## Contents

```
%---------------------------------------------------------------------------------------------
-
%A script to recontruct  a microtubule image, and find the bending modes
%Authors: David Apigo, Arooj Aslam, John Palmieri
%Date: September 5,
%Version Number: 1.2
%---------------------------------------------------------------------------------------------
-
clearvars -except snakes
```

## Image Import

```
%==============
ii = 0;
%input('How many microns per pixel: ');  %100x Mag has 15.3 pix/micron
u = .06;

%User Input
%==========
display('Select the file for the original images')

%path and filename for original image
[filename_orig,path_orig] = uigetfile('*.tif','Select Original Image Stack');

frame_num = input('How many images are you processing: ');
begin = input('Number of the first image ');
increment = input('How many images do you want to increment by? ');
end_image = input('Number of the last image ');
pt_spacing = input('What is the point spacing along the MT? ');

%initializing index
index = begin;

%Inputs all images into a 3D matrix
for ii = 1:((end_image-begin)/increment)+1
    I_orig(:,:,ii) = imread(strcat(path_orig, filename_orig),'Index',index);
    index =  index + 1;
end
```

```
Select the file for the original images
```

```
Error using input
Cannot call INPUT from EVALC.

Error in MT_Filament_Tracking_full_FT_theta_s_2_snakes (line 22)
frame_num = input('How many images are you processing: ');
```

## Setting up or initializing parameters for the images that will be used later in the code

```
%===========================================================================================

%Initializing Certain Parameters
%===============================
%spline smoothing parameter parameter = .9;
sz = 10;

%initializing count for gauss fit errors
catch_count = 0;

%Pads the array with zeroes and gets image information for skeletonized image
I_orig_padded = padarray(I_orig,[sz sz]);

% I_skel = padarray(I_skel_unpadded,[sz sz]);

%Width and height of the image in pixels
[width,height] = size(I_orig_padded);

%Initializing the size of the reconstructed microtubule matrix to be big
%enough to include the recostructed pixels from ther largest skeletonized image
previous_frame = 1;

%number of coordinates ("beads")
N = 0;

count = 1;

for recon_size = 1:size(snakes,1)
    current_frame = snakes(recon_size,1);
    if previous_frame ~= current_frame
        N(count)= snakes(recon_size-1,2)+1;  %plus 1 b/c the count starts at zero in data file
        count = count+1;
    end
    previous_frame = current_frame;
end

%puts in count of points for last frame and finds max N
%======================================================
%number of beads in each frame
N = [N snakes(size(snakes,1),2)];

%max number of beads across all frames
N_max = max(N);

%Generate coord matrix using max N row dimension
coord_reconstructed = zeros(N_max,2,length(N)); %[number of pixels, x-y cols, number of images
]
```

## Filtering Original Image

```
%===========================
sigma = 2;
w = 20;

filter_image  = input('Does this image need to be filtered? ','s');

if filter_image == 'y' || filter_image == 'Y'
    filter_flag = 0;
else
    filter_flag =1;
    I_filtered = I_orig;
end

while filter_flag == 0

    %First smoothing function does a scanning average of the neighboring pixels
    %using a mask the size of w by w
    I_boxavg = imboxfilt3(I_orig,(2*w+1));

    %Second smoothing function uses a gaussian
    %convolution kernel
    I_gaussfilt = imgaussfilt3(I_orig,sigma);%gauss_smooth(I_orig,w);
    %I_gaussfilt = imgaussfilt3(I_orig);%gauss_smooth(I_orig,w);

    %Made uint8 a double to deal with neg values
    I_filtered = abs((I_gaussfilt) - (I_boxavg));
    imshow(I_filtered(:,:,1),[])

    % subplot(221)
    % imshow(I_boxavg(:,:,1),[])
    % subplot(222)
    % imshow(I_gaussfilt(:,:,1),[])
    % subplot(223)


    f = input('Is this filter satisfactory? Y/N ','s');

    if f == 'N' || f == 'n'

        %used as the region around the MT for image rotation
        w = input('What is the window size for box filter?');

        %used as the region around the MT for image rotation
        sigma = input('Standard deviation for Gaussian Filter?');

    end

    if f == 'Y' || f == 'y'
        filter_flag = 1;
    end

end
```

## Filament Reconstruction

```
%Looks at one image at a time and:
%Finds the fit using the skeletonized image to find the tangential angles
%Rotates small region of the bandpassed image around the points taken from the polyfit functio
n
%Fits a gaussian to the intensity profile in each column of the rotated image
%Generates a reconstructed MT using the max of the gaussian as the center of the MT
k =0;
N_previous = 0;

for filament_recon=1:size(I_orig,3) %size (I_skel,3) returns the number of images, so this run
s for each image in the tif file
    clear x;
    clear y;
    %Pads each image matrix with zeroes to deal with points near the edge
    I_new_filtered = padarray(I_filtered(:,:,filament_recon),[sz sz]);

    %Get points from snake
    x = snakes((N_previous+1):(N(filament_recon)+ N_previous),3)+10;
    y = snakes((N_previous+1):(N(filament_recon)+ N_previous),4)+10;

    N_previous = N(filament_recon)+ N_previous;

    %xx = [x(1):pt_spacing:N(filament_recon)]';

    %Fit a spline to the skeleton and evaluate its derivative
    spline = fit(x,y,'smoothingspline');
    %     plot(spline,x,y,'-')
    y_spline = spline(x);
    % %     hold on
    %      plot(xx,y_spline,'-')
    %     hold off
    differential_spline = differentiate(spline,x);

    %Go through each point in the differential to
    %find the angle of rotation
    theta = double(rot_angle(differential_spline,x));


    %Grab a section (+-w) of the image around x0 to rotate by theta(x0)
    %Take the rotated section and find the new x,y coord
    for k = 1:length(x)%1:size(x)

        try
            %zooming into a wxw section around the point of interest
            %row_range = y(k)-(w):y(k)+(w);    %following the points of the skel image
            %col_range = x(k)-(sz):x(k)+(sz);
            row_range = round(y_spline(k))-(sz):round(y_spline(k))+(sz);    %following the poin
ts of the spline
            col_range = round(x(k))-(sz):round(x(k))+(sz);



            I = I_new_filtered(row_range,col_range);
            I_rot = imagerot(I,(theta(k)));
            %Use for checking rotation of MT
            %          figure(2);
            %          imshow(I_rot,[]);
            %
            %
            %Find max intensity of the middle column of the rotated image
            %section
            I_midpoint = (size(I,1)+1)/2;
            rows_rot = 1:size(I_rot,1); %number of rows
            intensity = double(I_rot(:,I_midpoint)); %gaussfit needs Y data to be a double
            gaussfit = fit(rows_rot.',intensity,'gauss1');
            gauss_centroid = gaussfit.b1;  %x-coord of the peak, gives the y value of the rotated image section
            %         gauss_centroid_matrix(k) = gaussfit.b1;  %x-coord of the peak, gives the y value of the rotated image section
            gauss_centroid_matrix(k,filament_recon) = gaussfit.b1;  %x-coord of the peak, gives the y value of the rotated image section

            % Use for plotting the gaussian fit
            %          figure(5)
            %          plot(rows_rot,intensity,'-d')
            %          hold on
            %          plot(gaussfit,rows_rot,intensity)
            %          hold off
            %          pause
            % %

            % rotate the wxw section back
            coord_unrotated = [cos(-theta(k)) sin(-theta(k));-sin(-theta(k)) cos(-theta(k))]*[0;gauss_centroid-I_midpoint];
            coord_unrotated_gauss_peak(k,filament_recon) = coord_unrotated(2,1);
            %translate coordinate back to its position in the larger image using skel image coorinates

            %         row_reconstructed(k) = coord_unrotated(2,1)+(y(k));  %row number of final position
            %         col_reconstructed(k) = coord_unrotated(1,1)+x(k);  %column number of final position
            %
            %coord_reconstructed(k,1,i) = x;  %uses the x values from the
            %spline fit
            coord_reconstructed(k,1,filament_recon) = coord_unrotated(1,1)+round(x(k));  %column number (x-coord) of final position
            coord_reconstructed(k,2,filament_recon) = coord_unrotated(2,1)+(round(y_spline(k)));  %row number (y-coord) of final position


        catch exception
            catch_count = catch_count+1;
            continue   %if the gauss fit does not converge, then return to the beginning of the loop
        end
    end

    %Generate a concatenated [max number of pixels x 2 x number of images] matrix of coords for all images
    %coord_reconstructed(:,:,filament_recon)  = [col_reconstructed' row_reconstructed'];
    %     %Plot skel image, its fit, and the final reconconstructed MT
    %      f = figure(filament_recon);
    %      hold on
    %      plot(spline,x,y, 'b');     %Plot spline fit over skeletonized image
    % %      print(f,'MT_fit.tif')
    %
    %   %plot reconstructed MT on original image
    figure(6)
    imshow(I_new_filtered(:,:),[])
    hold on
    coord_reconstructed(coord_reconstructed == 0) = NaN;
    plot(coord_reconstructed(:,1,filament_recon),coord_reconstructed(:,2,filament_recon),'r-')
;
    hold off
end
```

## Use for saving coords of each frame so that it can be used in origin

```
Colx = 1;
Coly = 2;
for frame = 1:size(I_orig,3)
    if frame == 1
        excel_data(:,[Colx Coly]) = coord_reconstructed(:,[1 2],frame);
    else
        Colx = Colx +2;
        Coly = Coly +2;
        excel_data(:,[(Colx) (Coly)]) = coord_reconstructed(:,[1 2],frame);
    end


end
```

## Filament Tracking - Fourier Analysis

```
% %User Input
% mode_num = input('What mode number do you want to plot up to?');
% %initializing variables
% count = 1;
% frame = 1;
% a_k = 0;
% k = 0;
% n = 0;
%
%
% %Calculate fourier coefficients
%     for frame_fourier = 1:size(I_skel,3) %number of frames
%         %Grab coordinates of frame j
%         %row = find(~isnan(coord_reconstructed(:,1,j)),1,'last');
%         coord_xy_pixels = coord_reconstructed(:,:,frame_fourier);
%         coord_xy = (1/15.3).*coord_reconstructed(:,:,frame_fourier); %converting from pixels
 to microns 15.3 pix/um
%
%         %Get the angle and segment length of each segment
%         %Calculate the inverse fourier transform for each segment and save
%         %in a vector
%
%         %If there are zeroes in the coord_xy variable, they are ignored by
%         %forcing the last non-zero entry to be the last xy_coord for that
%         %frame
%         if ~any(coord_xy)
%             end_point = find(coord_xy == 0,1,'first')-1;
%         else
%             end_point =  find(coord_xy(:,1),1,'last');
%         end
%
%             for k = 1:end_point-1
%                 %Calculate angle of segment k
%                 delta_x = coord_xy(k+1,1) - coord_xy(k,1);
%                 delta_y = coord_xy(k+1,2) - coord_xy(k,2);
%                 slope = (delta_y/delta_x);
%                 theta_s(k) = atan(slope);
%                 theta_s(isnan(theta_s)) = 0;
%
%                 %Calculate segment k length
%                 delta_s(k) = sqrt(delta_x^2 + delta_y^2); %discrete ds
%                 count = k;
%             end
%             %Sum all the segments for all modes to get a vector of a1..an
%             L = sum(delta_s);
%
%             %for n = 1:mode_num
%                 for k = 1:end_point-1
%                     %Calculate s_mid, the position you are at along the MT
%                     if k == 1
%                         s_mid(k) = delta_s(k);
%                     else
%                         %s_mid(k) = sum(delta_s(k-1))+ .5.*delta_s(k);
%                         s_mid(k) = s_mid(k-1)+0.5.*delta_s(k-1)+0.5.*delta_s(k);
%
%                     end
% %                     a_k(k)= theta_s(k)*delta_s(k)*cos((n*pi()*s_mid(k))/L) %a_n vector for
 each k
%                 end
%                 %summation to get the final coeff based on all the segments, a_1 a_2 a_3, et
c
%                 %coeff = [coeff_mode_1 coeff_mode_2 coeff_mode_3...] for
%                 %one image
%
%             for n = 1:mode_num
%                 for k = 1:end_point-1
%                     a_k = a_k + theta_s(k)*delta_s(k)*exp(i*((n*pi()*s_mid(k))/L));
%                 end
%                 a_n(n) = sqrt(2/L)*a_k;
%                 a_k = 0;
%             end
%             %all the modes for all the frames
%             %coeff_frame = each row corresponds to each frame, each column
%             %corresponds to the coeff for each mode (col 1 is mode 1, col 2
%             %is mode 2..)
%             coeff_frame(frame,:) = a_n;
%             coeff_frame_stats_on_rows(:,frame) = a_n;
%             frame = frame + 1;
%             magnitude_an(frame,:) = abs(a_n);
%
%     end
%
%     figure
%
% %Plotting Results
%
%
%     for plot_mode = 1:mode_num
%             subplot(1,mode_num,plot_mode); %divides the current figure into an 1-by-n grid and creates axes in the position specified by the fram
%             plot(1:size(magnitude_an,1),magnitude_an(:,plot_mode),'.')
%        % linkaxes(mode_plot,'y')
%        ylim([-3 3]);
%     end
%
%     %plotting only reals
%     for i = 1:mode_num
%             subplot(1,mode_num,i); %divides the current figure into an 1-by-n grid and creates axes in the position specified by the fram
%             plot(1:size(coeff_frame,1),coeff_frame(:,i),'.')
%        % linkaxes(mode_plot,'y')
%        ylim([-3 3]);
%     end
%
%     %Plot variance of the amplitudes
%     %Get the squared  difference between the same coeff between each frame and the average of the coeff over all frames, then sum over all frames, and divide by Frames - 1
%     var_coeff = var(magnitude_an,0,1);
%     figure (4)
%     plot(1:n,var_coeff)
```

*Published with MATLAB® R2017a*