NORTHEASTERN UNIVERSITY

Computer Vision Project #4: Target Tracking

Jordan Helderman
Justin Hess

13 December 2017

Target tracking is an important topic in computer vision, having application in automatic surveillance applications and, in general, in video processing applications. One challenge in target tracking is detecting and recovery from occlusion. In this report, we present algorithms for occlusion detection and recovery. We use the peak-to-sidelobe ratio as a detection statistic for occlusion detection, and we experiment with two method for occlusion recovery, which both predict the location of the occluded object based on a model for the dynamics of the object. The first method is a simple constant velocity model, and the other is an adaptive model that automatically estimates the complexity of the motion with the Hankel matrix and predicts the position of the occluded object via a linear predictor.

1. INTRODUCTION

Given a video and a region in a frame corresponding to an object of interest, the goal of the target tracking problem is to find regions in the other frames of the video which follow the target's motion. Two problems in target tracking is change in appearance of the target and occlusion of the object due to other objects in the scene. The first can be mitigated by iterative algorithms which process frames in sequence and update the internal model for the appearance of the object on each iteration. For the solution to the first problem and the fundamental structure of the tracker, we will use the circulant matrix approach used by Henriques [2]. The focus of the work for this report will be the development of an approach to solve the second problem, occlusion detection and recovery. We will address the detection and recovery independently. For occlusion detection, we will use the peak-to-sidelobe ratio (PSR) as a detection statistic. For the occlusion recovery, we will use dynamical systems to model the movement of the object and predict the movement of object during periods of occlusion. We will consider two models for this purpose. The first is a constant velocity model, and the other is one which estimates the complexity of the underlying dynamics and linearly predicts the motion of the object using the Hankel matrix. In the following sections, we will describe these algorithms in more detail, present our experimental results, and qualitatively and numerically evaluate the quality of these algorithms.

2. Algorithm Descripion

The following section will describe the algorithms we developed for occlusion detection and recovery. First, we will discuss the PSR-based approach for occlusion detection, and then we will discuss the two tracking algorithms that we developed for occlusion recovery.

2.1. Occlusion Detection

As mentioned previously, the occlusion detection is done via the PSR. The PSR is defined by the following equation, where g_{max} is the peak value of the correlation of a target image with a candidate image and μ and σ are the mean and standard deviation of the sidelobe.

$$PSR = \frac{g_{max} - \mu}{\sigma}$$

Estimation of the PSR according to this equation can be done with the standard estimators for all of the necessary variables on the peak and the sidelobes, respectively.

The PSR makes a good detection statistic for occlusion because it acts as a quality indicator for the correlation of the target with the candidate. It is possible to understand the reasoning for this in the context of matched filtering. In matched filtering, we correlate a template signal with a candidate and take the peak as index of the candidate signal that aligns the candidate signal with the template. Since the integration of the product of two signals, the core operation of correlation, forms an inner product space over the set of signals, we know that the response of the correlation will be maximized at the offset which aligns the candidate signal with the template signal, i.e. the peak of the correlation will be at the offset that aligns the signals. In the presence of stocahstic noise, the sidelobe levels then represent the autocorrelation of the template plus the correltion of the template with the noise. Thus, detection is easy when the PSR is high and more error-prone when the PSR is low. For this reason, we can use the PSR as a quality metric for correlation based detections like the one used to track the target between frames in [2]

2.2. Occlusion Recovery

In our algorithm, occlusion recovery is done with models for the object dynamics. In this section, we will, individually, discuss the two models the two models used in our experiments.

2.2.1. Constant Velocity Model

The first model we used was a constant velocity model. In this model, the position, $x(t)$ is described by the following equation where x_0 is the initial position and v is the constant velocity.

$$x(t) = x_0 + vt$$

Consequently, assuming this model holds and there is no measurement noise, we can estimate the constant velocity as the derivative of the position with respect to time. However, it is reasonable to expect measurement noise on our position measurements. To mitigate the effect of this noise on our estimate, we can take the average of the instantaneous velocity over the period we have tracked the object. As we track the target, we record the position of the target at each timestep, so the estimation of the constnat velocity is straightforward. After the velocity is estimated, the position of the object is predicted during periods of occlusion by evaluating the model at the subsequent timestep.

One issue with this model is that it is a simplistic model for how objects move, in a general sense, because many objects move in an erratic way. For example, soccer players on a pitch will typically move in a straight line over a relatively short time window, but will run up and down the pitch many times over the course of an entire game.

2.2.2. Hankel Matrix

To overcome the simplicity of the constant velocity model, we also tried a more complex dynamical model based on the Hankel matrix. The Hankel matrix, itself, is define by the following equation for a series of measurements $y_1, y_2, \dots$.

$$H_y = \begin{bmatrix} y_1 & y_2 & \cdots & y_p \\ y_2 & y_3 & \cdots & y_{p+1} \\ \vdots & \vdots & \ddots & \vdots \\ y_m & y_{m+1} & \cdots & y_{m+p-1} \end{bmatrix}$$

The Hankel matrix based tracker assumes that we are using a linear, recursive prediction model like the one seen in the following equation, where n is the degree/complexity of recursive update equation.

$$y_k = \sum_{i=1}^{n} a_i y_{k-i}$$

We can automatically find the complexity of the recursive update equation by finding the rank of the Hankel matrix such that the Hankel matrix prediction equation becomes

$$y_k = \sum_{i=1}^{Rank(H_y)} a_i y_{k-i}$$

In a practical setting, we can use the Hankel matrix to predict future values of y by computing the Hankel matrix ending in value of y to be predicte, dividing it into block matricies of the following form, and computing the predicted value according to the subsequent equations.

$$H_y = \left[\begin{array}{cccc|c} y_1 & y_2 & \cdots & y_{p-1} & y_p \\ y_2 & y_3 & \cdots & y_p & y_{p+1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \hline y_{k-p+1} & y_{m+1} & \cdots & y_{k-1} & y_k \end{array}\right] = \begin{bmatrix} A & b \\ C & y_k \end{bmatrix}$$

$$Av = b$$

$$y_k = Cv$$

Therefore, we can solve for v, and consequently y_k, using regular inversion, if A is invertible, or by a least squares method if A is overdetermined.

3. Experimental Results and Discussion

In this section, we will discuss our experimental results on the benchmark set of videos. Specifically, we will, first, present some results that intuitively demonstrate the effectiveness of the occlusion detection. Then, we will look at some representative tracking results, and, finally, we will compare the tracking performance of the tracking algorithms we developed against those developed by [2] and [1].

3.1. Occlusion Detection

To begin, we will discuss the performance of the occlusion detection. Our evaluation of the performance on this point will be qualitative in nature. Specifically, we will look at the occlusion detection over time and correlate it to events where the object was the occluded. In the David video, there are three instances where the object of interest (David's face) is occluded. In figures 3.1-3, we can see that, for all these instances, there is a noticable decrease in the PSR in the frame the image was taken or slightly before, and that the PSR is below the threshold of 20 (although just barely at the time David takes his glasses off).

Additionally, we can see that there are other regions in the video with low PSR. These are primarily caused by changes in David's facial expression. One example of this can be seen in figure 3.4. In this frame David is sticking out his tounge, which is correlated with a decrease in the PSR.

3.2. Target Tracking

In this section, we will qualitatively asses the performance of the target tracking algorithms by looking at the target tracks over time. We will consider both of the algorithms that we developed. In addition, we will consider the tracking algorithm upon which ours was based from [2] and the five instances of the Multiple Instance Learning (MIL) tracker from [1]. Plots of these tracks at key points in the David video can be seen in figures 3.5-8, and the tracks

Figure 3.1: PSR (David's Turn)

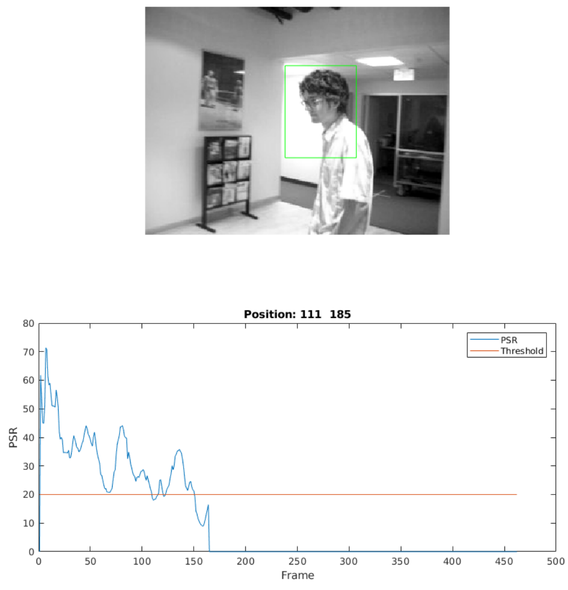


Figure 3.2: PSR (Taking off glasses)

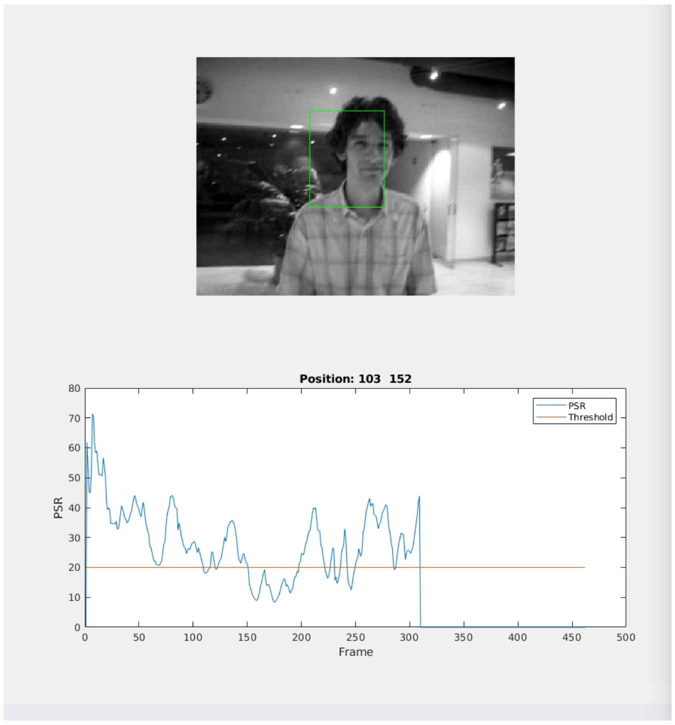


Figure 3.3: PSR (Putting on glasses)

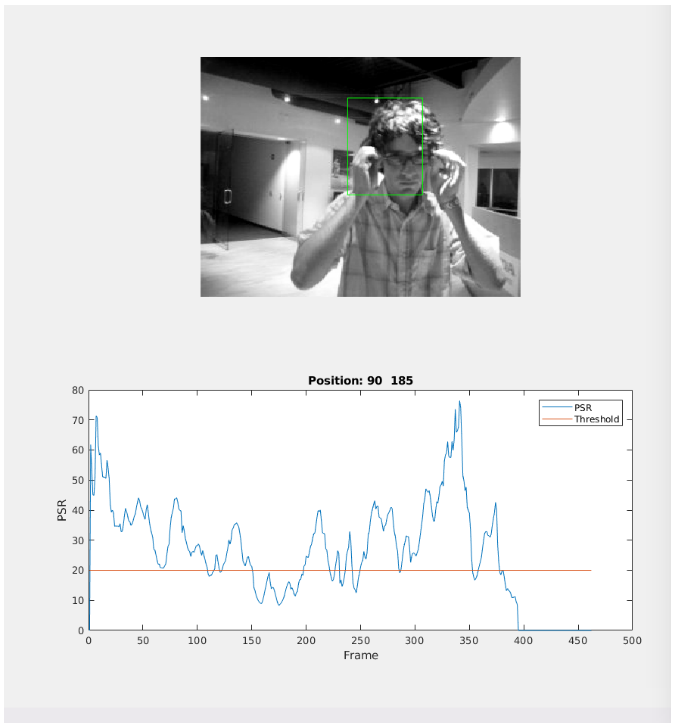


Figure 3.4: PSR (David sticking out tounge)

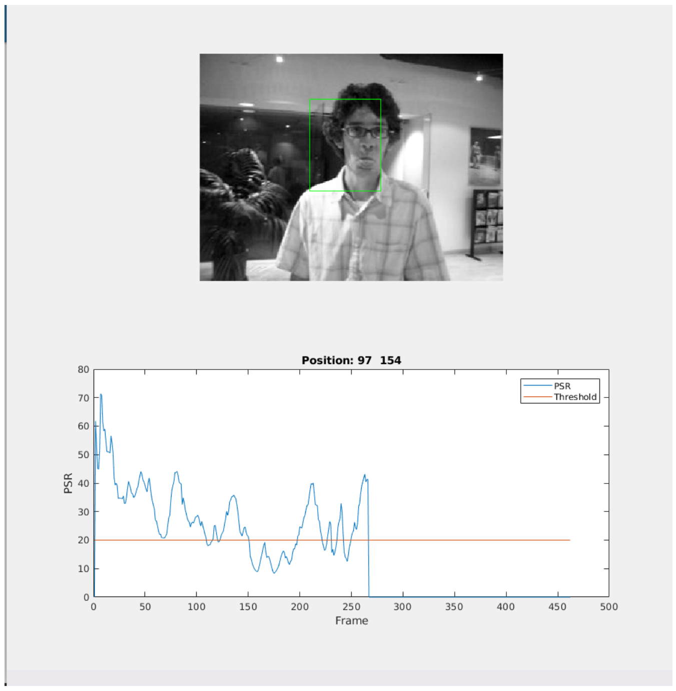
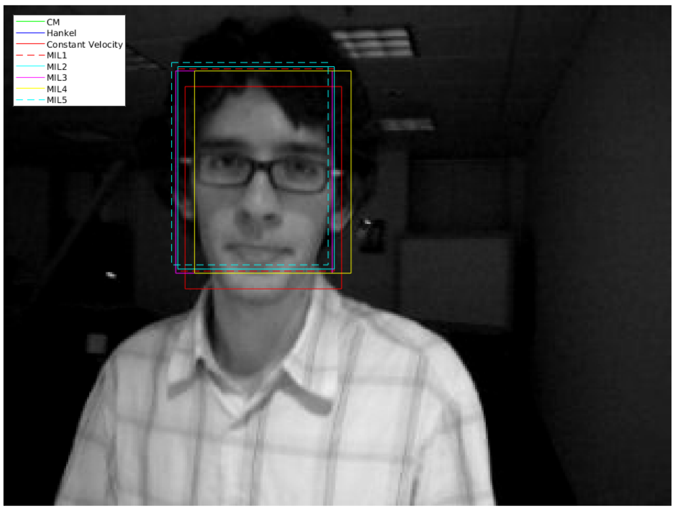


Figure 3.5: Tracks in the beginning of the video



can be identified by the legend. Additionally, the points in the video that we will consider are the beginning of the video, the moment after David turns to the side, obscuring his face, the moment after David sticks out his tounge, and the moment after David puts back on his glasses.

In the beginning of the video we can see that all the trackers are roughly the same in terms of tracked position. This makes sense due to the fact that the beginning of the video does not contain many changes to David's appearance or occlusions. The first difference comes when David turns to the side momentarily. The tracks after this moment are more spread than in the beginning with some of the MIL trackers maintaining the best lock on David's face. In the image that was taken after David sticks out his tounge, the tracks are even further spread, with the MIL4 and the Hankel trackers maintaining the best lock, and some of the others, notably the constant velocity tracker, starting to drift away from David's face. This is likely due to the model mismatch between the constant velocity tracker and the movements that David executes. Finally, looking at the last image, we can see that the tracks are even futher spread. This is probably at least in part due to the classifier losing track of the object it is supposed to be tracking. Since the model parameters are updated on each iteration, this can sometimes happen. In a deployed system, this could potentially be rectified by maintaining a pristine copy of the object and periodically doing a scan of the image to regain the track.

Figure 3.6: Tracks after the turn

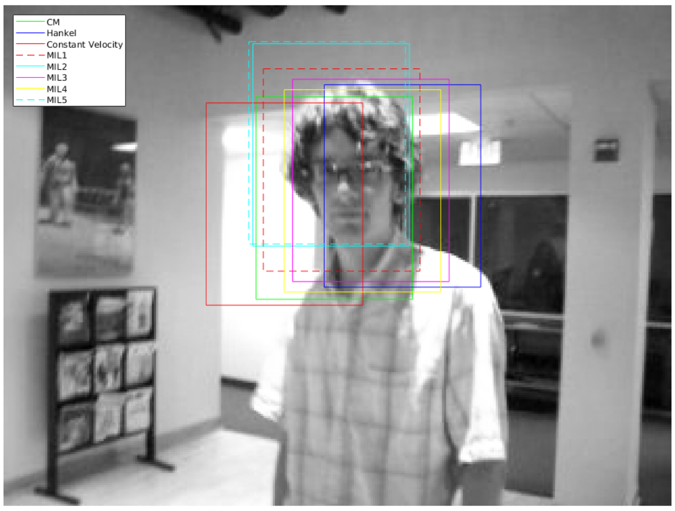


Figure 3.7: Tracks after the tounge

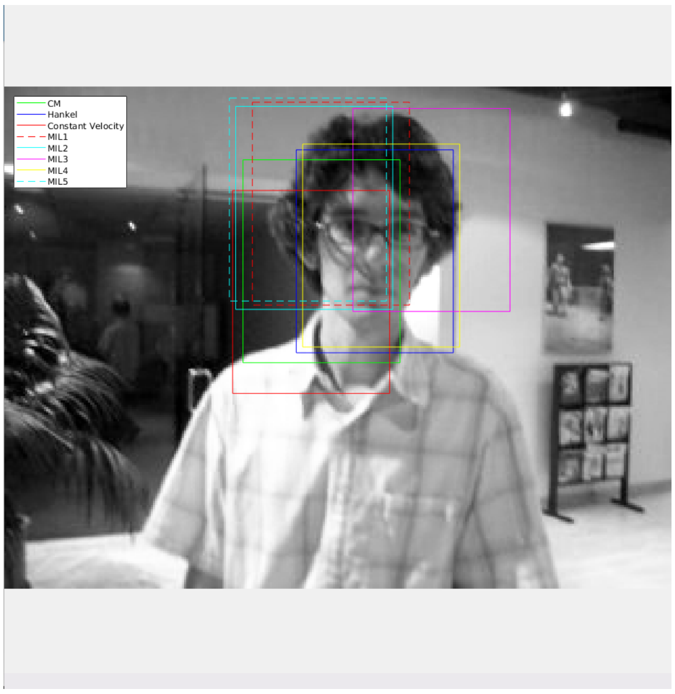
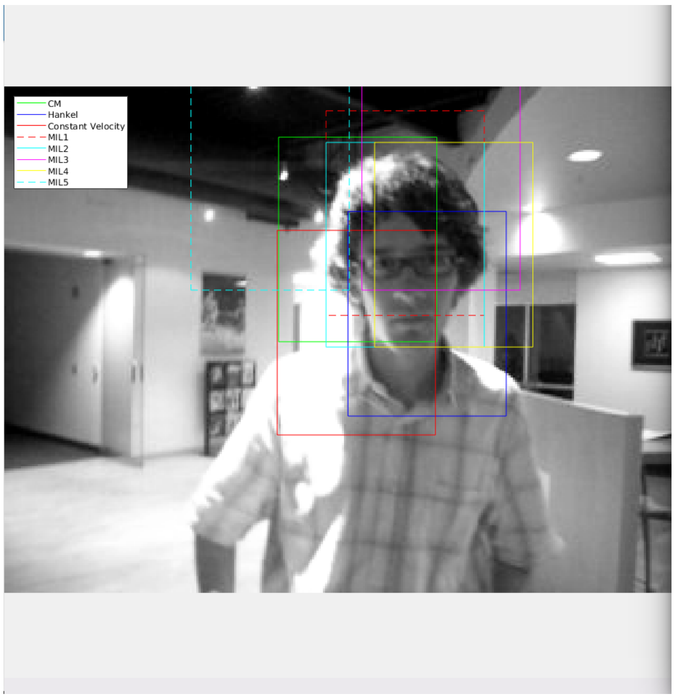


Figure 3.8: Tracks after putting the glasses back on



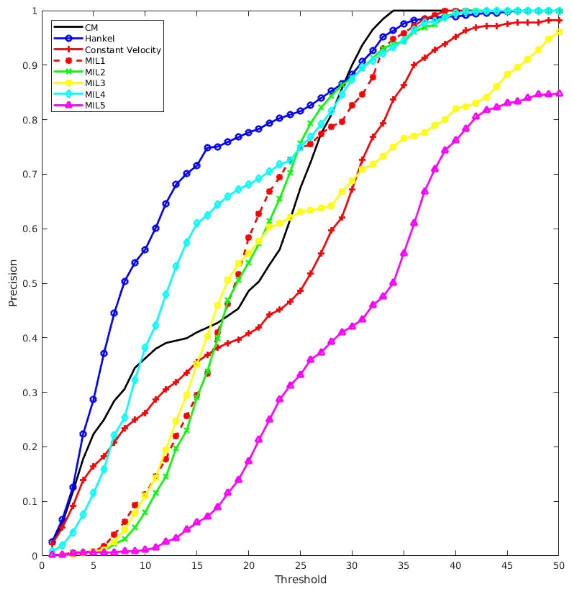
3.3. Comparison

Finally, we will do a numeric comparison of the tracking algorithms by comparing the precision of the detection algorithms for the David video. We compute this measure by comparing the tracked position to ground truth for David's position for each tracking algorithm and computing the precision as the number of instances where the position was less than some threshold away from the ground truth, while varying the threshold to obtain the curve. The plot of this quantity for each of the trackers can be seen in figure 3.9. The ideal curve would be a step function that steps from zero to one at a threshold of zero. As a result, curves which are higher/to the left of others indicate better performance. We can see that the best performance is obtained by the Hankel tracker, followed by the fourth instance of the MIL tracker. We note that the constant velocity tracker actually performed worse than the original tracker. This is possibly due to the model mismatch between the tracking algorithm and the way that David moves.

4. Conclusion

To summarize, we developed algorithms to do occlusion detection and recovery in a video-based object tracking application. We showed that the PSR can be used to detect occlusions and that dynamical models can be used to predict object movement during occlusions. In particular, we showed that the Hankel matrix based model obtained better performance than

Figure 3.9: Precision Plot vs Slack Threshold



the tracking algorithms from [2] and [1].

APPENDIX A TRACKING CODE

The following code implements and tests the tracking algorithms we describe in this report. The tracking code and some of the utility functions originated from [2].

Script for Visualizing the PSR

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

clc; clear; warning off; close all
%choose the path to the videos (you'll be able to choose one with the GUI)
base_path = './data/';

% occlusion detection parameters
peak_sz = [21, 21];
```

```
occlusion_thresh = 20;
diagnostic_plots = false;
num_past = 5;

%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 = choose_video(base_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);

% tracking
occluded = false(numel(img_files), 1);
tracked_location = nan(numel(img_files), 2);
track_v = nan;
track_mem = 200;
tracking_type = 'none';
max_hankel_deg = 5;
max_hankel_mem = -1;

%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 = nan(numel(img_files), 2);  %to calculate precision
```

```
num_frames = numel(img_files);
psr_log = zeros(num_frames, 1);

for frame = 1:numel(img_files)
    %% load and process image
        % load image
        im = imread([video_path img_files{frame}]);
    % convert to grayscale
    if size(im,3) > 1
                im = rgb2gray(im);
    end
    % resize
        if resize_image
                im = imresize(im, 0.5);
        end

        tic()

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

    %% locate target in the subwindow
    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)

        %% check for occlusion
        [gmax, gmax_idx] = max(response(:));
        [row, col] = ind2sub(size(response), gmax_idx);
        peak_c1 = [row - (peak_sz(1) - 1)/2, col - (peak_sz(2) - 1)/2];
        peak_c1 = max([1, 1], peak_c1);
        peak_c2 = peak_c1 + peak_sz - 1;
        peak_c2 = min(peak_c2, size(response));
        sidelobe_region = true(size(response));
        sidelobe_region(peak_c1(1):peak_c2(1), peak_c1(2):peak_c2(2)) = false;
        sidelobe_mean = mean(response(sidelobe_region));
        sidelobe_stdev = std(response(sidelobe_region));
        psr = (gmax - sidelobe_mean) / sidelobe_stdev;
        occluded(frame) = psr < occlusion_thresh;
        psr_log(frame) = psr;

                %% target location is at the maximum response
        [row, col] = find(response == max(response(:)), 1);
        pos = pos - floor(sz/2) + [row, col];

        %% tracking update
        if occluded(frame)
            if strcmpi(tracking_type, 'const-velocity')
                if isnan(track_v)
                    start_meas = max(1, frame - track_mem);
                    prev_pos = positions(start_meas:frame-1, :);
                    track_v = mean(diff(prev_pos, 1, 1), 1);
                end
                tracked_location(frame, :) = ...
                    tracked_location(frame-1,:) - track_v;
            elseif strcmpi(tracking_type, 'hankel')
                hankel_deg = min(frame-1, max_hankel_deg);
                if max_hankel_mem ~= -1
                    start_meas = max(1, frame - max_hankel_mem);
                else
                    start_meas = 1;
                end
                tracked_location(frame, :) = ...
                    hankel_track(positions(start_meas:frame,:), hankel_deg);
            elseif strcmpi(tracking_type, 'none')
                % no tracking
                tracked_location(frame, :) = pos;
            else
                error('tracking_type not recognized');
            end
            pos = tracked_location(frame, :);
        else
            track_v = nan;
            tracked_location(frame, :) = pos;
        end

        %% update position
        pos = max(pos, [1, 1]);
        pos = min(pos, size(im));
        positions(frame,:) = pos;
    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)
```

```
if ~occluded(frame) || strcmpi(tracking_type, 'none')
    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
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])];
start_frame = max(1, frame - num_past + 1);
end_frame = frame;
    if frame == 1  %first frame, create GUI
            figure('Name',['Tracker - ' video_path])
    subplot(2, 1, 1);
            im_handle = imshow(im, 'Border','tight', 'InitialMag',200);
            rect_handle = rectangle('Position',rect_position, 'EdgeColor','g');
    subplot(2, 1, 2);
    psr_plot_handle = plot(psr_log);
    hold on;
    plot([1, length(psr_log)], [occlusion_thresh, occlusion_thresh]);
    legend('PSR', 'Threshold');
    xlabel('Frame');
    ylabel('PSR');
else
            try  %subsequent frames, update GUI
        title(['Position: ', num2str(pos)]);
                    set(im_handle, 'CData', im);
        set(rect_handle, 'Position', rect_position);
        set(psr_plot_handle, 'YData', psr_log);
            catch  %, user has closed the window
                    return
            end
            end

            drawnow
            pause(0.05)  %uncomment to run slower
end

if resize_image, positions = positions * 2; end

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

%show the precisions plot
figure('Name',['Precisions - ' video_path])
show_precision(positions, ground_truth)
ylim([0, 1]);
```

Script for Evaluating Tracker Perofrmance

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

% clc; clear; warning off; close all
%choose the path to the videos (you'll be able to choose one with the GUI)
base_path = './data/';

% occlusion detection parameters
peak_sz = [21, 21];
occlusion_thresh = 20;
diagnostic_plots = false;
num_past = 5;

%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 = choose_video(base_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);
video_base = video_path(1:end-5);
vid_dirs = regexp(video_base, '/', 'split');
vid_name = vid_dirs{end-1};
track1_path = [video_base, vid_name, '_MIL_TR001.txt'];
track2_path = [video_base, vid_name, '_MIL_TR002.txt'];
track3_path = [video_base, vid_name, '_MIL_TR003.txt'];
track4_path = [video_base, vid_name, '_MIL_TR004.txt'];
track5_path = [video_base, vid_name, '_MIL_TR005.txt'];

% tracking
tracked_location = nan(numel(img_files), 2);
track_v = nan;
track_mem = 100;
occluded = false;
tracking_type = 'hankel';
max_hankel_deg = 5;
max_hankel_mem = -1;

%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 = nan(numel(img_files), 2);  %to calculate precision

num_frames = numel(img_files);
psr_log = zeros(num_frames, 1);

cm_tracker = CMTracker(...
    yf, pos, interp_factor, sigma, lambda, cos_window, target_sz, padding);
```

```
hank_tracker = HankelTracker(...
    yf, pos, interp_factor, sigma, lambda, cos_window, target_sz,...
    padding, peak_sz, occlusion_thresh, max_hankel_deg);
constv_tracker = ConstVTracker(...
    yf, pos, interp_factor, sigma, lambda, cos_window, target_sz,...
    padding, peak_sz, occlusion_thresh, track_mem);
mil_track1 = PreCompTracker(track1_path, pos, target_sz);
mil_track2 = PreCompTracker(track2_path, pos, target_sz);
mil_track3 = PreCompTracker(track3_path, pos, target_sz);
mil_track4 = PreCompTracker(track4_path, pos, target_sz);
mil_track5 = PreCompTracker(track5_path, pos, target_sz);

for frame = 1:numel(img_files)
    %% load and process image
        % load image
        im = imread([video_path img_files{frame}]);
    % convert to grayscale
    if size(im,3) > 1
                im = rgb2gray(im);
    end
    % resize
        if resize_image
                im = imresize(im, 0.5);
        end

        tic()

    %% run the tracker
    [cm_tracker, pos] = cm_tracker.track(im);
    [hank_tracker, hpos] = hank_tracker.track(im);
    [constv_tracker, cvpos] = constv_tracker.track(im);
    [mil_track1, mil1pos] = mil_track1.track(im);
    [mil_track2, mil2pos] = mil_track2.track(im);
    [mil_track3, mil3pos] = mil_track3.track(im);
    [mil_track4, mil4pos] = mil_track4.track(im);
    [mil_track5, mil5pos] = mil_track5.track(im);

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

        %% visualization
        if frame == 1 %first frame, create GUI
                figure('Name', 'Tracker')
                im_handle = imshow(im, 'Border','tight', 'InitialMag',200);
        cm_tracker = cm_tracker.plot('g');
        hank_tracker = hank_tracker.plot('b');
        constv_tracker = constv_tracker.plot('r');
        mil_track1 = mil_track1.plot('r', '--');
        mil_track2 = mil_track2.plot('c');
        mil_track3 = mil_track3.plot('m');
        mil_track4 = mil_track4.plot('y');
        mil_track5 = mil_track5.plot('c', '--');
        legend('CM', 'Hankel', 'Constant Velocity', 'MIL1',...
            'MIL2', 'MIL3', 'MIL4', 'MIL5', 'Location', 'northwest');
    else
                try  %subsequent frames, update GUI
%               title(['Position: ', num2str(pos)]);
                        set(im_handle, 'CData', im);
            cm_tracker = cm_tracker.update();
            hank_tracker = hank_tracker.update();
            constv_tracker = constv_tracker.update();
            mil_track1 = mil_track1.update();
            mil_track2 = mil_track2.update();
            mil_track3 = mil_track3.update();
            mil_track4 = mil_track4.update();
            mil_track5 = mil_track5.update();
                catch  %, user has closed the window
                        return
                end
        end

        drawnow
        pause(0.05)  %uncomment to run slower
end

if resize_image, positions = positions * 2; end

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

%show the precisions plot
figure('Name',['Precisions - ' video_path])
show_precision(positions, ground_truth, 'k')
hold on;
show_precision(hank_tracker.positions, ground_truth, 'b-o');
show_precision(constv_tracker.positions, ground_truth, 'r-+');
show_precision(mil_track1.positions, ground_truth, 'r--*');
show_precision(mil_track2.positions, ground_truth, 'g-x');
show_precision(mil_track3.positions, ground_truth, 'y-s');
show_precision(mil_track4.positions, ground_truth, 'c-d');
show_precision(mil_track5.positions, ground_truth, 'm-^');
hold off;
legend('CM', 'Hankel', 'Constant Velocity', 'MIL1', 'MIL2', 'MIL3',...
    'MIL4', 'MIL5', 'Location', 'northwest');
ylim([0, 1]);
```

Code to Plot the Precision

```
function show_precision(positions, ground_truth, ls)
%SHOW_PRECISION
%   Calculates precision for a series of distance thresholds (percentage of
%   frames where the distance to the ground truth is within the threshold).
%   The results are shown in a new figure.
%
%   Accepts positions and ground truth as Nx2 matrices (for N frames), and
%   a title string.
%
%   Joao F. Henriques, 2012
%   http://www.isr.uc.pt/~henriques/

        if nargin < 3
        ls = 'k-';
    end
        max_threshold = 50;  %used for graphs in the paper


        if size(positions,1) ~= size(ground_truth,1),
                disp('Could not plot precisions, because the number of ground')
                disp('truth frames does not match the number of tracked frames.')
                return
        end

        %calculate distances to ground truth over all frames
        distances = sqrt((positions(:,1) - ground_truth(:,1)).^2 + ...
                                        (positions(:,2) - ground_truth(:,2)).^2);
        distances(isnan(distances)) = [];

        %compute precisions
        precisions = zeros(max_threshold, 1);
        for p = 1:max_threshold,
                precisions(p) = nnz(distances < p) / numel(distances);
        end
        %plot the precisions
        plot(precisions, ls, 'LineWidth',2)
        xlabel('Threshold'), ylabel('Precision')

end
```

Object to Encode Original Circulant Matrix based Tracker

```
classdef CMTracker
    %CMTracker: A class encapsulating the Circulant Matrix based tracker

    properties
        yf;
        pos;
        z;
        alphaf;
        interp_factor;
        cos_window;
        sigma;
        lambda;
        sz;
        target_sz;
        rect_h;
        positions;
    end

    methods
        function obj = CMTracker(...
                yf, pos, interp_factor, sigma, lambda, ...
                cos_window, target_sz, padding)
            obj.yf = yf;
            obj.pos = pos;
            obj.z = nan;
            obj.alphaf = nan;
            obj.interp_factor = interp_factor;
            obj.sigma = sigma;
            obj.lambda = lambda;
            obj.cos_window = cos_window;
            obj.target_sz = target_sz;
            obj.sz = floor(target_sz * (1 + padding));
            obj.positions = [];
        end

        function [obj, pos] = track(obj, img)
        %% extract and pre-process subwindow
        x = get_subwindow(img, obj.pos, obj.sz, obj.cos_window);

        %% locate target in the subwindow
        if length(obj.z) ~= 1
            %% calculate response of the classifier at all locations
            k = dense_gauss_kernel(obj.sigma, x, obj.z);
            response = real(ifft2(obj.alphaf .* fft2(k)));   %(Eq. 9)

            %% target location is at the maximum response
            [row, col] = find(response == max(response(:)), 1);
            obj.pos = obj.pos - floor(obj.sz/2) + [row, col];

            %% update position
            obj.pos = max(obj.pos, [1, 1]);
            obj.pos = min(obj.pos, size(img));
        end

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

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

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

        %% return
        pos = obj.pos;
        obj.positions = [obj.positions; pos];
    end

    function obj = plot(obj, c, line_style)
```

```
            if nargin < 2
                c = 'g';
            end
            if nargin < 3
                line_style = '-';
            end
            rect_position = [obj.pos([2,1]) - obj.target_sz([2,1])/2, ...
                             obj.target_sz([2,1])];
            obj.rect_h = rectangle('Position',rect_position, ....
                'EdgeColor',c, 'LineStyle', line_style);
            % create an invisible line so that the legend works right...
            line(NaN,NaN,'LineStyle',line_style,'Color',c);
        end

        function obj = update(obj, h)
            if nargin > 1
                obj.rect_h = h;
            end
            rect_position = [obj.pos([2,1]) - obj.target_sz([2,1])/2,...
                             obj.target_sz([2,1])];
            set(obj.rect_h, 'Position', rect_position);
        end
    end
end
```

Object to Encode Hankel Tracker

```
classdef HankelTracker
    %CMTracker: A class encapsulating the Circulant Matrix based tracker
    %with occlusion detection via the PSR and occlusion recovery via a
    %constant velocity model

    properties
        yf;
        pos;
        z;
        alphaf;
        interp_factor;
        cos_window;
        sigma;
        lambda;
        sz;
        target_sz;
        rect_h;
        peak_sz;
        occlusion_thresh;
        positions;
        max_deg;
        max_hankel_mem;
    end

    methods
        function obj = HankelTracker(...
                yf, pos, interp_factor, sigma, lambda, cos_window,...
                target_sz, padding, peak_sz, occlusion_thresh, max_deg)
            obj.yf = yf;
            obj.pos = pos;
            obj.z = nan;
            obj.alphaf = nan;
            obj.interp_factor = interp_factor;
            obj.sigma = sigma;
            obj.lambda = lambda;
            obj.cos_window = cos_window;
            obj.target_sz = target_sz;
            obj.sz = floor(target_sz * (1 + padding));
            obj.occlusion_thresh = occlusion_thresh;
            obj.peak_sz = peak_sz;
            obj.positions = pos;
            obj.max_deg = max_deg;
            obj.max_hankel_mem = -1;
        end

        function [obj, pos] = track(obj, img)
            occluded = false;
            %% extract and pre-process subwindow
            x = get_subwindow(img, obj.pos, obj.sz, obj.cos_window);

            %% locate target in the subwindow
            if length(obj.z) ~= 1
                %% calculate response of the classifier at all locations
                k = dense_gauss_kernel(obj.sigma, x, obj.z);
                response = real(ifft2(obj.alphaf .* fft2(k)));   %(Eq. 9)

                %% check for occlusion
                occluded = is_occluded(response, obj.occlusion_thresh, obj.peak_sz);

                %% target location is at the maximum response
                [row, col] = find(response == max(response(:)), 1);
                pos = obj.pos - floor(obj.sz/2) + [row, col];
    %% tracking update
    if occluded
        hankel_deg = min(size(obj.positions, 1)-1, obj.max_deg);
        if obj.max_hankel_mem ~= -1
            start_meas = max(...
                1, size(obj.positions, 1) - obj.max_hankel_mem);
        else
            start_meas = 1;
        end
        obj.positions = ...
            [obj.positions; ...
             hankel_track(obj.positions(start_meas:end,:), hankel_deg)];
    else
        obj.positions = [obj.positions; pos];
    end

    %% update position
    obj.pos = obj.positions(end, :);
    obj.pos = max(obj.pos, [1, 1]);
    obj.pos = min(obj.pos, size(img));
end

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

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

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

```
            end

            %% return
            pos = obj.pos;
        end

        function obj = plot(obj, c, ls)
            if nargin < 2
                c = 'g';
            end
            if nargin < 3
                ls = '-';
            end
            rect_position = [...
              obj.pos([2,1]) - obj.target_sz([2,1])/2, obj.target_sz([2,1])];
            obj.rect_h = rectangle(...
              'Position',rect_position, 'EdgeColor',c, 'LineStyle', ls);
            % create an invisible line so that the legend works right...
            line(NaN,NaN,'LineStyle',ls,'Color',c);
        end

        function obj = update(obj, h)
            if nargin > 1
                obj.rect_h = h;
            end
            rect_position = [...
              obj.pos([2,1]) - obj.target_sz([2,1])/2, obj.target_sz([2,1])];
            set(obj.rect_h, 'Position', rect_position);
        end
    end
end
```

Object to Encode Constant Velocity Tracker

```
classdef ConstVTracker
    %CMTracker: A class encapsulating the Circulant Matrix based tracker
    %with occlusion detection via the PSR and occlusion recovery via a
    %constant velocity model

    properties
        yf;
        pos;
        z;
        alphaf;
        interp_factor;
        cos_window;
        sigma;
        lambda;
        sz;
        target_sz;
        rect_h;
        peak_sz;
        occlusion_thresh;
        positions;
        track_v;
        track_mem;
    end

    methods
        function obj = ConstVTracker(...
                yf, pos, interp_factor, sigma, lambda, cos_window, target_sz, ...
                padding, peak_sz, occlusion_thresh, track_mem)
            obj.yf = yf;
            obj.pos = pos;
            obj.z = nan;
            obj.alphaf = nan;
            obj.interp_factor = interp_factor;
            obj.sigma = sigma;
            obj.lambda = lambda;
            obj.cos_window = cos_window;
            obj.target_sz = target_sz;
            obj.sz = floor(target_sz * (1 + padding));
            obj.occlusion_thresh = occlusion_thresh;
            obj.peak_sz = peak_sz;
            obj.positions = pos;
            obj.track_v = nan;
            obj.track_mem = track_mem;
        end

        function [obj, pos] = track(obj, img)
            occluded = false;
            %% extract and pre-process subwindow
            x = get_subwindow(img, obj.pos, obj.sz, obj.cos_window);

            %% locate target in the subwindow
            if length(obj.z) ~= 1
                %% calculate response of the classifier at all locations
                k = dense_gauss_kernel(obj.sigma, x, obj.z);
                response = real(ifft2(obj.alphaf .* fft2(k)));   %(Eq. 9)
        %% check for occlusion
        occluded = is_occluded(response, obj.occlusion_thresh, obj.peak_sz);

        %% target location is at the maximum response
        [row, col] = find(response == max(response(:)), 1);
        pos = obj.pos - floor(obj.sz/2) + [row, col];

        %% tracking update
        if occluded
            if isnan(obj.track_v)
                frame = size(obj.positions, 1);
                start_meas = frame - obj.track_mem;
                prev_pos = obj.positions(start_meas:frame-1, :);
                obj.track_v = mean(diff(prev_pos, 1, 1), 1);
            end
            obj.positions = ...
                [obj.positions; obj.positions(end,:) - obj.track_v];
        else
            obj.track_v = nan;
            obj.positions = [obj.positions; pos];
        end

        %% update position
        obj.pos = obj.positions(end, :);
        obj.pos = max(obj.pos, [1, 1]);
        obj.pos = min(obj.pos, size(img));
    end

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

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

    if length(obj.z) == 1
        % first frame, train with a single image
        obj.alphaf = new_alphaf;
        obj.z = x;
    else
```

```
                % subsequent frames, interpolate model
                obj.alphaf = (1 - obj.interp_factor) * obj.alphaf +...
                    obj.interp_factor * new_alphaf;
                obj.z = (1 - obj.interp_factor) * obj.z + obj.interp_factor * new_z;
            end
%             end

            %% return
            pos = obj.pos;
        end

        function obj = plot(obj, c, ls)
            if nargin < 2
                c = 'g';
            end
            if nargin < 3
                ls = '-';
            end
            rect_position = [...
              obj.pos([2,1]) - obj.target_sz([2,1])/2, obj.target_sz([2,1])];
            obj.rect_h = rectangle(...
              'Position',rect_position, 'EdgeColor',c, 'LineStyle', ls);
            % create an invisible line so that the legend works right...
            line(NaN,NaN,'LineStyle',ls,'Color',c);
        end

        function obj = update(obj, h)
            if nargin > 1
                obj.rect_h = h;
            end
            rect_position = [...
              obj.pos([2,1]) - obj.target_sz([2,1])/2, obj.target_sz([2,1])];
            set(obj.rect_h, 'Position', rect_position);
        end
    end
end
```

Object to Encode Trackers based on Pre-computed values like the MIL Trackers

```
classdef PreCompTracker
    %PreCompTracker: A class encapsulating a tracker with precomputed
    %values

    properties
        target_sz;
        positions;
        n;
        pos;
        rect_h;
    end

    methods
        function obj = PreCompTracker(path, pos, target_sz)
            obj.target_sz = target_sz;
            obj.n = 1;
            track_data = csvread(path);
            obj.positions = [...
                track_data(:,2) + 0.5 * track_data(:,4),...
                track_data(:,1) + 0.5 * track_data(:,3)];
            obj.pos = pos;
        end

        function [obj, pos] = track(obj, img)
            if obj.n > size(obj.positions, 1)
                error('no more positions');
            end

            %% pull the position
            obj.pos = obj.positions(obj.n, :);
            obj.n = obj.n + 1;

            %% return
            pos = obj.pos;
        end

        function obj = plot(obj, c, ls)
            if nargin < 2
                c = 'g';
            end
            if nargin < 3
                ls = '-';
            end
            rect_position = [...
              obj.pos([2,1]) - obj.target_sz([2,1])/2, obj.target_sz([2,1])];
            obj.rect_h = rectangle(...
              'Position',rect_position, 'EdgeColor',c, 'LineStyle', ls);
            % create an invisible line so that the legend works right...
            line(NaN,NaN,'LineStyle',ls,'Color',c);
        end
        function obj = update(obj, h)
            if nargin > 1
                obj.rect_h = h;
            end
            rect_position = [...
              obj.pos([2,1]) - obj.target_sz([2,1])/2, obj.target_sz([2,1])];
            set(obj.rect_h, 'Position', rect_position);
        end
    end
end
```

Function to Perform Hankel Prediction

```
function yhat = hankel_track(y, deg)
num_rows = length(y) - deg + 1;
yhat = zeros(1, size(y, 2));
for m = 1:size(y,2)
    %% compute the hankel matrix
    H = zeros(num_rows, deg);
    for n = 1:num_rows
        H(n, :) = y(n:n+deg-1, m);
    end
    %% estimate the new point
    A = H(1:end-1, 1:end-1);
    b = H(1:end-1, end);
    C = H(end, 1:end-1);
    yhat(m) = C * pinv(A) * b;
end
end
```

Function to Detect Occlusion

```
function occluded = is_occluded(response, thresh, peak_sz)
%% check for occlusion
[gmax, gmax_idx] = max(response(:));
[row, col] = ind2sub(size(response), gmax_idx);
peak_c1 = [row - (peak_sz(1) - 1)/2, col - (peak_sz(2) - 1)/2];
peak_c1 = max([1, 1], peak_c1);
peak_c2 = peak_c1 + peak_sz - 1;
peak_c2 = min(peak_c2, size(response));
sidelobe_region = true(size(response));
sidelobe_region(peak_c1(1):peak_c2(1), peak_c1(2):peak_c2(2)) = false;
sidelobe_mean = mean(response(sidelobe_region));
sidelobe_stdev = std(response(sidelobe_region));
psr = (gmax - sidelobe_mean) / sidelobe_stdev;
```

```
occluded = psr < thresh;
end
```

REFERENCES

[1] Boris Babenko and Ming-Hsuan Yang Serge Belongie. “Robust Object Tracking with Online Multiple Instance Learning”. In: 2011.

[2] João F. Henriques et al. “Exploiting the Circulant Structure of Tracking-by-detection with Kernels”. In: *Proceedings of the 12th European Conference on Computer Vision - Volume Part IV*. ECCV’12. Florence, Italy: Springer-Verlag, 2012, pp. 702–715. ISBN: 978-3-642-33764-2. DOI: `10.1007/978-3-642-33765-9_50`. URL: `http://dx.doi.org/10.1007/978-3-642-33765-9_50`.