# Experiment 2 - Defeating the Diffraction limit

# Project Report

**Team**

**Names:**

Tiffany Ly(ID: ttl6xa)

Sharath (ID: vp4pa)

**Objective:** Localize the single biomolecules in a 3D space with a precision of a few nanometers using super resolution imaging techniques.

**Introduction:**

We were able to overcome the typical fluorescence microscopy diffraction limit of 250 nm using correlation and other super resolution techniques. We read in a stack of 100 images containing fluorescent proteins that randomly go on and off. We also had a template stack of 60 images of pairs of Gaussians with known orientation. We found the correlation of each 100 images with each pair of Gaussians from the template. The maximum correlation values were used for Super Resolution. We used a triple Gaussian function to fit these correlations to find the reliable Gaussians’ orientation, separation distance and midpoint.



Figure Methodology Flowchart

## Methodology:

1. Read the stack of images frame by frame
2. Preprocess each frame by Thresholding
   1. Threshold stack to binary image
   2. Alternative: use ImageJ preprocessing. We tried both method and got the same result.

-Differencing of Gaussian filters to reduce noise and remove background and retain high intensity pixels

-De-speckle

-Convert into binary using Max entropy method

-Save the final binarized stack separately

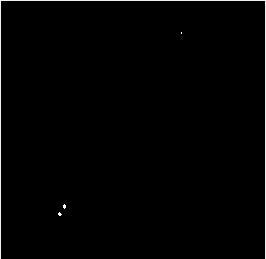
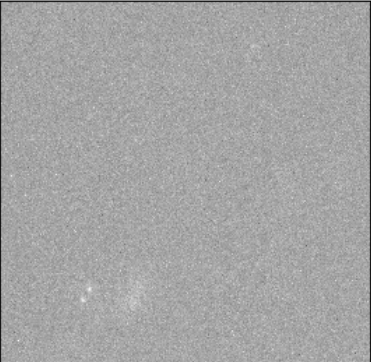


Figure Preprocess the original stack of images

1. Load Templates.mat. It has 60 templates which are used to identify the candidates for super resolution techniques

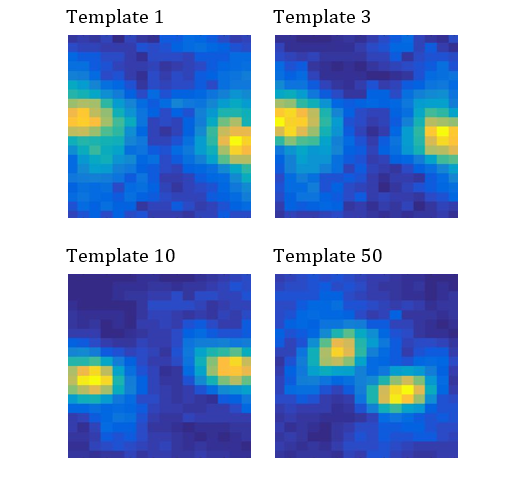


Figure Load stack of 60 stacks. Templates 1, 3, 10 and 50 shown above.

1. Find correlation between a frame and templates. The frames with maximum correlation for a given template are considered to have regions of interest for our further processing. The template that has maximum correlation is considered as the matched template for that molecule
   1. Calculate correlation value for every pixel in a frame for templates from 1 to 60 as in **Figure 4**.
   2. Locations of pixels with maximum correlation values greater than .7 are stored for further evaluation along with the matched templates (orientation) and corresponding frame numbers as in **Figure 5**.
   3. Each point shall be further localized using super resolution imaging technique

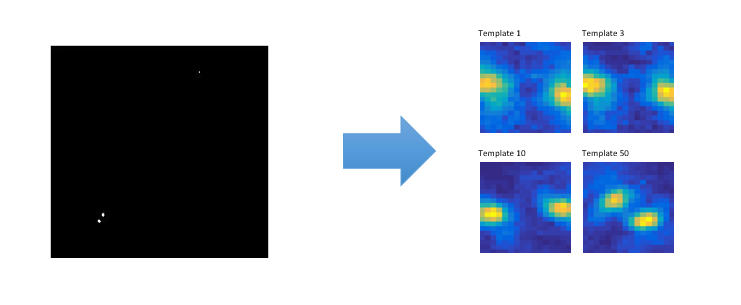


Figure Find correlation of 100 images with each of the 60 template images.

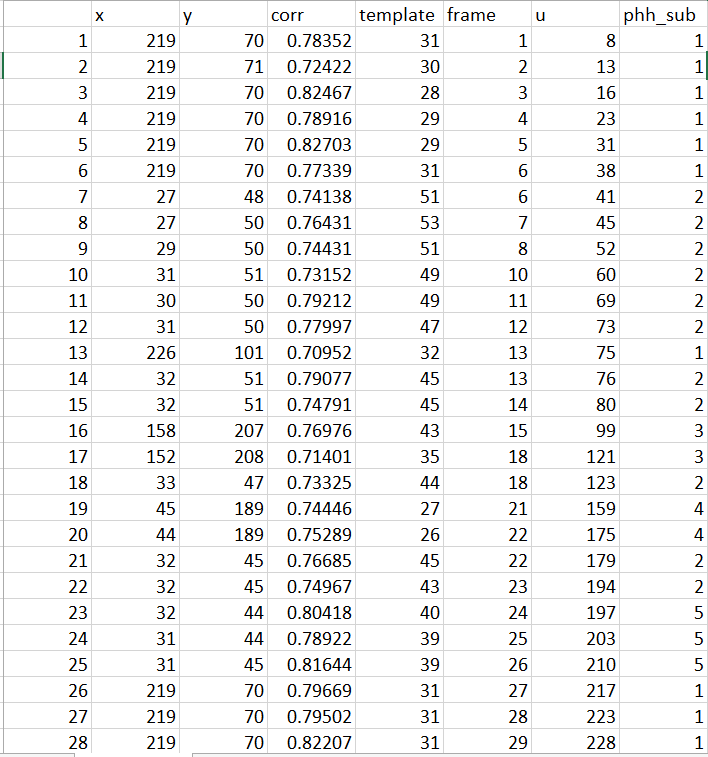


Figure 28/94 double helixes found on the 100 frames and their corresponding template number.

1. Clustering
   1. We grouped the pair of Gaussians with similar x and y coordinate into individual clusters as shown in **Figure 5**. Each number under column “phh\_sub” represents an individual cluster, or molecule.
   2. The coordinates were grouped into clusters using Euclidean method of pdist function. Then, hierarchally clustered into sqrt(the number of rows)/2 which came out to seven clusters.
   3. We end up with 5 real clusters. The algorithm resulted in seven but two of the seven only had one coordinate each which is not sufficient to make a Gaussian fit. Thus, it is assumed to be noise.
2. Super resolution using Gaussians fit

For every cluster, we fitted a triple Gaussian function with the non linear least squares method. The curve is a triple Gaussian because there will be significant correlation when one of the lobes of the double helix in the original image matches with one lobe of the template. However, we will attain the largest Gaussian (center) when the both lobes of the double helix of the image matches with the pair on the template.

* 1. Extract region of interest from correlation matrix by creating a square of 10 pixels around every point selected above.
  2. Fit a triple Gaussian function of following form:

where,

b1, b2, and b3 are x-coordinates for 3 Gaussian peaks

c1, c2 and c3 are y-coordinates

* 1. When the intensities of a double helix point follows a double Gaussian, the correlation values of the double helix point with the highly matched template follows a triple Gaussian as below

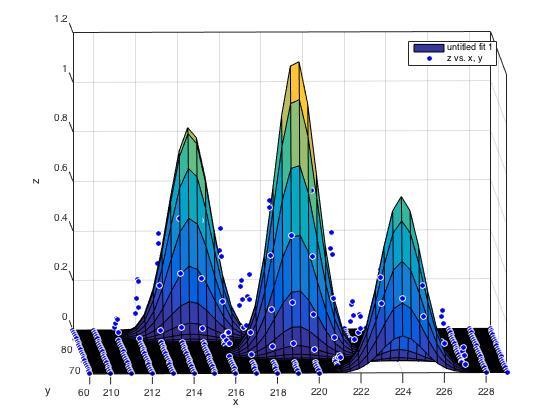


Figure Get the Triple Gaussian Curve for each cluster.

* 1. The midpoint of each double helix is the peak of the center triple Gaussian.
  2. Extract the peak points from the fitted function
     1. Inter lobe distance = distance between (b3,c3) and (b1, c1). So from
     2. (b2,c2) is the real valued pixel point. Maximum likelihood location

1. Determination of Orientation of a point
   1. The orientation of a point corresponds to the orientation of the best fitted template. See step 4.

## Results:

1. Gaussians of # 1, 30, 50, 70, 100 highlighted on table below

Frame 1: (218.3555, 70.1726)

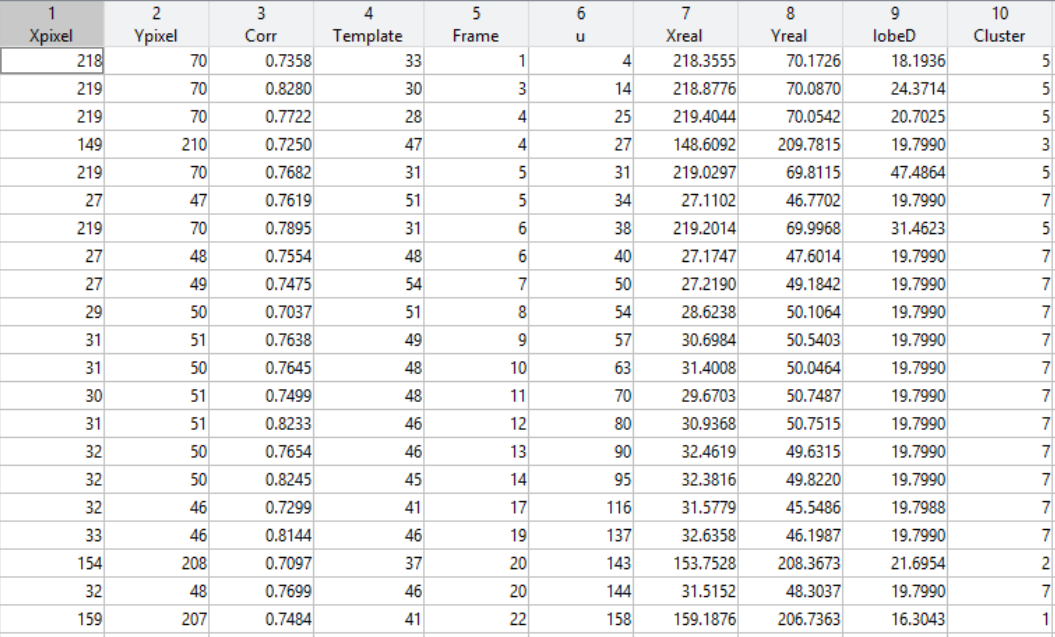
Frame 30: (33.4422, 43.7373)

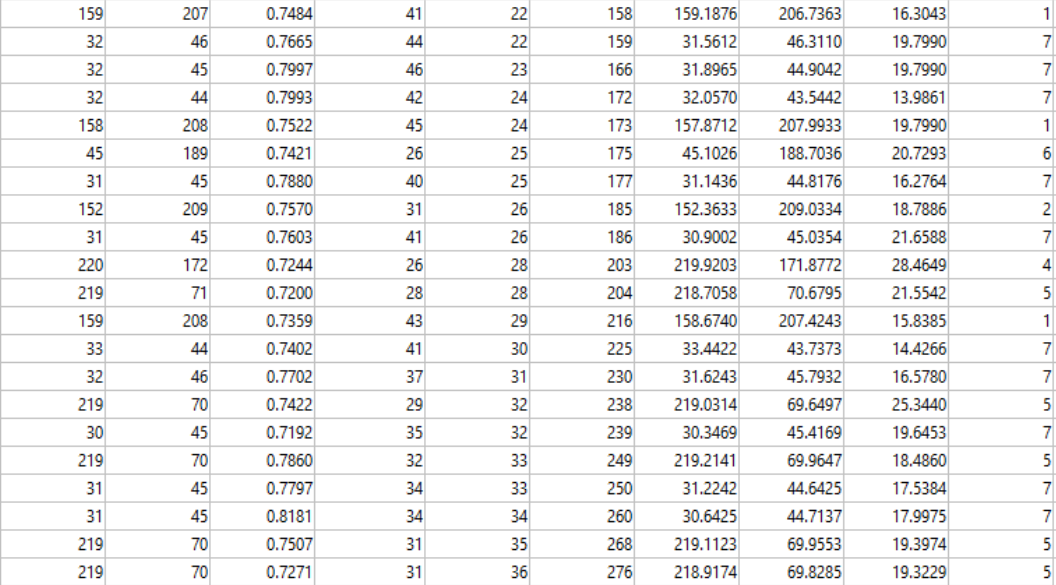
Frame 50: (214.3251, 60.4082)

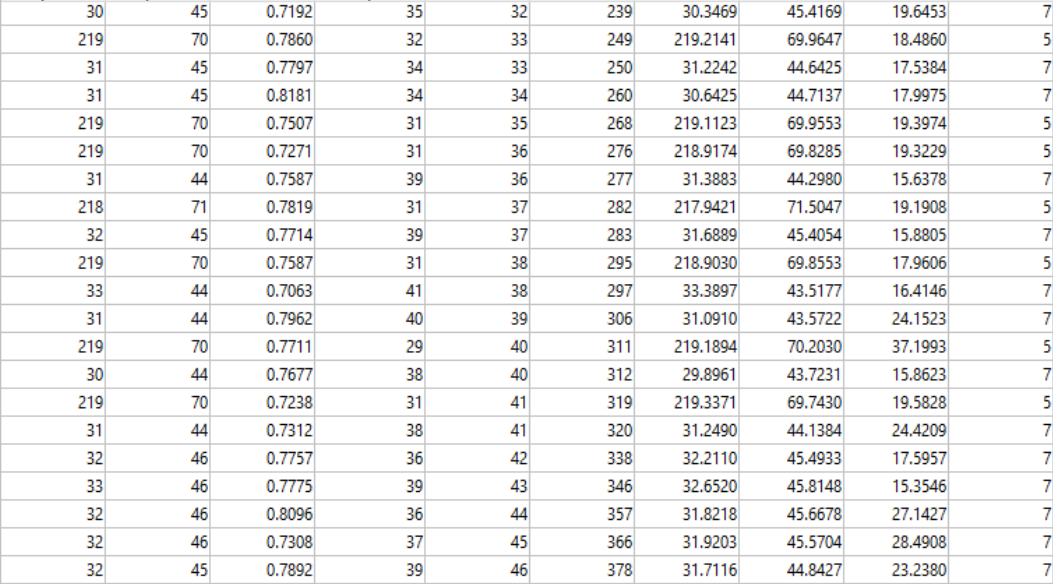
Frame 70: (44.3915, 188.3301)

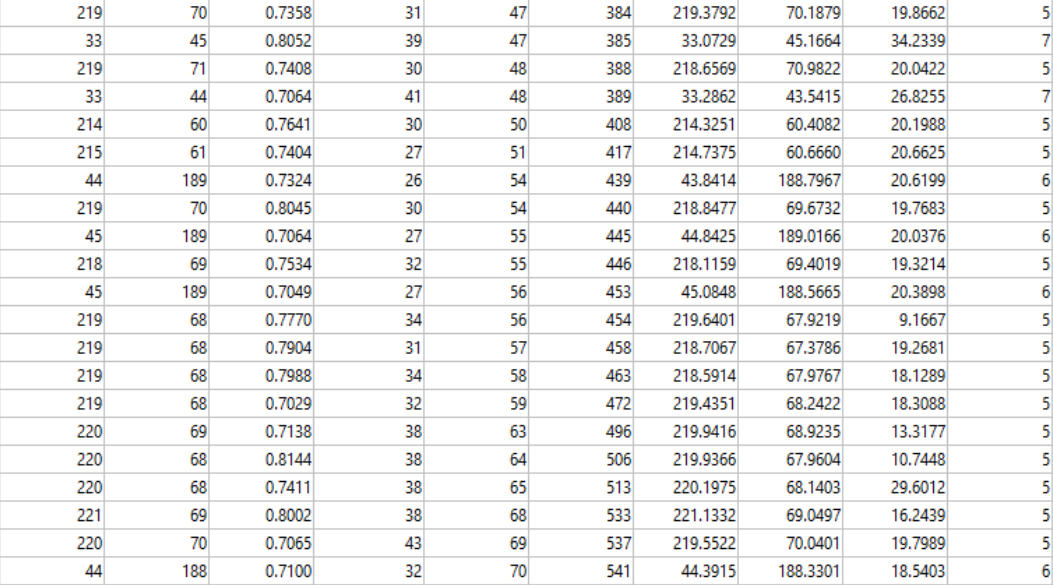
Frame 100: none

1. A table with final set of points: Pixel location, Frame number, center location, index of the matched template, frame pixel is on, unique number, the x and y coordinate found from the Gaussian centroid, the lobe distance and cluster number is shown below.









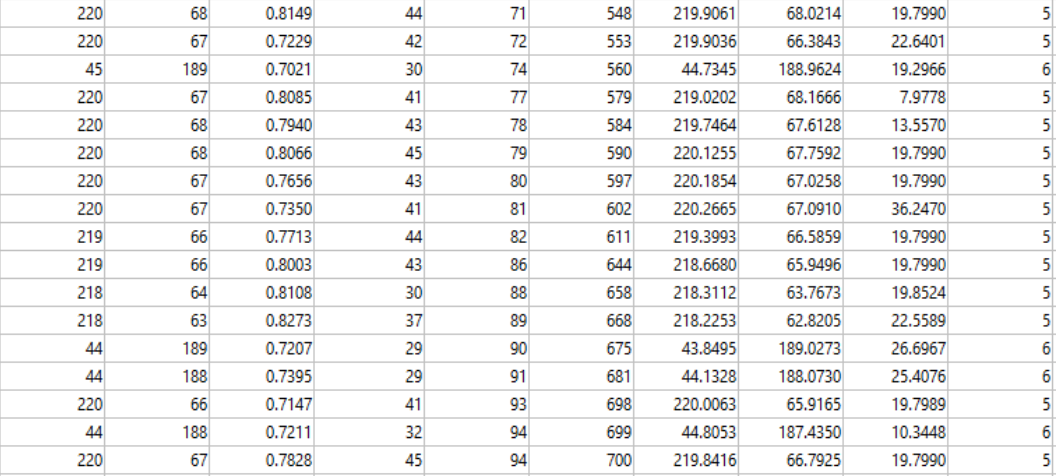


Figure Final table used for reconstruction and Gaussian curve.

1. Reconstruction of final set of points into a single frame

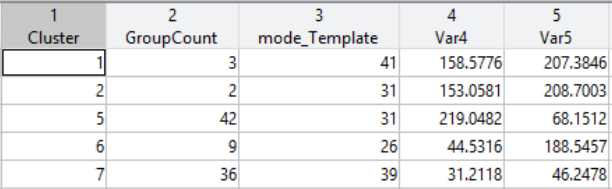


Figure Points and template for reconstruction of helixes

1. The centroid coordinate and the template’s orientation is used to reconstruct the each final pair of Gaussians onto one frame.

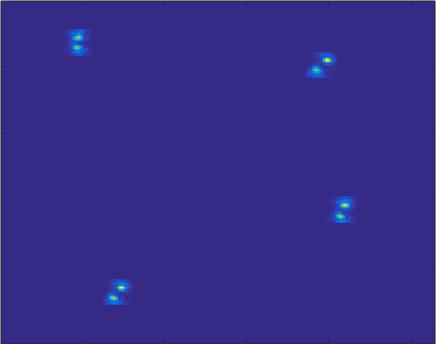


Figure Reconstruction of helixes.

1. Code efficiency
2. Extracted first set of candidate points for Gaussian fitting. Used threshold levels of 0.7 on correlation values to identify most likely candidates. Only those selected candidates are processed for super resolution which is an expensive operation. It helped us cut down on processing time
3. We have given initial start points for faster convergence in the Gaussian fitting
4. Code ran on Intel Core I5-5200U CPU @ 2.20GHz 2.19 GHz

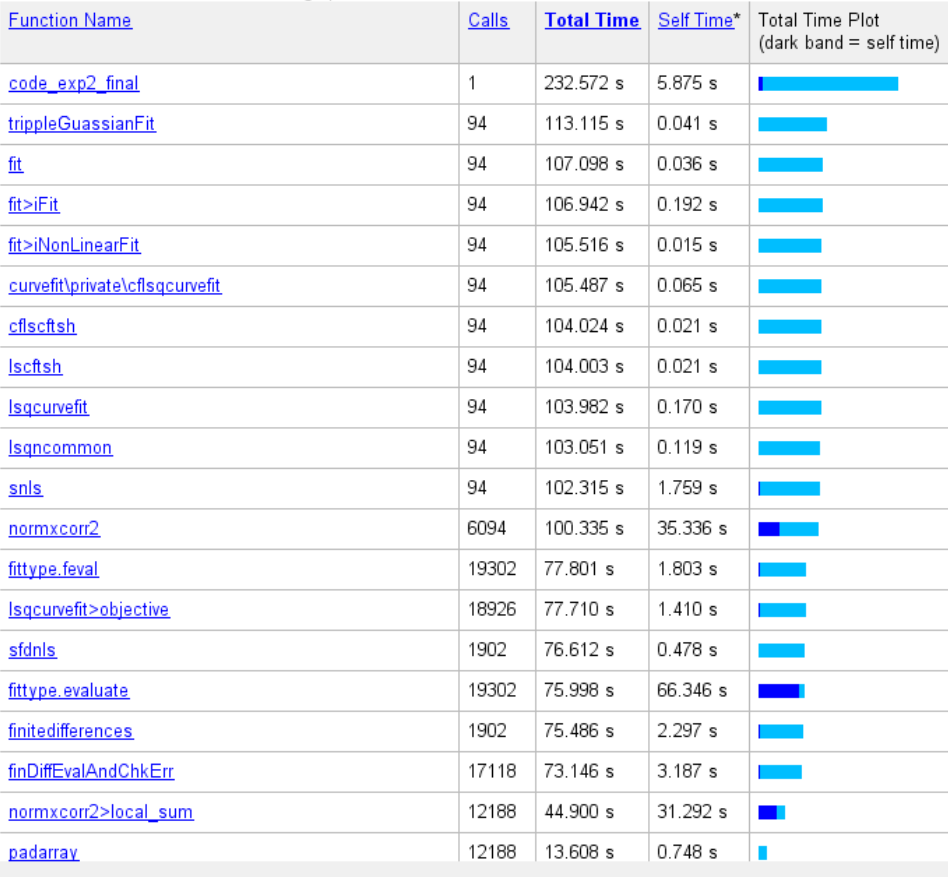


Figure Code run time

## Challenges

* Preprocessing: getting some noise and artifacts when smoothing and despeckling. Also, got some fiducial specks in our preprocessed images. However, these were tackled by repeated localizations and correlation function.
* Gaussian curve: Some challenges were figuring out which function to use and how to fit the curve efficiently. Originally, we had thought to fit using intensity. However, our stack of images was binary so we decided to fit with the correlation values we already had. It was also challenging to decide how many curves to use but the triple Gaussian seemed to work the best.

## Contributions

Tiffany: Preprocessing

Sharath: Triple Gaussian algorithm

Both: Correlation and clustering algorithm, research different methodologies, report

Sources:

*Simultaneous multiple-emitter fitting for single molecule super-resolution imaging*

*Fang Huang,1 Samantha L. Schwartz,2 Jason M. Byars,1 and Keith A. Lidke1,\**

[*http://faculty.sites.uci.edu/ianfsmithlab/cartcic/super-resolution-imaging/*](http://faculty.sites.uci.edu/ianfsmithlab/cartcic/super-resolution-imaging/)

[*http://www.gahlmannlab.com/research*](http://www.gahlmannlab.com/research)

## Code

**Preprocessing:**

delete('processedImages.tif');

stack = 'TheImages.tif';

info = imfinfo(stack);

for k = 1:numel(info)

I = imread(stack, k);

% upperThresh=160; %fiducials 1000

% lowerThresh=150; %background 100

threshold=140;

% I(I>upperThresh)=0;

binaryImage = I>threshold;

filt=gray2ind(medfilt2(medfilt2(binaryImage)),256);

% imshow(filt);

imwrite(filt, 'processedImages.tif', 'writemode', 'append');

end

**Main Code (Correlation, Clustering, Reconstruction)**

% This program reads a binary stack of images with repeated exposure of a

% cell and finds the real valued pixel positions of molecules in it using

% Super resolution techniques

stack = 'FinalBinary.tif';

info = imfinfo(stack);

load TheTemplates.mat;

% Initializing for speed

points = zeros(60,6);

fr\_points = zeros(0,6);

final\_points = zeros(0,6);

u = 1; %Unique point identifier

% Correlation values are calculated for every pixel of a frame with every template

% and initial candidates are identified for further processing

for f = 1:numel(info) % For every frame

Frame = imread(stack,f);

for t = 1:60

temp = template(t,:,:);

temp1 = reshape(temp,[20 20]);

corr = normxcorr2(temp1, Frame);

maxcorr = max(max(corr));

[x y] = find(corr == maxcorr,1);

% For every template create an unique point identifier

% If the previously identified Max correlated point with ‘t-1’ is within the neighborhood of 5 pixels, then it is the same point even it has max correlation value

if t > 1

if abs(prevx-x) <= 5 & abs(prevy-y) <= 5

u = u;

else

u = u + 1;

end

% else

% u = 1;

end

points(t,:) = [x y maxcorr t f u];

prevx = x;

prevy = y;

end

%[t1] = find(points(:,3) == max(points(:,3)));

%[t1] = find(points(:,3) > .6);

points = points(points(:,3) > 0.7,:);

fr\_points = vertcat(fr\_points, points);

end

% Filter out final Double helix candidates by getting the points with max correlation value within the same set of unique identifier

i = 1;

for k = (unique(fr\_points(:,6)'))

indx = fr\_points(:,6) == k;

max\_indx = fr\_points(indx,3) == max(fr\_points(indx,3));

subset = fr\_points(indx,:);

final\_points = vertcat(final\_points,subset(max\_indx,:));

i = i+1;

% disp(k);

end

text = sprintf('No. of points captured is %d', i);

disp(text);

%csvwrite('points.csv', final\_points);

% Following code calculates real valued pixel positions using Gaussian fitting on every point extracted above

n = size(final\_points);

ROI\_width = 10;

size1 = [(2\*ROI\_width+1)^2 1];

% For every point a neighborhood of 10 pixels is created and Gaussians are fit on the correlation values of the best matched template

for p = 1:n(1)

[xIdx, yIdx] = meshgrid(final\_points(p,1)-ROI\_width:final\_points(p,1)+ROI\_width, ...

final\_points(p,2)- ROI\_width:final\_points(p,2)+ ROI\_width);

frame = imread(stack,final\_points(p,5));

temp = template(final\_points(p,4),:,:);

temp = reshape(temp,[20 20]);

corr = normxcorr2(temp, frame);

frame\_sub = corr((xIdx(1,:))', yIdx(:,1));

frame\_sub(frame\_sub < 0.1) = 0;

%frame\_sub = frame((yIdx(:,1)), xIdx(1,:));

x = reshape(xIdx, size1);

y = reshape(yIdx, size1);

z = reshape(frame\_sub,size1);

x0 = final\_points(p,1);

y0 = final\_points(p,2);

[fit gof] = trippleGuassianFit(x,y,z,x0,y0);

lobe\_dist = sqrt((fit.b1-fit.b3)^2 + (fit.c1-fit.c3)^2);

Xmid\_point = fit.b2;

Ymid\_point = fit.c2;

final\_points(p,7:9) = [Xmid\_point Ymid\_point lobe\_dist];

end

csvwrite('fit\_points.csv', final\_points);

%Group similar coordinates

% Cluster the final points to get Molecules' positions

% We assume that the same point repeated in multiple frames belong to the

% same molecule

coord=final\_points(:,[1:2,4]);

dist=pdist(coord);

tree=linkage(dist, 'single'); %linkn based on shortest distance

%Clustering using sqrt(n/2) as the numebr clusters, where 'n' is the number

%of final points

final\_points(:,10)= cluster(tree,'maxclust',round(sqrt(n(1)/2)));

% Name columns of the dataset

final\_points = dataset({final\_points 'Xpixel','Ypixel','Corr','Template',...

'Frame','u','Xreal','Yreal','lobeD','Cluster'});

% Group points by Cluster Index to get final position of Molecules

% Get the Orientation of the molecule -> Template with maximum mode

group = grpstats(final\_points, {'Cluster'}, {'mode'},'Datavars','Template');

% Get final position coordinates

% Assuming that mean position across all the frames as the final position

% coordinates

group2 = grpstats(final\_points, {'Cluster'}, {'mean'},'Datavars',{'Xreal','Yreal'});

% Prepare final table for Image reconstruction

group(:,4:5) = group2(:,3:4);

% Assuming that point appearing in only one frame could be a Noise

group = group(group.GroupCount > 1,:);

% Image reconstruction

Im = zeros(size(frame));

n = size(group);

for p = 1:n(1)

xPoint = round(group.Var4(p));

yPoint = round(group.Var5(p));

tIndex = group.mode\_Template(p);

Im(xPoint-9:xPoint+10,yPoint-9:yPoint+10) = template(tIndex,:,:);

end

imagesc(Im)

title('Reconstructued Image');

**Tripple Gaussian Function**

function [fitresult, gof] = trippleGaussianFit(x, y, z, x0, y0)

%% Customized the auto generated Curve Fitting function.

[xData, yData, zData] = prepareSurfaceData( x, y, z );

% Set up fittype and options.

ft = fittype( '(a1/sqrt(2\*pi).\*exp(-((x-b1).^2/2)-((y-c1).^2/2))) + (a2/sqrt(2\*pi).\*exp(-((x-b2).^2/2)-((y-c2).^2/2)))+(a3/sqrt(2\*pi).\*exp(-((x-b3).^2/2)-((y-c3).^2/2)))', 'independent', {'x', 'y'}, 'dependent', 'z' );

opts = fitoptions( 'Method', 'NonlinearLeastSquares' );

opts.Display = 'Off';

b1 = x0-7;

b2 = x0;

b3 = x0+7;

c1 = y0+7;

c2 = y0;

c3 = y0-7;

opts.StartPoint = [0.178132454400338 0.128014399720173 0.190433267179954 b1 b2 b3 c1 c2 c3];

% Fit model to data.

[fitresult, gof] = fit( [xData, yData], zData, ft, opts );

% Plot fit with data.

figure( 'Name', 'untitled fit 1' );

h = plot( fitresult, [xData, yData], zData );

legend( h, 'untitled fit 1', 'z vs. x, y', 'Location', 'NorthEast' );

% Label axes

xlabel x

ylabel y

zlabel z

grid on

view( -2.3, 8.4 );