SettlingMPs

```
Script to detect particles in images acquired with LabSFLOC camera, compute their size and settling velocity.
Developed by Roberto Fernández (October-November 2020)
Reviewed on March 2022.
Contact: @IceMeanders (Twitter) - R.Fernandez@hull.ac.uk

This code has enough comments that any user can pick it up and understand each step.

Potential Improvements:
```

1. Adjust particle size based on how blurry they are (for out of focus particles)
2. Tracking that includes potential particle rotation as it settles (for example a fibre that starts vertically and ends horizontally).

Contents

Clean up

Clear all variables in the workspace

```
clearvars
% Close all open figure windows
close all
```

Input values

Indicate if you want the code to display intermediate results by using a value of one (1). Use any other number or character to avoid this (code will run faster if intermediate results are not displayed).

```
intRes = 1;
% Specify threshold (area in pixels²) to discard particles in detection routines
% This value might need to be revisited depending on particle size
% expected.
aT = 7000;
% Specify image size (rows - number of pixels in the vertical direction,
% and cols - number of pixels in the horizontal direction).
% NOTE: When working with images, the coordinate (x,y) = (0,0), i.e. the
% origin of the image is always the top left corner. 'x' grows to the right
% (columns), and 'y' grows to the bottom (rows).
rows = 1448;
cols = 1928;
% Specify minimum cross-correlation value to accept a particle pair as a
% match. A value of 1 would require a perfect match. Recommended values vary with image quality
% If images have good quality the value can be as high as 0.9. Reduce value using intermediate
% results visualization to determine an appropriate one for your specfic requirements.
minR = 0.80;
```

Input

The code assumes that the user chooses *.tif or *.jpg files. If a different file format is chosen it will crash. Get image series file - returns the names of all files selected by the user. If only one it is returned as a string. If multiple as a cell. For tracking, the user MUST select more than one image. Tracking requires an image pair (at least). path - returns the folder in which those files are stored (only single path is possible)

```
[file, path] = uigetfile('*.tif', '*.jpg', 'Select an image or an image series', 'MultiSelect', 'on');

% Get number of images (nImages) in the series selected by the user
% Check if the format of file is a cell (this means that the user selected
% multiple images)
if iscell(file)
    % If true, number of images is the number of columns of the cell
    nImages = size(file,2);
elseif file ~= 0
    % If false, and the value isn't zero, convert the variable file to a cell (When only
    % one image chosen, file is a string. However, it is easier to turn into a cell here than to
    % deal with multiple formats later in the code).
    file = {file};
    % Number of images
    nImages = 1;
else
    % Otherwise (if user did not choose a single file) display a message on
    % screen and finish running the code.
    disp('No images selected')
    return
end
```

Detection

The steps for detection are the following

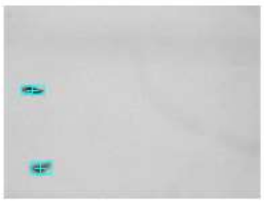
1. Load image
2. Invert image
3. Make image binary (speeds things up)
4. Determine properties of areas within image (particles)

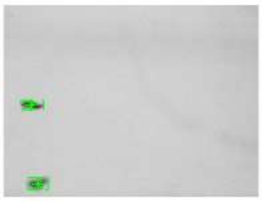
```
% Create array (called structure in Matlab) to store results from all images in a given image series.
% The first field in the structure is the image number (within the series)
% and the second field is another structure that holds the properties of
% all particles detected in the image. (mpProps - microplastic properties)
% Type 'doc struct' or 'help struct' in the Command Window for more information about
% this kind of variable.
allStats = struct('image', NaN, 'mpProps', NaN);

%  Loop through all images in the series. If user selects one image the code
%  runs only once. Otherwise it runs as many times as the number of chosen images.
for j = 1:nImages
    % Call function to detect particles in image and save thier properties in a structure
    % called mpStats (microplastic statistics). The function is included at
    % the end of this file (scroll down).
    mpStats = rf_DetectMPs(path, file{j});
    % Call function to filter out particles that do not meet the area
    % threshold criteria. The function is included at the end of this file
    % (scroll down). Any particles with area smaller than aT (area
    % threhsold) are removed from the mpStats structure.
    mpStats = rf_FilterDetMPs(mpStats, aT);
    % Add image number to allStats structure (one row 'j' per image)
    allStats(j).image = j;
    % Save the properties of all particles in that image as a structure in
    % the second column (or second field).
    allStats(j).mpProps = mpStats;
    % Following lines of code for plotting purposes only (they slow down
    % the code and might be ignored by setting intRes to a value different than one
    % when user knows what they are doing and trusts the code).
    % If user wants to display intermediate results (intRes = 1) at the beginning of the code.
    if intRes == 1
        % Grab centroids of all particles detected in the current image (image
        % j in the series)
        centroids = cat(1, mpStats.Centroid);
        % Grab bounding boxes of all particles detected in the current image
        % (image j in the series). The boxes have four values. The first value
        % is the top left 'x' (column) coordinate of the box; the second value is the
        % top left 'y' (row) coordinate of the box; the third value indicates
        % the horizontal width of the box (number of columns); the fourth
        % column inidcates the vertical height of the box (number of rows).
        boxes = cat(1, mpStats.BoundingBox);
        % Get the 'x' (column) coordinates of all box corners
        bx = [boxes(:,1) boxes(:,1)+boxes(:,3) boxes(:,1)+boxes(:,3) boxes(:,1) boxes(:,1)];
        % Get the 'y' (row) coordinates of all box corners
        by = [boxes(:,2) boxes(:,2) boxes(:,2)+boxes(:,4) boxes(:,2)+boxes(:,4) boxes(:,2)];
        % Clear figure (not needed when j = 1 but needed for j > 1)
        clf
        % Make sure images are plotted in Figure 1
        figure(1)
        % Display image j in the series and hold on (hold on makes sure that
        % new items added to the plot are displayed over the contents already
        % there. Otherwise, the new line deletes the previous contents and only
        % shows the plot/image corresponding to the last line of code).
        imshow(imread(fullfile(path, file{j}))), hold on
        % Plot particle centroids over the image.
        % Centroids will be cyan crosses given by '+c')
        plot(centroids(:,1), centroids(:,2), '+c')
        % Plot the bounding boxes using solid lines '-'
        % Matlab will automatically asign different colors to different boxes.
        plot(bx', by', '-')
        % Pause execution for 0.1 seconds to give the user enough time to see
        % what the code is doing.
        pause(0.1)
        % Last line related to intermediate results plots that may be left out
    end

    % Clear (delete) variables storing information used for plotting
    clear centroids boxes bx by
end
```

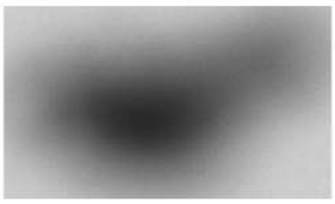
Matching

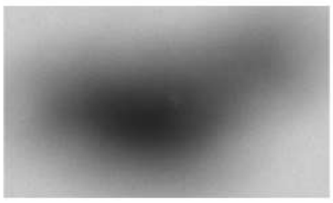
```
Assumes particles settle without rotating
The steps for tracking are the following:
1 Read image pair
2 Grab first particle in image 1
3 Use bounding box of particle in image 1 and extract a subimage (the size of the bounding box) from image 1.
4 Compare to each particle in second image via cross-correlation
5 Identify matches and save relevant data
6 Repeat with next particle in image 1
7 Repeat with next image pair
```

```
% Create structure to save all information related to image pairs
% (allPairs). It contains the following fields:
% image1 - saves the number of the image within the series
% image 2 - saves the number of the image within the series
% nParts1 - saves the number of particles in the first image
% nParts2 - saves the number of particles in the second image
% ccR - saves the cross correlation value (result of comparing the subimage
%       extracted from image 1 with the subimages extracted from image 2). A
%       positive value close to 1 means very good correlation (possible match). Smaller values
%       and negative values indicate poor correlation (no match).
% file1 - stores the string containing the folder and file name of image 1 to help ID
%       later on when postprocessing
% file2 - stores the string containing the folder and file name of image 2 to help ID
%       later on when postprocessing
allPairs = struct('image1', NaN, 'image2', NaN, 'nParts1', NaN, 'nParts2', NaN, 'ccR', NaN, 'file1', NaN, 'file2', NaN);

% Loop through all image pairs (loop goes up to number of images minus one)
% NOTE: This loop could run slightly faster if all things corresponding to
% img2 are assigned to img1 before next iteration (instead of re-reading
% the image and grabbing all values again).
% Would need to do img1 of j=1 before the loop and modify some code near
% the end of it.
if nImages >1
for j=1:nImages-1
    % Read the first image (img1) - j image
    img1 = imread(fullfile(path, file{j}));
    % Read the second image (img2) - j+1 image, next image in the series
    img2 = imread(fullfile(path, file{j+1}));
    % Grab centroids of particles detected in image1 (and not filtered via aT)
    centroids1 = cat(1, allStats(j).mpProps.Centroid);
    % Grab centroids of particles detected in image2 (and not filtered via aT)
    centroids2 = cat(1, allStats(j+1).mpProps.Centroid);
    % Grab bounding boxes of particles detected in image1 (and not filtered via aT)
    boxes1 = cat(1, allStats(j).mpProps.BoundingBox);
    % Grab bounding boxes of particles detected in image2 (and not filtered via aT)
    boxes2 = cat(1, allStats(j+1).mpProps.BoundingBox);
    % Following lines of code for plotting purposes only (they slow down
    % the code and might be ignored by setting intRes to a value different than one
    % when user knows what they are doing and trusts the code).
    % If user wants to display intermediate results (intRes = 1)
    if intRes == 1
        % Get the 'x' (column) coordinates for the corners of all boxes (particles) in image 1
        bx1 = [boxes1(:,1) boxes1(:,1)+boxes1(:,3) boxes1(:,1)+boxes1(:,3) boxes1(:,1) boxes1(:,1)];
        % Get the 'y' (row) coordinates for the corners of all boxes (particles) in image 1
        by1 = [boxes1(:,2) boxes1(:,2) boxes1(:,2)+boxes1(:,4) boxes1(:,2)+boxes1(:,4) boxes1(:,2)];
        % Get the 'x' (column) coordinates for the corners of all boxes (particles) in image 2
        bx2 = [boxes2(:,1) boxes2(:,1)+boxes2(:,3) boxes2(:,1)+boxes2(:,3) boxes2(:,1) boxes2(:,1)];
        % Get the 'y' (row) coordinates for the corners of all boxes (particles) in image 2
        by2 = [boxes2(:,2) boxes2(:,2) boxes2(:,2)+boxes2(:,4) boxes2(:,2)+boxes2(:,4) boxes2(:,2)];
        % Open new figure window (Figure 2)
        figure(2)
        % Clear figure (useless in first iteration but good to have for
        % following iterations)
        clf
        % Create a figure with two rows, one column, and work with the first row
        subplot(2,1,1)
        % Show the first image and hold on
        imshow(img1), hold on
        % Plot centroids over image1 as cyan crosses '+c'
        plot(centroids1(:,1), centroids1(:,2), '+c')
        % Plot bounding boxes over image 1 using cyan solid lines '-c'
        plot(bx1', by1', '-c')
        % Move to second row (plot with 2 rows, 1 column, second row active)
        subplot(2,1,2)
        % Show image 2 and hold on
        imshow(img2), hold on
        % Plot centroids over image 2 as green stars '*g'
        plot(centroids2(:,1), centroids2(:,2), '*g')
        % Plot bounding boxes over image 2 using green dashed lines '--g'
        plot(bx2', by2', '--g')
       % Last line related to intermediate results plots that may be left out
    end
    % Determine number of identified particles in image 1 (nP1)
    nP1 = size(centroids1,1);
    % Determine number of identified particles in image 2 (nP2)
    nP2 = size(centroids2,1);
    % Initialize matrix to store cross correlation results between
    % particles in image one and potential matches (candidate particles) in
    % image 2. Initial values are all zeros.
    R = zeros(nP1, nP2);
    % Save image1 (j) number to allPairs structure
    allPairs(j).image1 = j;
    % Save image2 (j+1) number to allPairs structure
    allPairs(j).image2 = j+1;
    % NOTE: The following two lines of code only work if running the code
    % from Box. The '29' will need to be different if the code is moved to
    % a different folder.
    % Save folder and file name of image1 (j) to allPairs structure
    allPairs(j).file1 = fullfile(path(29:end), file{j});
    % Save folder and file name of image2 (j+1) to allPairs structure
    allPairs(j).file2 = fullfile(path(29:end), file{j+1});
    % Save number of particles in image1 (j) to allPairs structure
    allPairs(j).nParts1 = nP1;
    % Save number of particles in image2 (j+1) to allPairs structure
    allPairs(j).nParts2 = nP2;

    % Loop through all particles in image 1 (nP1)
    for k = 1:nP1
        % Determine size of bounding box of particle k in image1
        boxsz = [boxes1(k,4) boxes1(k,3)];
        % Grab centroid coordinates of particle k in image1
        cntrd1 = centroids1(k,:);
        % Determine the 'x,y' (column, row) coordinates for the sides of the box
        % bounding particle k in image1 (image j)
        % Top row (y) - centroid position minus half the box height rounded
        % towards minus infinity (floor)
        top1 = floor(cntrd1(2)-ceil(boxsz(1)/2)); % y increases from top to bottom
        % Bottom row (y) - centroid position plus half the box height
        % rounded towards plus infinity (ceil)
        bottom1 = ceil(cntrd1(2)+ceil(boxsz(1)/2)); % y increases from top to bottom
        % Left column (x) - centroid position minus half the box width rounded
        % towards minus infinity (floor)
        left1 = floor(cntrd1(1)-ceil(boxsz(2)/2));
        % Right column (x) - centroid position plus half the box width rounded
        % towards plus infinity (ceil)
        right1 = ceil(cntrd1(1)+ceil(boxsz(2)/2));
        % Verify that box is not touching image1 edges on any side
        % If top side smaller than first row (y = 1) OR
        % If bottom side is larger than image number of rows (y = rows) OR
        % If left side is smaller than first column (x = 1) OR
        % If right side is larger than image number of columns (y = cols)
        if top1 < 1 || bottom1 > rows || left1 < 1 || right1 > cols
            % If true make subimg1 equal to nans (Not a number NaN)
            subimg1 = uint8(nan(bottom1-top1+1, right1-left1+1));
        else
            % If no side of bounding box near image edges
            % Extract sub image from image1 using the top-bottom,
            % left-right coordinates.
            subimg1 = img1(top1:bottom1, left1:right1);
        end

        % Following lines of code for plotting purposes only (they slow down
        % the code and might be ignored by setting intRes to a value different than one
        % when user knows what they are doing and trusts the code).
        % If user wants to display intermediate results (intRes = 1)
        if intRes == 1
            % Open new figure window - Figure 3
            figure(3)
            % Clear figure (uselss in first iteration but good to have for
            % following iterations)
            clf
            % Two rows, one column, go to first row
            subplot(2,1,1)
            % Show subimg1 (small image containing particle extracted from
            % image1 or black box = NaN if it was touching the edge of the original image)
            imshow(subimg1), hold on
            % Last line related to intermediate results plots that may be left out
        end
        % Loop through all particles in image2 (all candidates)
        for m = 1:nP2
            % Grab centroid paticles of particle m in image 2
            cntrd2 = centroids2(m,:);
            % Determine the 'x,y' (column, row) coordinates for the sides of the box
            % bounding particle m in image2 (image j+1)
            % Top row (y) - centroid position minus half the box height rounded
            % towards minus infinity (floor)
            top2 = floor(cntrd2(2)-ceil(boxsz(1)/2)); % y increases from top to bottom
            % Bottom row (y) - centroid position plus half the box height
            % rounded towards plus infinity (ceil)
            bottom2 = ceil(cntrd2(2)+ceil(boxsz(1)/2)); % y increases from top to bottom
            % Left column (x) - centroid position minus half the box width rounded
            % towards minus infinity (floor)
            left2 = floor(cntrd2(1)-ceil(boxsz(2)/2));
            % Right column (x) - centroid position plus half the box width rounded
            % towards plus infinity (ceil)
            right2 = ceil(cntrd2(1)+ceil(boxsz(2)/2));
            % Verify that box is not touching image2 edges on any side
            % If top side smaller than first row (y = 1) OR
            % If bottom side is larger than image number of rows (y = rows) OR
            % If left side is smaller than first column (x = 1) OR
            % If right side is larger than image number of columns (y = cols)
            if top2 < 1 || bottom2 > rows || left2 < 1 || right2 > cols
                % If true make subimg2 equal to nans (Not a number NaN)
                subimg2 = uint8(nan(bottom2-top2+1, right2-left2+1));
            else
                % If no side of bounding box near image edges
                % Extract sub image from image2 using the top-bottom,
                % left-right coordinates.
                subimg2 = img2(top2:bottom2, left2:right2);
            end

            % Following lines of code for plotting purposes only (they slow down
            % the code and might be ignored by setting intRes to a value different than one
            % when user knows what they are doing and trusts the code).
            % If user wants to display intermediate results (intRes = 1)
            if intRes == 1
                % Activate Figure 3 window
                figure(3)
                % Two rows, one column, go to second row
                subplot(2,1,2)
                % Show subimg2 (small image containing particle extracted from
                % image2 or black box = NaN if it was touching the edge of the original image)
                imshow(subimg2), hold on
                % Pause code execution for 0.1 seconds to allow user to see
                % what the code is doing.
                pause(0.1)
                % Last line related to intermediate results plots that may be left out
            end
            % Verify that subimg1 and subimg2 have the same dimensions
            if(size(subimg1) == size(subimg2))
                % If true (subimg1 and subimg2 have same size)
                % Compute cross correlation between both subimages
                R(k,m) = corr2(subimg1, subimg2);
                % Filter right away those particle pairs for which the
                % cross correlation coefficient is les sthan minR
                % (threshold to accept a pair of particles as a successful
                % match)
                % If R < minR
                if(R(k,m)<minR)
                   % Assign a NaN if R value suggests that particles are different.
                   R(k,m) = NaN;
                end
            else
                % If false (i.e. subimg1 and subimg2 have different sizes)
                % Assign NaN to the cross correlation value
                R(k,m) = NaN;
            end
            % Clear variables before next iteration
            clear cntrd2 top2 bottom2 left2 right2 subimg2
        end
        % Clear variables before next iteration
        clear boxsz cntrd1 top1 bottom1 left1 right1 subimg1
    end

        % Save all cross correlation results (matrix with nP1 rows and nP2
        % columns) to the allPairs structure.
        allPairs(j).ccR = R;
        % Clear variables before next iteration
        clear R img1 img2 centroids1 centroids2 boxes1 boxes2 bx1 bx2 by1 by2
end
end
clear j k m
save('results.m', 'allStats', 'allPairs')
```









Displacements and Areas

Compute displacements of succesfully matched particles and save their relevant details as results (mpTracks)

```
%  Create matrix to save all information related to particle tracking
%  (mpTracks). It contains the following columns:
%  cx1 - stores the 'x' coordinate (column) of a particle in image 1
%  cy1 - stores the 'y' coordinate (row) of a particle in image 1
%  cx2 - stores the 'x' coordinate (column) of the same particle in image 2
%  cy2 - stores the 'y' coordinate (row) of the same particle in image 2
%  dy - displacement in 'y' (associated to fall velocity)
%  a1 - area of particle in image 1
%  a2 - area of same particle in image 2

mpTracks = nan(1,7);


% Initialize results variable
results = [NaN NaN];
% Loop through all image pairs
for j = 1:nImages-1
    % Go through all pairs and compute displacements between frames.
    % Find indices for matched micro plastic particles (those locations
    % where ccR is not NaN)
    mMP = ~isnan(allPairs(j).ccR);
    % Get corresponding indices (positions within the matrix)
    idx = find(mMP>0);
    if ~isempty(idx)
        % Particle(s) on image 1 and particle(s) on image 2
        [pim1, pim2] = ind2sub(size(allPairs(j).ccR), idx);
        % Grab centroids of particles detected in image1
        centroids1 = cat(1, allStats(j).mpProps.Centroid);
        % Grab centroids of particles detected in image2
        centroids2 = cat(1, allStats(j+1).mpProps.Centroid);
        % Grab 'x' coordinate of centroids (for particle tracking)
        cx1 = centroids1(pim1,1);
        cx2 = centroids2(pim2,1);
        % Grab 'y' coordinate of centroids to compute displacements
        cy1 = centroids1(pim1,2);
        cy2 = centroids2(pim2,2);
        % Compute vertical displacements between matched particles (only in
        % 'y' not total displacements).
        dy = cy2 - cy1;
        % Grab area of particles detected in image1
        areas1 = cat(1, allStats(j).mpProps.Area);
        % Grab area of particles detected in image2
        areas2 = cat(1, allStats(j+1).mpProps.Area);

        a1 = areas1(pim1);
        a2 = areas2(pim2);
        % Concatenate intermediate results
        intResult = [cx1 cy1 cx2 cy2 dy a1 a2];
        % Get rid of particles with fall velocities smaller or equal to
        % zero (not moving or moving upwards).
        intResult(intResult(:,5) <= 0, :) = [];
        % Concatenate results matrix
        if ~isempty(intResult)
            mpTracks = [mpTracks; intResult];
        end
    end
    % clear variables before next iteration
    clear mMP idx pim1 pim2 centroids1 centroids2 cy1 cy2 dy areas1 areas2 mpAreas intRes
end
% Remove first row containing NaNs
mpTracks(1, :) = [];
```

Tracking

Go through all matched particles and identify if same particle is matched
in more than one image pair (i.e. track it).

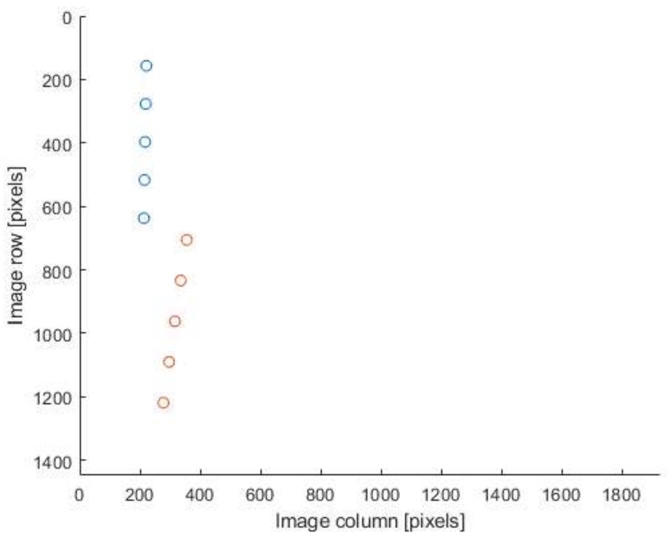
```
% Obtain total number of particle matches
mpTotal = size(mpTracks, 1);
% Add column to mpTracks to include particle id
mpTracks = [zeros(mpTotal,1) mpTracks];
% Initialize particle counter
mpCounter = 1;
% Give row 1 a particle ID of 1
mpTracks(1,1) = 1;

% The routine below has too many if-else statements and I am sure there is
% a more elegant solution. It started as a simple if statement but through
% debugging and having issues with certain image series it became messy.
% Loop through all particles to track centroids
for j=2:mpTotal
   % Find the column that has a cx1 == cx2 (same 'centroid 'x' coordinate in current and previous image).
   posx = find(mpTracks(j,2) == mpTracks(:,4));
   % Find the row that has a cy1 == cy2 (same 'centroid 'y' coordinate in current and previous image).
   if isempty(posx)
       % Increase the counter by one
       mpCounter = mpCounter + 1;
       % Assign new particle ID
       mpTracks(j,1) = mpCounter;
   else
       posy = find(mpTracks(j,3) == mpTracks(:,5));
       % If no matches found or if the rows for x,y do not match
       if isempty(posy)
           % Increase the counter by one
           mpCounter = mpCounter + 1;
           % Assign new particle ID
           mpTracks(j,1) = mpCounter;
       % Else, if matches found
       elseif posx ~= posy
           % Increase the counter by one
           mpCounter = mpCounter + 1;
           % Assign new particle ID
           mpTracks(j,1) = mpCounter;
       elseif size(posx,1)>1
           % Increase the counter by one
           mpCounter = mpCounter + 1;
           % Assign new particle ID
           mpTracks(j,1) = mpCounter;
       else
           % Assign the same particle ID
           mpTracks(j,1) = mpTracks(posy,1);
       end
   end
end
```

Independent Particle Results

Get median displacement and median area for particles identified in
multiple image pairs

```
% Max number of individual particles detected and matched
maxID = max(mpTracks(:,1));
```

```
% Create a figure to plot trajectories
figure(4)
set(gca, 'ydir', 'reverse')
xlim([0 cols]);
ylim([0 rows]);
hold on

% Loop through all particle IDs
for j = 1:maxID
   % Find all rows in mpTracks corresponding to the same particle ID
   id = find(mpTracks(:,1)== j);
   % Plot trajectory
   plot(mpTracks(id, 2), mpTracks(id,3), 'o')
   % Add plot labels
   xlabel('Image column [pixels]')
   ylabel('Image row [pixels]')
   % Create summary matrix including particle ID, median displacement and
   % median particle area.
   mpSummary(j,1) = j;
   mpSummary(j,2) = median(mpTracks(id, 6));
   mpSummary(j,3) = median(mpTracks(id, 7));
end
```



Output

Write results to spreadhseet Newer versions of matlab prefer to use 'writematrix' instead of 'xlswrite' but my current version predates the change.

```
[outfile,outpath] = uiputfile('*.xlsx', 'Save Results', 'results.xlsx');
if outpath ~= 0
    outfilename = fullfile(outpath,outfile);
    xlswrite(outfilename, mpTracks);
    xlswrite(outfilename, mpSummary, 2)
    save(fullfile(outpath, outfile(1:end-5)), 'mpSummary', 'mpTracks', 'allStats', 'allPairs')
end
```

Detection Function

```
The function receives two strings:
path - folder in which the image is stored
file - name of the image and an integer
After reading the image and making it binary, the code determines the
properties of the particles in the image by using Matlab's regionprops
function.
```

```
function [stats] = rf_DetectMPs(path, file)
%rf_DetectMPs finds particles in a grayscale image
    % Read greyscale image
    img = imread(fullfile(path, file));
    % Invert image (darker shades become lighter and viceversa)
    invimg = 255-img;
    % Convert image to black and white only
    BW = imbinarize(invimg, 'adaptive', 'Sensitivity', 0.3);
    %BW = imbinarize(invimg, 'global');
%     subplot(1,3,1), imshow(img)
%     subplot(1,3,2), imshow(invimg)
%     subplot(1,3,3), imshow(BW)
%     pause(0.1)
    % Determine properties of particles in the image
    stats = regionprops(BW, 'Area', 'BoundingBox', 'Centroid', 'EquivDiameter', 'MajorAxisLength', 'MinorAxisLength', 'Orientation');
end
```

Post-Detection Filtering

Filter results of detected particles in images Need to filter particles whose bounding box is near the edge of the image Filter based on particle size first then bounding box

```
%   areaThreshold - integer value used to filter out 'particles'
%   misidentified by the code. Any particle with an area (in pixel²)
%   smaller than areaThreshold will be removed.

%   Before returning the statistics (stats) of the particles, the code
%   filters the results to remove misidentified particles.
function [stats] = rf_FilterDetMPs(stats, areaThreshold)
% Create an array function to apply to all entries in the stats structure
fun = @(x) stats(x).Area < areaThreshold;
% Identify all stats entries that have an area smaller than
% areaThreshold
discard = arrayfun(fun, 1:numel(stats));
% Discard stats entries with smaller areas (not particles)
stats(discard) = [];
end
```