```
function spot_results = FindTrajects(image_label,start_frame,end_frame)
%
% Created by Isabel Llorente-Garcia, August 2011.
% If you use this code please acknowledge Isabel Llorente-Garcia in your
% publications.
%
% This function takes an image_label such as '513', '490', etc...
% which corresponds to a certain .sif image sequence in current folder,
% and then finds all trajectories of the bright fluorescent spots in time.
%
% Example of how to use this function:
% for image "Heiko_Thu Jun 24 2010_554.sif" in current folder:
% [numFrames2 frame_Ysize2 frame_Xsize2 image_data2] = extract_image_sequence_data('554');
%
% Reads .sif image sequence data and then for each frame:
% finds candidate spots,
% joins them to spots found and accepted on previous frame,
% eliminates coincidences in those candidates (points closer than 1 pixel),
% within all candidates, it finds actual spot centres through iterative Gaussian masking,
% and accepts only found spot centres with clipping flag equal to zero, constrained width and large enough SNR,
% eliminates coincidences from previously accepted found spot centres.
% Resulting final accepted spot centres are saved in the structure array spot_final.
% Then link spots into trajectory segments: For second frame do differently
% and check only spots in 1st frame and compare to spots in second frame.
% For rest of frames (for frame k): A) first check loose spots (TrajNumber=0) two frames
% ago (k-2) and compare to spots in current frame (k).
% B) then check all spots in previous frame (k-1) and compare to spots in
% current frame (k).
% To decide on the best asignment of pairs of spots (link), we build up
% matrices of pair-wise distances, ratio of intensities and ratio of sigmas
% of all pairs of spots in the two frames being compared, and take the
% winning asignment as that with the smallest pairwise distance.
% The intensity ratio and ratio of sigmas has to be within certain min and
% max bounds too.
%
% start_frame and end_frames are the frames through which the loop runs to
% find centres of bright spots.
```

```
38 %
39 % Output:
40 % The output, spot_results is a cell array with two elements:
41 % spot_results = {params, spot_final};
42 % The first element in the cell array, spot_results{1}, contains all parameters used to run
43 % the function: "params" (this is a structure array itself).
44 % The second element in the cell array, spot_results{2}, contains the track
45 % results: "spot_final".
46 % "spot_final", is itself a structure array with end_frame by L elements,
47 % each of which is a structure with fields: CentreX, CentreY, IspTot,
48 % Sigma, IbgAvg, IbgTot, IinnerTot, ClipFlag, FrameNumber and SpotNumber.
49 % Along the dimension L, we have the different bright spots found on each
50 % frame (L will usually be the number of spot centres found in frame_start,
51 % or the largest number of spots ever accepted).
52 % For example, spot_final(100,8) is a structure with the above fields
53 % corresponding to the eighth found spot on frame 100.
54 %
55 % Example of params: spot_results{1}:
56 %
57 %             image_label: '498'
58 %             start_frame: 7
59 %               end_frame: 500
60 %                       r: 5
61 %      max_num_candidates: 2000
62 %      subarray_halfwidth: 8
63 %     inner_circle_radius: 5
64 %        gauss_mask_sigma: 2
65 %            sigmaFit_min: 2
66 %            sigmaFit_max: 4
67 %                 SNR_min: 2
68 %                 rsq_min: 0.2000
69 %                 deflate: 1
70 %          d_coincid_cand: 1
71 %         d_coincid_found: 1
72 %               d_01_max: 5
73 %           Iratio_01_min: 0.5000
74 %           Iratio_01_max: 3
```

```
 75 %       SigmaRatio_01_min: 0.5000
 76 %       SigmaRatio_01_max: 2
 77 %                d_02_max: 5
 78 %           Iratio_02_min: 0.5000
 79 %           Iratio_02_max: 3
 80 %       SigmaRatio_02_min: 0.5000
 81 %       SigmaRatio_02_max: 2
 82 %                     rej: 10000
 83 %               file_name: 'Heiko_Thu Jun 24 2010_498.sif'
 84 %               numFrames: 500
 85 %             frame_Ysize: 512
 86 %             frame_Xsize: 512
 87 %
 88 % Example of how to call this function:
 89 % s0 = FindTrajects('470',5,500); uses
 90 % image sequence 470 and finds trajectories in it, for frames 5 to 500.
 91 % Another example: s1 = FindTrajects('490',100,110);.
 92 % ------------------------------------------------------
 93 % Note: for reading .sif files you need 'read_sif_data_direct.m',
 94 % 'GetAndorSifSize', etc. which are currently in path:
 95 % 'C:\Isabel\myMatlabFiles\IO_Input\'.
 96 %
 97 % NOTE: before running this function you should move into the directory which
 98 % contains the image sequence data (labelled by 'image_label').
 99
100
101 %% DEFINITIONS and PARAMETERS:
102
103 % I have saved all parameter values in the file "paramsForFindTrajects.m"
104 % in the current directory. Just calling the name of that file loads the
105 % parameter values into the workspace:
106 paramsForFindTrajects
107 % In this way, we can save different parameter sets for different data sets
108 % in an easy manner and run two Matlabs at the same time working with different parameter sets.
109
110 % %
111 % % define data (.sif image) directory:
```

```
% % dir_data = 'Z:\Leake\Heiko Data\';
% % dir_data = 'C:\Isabel\ExperimData\HeikoData\';
% % Print data directory on command window to guide user:
% disp(' ') % empty line
% disp(['The data directory (.sif images) is: ',cd]) % display current directory.
%
% % Save input parameters to "params":
% params.image_label = image_label;
% params.start_frame = start_frame;
% params.end_frame = end_frame;
%
% % Number of frames to average over when calculating a frame average in
% % order to get a Signal Mask to distinguish cell region from background
% % region:
% r = 5; % number of frames to average over, starting from start_frame. (default: 5, end_frame-start_frame)
% % Save parameters to results as structure "params":
% params.r = r;
%
% % If you don't want to use the cell mask to exclude spots found outside
% it, make the following parameter equal to 1 (then spots can be accepted anywhere on image):
% doNotUseCellMaskToTellSpotsRegion = 0;
% params.doNotUseCellMaskToTellSpotsRegion = doNotUseCellMaskToTellSpotsRegion;
%
% % Maximum number of candidate spots (if eg. 260000 candidate spots are
% % found, we get an error in function pdist: "Distance matrix has more
% % elements than the maximum allowed size in MATLAB"), hence, we limit the
% % max number of candidate spots:
% max_num_candidates = 2000; % should be around 200.
% % Save parameters to results as structure "params":
% params.max_num_candidates = max_num_candidates;
%
% % PARAMETERS for finding spot centres (see findSpotCentre1frame.m, inputs to the function):
% % The total integrated spot intensity is a bgnd corrected one, inside a
% % circular mask of radius inner_circle_radius.
% subarray_halfwidth = 8; % (Default: 8 pixels). Halfwidth of image square subarray
% % ROI, typically a square of size 17x17 pixels.
% inner_circle_radius = 5; % (Default: 5 pixels). Radius of inner circular mask that moves inside the fixed square subarray.
```

```
149 % gauss_mask_sigma = 2; % (Default: 2 pixels). Size in pixels of the applied Gaussian mask.
150 % guess_sigma_Fit = 3; % starting guess for Gaussian fit of brightspot intensity (default = 3).
151 % % Save parameters to results as structure "params":
152 % params.subarray_halfwidth = subarray_halfwidth;
153 % params.inner_circle_radius = inner_circle_radius;
154 % params.gauss_mask_sigma = gauss_mask_sigma;
155 % params.guess_sigma_Fit = guess_sigma_Fit;
156 %
157 % % PARAMETERS for deciding if we accept a spot centre found by the function
158 % % findSpotCentre1frame or not:
159 % sigmaFit_min = -inner_circle_radius; % minimum acceptable sigma of gaussian fit to spot, in pixels (2) (-3).
160 % sigmaFit_max = inner_circle_radius; % maximum acceptable sigma of gaussian fit to spot, in pixels (4) (3).
161 % SNR_min = 1.4; % minimum acceptable signal-to-noise ratio (at least 2) (SNR as defined in findSpotCentre1frame.m).
162 % rsq_min = 0.1; % minimum acceptable r-square value (0.2) (goodness of gaussian fit to spot).
163 % % Save parameters to results as structure "params":
164 % params.sigmaFit_min = sigmaFit_min;
165 % params.sigmaFit_max = sigmaFit_max;
166 % params.SNR_min = SNR_min;
167 % params.rsq_min = rsq_min;
168 %
169 % % PARAMETER to decide if deflation method is applied or not (subtract each
170 % % found spot to allow detection of weaker spots):
171 % deflate = 1;
172 % % Save parameters to results as structure "params":
173 % params.deflate = deflate;
174 %
175 % % PARAMETERS for eliminating coincident spots:
176 % d_coincid_cand = 1; % distance (in pixels) for eliminating coincidences in spot candidates.
177 % d_coincid_found = 1; % distance for eliminating coincidences in found spot centres.
178 % % Save parameters to results as structure "params":
179 % params.d_coincid_cand = d_coincid_cand; % distance for eliminating coincidences in spot candidates.
180 % params.d_coincid_found = d_coincid_found; % distance for eliminating coincidences in found spot centres.
181 %
182 % % PARAMETERS for building trajectories:
183 % % For linking spots in current and previous frames:
184 % d_01_max = 5; % max distance in pixels between spot centres in current and previous frames, for linking them into a↙
trajectory (5).
```

```
% Iratio_01_min = 0.5; % min ratio of total spot intensities (after bgnd subtraction) (0.5).
% Iratio_01_max = 3; % max ratio of total spot intensities (after bgnd subtraction) (frame k-1/frame k) (large enough value (3) to account for blinking).
% SigmaRatio_01_min = 0.5; % min ratio of spot widths (sigma of Gaussian fit) (0.5).
% SigmaRatio_01_max = 2; % max ratio of spot width (sigma of Gaussian fit) (2).
% % Save parameters to results as structure "params":
% params.d_01_max = d_01_max;
% params.Iratio_01_min = Iratio_01_min;
% params.Iratio_01_max = Iratio_01_max;
% params.SigmaRatio_01_min = SigmaRatio_01_min;
% params.SigmaRatio_01_max = SigmaRatio_01_max;
%
% % For linking loose spots in current frame and 2 frames ago (jump of 1 frame in trajectory):
% d_02_max = 5; % max distance in pixels between spot centres in current frame and 2 frames ago. (default: 5)
% Iratio_02_min = 0.5; % min ratio of total spot intensities (after bgnd subtraction).
% Iratio_02_max = 3; % max ratio of total spot intensities (after bgnd subtraction).
% SigmaRatio_02_min = 0.5; % min ratio of spot widths (sigma of Gaussian fit).
% SigmaRatio_02_max = 2; % max ratio of spot width (sigma of Gaussian fit).
% % Save parameters to results as structure "params":
% params.d_02_max = d_02_max;
% params.Iratio_02_min = Iratio_02_min;
% params.Iratio_02_max = Iratio_02_max;
% params.SigmaRatio_02_min = SigmaRatio_02_min;
% params.SigmaRatio_02_max = SigmaRatio_02_max;
%
% % Use a very large number (larger than image size in pixels) for rejected asignments:
% rej = 10000;
% params.rej = rej; % Save parameters to results as structure "params".
%
% % Parameter to exclude a region from accepting spots (see later on, lines 322 and 507):
% exclude_region = 0;
% % Note that when exclude_region is 1, at the moment is it set for an image
% % with two channels, where spots are excluded also from around a horizontal
% % line centred on the image.
% exclude_region_width = subarray_halfwidth; % exclude edges of image, only look at spots with a centre far enough from image edges.
% params.exclude_region = exclude_region;
```

```
% params.exclude_region_width = exclude_region_width;

% Alternative way of selecting image sequence file:
% uigetfile opens a file dialog box to choose data file:
% [file_data,path_data] = uigetfile({'*.sif'}, 'Chose image data sequence:');
% strcat('data (.sif image):','  ',path_data,file_data)
% open a file dialog box to choose analysis file:
% [file_analysis,path_analysis] = uigetfile({'*.xls'}, 'Chose analysis file (trajectory):');
% strcat('analysis file (.xls trajectory):','  ',path_analysis,file_analysis)

disp(' ') % empty line
disp(['The start frame for finding bright spot trajectories will be ',num2str(start_frame)]) % start_frame is an input.
disp(['The end frame for finding bright spot trajectories will be ',num2str(end_frame)]) % end_frame is an input.
% ------------------------------------------------------


%% Read in the image-sequence data:

% Read image-sequence file:
[numFrames frame_Ysize frame_Xsize image_data image_path] = extract_image_sequence_data(image_label);
% See "extract_image_sequence_data.m".
% numFrames is the number of frames in the image sequence.
% To get frame number "p" do: image_data(p).frame_data.
% Frame dimensions are frame_Ysize and frame_Xsize.
% ------------------------------------------------------------

% Save to parameters:
params.file_name = image_path;
params.numFrames = numFrames;
params.frame_Ysize = frame_Ysize;
params.frame_Xsize = frame_Xsize;


%% Calculate Signal Mask to distinguish cell region from background region:
% Use frame average (of first frames only) to calculate signal mask.

% Initialise frame accumulation in order to later calculate a frame average:
```

```
257 frame_accumul = zeros(frame_Ysize,frame_Xsize);
258
259 % r is the number of frames to average over, starting from start_frame.
260 % See PARAMETERS section.
261
262 for k = start_frame:start_frame+r  % loop through frames.
263     % Get frame data:
264     frame = image_data(k).frame_data; % extract frame data, stored in the field 'frame_data'.
265     frame = double(frame);
266     % Accummulate frames to then calculate frame average:
267     frame_accumul = frame_accumul + frame;
268 end
269
270 % Calculate frame average as the accumulation of all frames divided by the number of frames:
271 frame_avg = frame_accumul/(r+1);
272
273 frame_avg_Gray = mat2gray(frame_avg); % The input to function "getCellMaskAndBoundary" needs to be a grayscale image:
274
275 % Get SignalMask to know where cells are, to distinguish cells from background:
276 [SignalMask CellBoundaryMask] = getCellMaskAndBoundary(frame_avg_Gray);
277 % SignalMask is a matrix with 1 at positions where cells are and 0 at background.
278
279 % Or use getCellMaskAndBoundary2(frame,local_region),
280 % with local_region = [xleft xright ytop ybottom].
281 % [SignalMask CellBoundaryMask] = getCellMaskAndBoundary2(frame_avg_Gray,[1 frame_Xsize round(frame_Ysize/2) frame_Ysize]); %↙
Use bottom half of image only.
282 % Using a local region for thresholding and finding the cell mask, makes this function much faster,
283 % since only spots within that cell mask will be considered.
284
285 % CHECK: uncomment the following line if you don't want to use a signal
286 % mask. Detected spots don't need to be within a signal mask:
287
288 if doNotUseCellMaskToTellSpotsRegion ==1
289     SignalMask = ones(size(frame_avg_Gray,1),size(frame_avg_Gray,2));
290 end
291
292 %% Obtain candidate bright spots for start_frame (first frame), and find spot centres for those:
```

```
293
294 frame = image_data(start_frame).frame_data; % extract matrix data for first frame.
295 frame = double(frame);
296
297 disp(['frame number: ',num2str(start_frame)]) % print frame number to Command Window.
298
299 % Xpos is a matrix of the same size as frame, containing x values for all
300 % pixels and similarly for Ypos (used in future sections):
301 [Xpos,Ypos] = meshgrid(1:frame_Xsize,1:frame_Ysize);
302 % Note that the image thresholding occurrs in two halves: separating top and bottom halves.
303 % Find candidate-bright-spots on first frame:
304 frame_Gray = mat2gray(frame); % The input to function "findCandidateSpots" needs to be a grayscale image:
305
306 [candidate_spotsX_00 candidate_spotsY_00] = findCandidateSpots(frame_Gray,2); % Second input: use method 2, which seems to↙
work better.
307 % See C:\Isabel\myMatlabFiles\findCandidateSpots.m.
308 % candidate_spotsX and candidate_spotsY are two column vectors of the same
309 % length containing the x and y coordinates of the candidate bright spots found on the image.
310 % They contain integer numbers: coordinates or pixel numbers which give
311 % position on image plane.
312
313 % Reject candidate spots outside cell region (to speed up algorithm):
314 candidate_spotsX_0 = []; % initialise empty vectors before loop.
315 candidate_spotsY_0 = [];
316 for nn = 1:length(candidate_spotsX_00)
317    % Only use candidates for which there is a 1 in the SignalMask image:
318    if SignalMask(candidate_spotsY_00(nn),candidate_spotsX_00(nn))==1
319        candidate_spotsX_0 = [candidate_spotsX_0; candidate_spotsX_00(nn)];
320        candidate_spotsY_0 = [candidate_spotsY_0; candidate_spotsY_00(nn)];
321    end
322 end
323
324 disp(['no. of new candidate spots on start frame: ',num2str(length(candidate_spotsX_0))])
325
326 % Error control:
327     % Limit the max number of candidate spots (if eg. 260000 candidate spots are
328     % found, we will get an error in function pdist: "Distance matrix has more
```

```
329     % elements than the maximum allowed size in MATLAB").
330     % Select only the first max_num_candidates then.
331     if length(candidate_spotsX_0) > max_num_candidates
332         candidate_spotsX_0 = candidate_spotsX_0(1:max_num_candidates);
333         candidate_spotsY_0 = candidate_spotsY_0(1:max_num_candidates);
334         disp(['NOTE!! no. of candidate spots has been limited to ',num2str(max_num_candidates)])
335     end
336
337 % % Check graphically:
338 % imshow(frame_Gray,[]);
339 % hold on;
340 % plot(candidate_spotsX_0,candidate_spotsY_0,'*');
341 % figure;
342
343 % Find spot centres and decide if we accept them or not:
344 n =1; % Initialise index n (index for accepted spot centres which have a clipping flag equal to zero):
345 frame_to_search = frame; % Initialise frame to search for spot centres.
346
347 % Find spot centre through iterative masking:
348 for m = 1:size(candidate_spotsX_0,1) % loop through all candidate spots.
349     % Now find centre of bright spot using function findSpotCentre1frame:
350     % use candidate spots as initial estimates and then iterate to find spot centre.
351     % Image subarray ROI is a square of size 17x17 pixels (halfwidth is
352     % 8 pixels), inner circular mask that moves inside the fixed 17x17
353     % square has a radius of 5 pixels and the applied Gaussian
354     % mask has a sigma of 2 pixels:
355     spot_result = findSpotCentre1frame(frame_to_search,candidate_spotsX_0(m),candidate_spotsY_0(m),subarray_halfwidth,↙
inner_circle_radius,gauss_mask_sigma,guess_sigma_Fit);
356     spot_result.FrameNumber = start_frame; % Add new field containing frame number (time) to result structure.
357
358
359     if (spot_result.ClipFlag == 0 && spot_result.noConverge == 0 && ...
360             spot_result.SigmaFit <= sigmaFit_max && ...
361             spot_result.SigmaFit >= sigmaFit_min && ...
362             spot_result.SNR >= SNR_min &&...
363             spot_result.rsqFit >= rsq_min) &&...
364             (exclude_region == 0 || (exclude_region ==1 && (spot_result.CentreY <(frame_Ysize/2-exclude_region_width) ||↙
```

```
spot_result.CentreY >(frame_Ysize/2+exclude_region_width))))
            % Only accept and save result of found spot if clipping flag =0 and if values of sigmaFit, signal to noise and↙
rsquare of fit are acceptable.
            spot_result.SpotNumber = n; % Add new field containing spot number to result structure.
            spot_final(start_frame,n) = spot_result; % store "good" found spots.
            % This is also saved in the final result spot_final, structure array.
            % first index is for frame number, second index is for spot number.

            %--------------------------------
            if deflate==1 % see parameter section at the beginning.
                % "Deflation" process: subtract from raw frame image the corresponding
                % Gaussian fit of each found and accepted spot before finding next spot centre (enables acceptance of dimmer↙
spots).
                %
                % Matrices containing the x and y positions in the image frame: Xpos and Ypos
                % Xpos is a matrix of the same size as frame, containing x values for all pixels and similarly for Ypos.
                % Calculate Xpos, Ypos at the beginning: [Xpos,Ypos] = meshgrid(1:frame_Xsize,1:frame_Ysize);
                % Parameters of Gaussian fit of previously accepted spot:
                x_fit = spot_final(start_frame,n).CentreX;
                y_fit = spot_final(start_frame,n).CentreY;
                I_fit = spot_final(start_frame,n).I0Fit;
                sigma_fit = spot_final(start_frame,n).SigmaFit;
                % deflated frame (frame with found spot subtracted):
                deflated_frame = frame_to_search - I_fit*exp(-((Xpos-x_fit).^2+(Ypos-y_fit).^2)/(2*sigma_fit^2));
                frame_to_search = deflated_frame; % update frame to search for finding next spot-centre.
                % % Graphical check of deflated frames:
                % subplot(1,2,1); imshow(frame,[]); % frame is the original frame (always the same).
                % subplot(1,2,2); imshow(deflated_frame,[]);
            end
            %--------------------------------

            n = n+1; % advance index n for accepted spot centres.
    end
end

% % display the number of accepted spot-centres for this frame:
disp(['no. of accepted spot centres in first frame: ',num2str(n-1)])
```

```
399
400 % convert results of found spot-centre positions to a useful form that can
401 % be used as input candidate-spots on the following frame:
402 if (n-1) == 0 % error control: if no spots were accepted.
403     found_spot_CentreX = [];
404     found_spot_CentreY = [];
405     % I need to create the whole spot_final structure with all its fields
406     % here, just in case the number of accepted spots in the first frame is
407     % zero, in order not to get error: "Subscripted assignment between
408     % dissimilar structures".
409     % Save empty spot (we need this, otherwise if in the last frame the no. of accepted spots is 0, there will be no result↙
spot_final(end_frame,:) and the following functions will fail).
410     spot_final(start_frame,n).CentreX = [];
411     spot_final(start_frame,n).CentreY = [];
412     spot_final(start_frame,n).IspTot = [];
413     spot_final(start_frame,n).rsqFit = [];
414     spot_final(start_frame,n).SigmaFit = [];
415     spot_final(start_frame,n).I0Fit = [];
416     spot_final(start_frame,n).bg_noise_offset_afterBGsubtract = [];
417     spot_final(start_frame,n).BgNoiseStd = [];
418     spot_final(start_frame,n).IbgAvg = [];
419     spot_final(start_frame,n).IbgTot = [];
420     spot_final(start_frame,n).SNR = [];
421     spot_final(start_frame,n).IinnerTot = [];
422     spot_final(start_frame,n).ClipFlag = [];
423     spot_final(start_frame,n).noConverge = [];
424     spot_final(start_frame,n).TrajNumber = [];
425     spot_final(start_frame,n).FrameNumber = [];
426     spot_final(start_frame,n).SpotNumber = [];
427 else
428     found_spot_CentreX = [spot_final(start_frame,:).CentreX]'; % column vector with found CentreX positions of all candidate↙
spots.
429     found_spot_CentreY = [spot_final(start_frame,:).CentreY]'; % column vector with found CentreY positions of all candidate↙
spots.
430 end
431
432 % Check graphically:
```

```
433 figure;
434 imshow(frame_Gray,[]);
435 hold on;
436 plot(found_spot_CentreX,found_spot_CentreY,'o','Color','g','MarkerSize',10) % plot accepted spot centres in green.
437 pause(0.1); % this pause is needed to give time for the plot to appear
438 hold off;
439 % ---------------------------------------------------------------------
440
441
442 %% Loop through selected frames:
443
444 tr =1; % initialise trajectory index.
445
446 for k = (start_frame+1):end_frame
447     % to go through all frames do instead: for k = 1:length(sifData)
448
449     frame = image_data(k).frame_data; % extract frame data which is stored in field 'frame_data'.
450     frame = double(frame);
451
452     imshow(frame,[],'Border','tight','InitialMagnification',150); % show image scaled between its min and max values ([]).
453     hold on;
454
455     disp(['frame number: ',num2str(k)]) % print frame number to Command Window.
456
457     %-------------------------------------
458     % Find new candidate spots for this frame:
459     frame_Gray = mat2gray(frame); % The input to function "findCandidateSpots" needs to be a grayscale image:
460
461     [candidate_spotsX_00 candidate_spotsY_00] = findCandidateSpots(frame_Gray,2); % Second input: use method 2, which seems↙
to work better.
462     % the subindex "_00" in candidate_spotsX_00 indicates newly found spot
463     % candidates for the current frame. On the other hand, found_spot_CentreX and found_spot_CentreY are the
464     % accepted spot-centre positions coming from the previous frame.
465
466     % Reject candidate spots outside cell region (SignalMask) (to speed up algorithm):
467     candidate_spotsX_0 = []; % initialise empty vectors before loop.
468     candidate_spotsY_0 = [];
```

```
469     for nn = 1:length(candidate_spotsX_00)
470         % Only use candidates for which there is a 1 in the SignalMask image:
471         if SignalMask(candidate_spotsY_00(nn),candidate_spotsX_00(nn))==1
472             candidate_spotsX_0 = [candidate_spotsX_0; candidate_spotsX_00(nn)];
473             candidate_spotsY_0 = [candidate_spotsY_0; candidate_spotsY_00(nn)];
474         end
475     end
476
477     disp(['no. of new candidate spots: ',num2str(length(candidate_spotsX_0))])
478     %-------------------------------------
479
480     % Join accepted spot-centre positions from previous frame with
481     % candidate spots for this frame to use them as new candidates for this frame:
482     candidate_spotsX = [found_spot_CentreX; candidate_spotsX_0];
483     candidate_spotsY = [found_spot_CentreY; candidate_spotsY_0];
484
485 %         % Plot new candidate spots in yellow and found spot centres from
486 %         % previous frame in cyan
487 %          plot(candidate_spotsX_0,candidate_spotsY_0,'+','Color','y','MarkerSize',3);
488 %          pause(0.5);
489 %          plot(found_spot_CentreX,found_spot_CentreY,'+','Color','c','MarkerSize',3);
490 %          pause(0.5);
491
492     disp(['no. of initial total candidate spots: ',num2str(length(candidate_spotsX))])
493
494     % Error control:
495     % Limit the max number of candidate spots (if eg. 260000 candidate spots are
496     % found, we get an error in function pdist: "Distance matrix has more
497     % elements than the maximum allowed size in MATLAB").
498     % Select only the first max_num_candidates then.
499     if length(candidate_spotsX) > max_num_candidates
500         candidate_spotsX = candidate_spotsX(1:max_num_candidates);
501         candidate_spotsY = candidate_spotsY(1:max_num_candidates);
502         disp(['NOTE!! no. of initial total candidate spots has been limited to ',num2str(max_num_candidates)])
503     end
504
505     %-------------------------------------------
```

```
506     % Eliminate coincidences in spot candidates:
507
508     [candidate_spotsX candidate_spotsY pos] = eliminateCoincidentSpots(candidate_spotsX,candidate_spotsY,d_coincid_cand);
509     % see C:\Isabel\myMatlabFiles\eliminateCoincidentSpots.m
510     % The function checks the distances between all pairs of points with x
511     % and y coordinates candidate_spotsX and candidate_spotsY respectively,
512     % and removes those points (x,y) which are closer than one pixel (distance<1) to
513     % another point in the list.
514
515     %     % for debugging:
516     %     x2 = candidate_spotsX;
517     %     y2 = candidate_spotsY;
518
519     disp(['no. of total candidate spots after eliminating coincidences: ',num2str(length(candidate_spotsX))])
520
521     %     % Plot all candidate spots in magenta after removing coincidences:
522     %      plot(candidate_spotsX,candidate_spotsY,'+','Color','m','MarkerSize',3);
523     %      pause(0.5);
524     %-------------------------------------------------
525
526
527     n =1; % Initialise index n (index for accepted spot centres which have a clipping flag equal to zero):
528     frame_to_search = frame; % Initialise frame to search for spot centres.
529
530     for m = 1:size(candidate_spotsX,1) % for each frame, loop throuh all the candidate spots.
531         % Now find centre of bright spot using function findSpotCentre1frame:
532         % use candidate spots as initial estimates and then iterate to find spot centre.
533         % Image subarray ROI is a square of size 17x17 pixels (halfwidth is
534         % 8 pixels), inner circular mask that moves inside the fixed 17x17
535         % square has a radius of 5 pixels and the applied Gaussian
536         % mask has a sigma of 2 pixels:
537         spot_result = findSpotCentre1frame(frame_to_search,candidate_spotsX(m),candidate_spotsY(m),subarray_halfwidth,↙
inner_circle_radius,gauss_mask_sigma,guess_sigma_Fit);
538         % index k is for frame number, index m is for spot number
539         spot_result.FrameNumber = k; % Add new field containing frame number (time) to result structure.
540
541         % accepted spot centres:
```

```
542         if (spot_result.ClipFlag == 0 && spot_result.noConverge == 0 && ...
543                 spot_result.SigmaFit <= sigmaFit_max && ...
544                 spot_result.SigmaFit >= sigmaFit_min && ...
545                 spot_result.SNR >= SNR_min &&...
546                 spot_result.rsqFit >= rsq_min)&&...
547                 (exclude_region == 0 || (exclude_region ==1 && (spot_result.CentreY <(frame_Ysize/2-exclude_region_width) ||↙
spot_result.CentreY >(frame_Ysize/2+exclude_region_width))))
548             % Only accept and save result of found spot if clipping flag =0 and if values of sigmaFit, signal to noise and↙
rsquare of fit are acceptable.
549             spot(k,n) = spot_result; % store accepted found spots in this preliminary result.
550             % first index is for frame number, second index is for spot number.
551             %           %--------------------------------------------------
552             %           plot(spot(k,n).CentreX,spot(k,n).CentreY,'o','Color','r','MarkerSize',10); % Plot found centre spots↙
in red
553             %           %--------------------------------------------------
554             if deflate==1 % see parameter section at the beginning.
555                 % "Deflation" process: subtract from raw frame image the corresponding
556                 % Gaussian fit of each found and accepted spot before finding
557                 % next spot centre (enables acceptance of dimmer spots).
558                 % Xpos and Ypos are matrices of x and y positions on image frame.
559                 % Parameters of Gaussian fit of previously accepted spot:
560                 x_fit = spot(k,n).CentreX;
561                 y_fit = spot(k,n).CentreY;
562                 I_fit = spot(k,n).I0Fit;
563                 sigma_fit = spot(k,n).SigmaFit;
564                 % deflated frame (frame with found spot subtracted):
565                 deflated_frame = frame_to_search - I_fit*exp(-((Xpos-x_fit).^2+(Ypos-y_fit).^2)/(2*sigma_fit^2));
566                 frame_to_search = deflated_frame; % update frame for finding next spot-centre.
567                 % % Graphical check of deflated frames:
568                 % subplot(1,2,1); imshow(frame,[]); % frame is the original frame (always the same).
569                 % subplot(1,2,2); imshow(deflated_frame,[]);
570             end
571             %------------------------------
572
573             n = n+1; % and advance index n for accepted spot centres.
574         end
575
```

```
576     end
577
578     % display the number of accepted spot-centres for each frame:
579     disp(['no. of accepted spot centres: ',num2str(n-1)])
580
581     % % The following two lines are used together with the previous two
582     % "plot" and "imshow" (commented off) lines:
583     pause(0.1); % this pause is needed to give time for the plot to appear
584     %    hold off;
585
586     % convert results of found spot-centre positions to a useful form that can
587     % be used as input candidate-spots on the following frame:
588     if (n-1) == 0 % error control: if no spots were accepted.
589         found_spot_CentreX = [];
590         found_spot_CentreY = [];
591         spot_final(k,n).SpotNumber = []; % Save empty spot (we need this, otherwise if in the last frame the no. of accepted↙
spots is 0, there will be no result spot_final(end_frame,:) and the following functions will fail).
592     else
593         found_spot_CentreX = [spot(k,:).CentreX]'; % column vector with found CentreX positions of all candidate spots.
594         found_spot_CentreY = [spot(k,:).CentreY]'; % column vector with found CentreY positions of all candidate spots.
595
596         %-------------------------------
597         % Eliminate coincidences in result of last found spots for a given frame (for distance <1):
598         [found_spot_CentreX found_spot_CentreY pos_final] = eliminateCoincidentSpots(found_spot_CentreX,found_spot_CentreY,↙
d_coincid_found);
599         % see C:\Isabel\myMatlabFiles\eliminateCoincidentSpots.m
600         % pos_final contains positions of selected, kept spot centres.
601         %-------------------------------
602
603         % Save final spots to variable final_spots:
604         n=1; % index for final kept spot.
605         for ii = 1:length(pos_final)
606             mientras = spot(k,pos_final(ii)); % intermediate result.
607             mientras.SpotNumber = n; % Add new field containing spot number to result structure.
608             spot_final(k,n)=mientras; % final result structure of accepted spot centres.
609             n = n+1;
610         end
```

```
611         % Plot found spot centres:
612         pause(0.5);
613         plot(found_spot_CentreX,found_spot_CentreY,'o','Color','g','MarkerSize',10) % plot final accepted spot centres in↙
green.
614         pause(0.1); % this pause is needed to give time for the plot to appear
615         hold off;
616
617         disp(['no. of final found spot centres after eliminating coincidences: ',num2str(length(found_spot_CentreX))])
618     end
619
620
621
622     %--------------------
623     % LINKING SPOTS INTO TRAJECTORY SEGMENTS:
624
625     % Link found and accepted spots into trajectory segments:
626
627     % Trajectory index tr is initialised to 1 outside the loop through frames (k loop).
628
629     % Do differently FOR SECOND FRAME (k == start_frame+1): compare only accepted spots in
630     % previous and current frames:
631     if k == start_frame+1 && ... % If second frame and
632             (n-1)~=0 && ... % if the number of accepted spot centres is not zero and
633             isempty([spot_final(k-1,:).SpotNumber])==0 && ... % at least 1 accepted spot in previous frame and
634             isempty([spot_final(k,:).SpotNumber])==0 % at least 1 accepted spot in current frame.
635         % There are no trajectories jet, so compare accepted spots in previous and current frames:
636         N0 = max(cat(1,spot_final(k-1,:).SpotNumber));  % no. of accepted spots in previous frame.
637         % Note: cat(1,spot_final(k-1,:).SpotNumber) gives a column vector with the values of SpotNumber for all non-empty↙
accepted spots in frame k-1.
638         N1 = max(cat(1,spot_final(k,:).SpotNumber));  % no. of accepted spots in current frame.
639
640         % Create cell arrays with empty elements to pre-asign sizes:
641         d01 = cell(N0,N1); % Note: d01 is a cell array (matrix) but d_01 below is a scalar.
642         Iratio01 = cell(N0,N1); % Note: Iratio01 is a cell array (matrix) but Iratio_01 below is a scalar.
643         SigmaRatio01 = cell(N0,N1); % Note: SigmaRatio01 is a cell array (matrix) but SigmaRatio_01 below is a scalar.
644
645         for q0 = 1:N0 % loop though accepted spots in previous frame.
```

```
646             for q1 = 1:N1 % loop though accepted spots in current frame.
647                 % d_01: distance between spot centres in previous and current frames:
648                 d_01 = sqrt((spot_final(k-1,q0).CentreX-spot_final(k,q1).CentreX)^2+(spot_final(k-1,q0).CentreY-spot_final(k,↙
q1).CentreY)^2);
649                 % Iratio_01: ratio of intensities of spot centre in previous and current frames:
650                 Iratio_01 = spot_final(k-1,q0).IspTot/spot_final(k,q1).IspTot;
651                 % SigmaRatio_01: ratio of widths of spots (Gaussian fits) in previous and current frames:
652                 SigmaRatio_01 = spot_final(k-1,q0).SigmaFit/spot_final(k,q1).SigmaFit;
653
654                 %                 d_01
655                 %                 Iratio_01
656                 %                 SigmaRatio_01
657
658                 % Accept and save trajectory if spots in previous and
659                 % current frames fulfill the following conditions:
660                 if d_01 < d_01_max && ...  % see PARAMETERS at start of this function.
661                         Iratio_01_min <= Iratio_01 && Iratio_01 <= Iratio_01_max && ...
662                         SigmaRatio_01_min <= SigmaRatio_01 && SigmaRatio_01 <= SigmaRatio_01_max
663                     % Asign accepted values to cell array elements to store them:
664                     d01{q0,q1} = d_01; % use {} for cell arrays.
665                     Iratio01{q0,q1} = Iratio_01;
666                     SigmaRatio01{q0,q1} = SigmaRatio_01;
667                 else % rejected asignments:
668                     d01{q0,q1} = rej; % Use rej for asignments not accepted (images usually 512x512arrays, so rej pix is an↙
impossibly large distance, this is why it is chosen here).
669                     Iratio01{q0,q1} = rej; % Use rej for asignments not accepted.
670                     SigmaRatio01{q0,q1} = rej; % Use rej for rejected asignments.
671                 end
672             end
673
674 %                       d01
675 %                       [d01{q0,:}]
676             % Note that [d01{q0,:}] gives only non-empty elements of row q0
677             % in the cell array d01 as a row vector, that's why we had to
678             % give a numeric value rej to non-accepted asignments.
679
680             % Note that if all asignments in previous step are rejected,
```

```
681             % [d01{q0,:}] will be a list of rej values, and its minimum will
682             % be rej.
683             % If list of "linkable" spots, [d01{q0,:}], has no accepted
684             % asignments (all values are rej):
685             if min([d01{q0,:}]) == rej
686                 % Asign trajectory number 0 to the spot in the previous frame only:
687                 spot_final(k-1,q0).TrajNumber = 0;
688             else % if there is at least one accepted asignment for a given spot in the previous frame:
689
690                 % Decide of all possible accepted spots (in current frame)
691                 % that could be linked to spot q0 in previous frame, which one is the best:
692                 % We take the best as the closest one to spot q0:
693                 q1_chosen = find([d01{q0,:}] == min([d01{q0,:}])); % find position of the minimum pair-wise distance.
694
695 %                               q1_chosen
696
697                 % Check if there is a better competing asignment for a given spot q1 in the current
698                 % frame from another spot in the previous frame.
699                 % Hence, check also column-wise in matrix d01 to avoid asigning a traj
700                 % number to a spot q1 in the current frame that had already
701                 % had a traj number asigned to it linking it to a different spot q0 in
702                 % the previous frame, which might be at a shorter distance
703                 % from it than the current one.
704
705 %                               [d01{:,q1_chosen}] % chosen column of d01 matrix of distances.
706
707                 % Asign trajectory numbers to structure spot_final:
708                 % If the found distance in that column is not the minimum one:
709                 if q0 ~= find([d01{:,q1_chosen}] == min([d01{:,q1_chosen}]));
710                     spot_final(k-1,q0).TrajNumber = 0; % asign trajectory number 0 to spot in previous frame.
711                 else
712                     spot_final(k-1,q0).TrajNumber = tr; % asign trajectory number to spot in previous frame, to spot_final↙
structure.
713                     spot_final(k,q1_chosen).TrajNumber = tr; % asign same trajectory number to spot in current frame.
714                     tr = tr+1; % advance trajectory-number index.
715                 end
716             end
```

```
717         end
718
719
720
721     else % for FRAMES k >= start_frame+2, from third chosen frame on:
722
723         % A) Compare loose spots (TrajNumber is 0) two frames ago (k-2)
724         % to found spots in current frame (TrajNumber is []):
725         % XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
726         % DO maybe!!
727         if k ~= start_frame+1 && (n-1)~=0 && ... % If the number of accepted spot centres is not zero and
728                 isempty([spot_final(k-2,:).SpotNumber])==0 && ... % at least 1 accepted spot 2 frames ago and
729                 isempty([spot_final(k,:).SpotNumber])==0 % at least 1 accepted spot in current frame.
730
731             N0 = max(cat(1,spot_final(k-2,:).SpotNumber));  % no. of accepted spots 2 frames ago.
732             N1 = max(cat(1,spot_final(k,:).SpotNumber));  % no. of accepted spots in current frame.
733
734             % Create cell arrays with empty elements to pre-asign sizes.
735             d02 = cell(N0,N1); % Note: d02 is a cell array (matrix) but d_02 below is a scalar.
736             Iratio02 = cell(N0,N1); % Note: Iratio02 is a cell array (matrix) but Iratio_02 below is a scalar.
737             SigmaRatio02 = cell(N0,N1); % Note: SigmaRatio02 is a cell array (matrix) but SigmaRatio_02 below is a scalar.
738
739             for q0 = 1:N0 % loop though loose accepted spots 2 frames ago.
740                 if spot_final(k-2,q0).TrajNumber == 0 % only for loose (unlinked) spots (TrajNumber=0) two frames ago (so↙
only rows q0 in matrix d01 which have unlinked spots will fill up).
741                     for q1 = 1:N1 % loop though accepted spots in current frame.
742                         % d_01: distance between spot centres in previous and current frames:
743                         d_02 = sqrt((spot_final(k-2,q0).CentreX-spot_final(k,q1).CentreX)^2+(spot_final(k-2,q0).CentreY-↙
spot_final(k,q1).CentreY)^2);
744                         % Iratio_01: ratio of intensities of spot centre in previous and current frames:
745                         Iratio_02 = spot_final(k-2,q0).IspTot/spot_final(k,q1).IspTot;
746                         % SigmaRatio_01: ratio of widths of spots (Gaussian fits) in previous and current frames:
747                         SigmaRatio_02 = spot_final(k-2,q0).SigmaFit/spot_final(k,q1).SigmaFit;
748
749                         %                           d_02
750                         %                           Iratio_02
751                         %                           SigmaRatio_02
```

```
752
753                         % Accept and save trajectory if spots in previous and
754                         % current frames fulfill the following conditions:
755                         if d_02 < d_02_max && ...  % see PARAMETERS at start of this function.
756                                 Iratio_02_min <= Iratio_02 && Iratio_02 <= Iratio_02_max && ...
757                                 SigmaRatio_02_min <= SigmaRatio_02 && SigmaRatio_02 <= SigmaRatio_02_max
758                             % Asign accepted values to cell array elements to store them:
759                             d02{q0,q1} = d_02; % use {} for cell arrays.
760                             Iratio02{q0,q1} = Iratio_02;
761                             SigmaRatio02{q0,q1} = SigmaRatio_02;
762                         else % rejected asignments:
763                             d02{q0,q1} = rej; % Use rej for asignments not accepted (images usually 512x512arrays, so rej pix↙
is an impossibly large distance, this is why it is chosen here).
764                             Iratio02{q0,q1} = rej; % Use rej for asignments not accepted.
765                             SigmaRatio02{q0,q1} = rej; % Use rej for rejected asignments.
766                         end
767                     end
768
769 %                                       d02
770 %                                       [d02{q0,:}]
771                     % Note that [d02{q0,:}] gives only non-empty elements of the cell array d01 as a row vector.
772
773                     % Note that if all asignments in previous step are rejected,
774                     % [d02{q0,:}] will be a list of rej values, and its minimum will be rej.
775                     % If list of "linkable" spots, [d02{q0,:}], has no accepted asignments (all values are rej):
776                     if min([d02{q0,:}]) == rej
777                         % Asign trajectory number 0 to the spot two frames ago only:
778                         spot_final(k-2,q0).TrajNumber = 0; % point stays loose (unlinked).
779
780                     else % if there is at least one accepted asignment for a given spot two frames ago:
781
782                         % Decide of all possible accepted/saved spots (in current frame)
783                         % that could be linked to spot q0 in previous frame (of all possible asignments), which one is the↙
best:
784                         % We take the best as the closest one to spot q0:
785                         q1_chosen = find([d02{q0,:}] == min([d02{q0,:}])); % find position of the minimum pair-wise distance.
786
```

```
787 %                                           q1_chosen
788
789                         % Check if there is a better competing asignment for a given spot q1 in the current
790                         % frame from another spot q0 two frames ago.
791                         % Hence, check also column-wise in d01 to avoid asigning a traj
792                         % number to a spot q1 in the current frame that had already
793                         % had a traj number asigned to it linking it to a different spot q0 two frames ago which might be at↙
a shorter distance
794                         % from it than the current one.
795
796 %                                           [d02{:,q1_chosen}] % chosen column of d01 matrix of distances.
797
798                         % Asign trajectory numbers to structure spot_final:
799                         % If the found distance in that column is not the minimum one:
800                         if q0 ~= find([d02{:,q1_chosen}] == min([d02{:,q1_chosen}]));
801                             % Asign trajectory number 0 to the spot two frames ago only:
802                             spot_final(k-2,q0).TrajNumber = 0; % point stays loose (unlinked).
803                         else
804                             spot_final(k-2,q0).TrajNumber = tr; % asign trajectory number to spot two frames ago, to↙
spot_final structure.
805                             spot_final(k,q1_chosen).TrajNumber = tr; % asign same trajectory number to spot in current frame.
806                             tr = tr+1; % advance trajectory-number index.
807                         end
808                     end
809                 end
810             end
811         end
812
813
814         % B) Compare loose spots (TrajNumber is []) and trajectories (TrajNumber is >0)
815         % in previous frame (k-1) to found spots in current frame:
816         if (n-1)~=0 && ... % If the number of accepted spot centres is not zero and
817                 isempty([spot_final(k-1,:).SpotNumber])==0 && ... % at least 1 accepted spot in previous frame.
818                 isempty([spot_final(k,:).SpotNumber])==0 % at least 1 accepted spot in current frame.
819
820             % zzzzzzzzzzzzzzzzzzzzz
821             N0 = max(cat(1,spot_final(k-1,:).SpotNumber));  % no. of accepted spots in previous frame.
```

```
822             N1 = max(cat(1,spot_final(k,:).SpotNumber));  % no. of accepted spots in current frame.
823
824             % Create cell arrays with empty elements to pre-asign sizes.
825             d01 = cell(N0,N1);
826             Iratio01 = cell(N0,N1);
827             SigmaRatio01 = cell(N0,N1);
828
829             for q0 = 1:N0 % loop though accepted spots in previous frame.
830                 for q1 = 1:N1 % loop though accepted spots in current frame.
831                     % d_01: distance between spot centres in previous and current frames:
832                     d_01 = sqrt((spot_final(k-1,q0).CentreX-spot_final(k,q1).CentreX)^2+(spot_final(k-1,q0).CentreY-↙
spot_final(k,q1).CentreY)^2);
833                     % Iratio_01: ratio of intensities of spot centre in previous and current frames:
834                     Iratio_01 = spot_final(k-1,q0).IspTot/spot_final(k,q1).IspTot;
835                     % SigmaRatio_01: ratio of widths of spots (Gaussian fits) in previous and current frames:
836                     SigmaRatio_01 = spot_final(k-1,q0).SigmaFit/spot_final(k,q1).SigmaFit;
837
838                     %                       d_01
839                     %                       Iratio_01
840                     %                       SigmaRatio_01
841
842                     % Accept and save trajectory if spots in previous and
843                     % current frames fulfill the following conditions:
844                     if d_01 < d_01_max && ...  % see PARAMETERS at start of this function.
845                             Iratio_01_min <= Iratio_01 && Iratio_01 <= Iratio_01_max && ...
846                             SigmaRatio_01_min <= SigmaRatio_01 && SigmaRatio_01 <= SigmaRatio_01_max
847                         % Asign accepted values to cell array elements to store them:
848                         d01{q0,q1} = d_01; % use {} for cell arrays.
849                         Iratio01{q0,q1} = Iratio_01;
850                         SigmaRatio01{q0,q1} = SigmaRatio_01;
851                     else % rejected asignments:
852                         d01{q0,q1} = rej; % Use rej for asignments not accepted (images usually 512x512arrays, so rej pix is↙
an impossibly large distance, this is why it is chosen here).
853                         Iratio01{q0,q1} = rej; % Use rej for asignments not accepted.
854                         SigmaRatio01{q0,q1} = rej; % Use rej for rejected asignments.
855                     end
856                 end
```

```
857
858 %                                   d01
859 %                                   [d01{q0,:}] % last row of d01 matrix of distances.
860                 % Note that [d01{q0,:}] gives only non-empty elements
861                 % of row q0 in the cell array d01, as a row vector.
862
863                 % Note that if all asignments in previous step are rejected,
864                 % [d01{q0,:}] will be a list of rej values, and its minimum will
865                 % be rej.
866                 % If list of "linkable" spots, [d01{q0,:}], has no accepted
867                 % asignments (all values are rej):
868                 if min([d01{q0,:}]) == rej
869
870                     if isempty(spot_final(k-1,q0).TrajNumber) % if point in previous frame was not part of a trajectory↙
(TrajNumber=[]):
871                         % Asign trajectory number 0 to the spot in the previous frame only:
872                         spot_final(k-1,q0).TrajNumber = 0;
873                     end
874
875                 else % if there is at least one accepted asignment for a given spot two frames ago:
876
877                     % Decide of all possible accepted/saved spots (in current frame)
878                     % that could be linked to spot q0 in previous frame, which one is the best:
879                     % We take the best as the closest one to spot q0:
880                     q1_chosen = find([d01{q0,:}] == min([d01{q0,:}])); % find position of the minimum pair-wise distance.
881
882 %                                       q1_chosen
883
884                     % Check if there is a better competing asignment for a given spot q1 in the current
885                     % frame from another spot q0 in the previous frame.
886                     % Hence, check also column-wise in d01 to avoid asigning a traj
887                     % number to a spot q1 in the current frame that had already
888                     % had a traj number asigned to it linking it to a different spot q0 in
889                     % the previous frame which might be at a shorter distance
890                     % from it than the current one.
891
892 %                                       [d01{:,q1_chosen}]  % chosen column of d01 matrix of distances.
```

```
893
894                     % If the found distance in that column is not the minimum one:
895                     if q0 ~= find([d01{:,q1_chosen}] == min([d01{:,q1_chosen}]));
896                         if isempty(spot_final(k-1,q0).TrajNumber) % if point in previous frame was not part of a trajectory
(TrajNumber=[]):
897                             % Asign trajectory number 0 to the spot in the previous frame only:
898                             spot_final(k-1,q0).TrajNumber = 0;
899                         end
900                     else
901                         % Asign trajectory numbers to structure spot_final:
902                         if spot_final(k-1,q0).TrajNumber > 0 % if point in previous frame was already part of a trajectory:
903                             spot_final(k,q1_chosen).TrajNumber = spot_final(k-1,q0).TrajNumber; % asign that trajectory
number to spot in current frame.
904                         else % if point in previous frame was not part of a trajectory:
905                             spot_final(k-1,q0).TrajNumber = tr; % asign new trajectory number to spot in previous frame.
906                             spot_final(k,q1_chosen).TrajNumber = tr; % asign same trajectory number to spot in current frame.
907                             tr = tr+1; % advance trajectory-number index.
908                         end
909                     end
910                 end
911             end
912             % zzzzzzzzzzzzzzzzzzzzz
913         end
914     end
915     %-----------------
916
917
918 end  % loop through selected frames
919
920
921
922 %% OUTPUT OF SPOT-FINDING PROCESS: final output spot_results:
923 %
924 spot_results = {params, spot_final};
925 % params is a structure array containing all parameters used to run the
926 % function.
927 % spot_final, is a structure array with end_frame x L elements,
```

```
928 % each of which is a structure with fields:
929 %     'CentreX'
930 %     'CentreY'
931 %     'IspTot'
932 %     'rsqFit'
933 %     'SigmaFit'
934 %     'I0Fit'
935 %     'BgNoiseStd'
936 %     'IbgAvg'
937 %     'IbgTot'
938 %     'SNR'
939 %     'IinnerTot'
940 %     'ClipFlag'
941 %     'TrajNumber'
942 %     'FrameNumber'
943 %     'SpotNumber'
944 %
945 % Along the dimension L, we have the different bright spots found on each
946 % frame (L will often be the number of spot centres found in frame_start,
947 % it is always the largest number of spots ever accepted on one frame).
948 % For example, spot_final(100,8) is a structure with the above fields
949 % corresponding to the eighth found spot on frame 100.
950 %
951 % Note that even if we only analyse from start_frame to end_frame,
952 % spot_final is a list containing empty structure arrays from index 1 to
953 % index start_frame, and then the found spots for the analysed frames start_frame to end_frame.
954 %
955 % The result is padded to a fixed number of spot structures for reach
956 % frame (the maximum no. of accepted found spots of all frames), so that for a given
957 % frame in which less found spots have been accepted, the remaining
958 % elements are padded with empty structures with empty fields [].
959 % To check if a given spot is empty: isempty(spot_final(101,10).CentreX)
960 % gives 1 if field "CentreX" of tenth spot found and accepted in frame 101
961 % is empty (equal to []).
962 %
963 % e.g. cat(1,spot_final(100,:).SigmaFit) gives a vector column with all the
964 % non-empty SigmaFit values for all spot centres accepted in frame 100.
```

965