detection, then it doesn't matter if this is set such that it would be included.

```
%Minimum length (frames) for track segment analysis
parameters.lenForClassify = 10;
```

This sets the minimum number of frames used to analyze the tracks and classify them as random Brownian motion or linear motion. Any track under this length will not be classified. This is also used to determine how the search radius will scale (Brownian parameters above or Linear parameters below) for each track. I don't think it is necessary to change this. You could lower it and see if there are better tracks as a result of the scaling being applied to shorter segments

```
%Standard deviation multiplication factor along preferred direction of
%motion
parameters.linStdMult = 3*ones(gapCloseParam.timeWindow,1);
```

Here it's looking at linear motion, and this determines how the standard deviation form the linear direction of motion is allowed to look. I don't think it's necessary to mess with this.

```
%Formula for scaling the linear search radius with time.
parameters.linScaling = [0.5 0.01]; %power for scaling the linear
search radius with time (similar to brownScaling).
parameters.timeReachConfL = gapCloseParam.timeWindow; %similar to
timeReachConfB, but for the linear part of the motion.
```

This is basically the same idea as with the Brownian scaling above, but now applied to tracks classified as linear motion and >10 frames in length.

```
%Maximum angle between the directions of motion of two linear track
%segments that are allowed to get linked
parameters.maxAngleVV = 30;
```

Like it says, the maximum angle that two track segments can be at and still be linked together. It doesn't make much sense to allow two tracks moving perpendicular to each other to be connected, 30 degrees is reasonable though. I would stay <45 degrees for this.

```
%Gap length penalty (disappearing for n frames gets a penalty of
%gapPenalty^n)
%Note that a penalty = 1 implies no penalty, while a penalty < 1
implies
%that longer gaps are favored
parameters.gapPenalty = 1;
```

I've just always left this at one. I haven't had a problem with it so far considering my gap length is so low. If you start getting weirdly connected tracks like I described for the gap length closing parameters, it would be useful to increase this number.

```
%Resolution limit in pixels, to be used in calculating the merge/split
search radius
```

```
%Generally, this is the Airy disk radius, but it can be smaller when
%iterative Gaussian mixture-model fitting is used for detection
parameters.resLimit = 3.4;
```

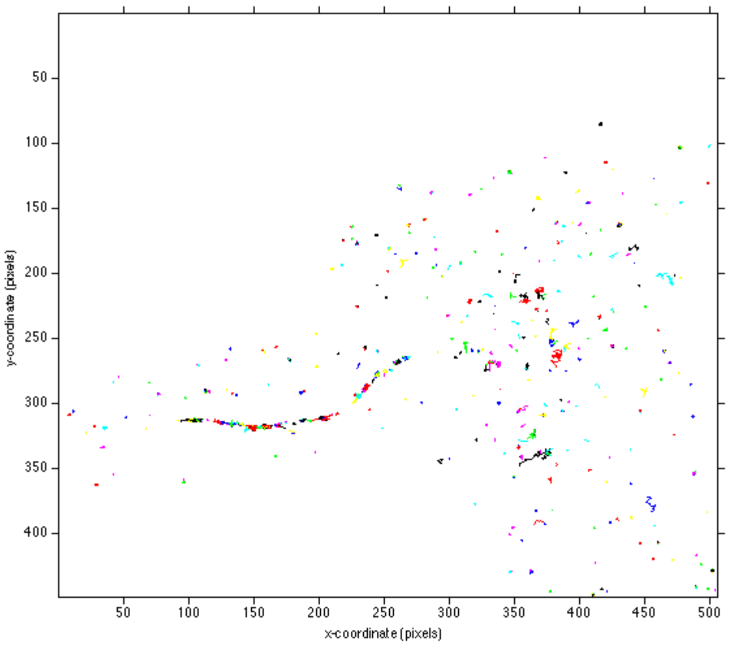
Since I haven't been doing multiple Gaussian kernel fitting on my tracks, I've just left this at 3.4. If you end up setting doMMF = 1 in your detection script, you may be able to lower this number, though I'm not sure how you'd scale it.

```
%Store parameters for function call
costMatrices(2).parameters = parameters;
clear parameters

%-----------------------------------------------------------------------
----

%% additional input

%saveResults
saveResults.dir = '/Users/Kristen/Documents/TamkunLab/5.2.13/Cell 1 -
5100/Tif - 0.4%/'; %directory where to save input and output
saveResults.filename = 'Cell1_Tracking_2001-4000_SR2_TW2.mat'; %name of
file where input and output are saved
% saveResults = 0; %don't save results

%verbose
verbose = 1;

%problem dimension
probDim = 2;
```

Again, this would be the place to define the directory you want the tracking results to be saved in and what the title should be for future access. Problem dimension should remain 2, since we are only looking at 2D motion. I don't know what verbose means in this context, but I've just left this as is.

Plotting the Results

1. In order to plot the tracks, you need to have the output open in the Matlab workspace. I like to plot everything right after the tracking is finished. If you have to wait to plot it or want to go back to older results and plot them, you'll need to open your tracking results. This will put in a variable called "tracksFinal" that has the information needed to make plots & overlay them on movies.
2. In the command window, I generally type in `plotTracks2D(tracksFinal, [], 2, [], 0, 1, [], 0, 0, [], 10)` to get plots like the one below…



If you open up the plotTracks2D script, it gives very helpful info on what to enter for each input. Anything that you want to just keep at a default value, you should enter [] instead of a number.
I think the comments are good at explaining it, so the first input variable is always going to be "tracksFinal" since that is where the program stores the information after doing the tracking. Then the following input comes next…

```
%        timeRange          : 2-element row vector indicating time range
                                   to plot.
```

For this, I generally use [ ] to indicate that I want the entire movie to be plotted. If you only wanted to look at the range 250-500, you'd enter [250 500] instead

```
%                             Optional. Default: whole movie.
%        colorTime          : String with the following options:
%                             -'1' if time is to be color-coded (green in
                                   the beginning, blue in the middle, red in
                                   the end).
%                             -'k', 'b', 'r', etc. if all tracks are in
                                   black, blue, red, etc.
%                             -'2' if tracks are colored by cycling
                                   through the plot's default color order.
%                             -'3' as 2, but using extendedColors (23
%                                different colors).
%                             Optional. Default: 'k'.
```

I always set this to 2, so that we get the rainbow plots. This makes it easy to distinguish between individual tracks. Though, I think setting it to one would be fun to look at because it'll show you where the start and end of each track is. Probably not exciting for the AIS, but could be fun to look at in the soma.

```
%        markerType         : String indicating marker type for plotting.
%                             Only used if colorTime is not '1'.
%                             Optional. Default: 'none'.
```

If you wanted to use something other than a solid line, you'd enter say '--' for a dashed line for example. However, I just leave it set to [ ] which is the default.

```
%        indicateSE         : 1 if track starts and ends are to be
                                   indicated
```

```
%                           with circles and squares, respectively; 0
%                           otherwise. Optional. Default: 1.
```
This is just another what to see where the tracks start and end, though here they'd be represented by circles and squares rather than a color, which could end up looking messy. I always leave this at 0.
```
%       newFigure         : 1 if plot should be made in a new figure
%                           window, 0 otherwise (in which case it will
                                 be plotted in an existing figure window).
%                           newFigure can also be a handle to the axes
                                 in which to plot.
%                           Optional. Default: 1.
```
I typically leave this on 1, else I may get tracks overlaid on another figure I have open. Which would be fine if I did this after each set of tracks I plotted for the same movie.
```
%       image             : An image that the tracks will be overlaid
                                 on if newFigure=1. It will be ignored if
                                 newFigure=0.
%                           Optional. Default: no image
```
If you wanted to overlay your tracks on an image, then you'd put in this spot '/directorywithfile/filename.extension'. Otherwise I just leave it at [ ]
```
%        flipXY             : 1 if x and y coord should be flipped for
%                             plotting. Optional. Default: 0.
```
If you want to flip the x and y coordinates in your plot. Since the diffusion analysis flips the coordinates from what I can remember, this would make it easier to compare (potentially).
```
%       ask4sel           : 1 if user should be asked to select tracks
                                 in plot in order to show track information.
%                           Optional. Default: 1.
```
If you wanted to look at individual track information by clicking on the track in the plot, I usually just set this =0 since I don't look at more than just what's on the plot.
```
%       offset            : [dx,dy] that is to be added to the
                                 coordinates.
%                           Optional. Default: [0,0]
```
If you wanted to shift the coordinates by a certain amount, you'd enter them in here, though I don't see why we would need this, so I keep it at the default by entering [ ]
```
%       minLength         : Minimum length of tracks to be plotted.
%                           Optional. Default: 1.
```
If you wanted to look at only the longer tracks, set this value to whatever the minimum length you want is. However, the program will only make tracks

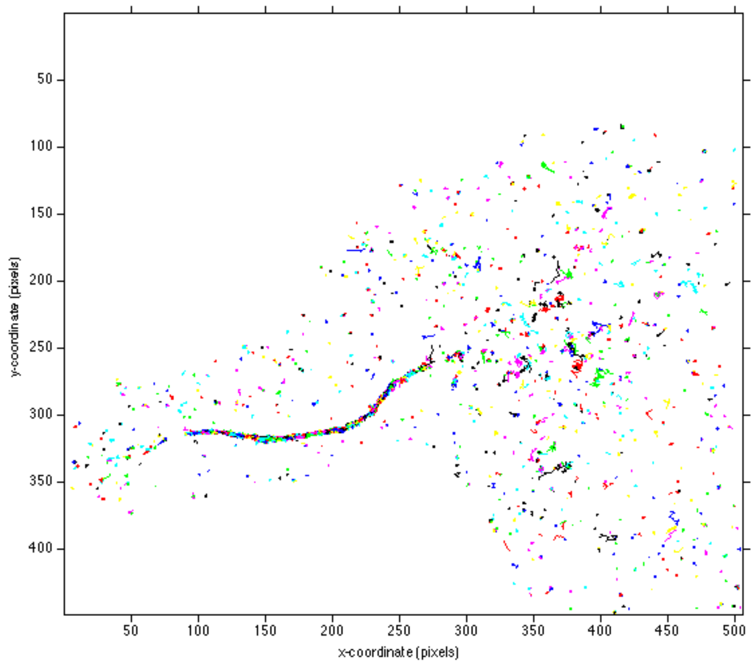
3. Now since I've generally been splitting up the larger movies to shorten the runtime of the program – even just a few thousand tracks can take a few hours to analyze. Sometimes the tracking won't even work on the Tamkun computer if I don't split it up because we've run out of memory. So, if you end up splitting your movie into segments, I wrote a quick and dirty script for plotting all the tracks onto one figure. It will require that you edit it by typing in the full location of your saved tracking results for each analysis. If you need more, just copy+paste the repeated commands and edit as needed. If you need less, just comment out the code you don't need with a "%" in front of each line. I've copied the script below, but I'll send it in an email as well, just save it in the same folder as the rest of the uTrack scripts.

To run this script after you are done editing, just type `plot2Dtracks_full` into the command window.

```
%plotting tracking results from split up movies
%rough script, so it doesn't take input, but you can edit it easily to %do this
if you want.

function plot2Dtracks_full

%clear all the variables currently in the matlab workspace
clear all;

%load the matlab file saved after tracking
load('/Users/Kristen/Documents/TamkunLab/4.12.13/cell 5 -5135/Tif -
0.3%/Cell1_Tracking_1.mat');

%rename the tracksFinal variable
tracksFinal1=tracksFinal;
clear tracksFinal;

%repeat this set of commands for the number of tracking results you %have
saved. There is probably an actual efficient way of doing this, %but I just
thought it would be ok to do it this way. I just copied + %pasted the first set
of commands and edited the file name by updating %the numbers. You only need to
make sure that it is opening the correct %saved Tracking file. And that you
have all of them opening. Say, if %you only had 2 sets of tracks,then you could
comment out the last 3 %sets of commands by putting a % in front of each line.

%load the matlab file saved after tracking
load('/Users/Kristen/Documents/TamkunLab/4.12.13/cell 5 -5135/Tif -
0.3%/Cell1_Tracking_2.mat');

%rename the tracksFinal variable
tracksFinal2=tracksFinal;
clear tracksFinal;

%load the matlab file saved after tracking
load('/Users/Kristen/Documents/TamkunLab/4.12.13/cell 5 -5135/Tif -
0.3%/Cell1_Tracking_3.mat');

%rename the tracksFinal variable
tracksFinal3=tracksFinal;
clear tracksFinal;

%load the matlab file saved after tracking
load('/Users/Kristen/Documents/TamkunLab/4.12.13/cell 5 -5135/Tif -
0.3%/Cell1_Tracking_4.mat');

%rename the tracksFinal variable
tracksFinal4=tracksFinal;
clear tracksFinal;

%load the matlab file saved after tracking
load('/Users/Kristen/Documents/TamkunLab/4.12.13/cell 5 -5135/Tif -
0.3%/Cell1_Tracking_5.mat');

%rename the tracksFinal variable
tracksFinal5=tracksFinal;
clear tracksFinal;

%now for the plotting...you don't need to touch this as long as the
%settings for the plots stays the same, unless you are plotting more or %fewer
results. When plotting fewer, just comment out the ones you %don't need by
putting a % in front of that line.
```

```
hold on %this ensures that the plots stay on the same figure.

%generates the first plot in a new window
plotTracks2D(tracksFinal1, [], 2, [], 0, 1, [], 0, 0, [], 10);
%each other plot will be on the same figure (changed the 1 to a 0)
plotTracks2D(tracksFinal2, [], 2, [], 0, 0, [], 0, 0, [], 10);
plotTracks2D(tracksFinal3, [], 2, [], 0, 0, [], 0, 0, [], 10);
plotTracks2D(tracksFinal4, [], 2, [], 0, 0, [], 0, 0, [], 10);
plotTracks2D(tracksFinal5, [], 2, [], 0, 0, [], 0, 0, [], 10);

hold off
```

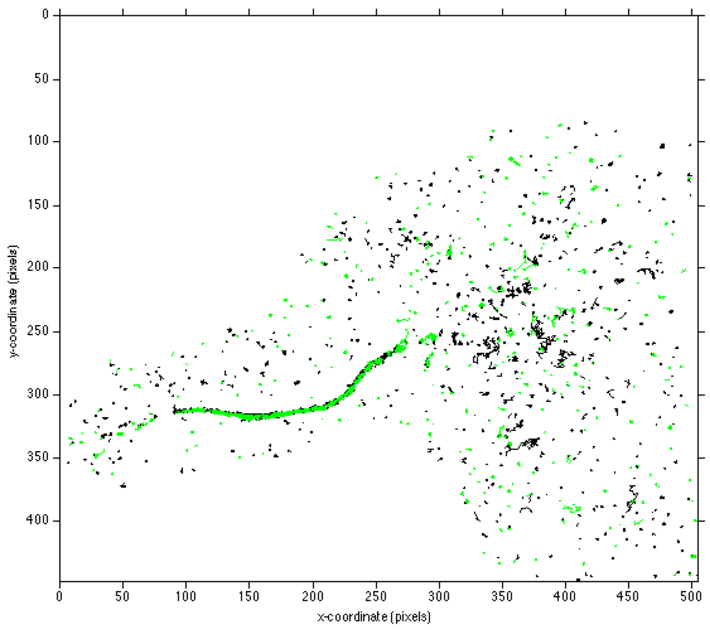
And now this gives me a plot that looks like this:



So we can now see where all the tracks were over the whole movie without flipping back and forth. This could be useful in identifying areas of high activity at a glance. It could also be useful if you had some experimental change at a certain point in the movie. Say we had taken a movie and at frame # 6001, something was added to potentially change the dynamics of the sodium channels. An easy way to quickly compare the tracks between the two would be to overlay the separate tracks using this script with some minor changes that I'll highlight.

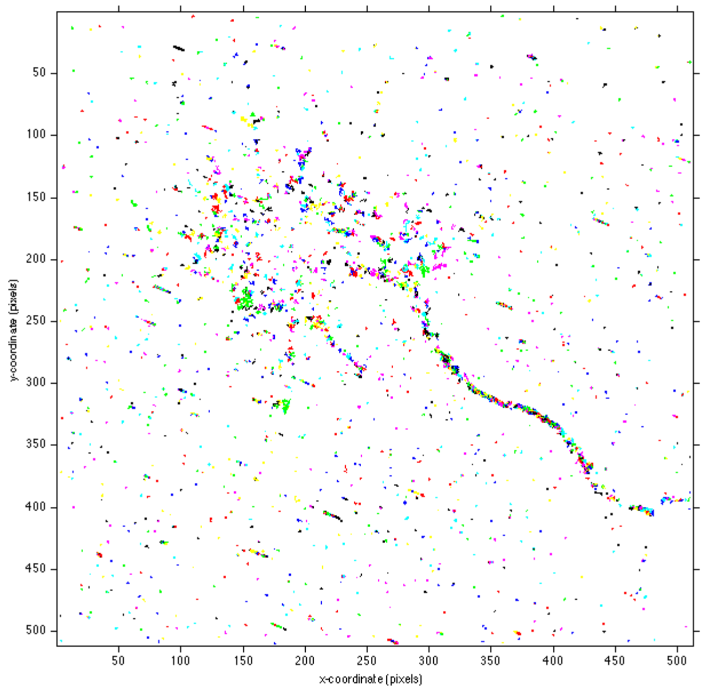
```
hold on %this ensures that the plots stay on the same figure.

%generates the first plot in a new window
plotTracks2D(tracksFinal1, [], 'k', [], 0, 1, [], 0, 0, [], 10);
%each other plot will be on the same figure (changed the 1 to a 0)
plotTracks2D(tracksFinal2, [], 'k', [], 0, 0, [], 0, 0, [], 10);
plotTracks2D(tracksFinal3, [], 'k', [], 0, 0, [], 0, 0, [], 10);
plotTracks2D(tracksFinal4, [], 'g', [], 0, 0, [], 0, 0, [], 10);
plotTracks2D(tracksFinal5, [], 'g', [], 0, 0, [], 0, 0, [], 10);

hold off
```

So, what I've done here is change the plotting parameters so that the first 6000 tracks are in black and the last 6001-10000 tracks are in green. This is what I get:



This would be a quick glance to see if there is something obviously different between the two, say there's less clustering in one area or less overall movement in the axon. Since you can't distinguish individual tracks like you can with the rainbow coloring, it would only be something very broad. You'd really have to just do it to see if this is at all useful, I just thought I'd throw it in just in case.

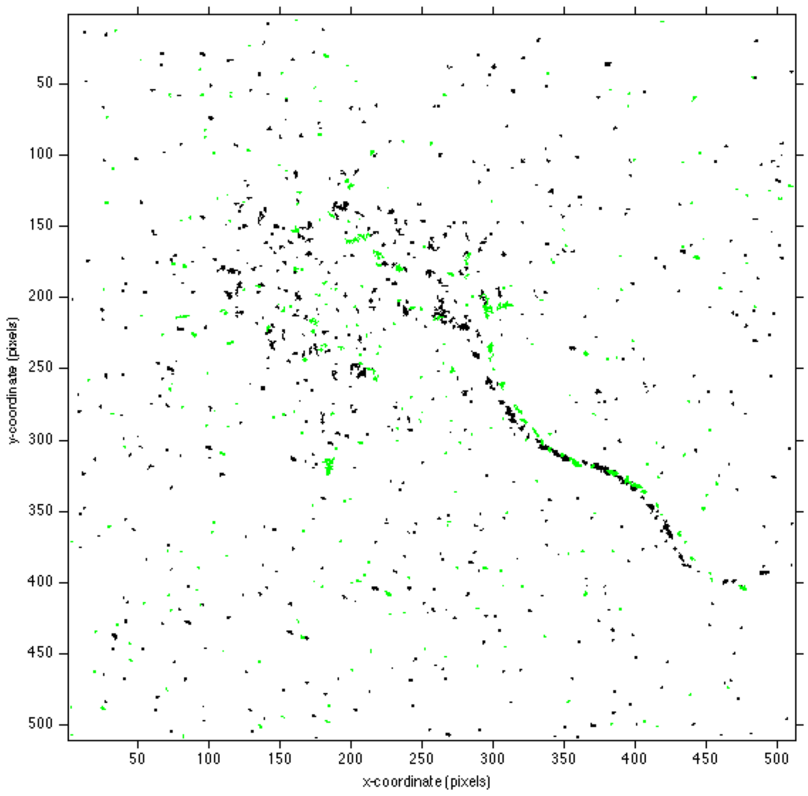
I did find this to be useful when I looked at this particular plot:



and the "dirt" seemed to be drifting over the course of the movie. If that were the case, then my tracks were also likely to be drifting, and the plot wouldn't be useful to me. So, I commented out everything except for the first and last set of tracks in the script to compare the beginning and end. Script is below, so you can see how I edited it to do this.

```
%plotting tracking results from split up movies
%rough script, so it doesn't take input, but you can edit it easily to do
%this if you want.

function plot2Dtracks_full

%clear all the variables currently in the matlab workspace
clear all;

%load the matlab file saved after tracking
load('/Users/Kristen/Documents/TamkunLab/5.2.13/Cell 1 - 5100/Tif -
0.4%/Cell1_Tracking_1-2000_SR3_TW2_Random.mat');

%rename the tracksFinal variable
tracksFinal1=tracksFinal;
clear tracksFinal;

%repeat this set of commands for the number of tracking results you have
%saved. There is probably an actual efficient way of doing this, but I just
%thought it would be ok to do it this way. I just copied + pasted the first
%set of commands and edited the file name by updating the numbers. You only
%need to make sure that it is opening the correct saved Tracking file. And
%that you have all of them opening. Say, if you only had 2 sets of tracks,
%then you could comment out the last 3 sets of commands by putting a % in
%front of each line.

%load the matlab file saved after tracking
%load('/Users/Kristen/Documents/TamkunLab/5.2.13/Cell 1 - 5100/Tif -
0.4%/Cell1_Tracking_2001-4000_SR3_TW2_Random.mat');

%rename the tracksFinal variable
%tracksFinal2=tracksFinal;
%clear tracksFinal;

%load the matlab file saved after tracking
%load('/Users/Kristen/Documents/TamkunLab/5.2.13/Cell 1 - 5100/Tif -
0.4%/Cell1_Tracking_4001-6000_SR3_TW2_Random.mat');

%rename the tracksFinal variable
%tracksFinal3=tracksFinal;
%clear tracksFinal;

%load the matlab file saved after tracking
%load('/Users/Kristen/Documents/TamkunLab/5.2.13/Cell 1 - 5100/Tif -
0.4%/Cell1_Tracking_6001-8000_SR3_TW2_Random.mat');

%rename the tracksFinal variable
%tracksFinal4=tracksFinal;
%clear tracksFinal;

%load the matlab file saved after tracking
load('/Users/Kristen/Documents/TamkunLab/5.2.13/Cell 1 - 5100/Tif -
0.4%/Cell1_Tracking_8001-10000_SR3_TW2_Random.mat');

%rename the tracksFinal variable
tracksFinal5=tracksFinal;
clear tracksFinal;
```

```
%now for the plotting...you don't need to touch this as long as the
%settings for the plots

hold on %this ensures that the plots stay on the same figure.

%generates the first plot in a new window
plotTracks2D(tracksFinal1, [], 'k', [], 0, 1, [], 0, 0, [], 10);
%each other plot will be on the same figure (changed the 1 to a 0)
%plotTracks2D(tracksFinal2, [], 2, [], 0, 0, [], 0, 0, [], 10);
%plotTracks2D(tracksFinal3, [], 2, [], 0, 0, [], 0, 0, [], 10);
%plotTracks2D(tracksFinal4, [], 2, [], 0, 0, [], 0, 0, [], 10);
plotTracks2D(tracksFinal5, [], 'g', [], 0, 0, [], 0, 0, [], 10);

hold off
```



And so you can see the first set of tracks in black and the last in green. There is clearly a drift in the movie as you can see from the AIS. This could be fixed in the plots by changing the offset input for each one to match how much the shift is in that time frame [dx dy], but that will only fix the plot. If we want to actually analyze the diffusion on this, it's probably best to go work with the movie itself and see if we can correct the drift.

I like to save them as a Matlab figure, then I can go back and zoom in on tracks if need be. If you zoom in on a spot, then save it as a ".png" or ".bmp" or whatever, then it'll save just that chunk of the plot.

Visualizing Tracks in a Movie

Now comes the fun part. The uTrack software comes with a script that allows you to watch the tracks being made overlaid on your raw movie data. This is extremely useful for double-checking your tracks, or just for looking at particular areas of interest. It also shows you how many tracks actually get made at a time to give you your final plots. I like to plot the tracks first so I can pick out areas of interest to visualize. This would be the one-segment (1-2000 for example) plot, not the full thing since they aren't combined into a single file, the overlay is easiest to do with just the one set of tracks.

To do this, you'll need to type into the command window

```
overlayTracksMovieNew(tracksFinal,startend,dragtailLength,
    saveMovie,movieName,filterSigma,classifyGaps,highlightES,showRaw,
    imageRange,onlyTracks,classifyLft,diffAnalysisRes,intensityScale,
    colorTracks,firstImageFile,dir2saveMovie,minLength,plotFullScreen,
    movieType)
```

Where `overlayTracksMovieNew(INPUTS)`is the format. Each one of those inputs has a default value, so you could just type in `overlayTracksMovieNew(tracksFinal)` into the command window and watch a movie. However, I like to use something like this:

```
overlayTracksMovieNew(tracksFinal, [], 10, 0, [], 1.044, 0, 0, 1, [200
400; 300 500], 0, 0, [], 1, 0, [], [], 10, 1, [])
```

Again, with this, the comments in the actual script are very helpful

```
INPUT  tracksFinal   : Output of trackCloseGapsKalman.
```

Here the output of the script is tracksFinal, so we enter this as the first input.

```
%       startend      : Row vector indicating first and last frame to
%                       include in movie. Format: [startframe endframe].
%                       Optional. Default: [(first frame with tracks) (last
                        frame with tracks)]
```

Like it says, just enter the first and last frame you want to look at. I normally look at the whole movie, so I enter [ ].

```
%       dragtailLength: Length of drag tail (in frames).
%                       Optional. Default: 10 frames.
%                       ** If dragtailLength = 0, then no dragtail.
%                       ** To show tracks from their beginning to their end,
%                       set dragtailLength to any value longer than the
%                       movie.
%                       ** To show tracks statically while features dance
%                       on them, use -1.
%                       ** To show tracks from their beginning to their
%                       end, and to retain tracks even after the particle
%                       disappears, use -2.
```

Normally I just set this to 10. Basically it's here to give you tracks that the red circles in the movies trace out, for x number of frames. If you wanted to watch the tracks being drawn out like we would see in the plot use -2, which could be nice to see what is happening when tracks overlap.

```
%       saveMovie     : 1 to save movie (as Quicktime), 0 otherwise.
%                       Optional. Default: 0.
```

Unfortunately, I've not been able to get the program to actually save the movie for me. I don't think it was my computer, but you could try to save it if you wanted to. So normally I've just set this =0.

```
%       movieName     : filename for saving movie.
%                       Optional. Default: TrackMovie (if saveMovie = 1).
```

If you try to save the movie, then enter 'thenameyouwant' for this input. I normally just leave it at the default [ ], since I don't save the movie.

```
%       filterSigma   : 0 to overlay on raw image, PSF sigma to overlay on
%                       image filtered with given filterSigma.
%                       Optional. Default: 0.
```

I find this hasn't really made much of a difference if I enter 0 or whatever the final PSFsigma variable is in the workspace at the time. The movies are still overlaid on a brighter, less contrasted image than we get from editing. It's interesting, because when we overlay detections on the images, they look more like what we got from editing, so I'm not sure why this acts differently.

```
%       classifyGaps  : 1 to classify gaps as "good" and "bad", depending
%                       on their length relative to the legnths of the
%                       segments they connect, 0 otherwise.
%                       Optional. Default: 1.
```

I normally leave this at 0, because I usually use pretty strict criteria for gap length, so any track jumping is normally pretty obvious. However, could still be nice to have just to see where the gaps are being connected, good or bad.

```
%       highlightES   : 1 to highlight track ends and starts, 0 otherwise.
%                       Optional. Default: 1.
```

I normally leave this at 0 because it doesn't matter to me where the tracks start and end when I look at a movie.

```
%       showRaw       : 1 to add raw movie to the left of the movie with
%                       tracks overlaid, 2 to add raw movie at the top of
%                       the movie with tracks overlaid, 0 otherwise.
%                       Optional. Default: 0.
```

I like having the raw movie to the left sometimes, but it's not really necessary, so I leave it at 0.

```
%       imageRange    : Image region to make movie out of, in the form:
%                       [min pixel X, max pixel X; min pixel Y, max pixel Y].
%                       Optional. Default: Whole image.
```

This is why I like to plot the results when I do a movie overlay. This input is for watching the movie at only a specific section of the movie. So, I find my region of interest in the plot, zoom in, and look at the x and y axis to determine where I want to watch the movie. In this comment, it says that you do [xmin xmax; ymin ymax], but I think the script itself is actually flipped, because I've found it to work best when I do the opposite [ymin ymax; xmin xmax].

```
%       onlyTracks    : 1 to show only tracks without any symbols showing
%                       detections, closed gaps, merges and splits; 0 to
%                       show symbols on top of tracks.
%                       Optional. Default: 0.
```

I normally leave this at zero, I don't see a reason not to mark gaps/merges/splits. It's nice to see where the program is doing these merges/splits just to see if they are at least accurate.

```
%       classifyLft   : 1 to classify objects based on (1) whether they
%                       exist throughout the whole movie, (2) whether they
%                       appear OR disappear, and (3) whether they appear
%                       AND disappear; 0 otherwise.
%                       Optional. Default: 1.
```

I normally leave this at 0 because there won't be spots that stick around the whole movie, considering how long the movies we take are.

```
%       diffAnalysisRes:Diffusion analysis results (either output of
%                       trackDiffusionAnalysis1 or
                        trackTransientDiffusionAnalysis2).
%                       Needed if tracks/track segments are to be
%                       colored based on their diffusion classification.
%                       With this option, classifyGaps, highlightES and
%                       classifyLft are force-set to zero, regardless of input.
%                       Optional. Default: None.
```

I normally leave this at the default, [ ], because we don't get diffusion analysis results from our tracks. I assume this can be done, but I didn't find a specific script in the old version of the u-track package. When I look at the diffusion analysis from the new package, it wasn't very useful as there were many unclassified tracks.

```
%       intensityScale: 0 to autoscale every image in the movie, 1
%                       to have a fixed scale using intensity mean and std,
%                       2 to have a fixed scale using minimum and maximum
%                       intensities.
%                       Optional. Default: 1.
```

I think that leaving this =1 is fine.

```
%       colorTracks    : 1 to color tracks by rotating through 7 different
%                       colors, 0 otherwise. With this option,
%                       classifyGaps, highlightES and classifyLft are
%                       force-set to zero, regardless of input.
%                       Option ignored if diffAnalysisRes is supplied.
%                       Optional. Default: 0.
```

If you want each track to be a different color, like in the 2D plots, set this =1. Otherwise the default color is a peach color.

```
%       firstImageFile: Name of the first image file in the folder of
%                       images that should be overlaid. The file has to be
%                       the first image that has been analyzed even if not
%                       plotted. If file is not specified [], user will be
%                       prompted to select the first image.
%                       Optional. Default: [].
```

Usually I just leave this as default [ ], because it'll ask you to open the file, which to me is easier than typing in the full directory and name. If you want to do that however, you can by entering '/directory/folder/imagename.ext' for your image (the first one in the movie you are overlaying).

```
%       dir2saveMovie:  Directory where to save output movie.
%                       If not input, movie will be saved in directory where
%                       images are located.
%                       Optional. Default: [].
```

Again, I leave this at the default, [ ], because I don't get to save the movies, so it doesn't matter.

```
%       minLength      : Minimum length of tracks to be ploted.
%                       Optional. Default: 1.
```

I leave this at 10, but you could just put in whatever you decided to make your minimum track length in the tracking script. This would only be worthwhile to change if we wanted to only look at the longer tracks, but we don't get many of those in the movies anyway to distinguish shorter or longer tracks.

```
%       plotFullScreen: 1 the figure will be sized to cover the whole
%                       screen. In this way the movie will be of highest
%                       possible quality. default is 0.
```

I usually set this =1 because I like being able to see it on the full screen.

```
%       movieType      : 'mov' to make a Quicktime movie using MakeQTMovie,
%                       'avi' to make AVI movie using Matlab's movie2avi,
%                       'mp4_unix', 'avi_unix' to make an MP4 or AVI movie
%                       using ImageMagick and ffmpeg. These options works
%                       only under linux or mac.
%                       Optional. Default: 'mov'.
```

Since I don't save the movie, I just leave it at the default [ ], or just end the string of inputs for the command and enter a) after the plotFullScreen input instead.

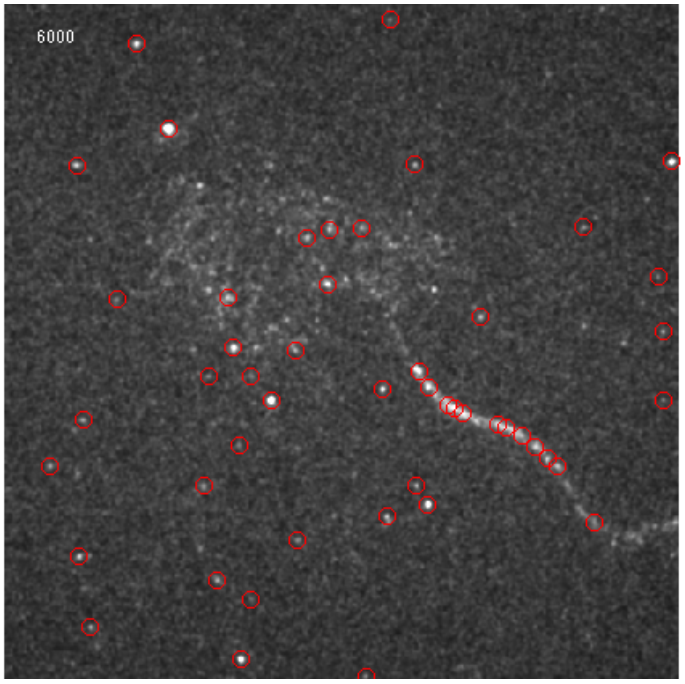
Now you can watch the tracks as they are being drawn in the movie and see what parameters you should play with to get better results if you're not happy with what you see. There is also an "overlayFeaturesMovie" script if you want to watch a movie like this, but looking at the detection results. So, much like the above script, the comments are useful and actually take the same kinds of inputs:

```
%INPUT  movieInfo   : Output of detectSubResFeatures2D_StandAlone.
%       startend    : Row vector indicating first and last frame to
%                     include in movie. Format: [startframe endframe].
%                     Optional. Default: [1 (maximum available frame)]
%       saveMovie   : 1 to save movie (as Quicktime), 0 otherwise.
%                     Optional. Default: 0
%       movieName   : filename for saving movie.
%                     Optional. Default: FeaturesMovie (if saveMovie = 1).
%       filterSigma : 0 to overlay on raw image, PSF sigma to overlay on image
%                     filtered with given filterSigma.
%                     Optional. Default: 0
%       showRaw     : 1 to add raw movie to the left of the movie with
%                     tracks overlaid, 2 to add raw movie at the top of
%                     the movie with tracks overlaid, 0 otherwise.
%                     Optional. Default: 0.
%       intensityScale: 0 to autoscale every image in the movie, 1
%                     to have a fixed scale using intensity mean and std, 2
%                     to have a fixed scale using minimum and maximum
%                     intensities.
%                     Optional. Default: 1.
%       firstImageFile: Name of the first image file in the folder of
%                     images that should be overlaid. The file has to be
%                     the first image that has been analyzed even if not
%                     plotted. If file is not specified [], user will be
%                     prompted to select the first image.
%                     Optional. Default: [].
%       dir2saveMovie: Directory where to save output movie.
%                     If not input, movie will be saved in directory where
%                     images are located.
%                     Optional. Default: [].
%       movieType   : 'mov' to make a Quicktime movie using MakeQTMovie,
%                     'avi' to make AVI movie using Matlab's movie2avi,
%                     'mp4_unix', 'avi_unix' to make an MP4 or AVI movie
%                     using ImageMagick and ffmpeg. These options works
%                     only under linux or mac.
%                     Optional. Default: 'mov'.
%       plotFullScreen: 1 the figure will be sized to cover the whole
%                       screen. In this way the movie will be of highest
%                       possible quality. default is 0.
%
```

So you want to enter the inputs as you would for the overlayTracksMovieNew script. Instead of tracksFinal though, you are using movieInfo. Make sure your detection results are actually open, with movieInfo in the workspace. For example, I'd type in:

```
overlayFeaturesMovie(movieInfo, [4001 6000], 0, [], 1.06, [], [], [],
1)
```

The movie output will look like this:



Where each red circle represents a detection, like in the track overlays. Unfortunately, there is no option for cropping the movie to look at a particular segment, so this is better for a broader look at your detection results. It's useful for seeing if you need more detected particles or if you need to adjust the tracking parameters more. You can see how many detections aren't necessarily being tracked if you compare with the tracking movie. Here we are missing several areas in the soma that aren't getting detected. While we have much more overall detection in the soma for the duration of the movie as indicated by the tracks, if we go back and adjust detection parameters, we can probably get more tracks and a better picture of what is going on in the soma.