Dixon Jin and Emerson Boyd
EECE 5639
12/12/17

Computer Vision Project 4 - Target Tracking

Abstract

The purpose of this experiment was to detect obfuscation of an object being tracked in a video and to attempt to recover from the loss of sight by predicting where the object would appear in the following frame. This project extended upon the work performed by João F. Henriques et al. in their project “Kernelized Correlation Filters”. The first step was to detect when the object being tracked was obfuscated, which was done using by calculating a “peak to sidelobe ratio” (PSR). Then, when obfuscation was detected, the position of the object was predicted using a Hankel matrix to create a linear combinations of previous x and y object coordinates. If obfuscation for any given frame was not detected, however, the position of the object was calculated as defined by the original authors of the project. To analyze the results of our work, we plotted the accuracy of our methods vs. the accuracy of the original methods.

Description of Algorithms

The majority of the code in this project is based off of the “Kernelized Correlation Filters” project developed by João F. Henriques et al. All of the parameters that define how our algorithms behave are listed at the top of the “run_tracker.m” file. The only files that was altered in the process of developing our algorithms was “run_tracker.m”. Three new files were created, as well: “PSR.m”, “predictLocation.m”, and “extractMIL.m”.

The first algorithm used was designed to predict a PSR value to detect occlusion of the object being tracked. The method takes, as input, the outer response window from locating the object target as well as the maximum response coordinates within that window. An 11x11 window is created surrounding the maximum response, then the mean and standard deviation of the remaining pixels are calculated. The PSR value is determined by taking the pixel value at the max response pixel and subtracting the mean of the pixel values outside the window, then dividing by the standard deviation of the pixel values outside the window. If the PSR value falls below a certain threshold, then we determine that obfuscation has occurred.

The next step in the process is to attempt to recover from occlusion once occlusion has been detected. Recovering from occlusion entails using previous x and y locations of the object to predict future locations. The function “predictLocation” takes, as inputs, the previous x coordinates of the object used to predict the future x coordinate, the previous y coordinates of the object used to predict the future y coordinate, and the number of frames to predict in the

future (more on that later). The Hanel matrix is a circulant matrix that consists of all of the known values passed as inputs to the function. The Hankel matrices (separate for x and y computation) are labeled as “Hx” and “Hy” in the code. The matrices are then split into their respective components “Ax”, “bx”, “Ay” and “by”, where a linear combination of the matrices is used to solve the equations A * v = b. This calculation was performed using the “A \ b” function in matlab to solve for “vx” and “vy”, which are the coefficient vectors used for converting previous x and y coordinates to more recent ones.

After creating Hankel matrices and coefficient vectors based off of the known values, we use those values to predict where the object’s coordinates will be. Often, we want to use data from several frames ago rather than data from the most recent frames in order to predict the object’s location because, if occlusion happens, the data from the most recent frames might be inaccurate. That is why we offer the option to use past data to predict any number of frames in the future. Using the equation “C * v = pred”, where C is previous object coordinates and v is the coefficient vector described above, we can solve for pred, the new predicted object coordinate. Using known object coordinates, we predict a new location of (predX, predY) using our Cx and Cy arrays. Then, depending on how many frames into the future the algorithm is asked to predict, the Cx and Cy arrays are rearranged to drop the object coordinate from longest ago, shift the remaining cells to the left, and place the most recently predX and predY values on the right of the arrays. Then, a new predX and predY are estimated based on the update Cx and Cy arrays. This process is repeated until the algorithm predicts as many x and y coordinates into the future as it is asked to predict. The final x and y predictions are returned upon calculation.

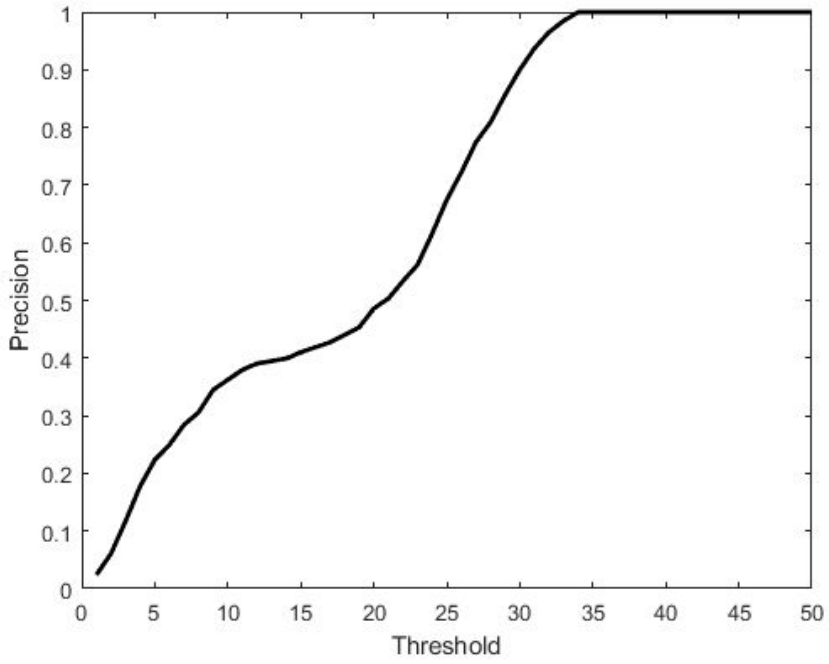
The parameters that dictate how our algorithms run are defined at the top of “run_tracker.m”. “PSR_THRESHOLD” defines the PSR value that the frame must be lower than in order for us to conclude that occlusion has occurred in that frame. “NUM_HANKEL_FRAMES” defines the amount of previous frames that we want to use as input to our Hankel matrix in order to predict future object locations. “PREV_FRAME_LOOKBACKS” indicates the amount of frames into the future where we want to predict the location of the object. For clarification of our algorithm, we present an example where the current frame number is 120, occlusion has been detected, NUM_HANKEL_FRAMES is set to 8, and PREV_FRAME_LOOKBACKS is set to 2. Because we use 8 frames for the Hankel matrix and want to look back 2 extra frames in order to predict the object’s location in frame 120, we create a “frameRange” of 110-117 to create our Hankel matrix. Then, within “predictLocation”, once the location at frame 118 has been predicted using the Hankel matrix, we know we need to predict 2 frames into the future. Thus, we predict the location at frame 119, then finally the location at frame 120.

If the user wants to run the code as the original CM tracker would, without detecting occlusion, the PSR_THRESHOLD can be set to 0. In order to analyze the accuracy of our results, we used the built-in “show_precision” function as defined by the original authors of the project, which shows the accuracy of locating the target within varying pixel ranges around our estimated target.

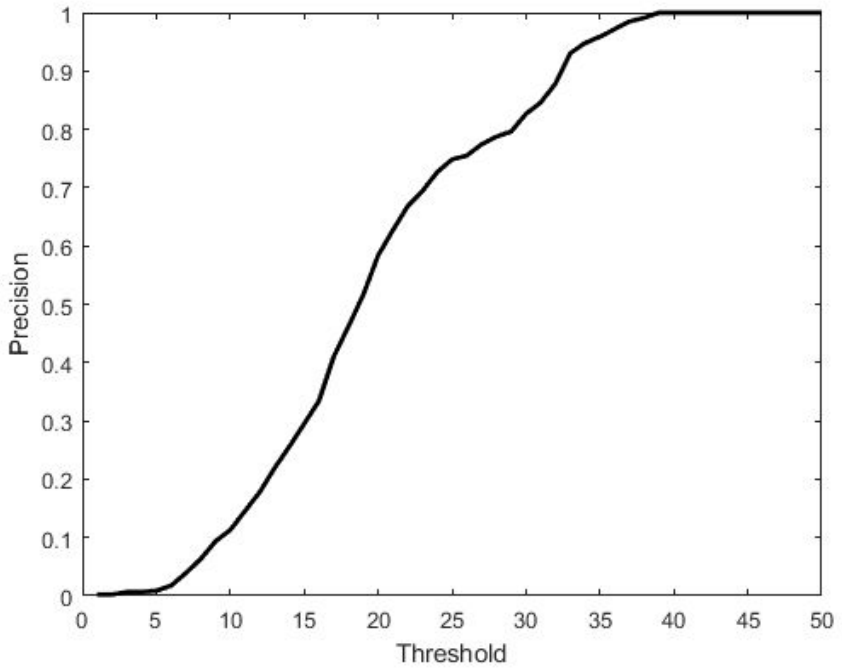
Experiments, Values of Parameters Used, and Observations

These experiments were run for four different groups of images. The first two sets of images had no occlusion, and therefore had approximately the same results for all three different forms of tracking. The second two sets of images had occlusion, and the results of the original CM tracker, MIL tracker, and the modified CM tracker with occlusion recovery all differed.

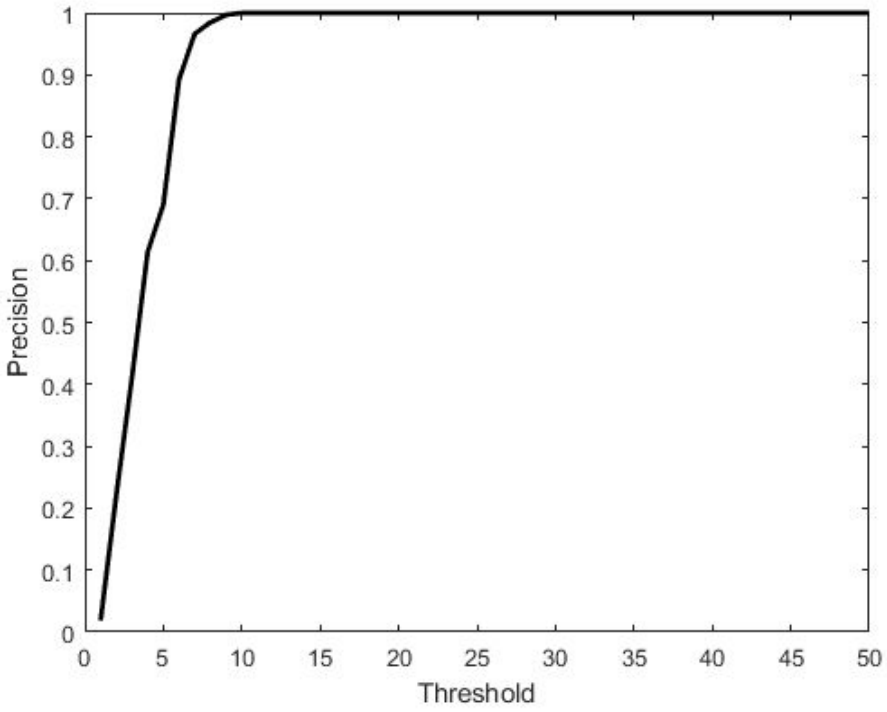
The first set of images was named “david” and tracked the face of a man walking. The second set of images was named “dollar” and tracked the movement of a dollar. The precision of the original CM tracker and the MIL tracker for the sets of images can be seen below. Since there was no occlusion for either set, the precision was the same after modifications to the original CM tracker code.



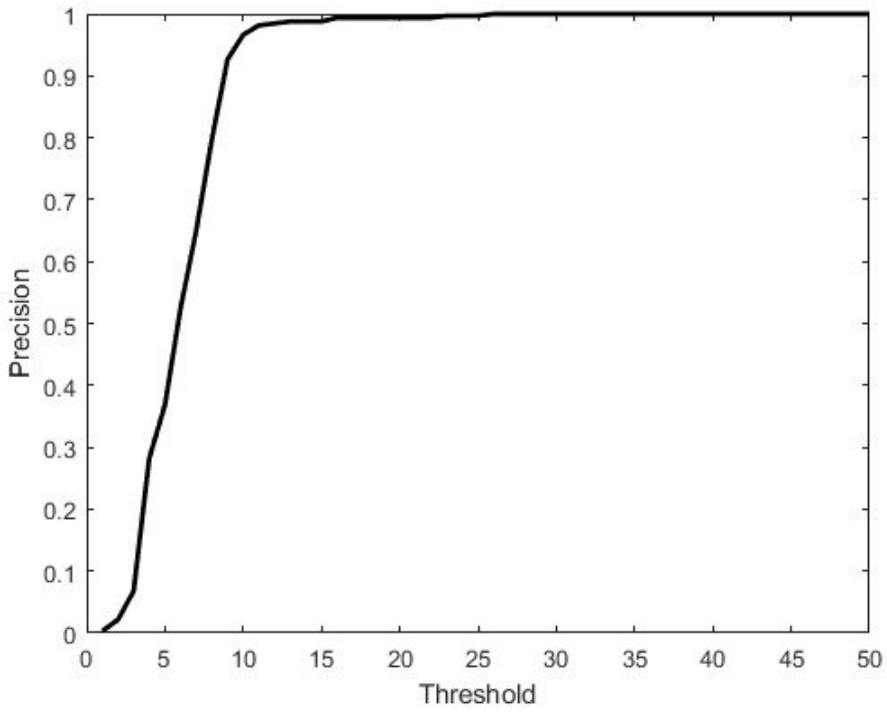
“David” Images - Precision - Original and Modified CM Tracker



“David” Images - Precision - MIL Tracker



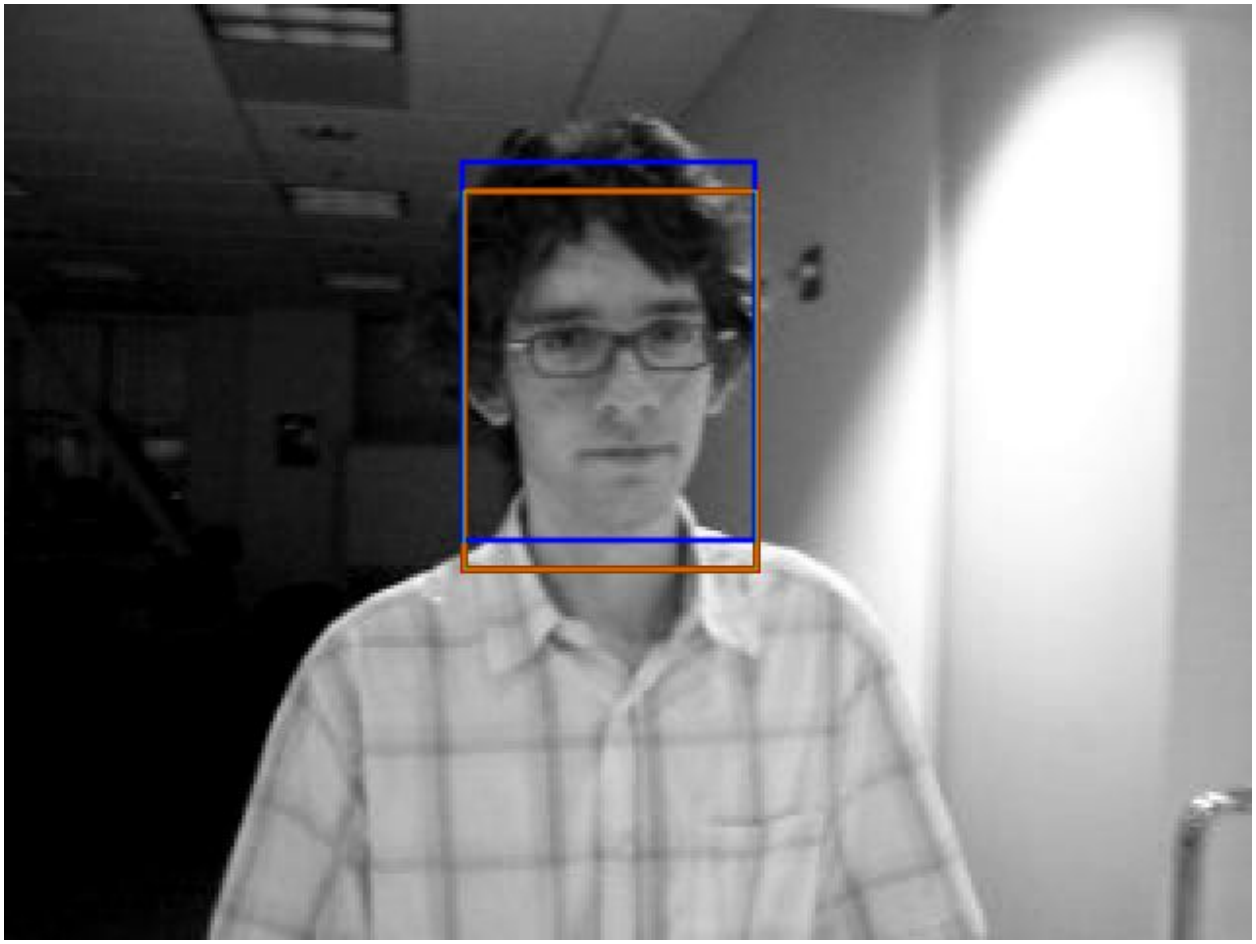
“Dollar” Images - Precision - Original and Modified CM Tracker



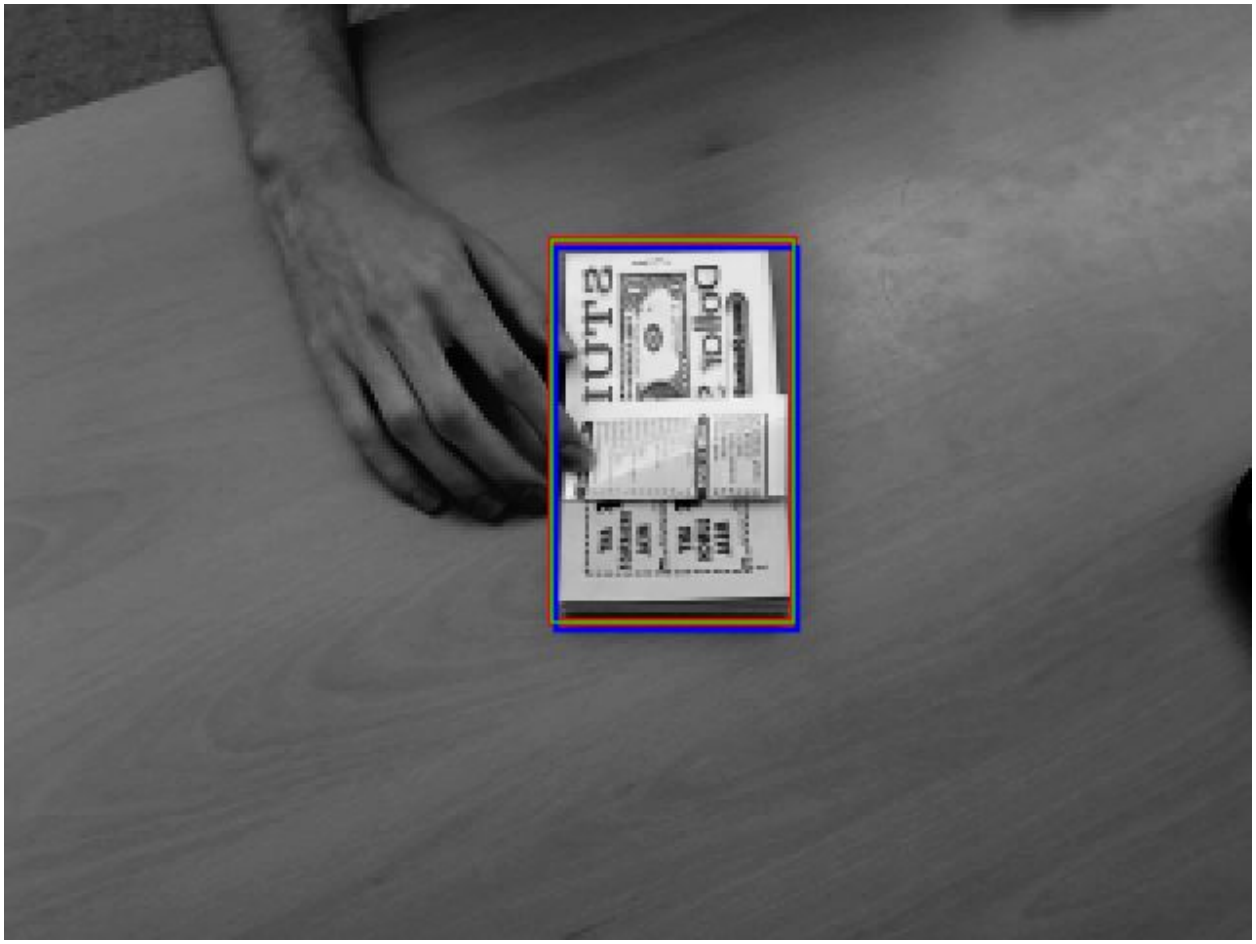
“Dollar” Images - Precision - MIL Tracker

The precision of all three filters were all nearing 1 at a threshold of 50, which means that a majority of the position values calculated by the filters were within the threshold of the ground truth values. The original CM filter actually performed a little better than the MIL tracker because there was no occlusion in the videos. In the second set of videos, occluding the object being tracked will greatly affect the precision of the original and modified CM filter.

Different frames of the videos also had the three different filters in the same area. The green box represents the position that the original CM filter calculated. The blue box represents the position that the MIL filter calculated, and the red box indicates the position that the modified CM filter calculated. In every frame of the two videos, the three boxes overlapped because they all came out to have approximately the same values. Below are some frames of the videos where all three filters lined up.



“David” Images - Frame 80



“Dollar” Images - Frame 80

As seen above, the three filters were all close in the different frames for each of the two videos. This is because neither of the videos had any occlusion to affect the detection of the object.

The second set of videos, named “tiger1” and “tiger2”, had occlusion within them, and the results for both the precision as well as the filter location differed.

When testing the modified CM tracker for “tiger1”, our parameters were set to:

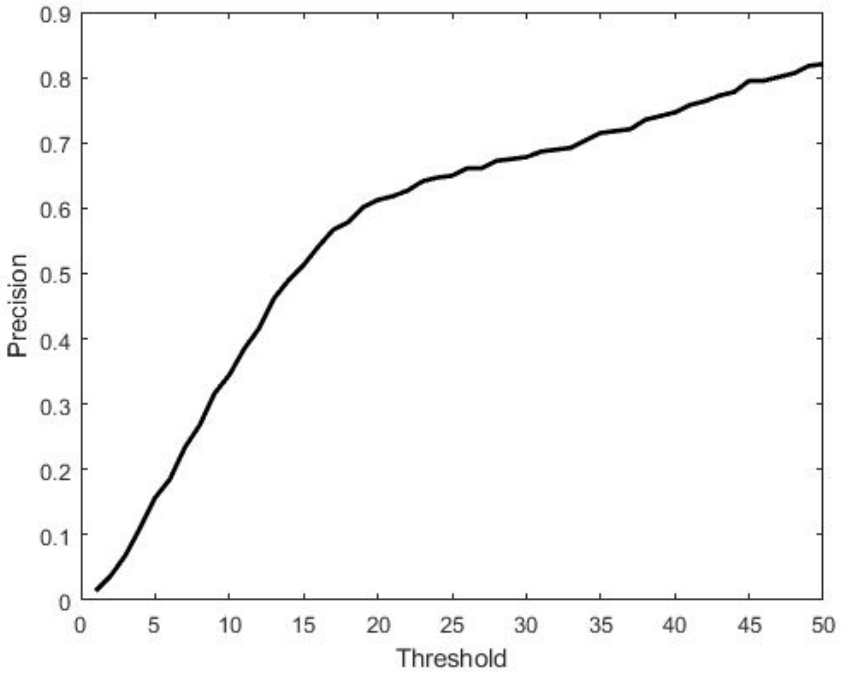
- PSR_THRESHOLD = 5.0
- NUM_HANKEL_FRAMES = 13
- PREV_FRAME_LOOKBACKS = 5

When testing the modified CM tracker for “tiger2”, our parameters were set to:

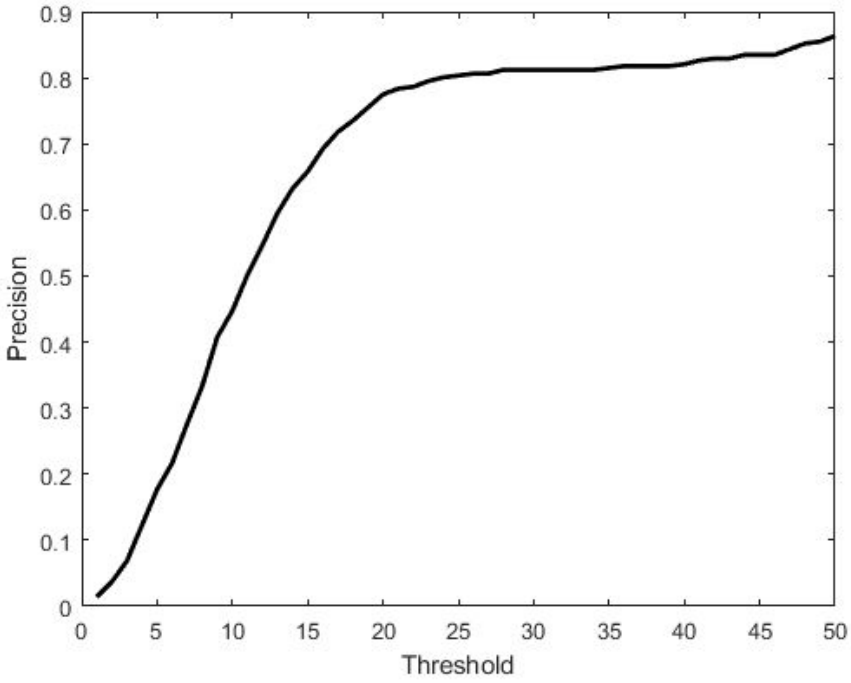
- PSR_THRESHOLD = 5.2
- NUM_HANKEL_FRAMES = 4
- PREV_FRAME_LOOKBACKS = 2

These parameters were determined by testing the algorithm several times with different values and determining that occlusion was best-detected and recovered from using these parameters. For “tiger1”, we found that occlusion was detected so late that the positions from a few frames before were incorrect as well. It was thus useful to use an aggregate of 13 frames between 18-6 frames before occlusion was detected to predict the tiger’s current location. For “tiger2”, we found that occlusion was detected more immediately, so we could instead use an aggregate of 4 frames from 6-3 frames before occlusion was detected.

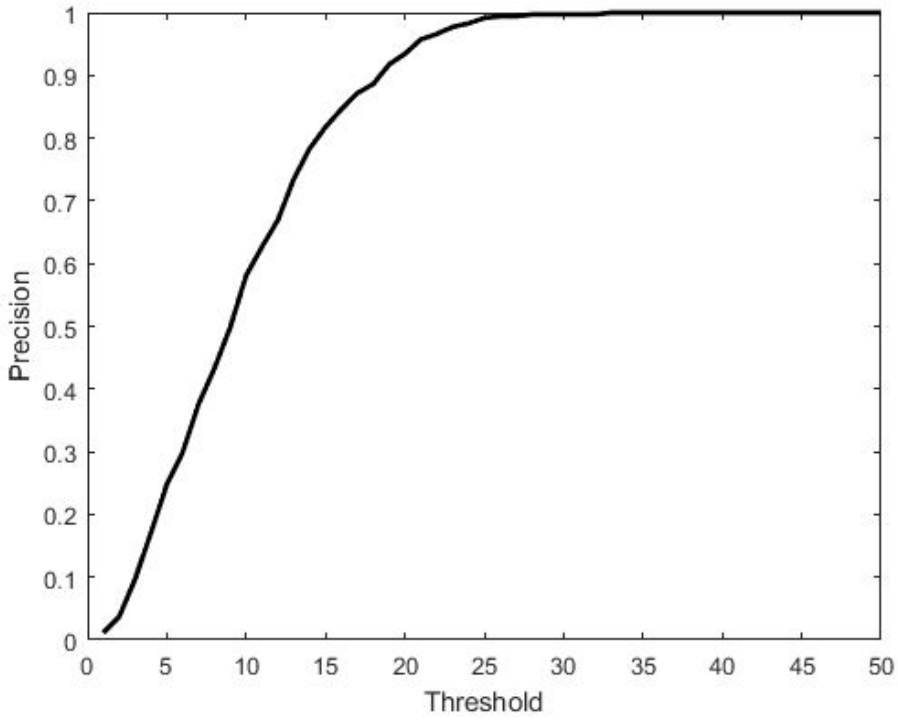
Below are the differences in the precision between the three filters for both sets of images.



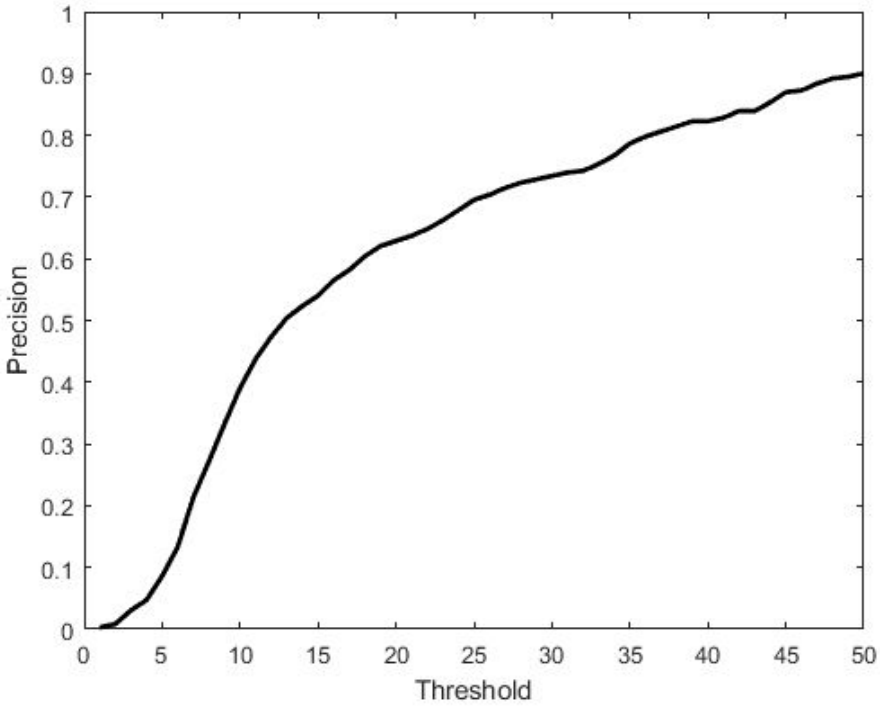
“Tiger1” Images - Precision - Original CM Tracker



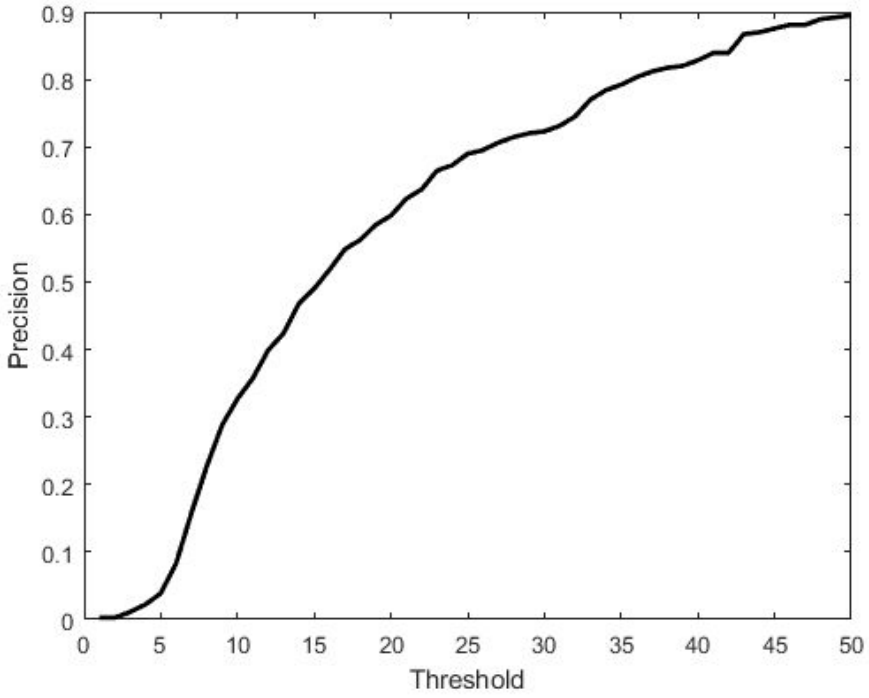
“Tiger1” Images - Precision - Modified CM Tracker



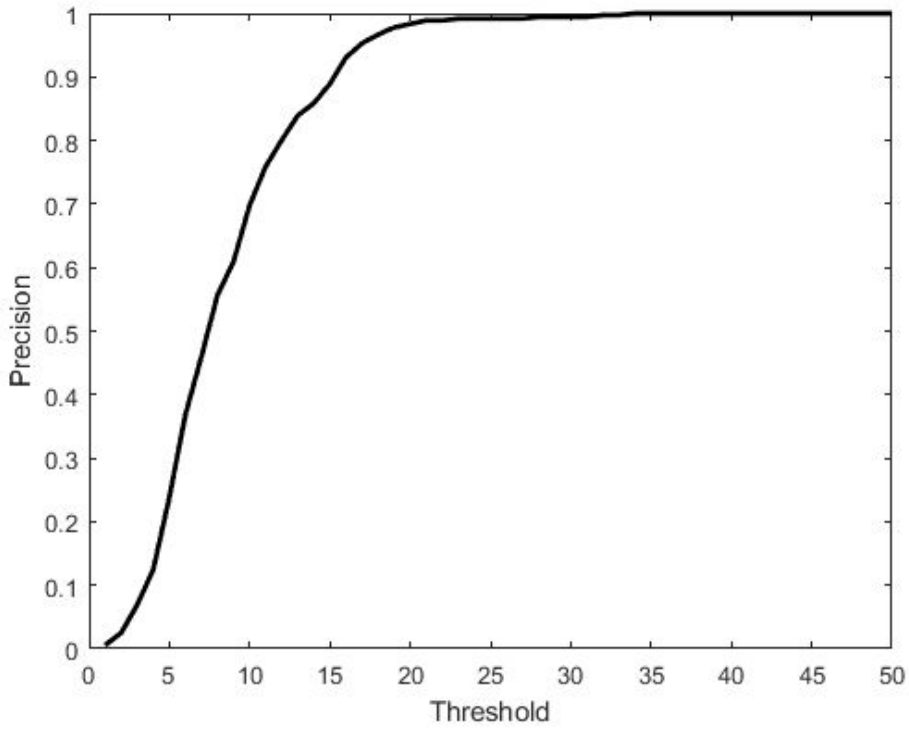
“Tiger1” Images - Precision - MIL Tracker



“Tiger2” Images - Precision - Original CM Tracker



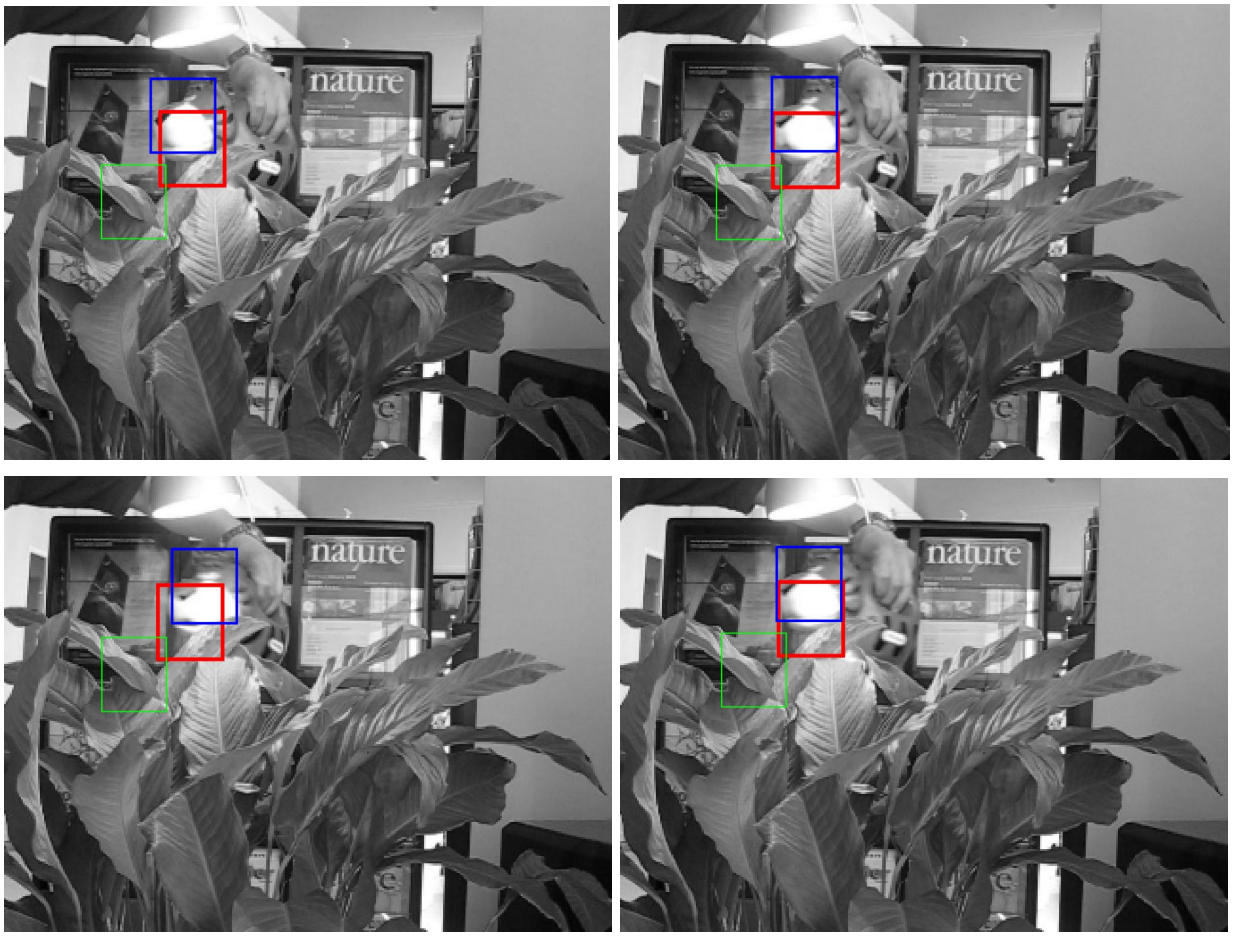
“Tiger2” Images - Precision - Modified CM Tracker



“Tiger2” Images - Precision - MIL Tracker

For both sets of images, the modified CM tracker that recovered from occlusion performed better than the original CM tracker. However, the MIL tracker outperformed even the modified CM tracker. This can be seen in the frames when occlusion was detected. The original CM tracker was unable to find the object for the remainder of the video, the modified CM tracker was looking for the new location and was eventually able to find the object, but the MIL tracker was able to track the object almost flawlessly throughout the video. Below are some of the frames where the tracking differed.

“Tiger1” Frames 235-238 with all tracking



“Tiger2” Frames 134-137 with all tracking

Above are frames 235-238 for the “tiger1” video and frames 134-137 for the “tiger2” video. The green rectangle represents the original CM tracker, the red rectangle represents the modified CM tracker, and the blue rectangle represents the MIL tracker. In the first frame for both the “tiger1” and “tiger2” video, occlusion was detected. The original CM tracker ends up the furthest away from the object being tracked, while the modified CM tracker predicts the location that the object will be found again. In the last frame of both sets, the modified CM tracker was able to find the object again, while the original CM tracker was not. However, the MIL tracker was able to follow the object almost perfectly in both cases, indicating that although our occlusion detection and recovery improved upon the original CM tracker, it still did not perform as well as the MIL tracker, which is further supported by the precision graphs shown earlier.

Conclusion

One benefit of using the Hankel matrix to predict the movement of an object is that it tends to be more accurate than if a constant velocity were being assumed. For example, a Hankel matrix can take in multiple previous frames to predict the location of the object in the future. An object moving in a circular motion would be far better predicted by the previous 4 frames than just the previous frame.

To sum up the results of the experiment, the improvements made to the CM tracker were efficient in detecting and recovering from occlusion. However, the MIL tracker still ended up being even more efficient than the improvements made to the original CM tracker.

Target tracking has numerous possible applications to the real world, especially if the findings are applied in greater depth. These higher-level algorithms could aid in object movement prediction and avoidance. For example, unmanned aerial vehicles (UAVs) need to be able to maneuver with minimal control from people. Implementing target tracking and prediction would allow for UAVs to avoid other objects flying around. Such a feature has already been proposed in the “Aerospace Science and Technology” journal [2].

Overall, the use of target tracking in computer vision is very powerful. It is interesting and fun to experiment with algorithms that are used by computer vision scientists today.

Bibliography

[1] João F. Henriques et al., Original CM Tracker, http://www.robots.ox.ac.uk/~joao/circulant/

[2]
Peng Yao, Honglun Wang, Zikang Su, Cooperative path planning with applications to target tracking and obstacle avoidance for multi-UAVs, In Aerospace Science and Technology, Volume 54, 2016, Pages 10-22, ISSN 1270-9638, https://doi.org/10.1016/j.ast.2016.04.002. (http://www.sciencedirect.com/science/article/pii/S1270963816301304)
Keywords: Lyapunov Guidance Vector Field (LGVF); Improved Interfered Fluid Dynamical System (IIFDS); Unmanned aerial vehicles (UAVs); Three-dimensional cooperative path planning; The rolling optimization strategy

```
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