```
clc
clear
close all
PSR_THRESHOLD = 5;
NUM_HANKEL_FRAMES = 13; % must be > 2
PREV_FRAME_LOOKBACKS = 5;

%default color for visualization
edgeColor = 'g';

%  Exploiting the Circulant Structure of Tracking-by-detection with
 Kernels
%
%  Main script for tracking, with a gaussian kernel.
%
%  João F. Henriques, 2012
%  http://www.isr.uc.pt/~henriques/


%choose the path to the videos (you'll be able to choose one with the
 GUI)
base_path = './data/';


%parameters according to the paper
padding = 1;     %extra area surrounding the target
output_sigma_factor = 1/16;  %spatial bandwidth (proportional to
 target)
sigma = 0.2;     %gaussian kernel bandwidth
lambda = 1e-2;     %regularization
interp_factor = 0.075;   %linear interpolation factor for adaptation



%notation: variables ending with f are in the frequency domain.

%ask the user for the video
[video_path, video_name] = choose_video(base_path);
MIL = extractMIL(video_path);
if isempty(video_path), return, end  %user cancelled
[img_files, pos, target_sz, resize_image, ground_truth, video_path]
 = ...
 load_video_info(video_path);

pos2 = pos;
saveFrame = 0;
%window size, taking padding into account
sz = floor(target_sz * (1 + padding));

%desired output (gaussian shaped), bandwidth proportional to target
 size
output_sigma = sqrt(prod(target_sz)) * output_sigma_factor;
```

```
[rs, cs] = ndgrid((1:sz(1)) - floor(sz(1)/2), (1:sz(2)) -
 floor(sz(2)/2));
y = exp(-0.5 / output_sigma^2 * (rs.^2 + cs.^2));
yf = fft2(y);

%store pre-computed cosine window
cos_window = hann(sz(1)) * hann(sz(2))';


time = 0;  %to calculate FPS
positions = zeros(numel(img_files), 2);  %to calculate precision

for frame = 1:numel(img_files),
 %load image
 im = imread([video_path img_files{frame}]);
 if size(im,3) > 1,
  im = rgb2gray(im);
 end
 if resize_image,
  im = imrepsize(im, 0.5);
 end

 tic()

 %extract and pre-process subwindow
 x = get_subwindow(im, pos, sz, cos_window);

 if frame > 1,
  %calculate response of the classifier at all locations
  k = dense_gauss_kernel(sigma, x, z);
  response = real(ifft2(alphaf .* fft2(k)));   %(Eq. 9)

  %target location is at the maximum response
  [row, col] = find(response == max(response(:)), 1);
        psrValue = PSR(response, row, col);
        pos2 = pos2 - floor(sz/2) + [row, col];
        if (psrValue < PSR_THRESHOLD)
            disp(['Occlusion detected at frame: ' num2str(frame)])
            saveFrame = 1;
            edgeColor = 'r';

            frameRange = frame-NUM_HANKEL_FRAMES-
PREV_FRAME_LOOKBACKS:frame-1-PREV_FRAME_LOOKBACKS;
            [posX, posY] = predictLocation(positions(frameRange,1),
 positions(frameRange,2), PREV_FRAME_LOOKBACKS);

            % we update the current position based on the predicted
 value
            pos = [posX, posY];

            outputFrames(frame,:,:,:) = 0;
        elseif (psrValue > PSR_THRESHOLD) && (edgeColor == 'r')
            disp(['Object recovered at frame: ' num2str(frame)])
            saveFrame = 1;
```

```
            edgeColor = 'b';
            pos = pos - floor(sz/2) + [row, col];
            outputFrames(frame,:,:,:) = [pos([2,1]) -
target_sz([2,1])/2, target_sz([2,1])];
        else
            saveFrame = 0;
            edgeColor = 'g';
            pos = pos - floor(sz/2) + [row, col];
            outputFrames(frame,:,:,:) = [pos([2,1]) -
target_sz([2,1])/2, target_sz([2,1])];
        end
end

%get subwindow at current estimated target position, to train
classifer
x = get_subwindow(im, pos, sz, cos_window);

%Kernel Regularized Least-Squares, calculate alphas (in Fourier
domain)
k = dense_gauss_kernel(sigma, x);
new_alphaf = yf ./ (fft2(k) + lambda);   %(Eq. 7)
new_z = x;

if frame == 1,  %first frame, train with a single image
 alphaf = new_alphaf;
 z = x;
else
 %subsequent frames, interpolate model
 alphaf = (1 - interp_factor) * alphaf + interp_factor * new_alphaf;
 z = (1 - interp_factor) * z + interp_factor * new_z;
end

%save position and calculate FPS
positions(frame,:) = pos;
time = time + toc();

%visualization

rect_position = [pos([2,1]) - target_sz([2,1])/2, target_sz([2,1])];
   rect_position2 = [MIL(frame, [2,1]) - target_sz([2,1])/2,
target_sz([2,1])];
   rect_position3 = [pos2([2,1]) - target_sz([2,1])/2,
target_sz([2,1])];

if frame == 1,  %first frame, create GUI
 figure('NumberTitle','off', 'Name',['Tracker - ' video_path])
 im_handle = imshow(im, 'Border','tight', 'InitialMag',200);
        % Creates 3 rectangles for different filters
        % Red - Modified CM Tracker
        % Blue - MIL Tracker
        % Green - Original CM Tracker
 rect_handle =
rectangle('Position',rect_position, 'EdgeColor','r','LineWidth',3);
```

```
        rect_handle2 =
 rectangle('Position',rect_position2, 'EdgeColor','b','LineWidth',2);
        rect_handle3 =
 rectangle('Position',rect_position3, 'EdgeColor','g','LineWidth',1);
 else
  try  %subsequent frames, update GUI
   set(im_handle, 'CData', im)
            set(rect_handle, 'Position', rect_position)
            set(rect_handle2, 'Position', rect_position2)
            set(rect_handle3, 'Position', rect_position3)
   %set(rect_handle, 'Position', rect_position, 'EdgeColor',
 edgeColor)
  catch  %#ok, user has closed the window
   return
  end
    end

 drawnow
    % Saves frames that detect occlusion or recover from occlusion
    %if saveFrame == 1
    %    saveas(gcf,([video_name '_Frame' num2str(frame) '.png']))
    %end

  %pause(0.05)  %uncomment to run slower
end

dlmwrite([video_name '-OutputFile.txt'], outputFrames)
if resize_image, positions = positions * 2; end

disp(['Frames-per-second: ' num2str(numel(img_files) / time)])

% Shows the precision plot for the MIL tracker
% show_precision(MIL, ground_truth, video_path)

%show the precisions plot
show_precision(positions, ground_truth, video_path)
```

Published with MATLAB® R2017a

```
function [ psrValue ] = PSR(response, maxRow, maxCol)

    boxFilterSize = 3;
    boxRadius = floor(boxFilterSize / 2);

    sizeX = size(response, 1);
    sizeY = size(response, 2);

    valuesOutsidePeak = [];
    x = 1;

    for i = 1:sizeX
        for j = 1:sizeY
            if (i < maxRow - boxRadius || i > maxRow + boxRadius) ...
                || (j < maxCol - boxRadius || j > maxCol + boxRadius)
                    valuesOutsidePeak(x) = response(i, j);
                    x = x + 1;
            end
        end
    end

    meanValue = mean(valuesOutsidePeak);
    standardDev = std(valuesOutsidePeak);

    psrValue = (response(maxRow, maxCol) - meanValue) / standardDev;

end
```

Published with MATLAB® R2017a

```
function [ predX, predY ] = predictLocation( maxResponseX,
 maxResponseY, frameLookbacks )

    numDataPoints = size(maxResponseX, 1);
    dimens = ceil(numDataPoints / 2);

    % Hx is the Hankel matrix for the x response values
    Hx = zeros(dimens, dimens);

    % Hy is the Hankel matrix for the y response values
    Hy = zeros(dimens, dimens);

    for i = 1:dimens
        for j = 1:dimens
            cellNum = i + j - 1;
            Hx(i, j) = maxResponseX(cellNum);
            Hy(i, j) = maxResponseY(cellNum);
        end
    end

    % all but the last column of the H matrix is A
    Ax = Hx(:, 1:dimens-1);
    Ay = Hy(:, 1:dimens-1);

    % the last column of the H matrix is b
    bx = Hx(:, dimens);
    by = Hy(:, dimens);

    vx = Ax \ bx;
    vy = Ay \ by;

    cLength = dimens-1;
    Cx = zeros(1, cLength);
    Cy = zeros(1, cLength);

    % Cx and Cy are used with vx and vy to predict a new X and Y
    for i = 1:cLength
        Cx(1, i) = maxResponseX(numDataPoints - (dimens - i) + 1);
        Cy(1, i) = maxResponseY(numDataPoints - (dimens - i) + 1);
    end

    % This iterates once for each future prediction to be made
    for i = 0:frameLookbacks
        predX = Cx * vx;
        predY = Cy * vy;

        % We want to shift the Cx and Cy arrays left and add the
        % newly-predicted X and Y's in case we want to predict X and Y
        % again
        Cx = circshift(Cx(:, 1:cLength), [0 cLength-1]);
        Cy = circshift(Cy(:, 1:cLength), [0 cLength-1]);
        Cx(:, cLength) = predX;
```

```
        Cy(:, cLength) = predY;
    end
end
```

Published with MATLAB® R2017a

```
function [MIL] = extractMIL(video_path)

    text_files = dir([video_path '*_MIL_TR001.txt']);
 assert(~isempty(text_files), 'No initial position and ground truth
 (*_gt.txt) to load.')

 f = fopen([video_path text_files(1).name]);
 MIL = textscan(f, '%f,%f,%f,%f');  %[x, y, width, height]
 MIL = cat(2, MIL{:});
 fclose(f);

    MIL = MIL(:,[2,1]) + MIL(:,[4,3]) / 2;

end
```

Published with MATLAB® R2017a