Target Tracking

Song Jiang and Mengyun Xia
EECE 5639, Computer Vision
April 16, 2016

Abstract

In this project, we will include a test that measures the response of the filter against the rest of the search window through the use of the "Peak to Sidelobe Ratio" (PSR) to check for occlusion. After detecting occlusion, we will use the Hankel matrix. We use the locations of the target in the past to predict where the target is now (even if it is behind some occluding object) and predict where it should be in the next frame.

1. Introduction

The Circulant Matrix tracker that we discussed in class is very efficient finding a translated copy of the target template (from the previous frame) by computing many convolutions in a single shot. This is accomplished by finding the peak response of a filter applied to a region of the current frame that is expected to include the target. This filter changes from frame to frame and it is computed based on the FFT of a larger region which contains the target in the current frame. But the CM tracker does not check for occlusion. Occlusion detection can be done by including a test that measures the response of the filter against the rest of the search window through the use of the "Peak to Sidelobe Ratio" (PSR). The tracker can attempt to find the target in the next frame at this predicted location. The prediction of the location of the target based on previous measurements can be done using Hankel matrix.

2. Description of Algorithms

The Circulant Matrix (CM) tracker is very efficient finding a translated copy of the target template (from the previous frame) by computing many convolutions in a single shot. This is accomplished by finding the peak response of a filter applied to a region of the current frame that is expected to include the target. This filter changes from frame to frame and it is computed based on the FFT of a larger region which contains the target in the current frame. (Efficiency is obtained by applying this filter in the frequency domain).

About detecting occlusion, to check for occlusion, one can do this by including a test that measures the response of the filter against the rest of the search window through the use of the "Peak to Sidelobe Ratio" (PSR). To do this, one should split the response of the filter into the maximum value and the "sidelobe" consisting of the rest of the pixels in the region, excluding a small window around the peak. Then the PSR is defined as: $(gmax - \mu)/\sigma$, where gmax is the value of the peak, and μ and σ are the mean value and the standard deviation of the sidelobe, respectively. A low PSR indicates a poor match and a possible occlusion. If occlusion is detected, the tracker should stop or attempt to hallucinate the target until it can detect it again.

About recovery from occlusion, when occlusion is detected, the tracker could use the locations of the target in the past to predict where the target is now (even if it is behind some occluding object) and predict where it should be in the next frame. In this way, rather than giving up, the tracker can attempt to find the target in the next frame at this predicted location. The prediction of the location of the target based on previous measurements can be done using Hankel matrix of the target locations.

The rank of the Hankel matrix is the order of the system. With perfect noise-free data, the minimum order realization can be easily obtained by keeping only the non-zero Hankel singular values. With real or noise-contaminated data, however, the Hankel matrix tends to be full rank, thus making the problem of determining a minimum-order state-space model non-trivial. A reduced-order model obtained by retaining only "significant" singular values tends to be poor in accuracy. Indeed, this issue remains one of the most unsatisfactory discrepancy between what is expected in theory and what is actually observed in practice. Real systems are both non-linear and infinite dimensional. A common procedure is to keep all Hankel singular values at the expense of a high-dimensional state-space model and use a separate post-identification procedure such as model reduction is to reduce the dimension of the identified model. However, any time model reduction is invoked, some accuracy is lost. It is preferable to have an identification method that produces a "true" or effective order model directly in the first place.

"Simple" dynamics might fail if there is prolonged occlusion. We can use tools from system identification (Hankel matrix) to predict future measurements without assuming a dynamic model.
Given a sequence of measurements of d-dimensional vectors:

$$y_1, y_2, y_3 \cdots$$

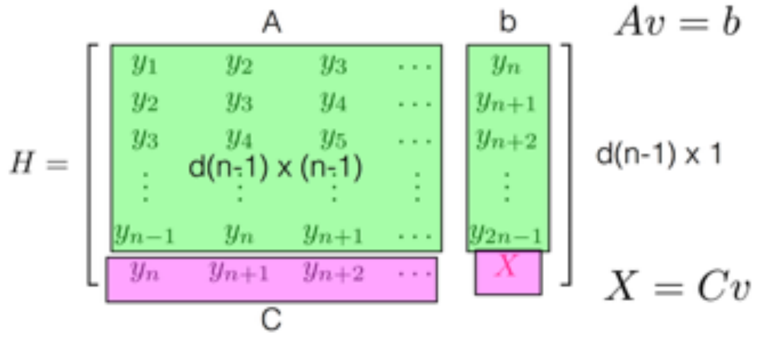
Its Hankel matrix is defined as:

$$H_y = \begin{bmatrix} y_1 & y_2 & \cdots & y_{n-1} & y_n & \cdots & y_p \\ y_2 & y_3 & \cdots & y_n & y_{n+1} & \vdots & y_{p+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ y_m & y_{m+1} & \cdots & y_{m+n-2} & y_{m+n-1} & \cdots & y_{m+p-1} \end{bmatrix}_{\text{md x p}}$$

Rank(Hy) measures the complexity of the underlying dynamics: after we have "enough" measurements the rank of the Hankel matrix does not increase.

Rank(H) = n =Complexity of Dynamics

The new measurement should not increase the complexity of the dynamics:
The last column must be a linear combination of the previous ones. (But we should know the order of the system n-1).



A
b
$Av = b$
$H =$
y_1 y_2 y_3 $\cdots$
y_2 y_3 y_4 $\cdots$
y_3 y_4 y_5 $\cdots$
d(n-1) x (n-1)
y_{n-1} y_n y_{n+1} $\cdots$
y_n y_{n+1} y_{n+2}
$\vdots$
y_{2n-1}
d(n-1) x 1
y_n y_{n+1} y_{n+2} $\cdots$
X
$X = Cv$
C

i. Assemble Hankel matrix with noise

$$H = \begin{bmatrix} y_1 + \eta_1 & y_2 + \eta_2 & y_3 + \eta_3 & \cdots & y_n + \eta_n \\ y_2 + \eta_2 & y_3 + \eta_3 & y_4 + \eta_4 & \cdots & y_{n+1} + \eta_{n+1} \\ y_3 + \eta_3 & y_4 + \eta_4 & y_5 + \eta_5 & \cdots & y_{n+2} + \eta_{n+2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ y_n + \eta_n & y_{n+1} + \eta_{n+1} & y_{n+2} + \eta_{n+2} & \cdots & y_{2n-1} + \eta_{2n-1} \end{bmatrix}$$

ii. Minimize rank(H) wrt noise

iii. Predict next measurement/update

3. Experiments and values of parameters used

1) Detection whether there is an occlusion

Split the response of the filter into the maximum value and the "sidelobe" consisting of the rest of the pixels in the region, excluding a small window (11×11) around the peak. Then the PSR is defined as: $gmax - \mu/\sigma$, where gmax is the value of the peak, and μ and σ are the mean value and the standard deviation of the sidelobe, respectively. A low PSR indicates a poor match and a possible occlusion. Here, we set a threshold to judge whether the PSR is enough low.

2) Recovery from the occlusion

Applying the Hankel matrix of the target locations to use the locations of the target in the past to predict where the target is now (even if it is behind some occluding object) and predict where it should be in the next frame. Here, we make two Hankel matrix, splitting the coordinates of the position of the tracking image and making them be the parameters in Hankel matrix. Through calculating, we get a new position under current frame and continually do the next detection of occlusion.

4. Observations

Use the show precision function to compare the results against the original CM tracker and the original Multiple Instance Learning (MIL) tracker. Add a visualization to indicate when detected occlusion and when recovered the target.

In the following images, the bounding box's width and height is 68 and 78. The blue, green and red bounding boxes are generated by MIL, CM and our tracker. At first, the green and red boxes are overlapped. When at the time of first image generated, the occlusion was detected and two boxes were separated. Later, our tracker still followed center position of the image. However, the green box was nearly unmoved. And the blue box had a big error.

nature

nature

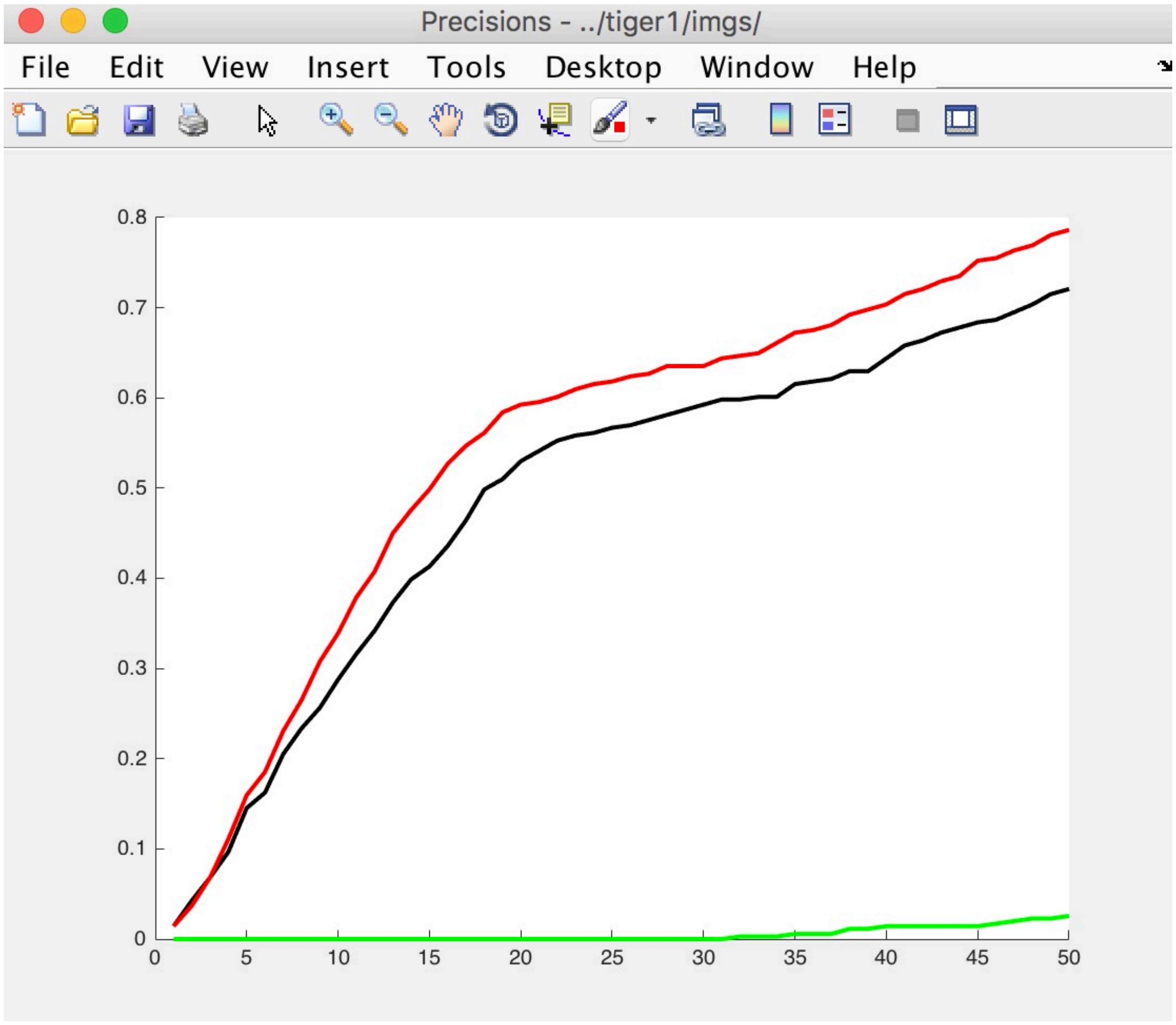
nature

nature

nature

nature

nature



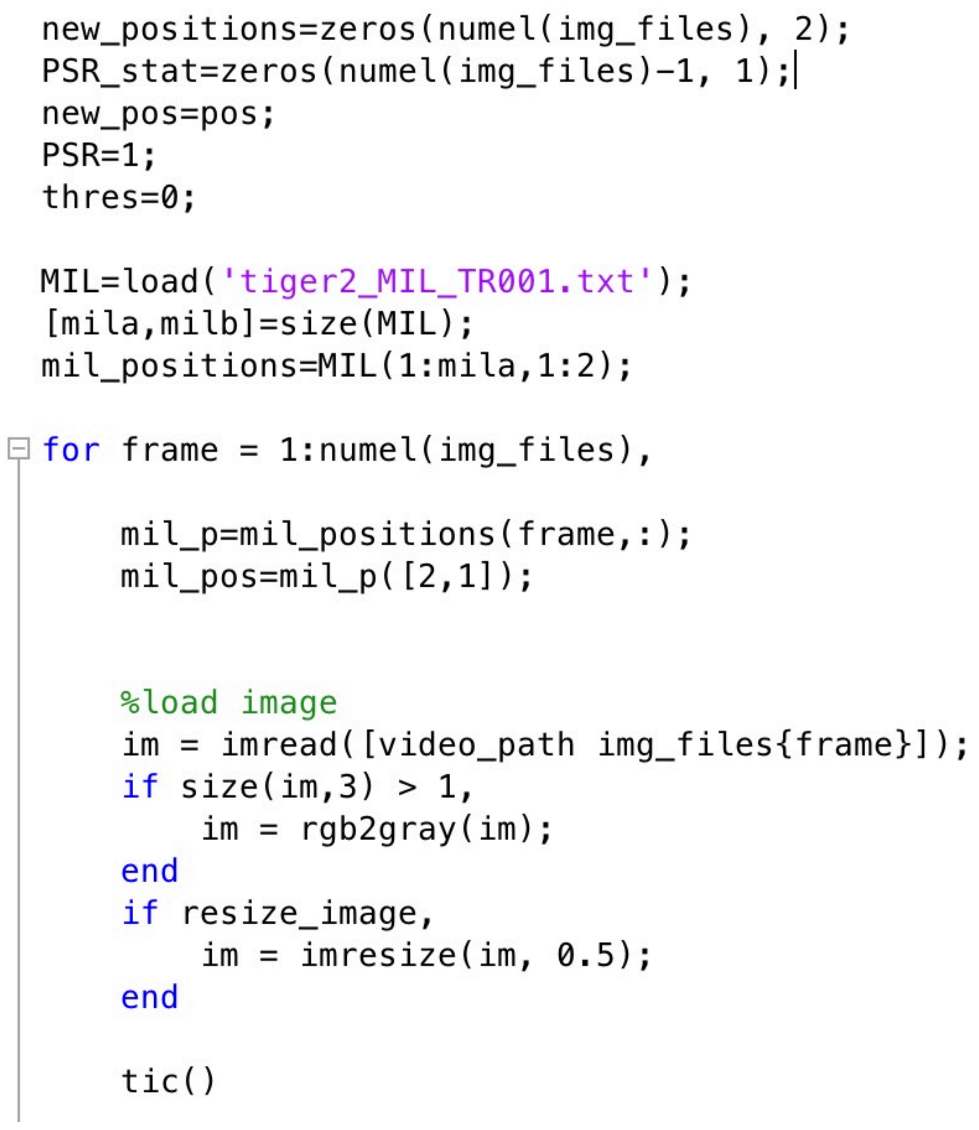
Calculates precision for a series of distance thresholds (percentage of frames where the distance to the ground truth is within the threshold). From this showing-precision image, we can see that our tracker is represented by the red line. It is similar to the green one which represents to the CM tracker at the beginning. And it has better precision.

5. Conclusion

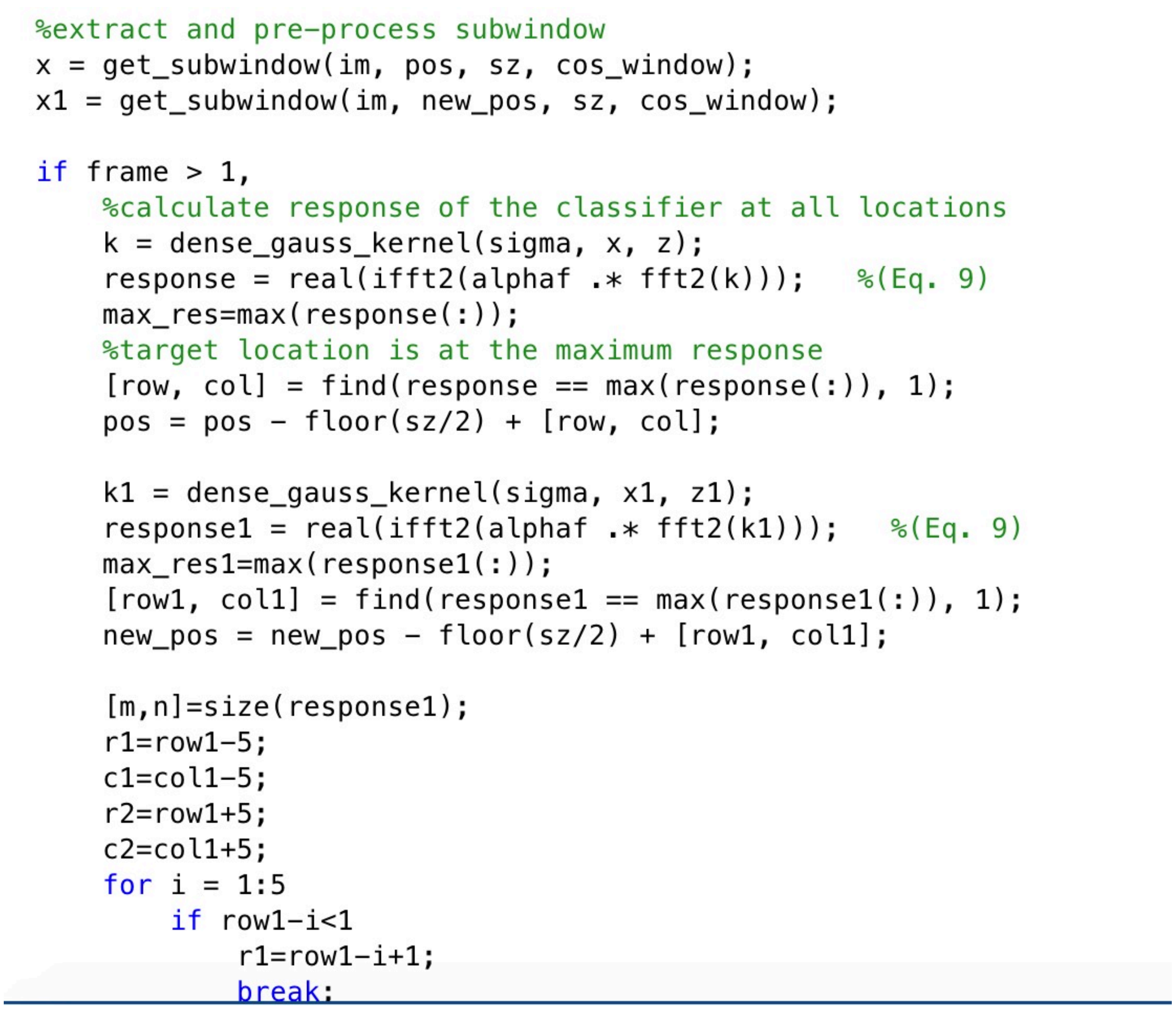
Using Hankel matrix is a good way to make a prediction. And it can help solve the problem of occlusion. In this way, rather than giving up, the tracker can attempt to find the target in the next frame at this predicted location.

6. Appendix

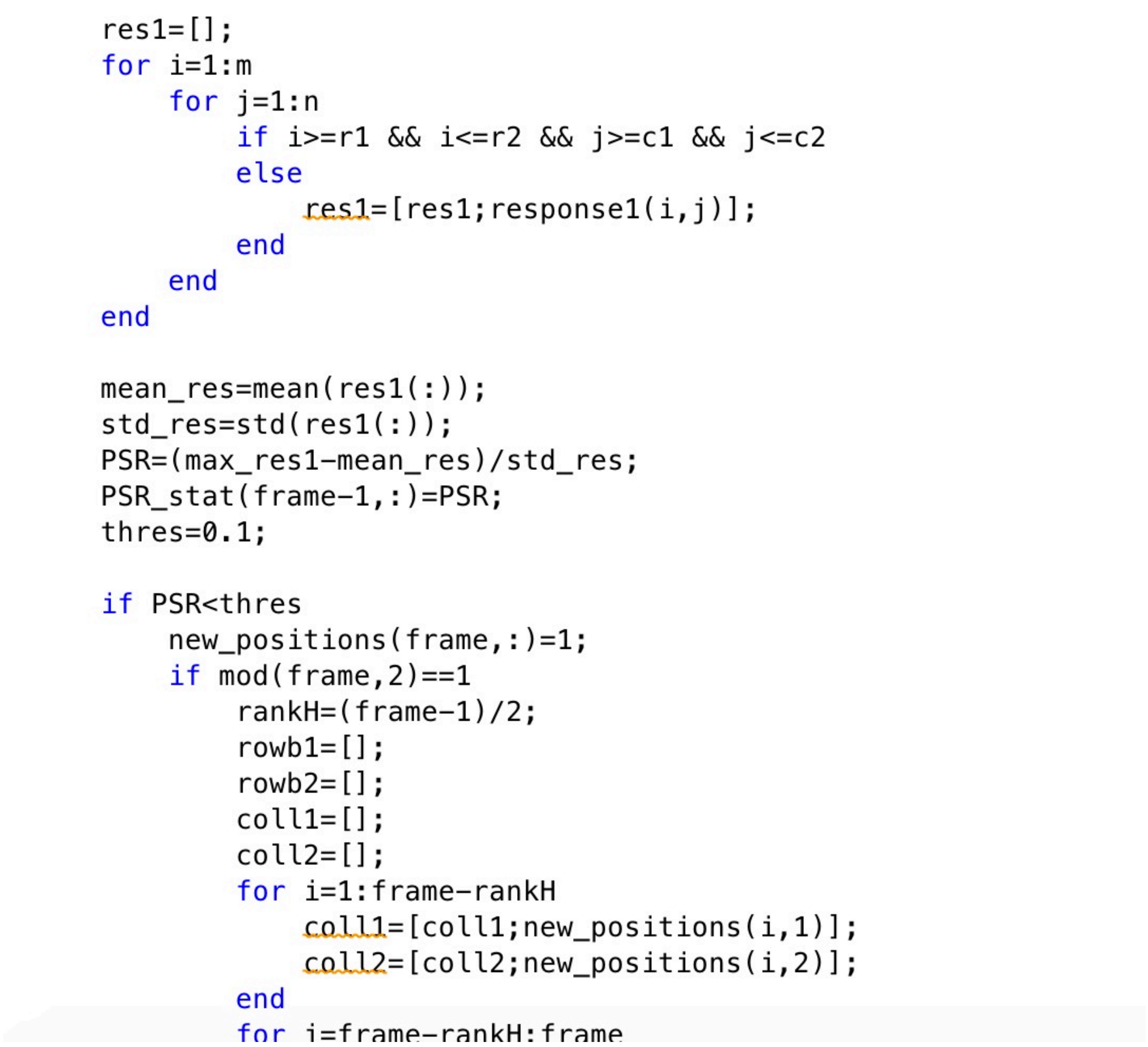
The programs written by ourselves:



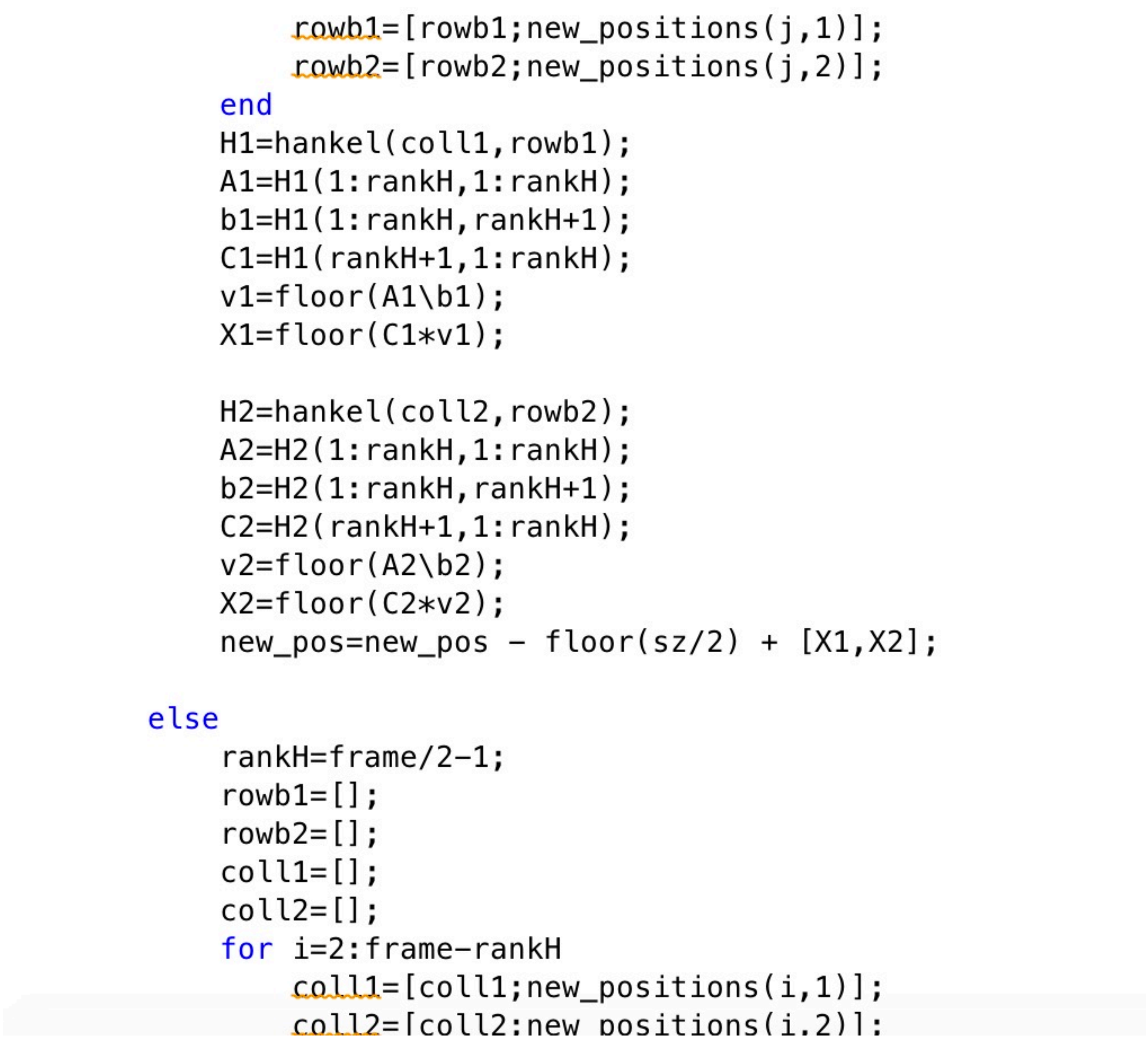
```
new_positions=zeros(numel(img_files), 2);
PSR_stat=zeros(numel(img_files)-1, 1);
new_pos=pos;
PSR=1;
thres=0;

MIL=load('tiger2_MIL_TR001.txt');
[mila,milb]=size(MIL);
mil_positions=MIL(1:mila,1:2);

for frame = 1:numel(img_files),

    mil_p=mil_positions(frame,:);
    mil_pos=mil_p([2,1]);


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

    tic()
```



```
%extract and pre-process subwindow
x = get_subwindow(im, pos, sz, cos_window);
x1 = get_subwindow(im, new_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)
    max_res=max(response(:));
    %target location is at the maximum response
    [row, col] = find(response == max(response(:)), 1);
    pos = pos - floor(sz/2) + [row, col];

    k1 = dense_gauss_kernel(sigma, x1, z1);
    response1 = real(ifft2(alphaf .* fft2(k1)));   %(Eq. 9)
    max_res1=max(response1(:));
    [row1, col1] = find(response1 == max(response1(:)), 1);
    new_pos = new_pos - floor(sz/2) + [row1, col1];

    [m,n]=size(response1);
    r1=row1-5;
    c1=col1-5;
    r2=row1+5;
    c2=col1+5;
    for i = 1:5
        if row1-i<1
            r1=row1-i+1;
            break;
        end
    end
    for i = 1:5
        if col1-i<1
            c1=col1-i+1;
            break;
        end
    end
    for i = 1:5
        if row1+i>m
            r2=row1+i-1;
            break;
        end
    end
    for i = 1:5
        if col1+i>n
            c2=col1+i-1;
            break;
        end
    end


    res1=[];
    for i=1:m
        for j=1:n
            if i>=r1 && i<=r2 && j>=c1 && j<=c2
            else
```



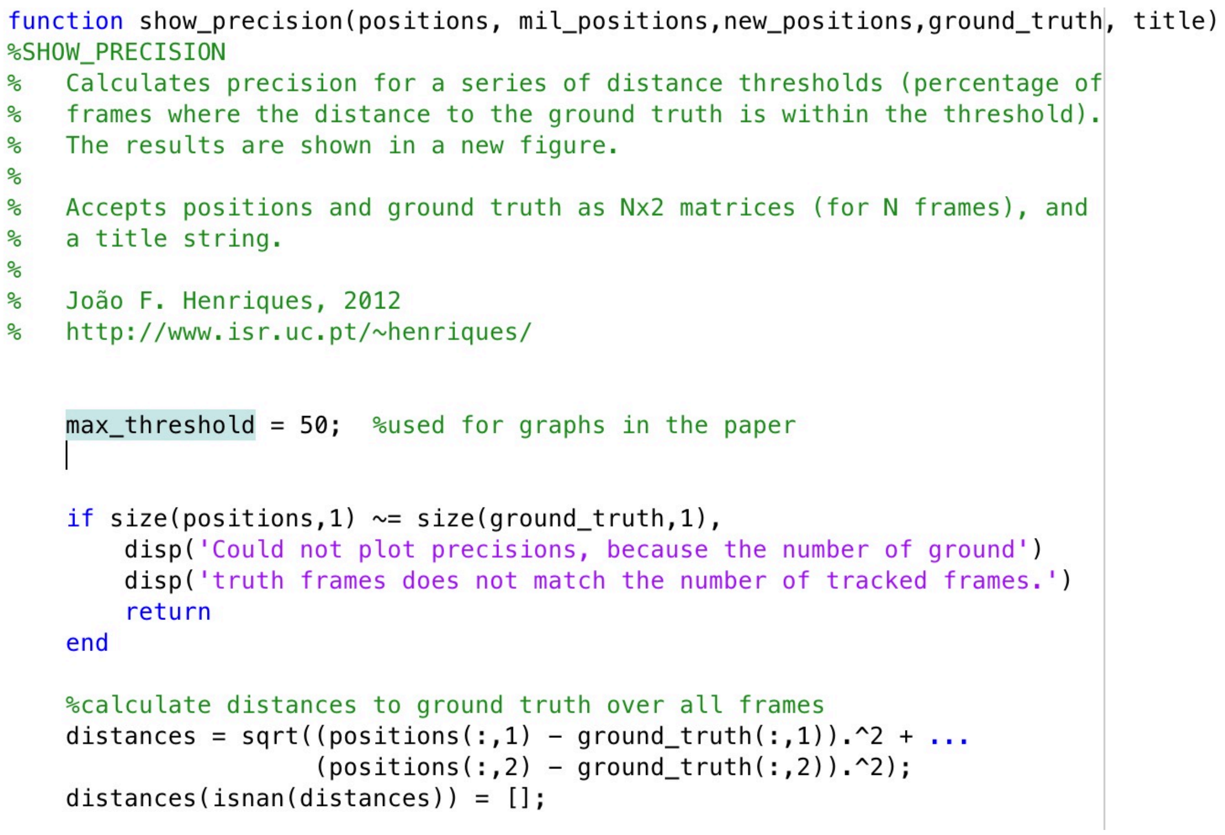
```
res1=[];
for i=1:m
    for j=1:n
        if i>=r1 && i<=r2 && j>=c1 && j<=c2
        else
            res1=[res1;response1(i,j)];
        end
    end
end

mean_res=mean(res1(:));
std_res=std(res1(:));
PSR=(max_res1-mean_res)/std_res;
PSR_stat(frame-1,:)=PSR;
thres=0.1;

if PSR<thres
    new_positions(frame,:)=1;
    if mod(frame,2)==1
        rankH=(frame-1)/2;
        rowb1=[];
        rowb2=[];
        coll1=[];
        coll2=[];
        for i=1:frame-rankH
            coll1=[coll1;new_positions(i,1)];
            coll2=[coll2;new_positions(i,2)];
        end
        for i=frame-rankH:frame
```



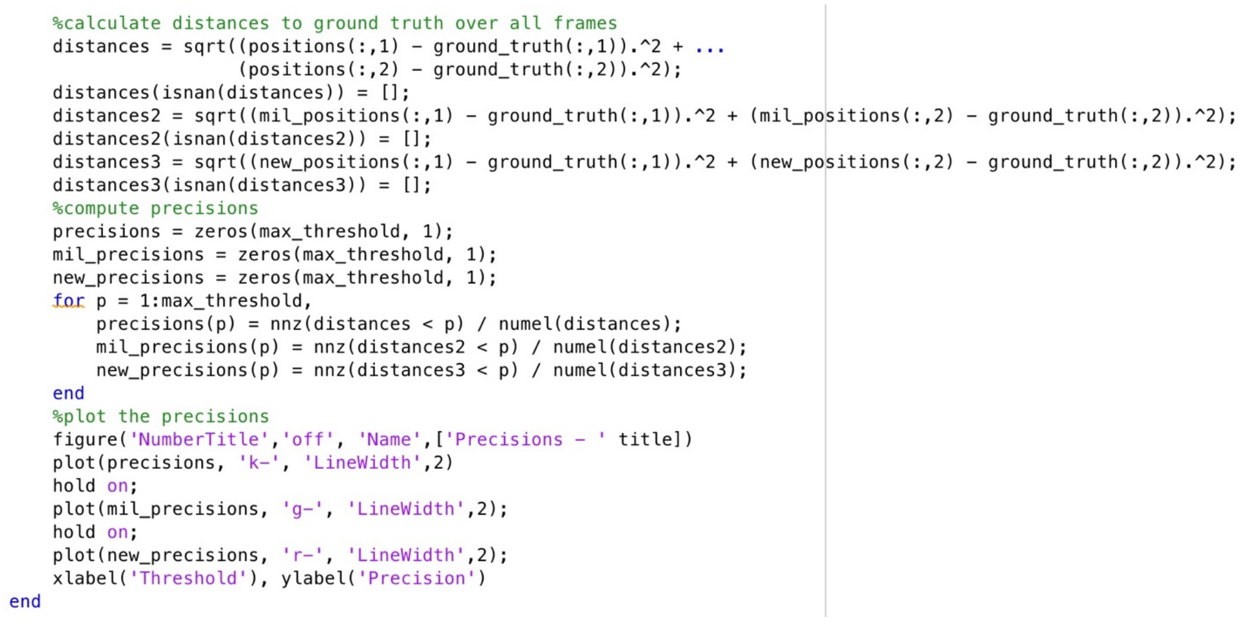
```
            rowb1=[rowb1;new_positions(j,1)];
            rowb2=[rowb2;new_positions(j,2)];
        end
        H1=hankel(coll1,rowb1);
        A1=H1(1:rankH,1:rankH);
        b1=H1(1:rankH,rankH+1);
        C1=H1(rankH+1,1:rankH);
        v1=floor(A1\b1);
        X1=floor(C1*v1);

        H2=hankel(coll2,rowb2);
        A2=H2(1:rankH,1:rankH);
        b2=H2(1:rankH,rankH+1);
        C2=H2(rankH+1,1:rankH);
        v2=floor(A2\b2);
        X2=floor(C2*v2);
        new_pos=new_pos - floor(sz/2) + [X1,X2];

    else
        rankH=frame/2-1;
        rowb1=[];
        rowb2=[];
        coll1=[];
        coll2=[];
        for i=2:frame-rankH
            coll1=[coll1;new_positions(i,1)];
            coll2=[coll2;new_positions(i,2)];
```

```
                end
                for i=frame-rankH:frame
                    rowb1=[rowb1;new_positions(i,1)];
                    rowb2=[rowb2;new_positions(i,2)];
                end
                H1=hankel(coll1,rowb1);
                A1=H1(1:rankH,1:rankH);
                b1=H1(1:rankH,rankH+1);
                C1=H1(rankH+1,1:rankH);
                v1=floor(A1\b1);
                X1=floor(C1*v1);

                H2=hankel(coll2,rowb2);
                A2=H2(1:rankH,1:rankH);
                b2=H2(1:rankH,rankH+1);
                C2=H2(rankH+1,1:rankH);
                v2=floor(A2\b2);
                X2=floor(C2*v2);
                new_pos=new_pos - floor(sz/2) + [X1,X2];
            end
        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;


    x1 = get_subwindow(im, new_pos, sz, cos_window);
        %Kernel Regularized Least-Squares, calculate alphas (in Fourier domain)
        k1 = dense_gauss_kernel(sigma, x1);
        new_alphaf1 = yf ./ (fft2(k1) + lambda);   %(Eq. 7)
        new_z1 = x1;



    if frame == 1,  %first frame, train with a single image
        alphaf = new_alphaf;
        z = x;

        alphaf1 = new_alphaf1;
        z1 = x1;
```

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

    alphaf1 = (1 - interp_factor) * alphaf1 + interp_factor * new_alphaf1;
    z1 = (1 - interp_factor) * z1 + interp_factor * new_z1;

end

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

%visualization
rect_position = [pos([2,1]) - target_sz([2,1])/2, target_sz([2,1])];
rect_mil_position = [mil_pos([2,1]) - target_sz([2,1])/2, target_sz([2,1])];
rect_new_position = [new_pos([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);
    rect_handle = rectangle('Position',rect_position, 'EdgeColor','g');
    rect_handle1 = rectangle('Position',rect_new_position, 'EdgeColor','r');
    rect_handle2 = rectangle('Position',rect_mil_position, 'EdgeColor','b');
    if frame == 1,  %first frame, create GUI
        figure('NumberTitle','off', 'Name',['Tracker - ' video_path]);
        im_handle = imshow(im, 'Border','tight', 'InitialMag',200);
        rect_handle = rectangle('Position',rect_position, 'EdgeColor','g');
        rect_handle1 = rectangle('Position',rect_new_position, 'EdgeColor','r');
        rect_handle2 = rectangle('Position',rect_mil_position, 'EdgeColor','b');
    else
        try  %subsequent frames, update GUI
            set(im_handle, 'CData', im)
            set(rect_handle, 'Position', rect_position)
            set(rect_handle1, 'Position', rect_new_position)
            set(rect_handle2, 'Position', rect_mil_position)
        catch  %#ok, 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
show_precision(positions,mil_positions,new_positions, ground_truth, video_path)
```



```
function show_precision(positions, mil_positions,new_positions,ground_truth, title)
%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.
%
%   João F. Henriques, 2012
%   http://www.isr.uc.pt/~henriques/


    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)) = [];
```



```
    %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)) = [];
    distances2 = sqrt((mil_positions(:,1) - ground_truth(:,1)).^2 + (mil_positions(:,2) - ground_truth(:,2)).^2);
    distances2(isnan(distances2)) = [];
    distances3 = sqrt((new_positions(:,1) - ground_truth(:,1)).^2 + (new_positions(:,2) - ground_truth(:,2)).^2);
    distances3(isnan(distances3)) = [];
    %compute precisions
    precisions = zeros(max_threshold, 1);
    mil_precisions = zeros(max_threshold, 1);
    new_precisions = zeros(max_threshold, 1);
    for p = 1:max_threshold,
        precisions(p) = nnz(distances < p) / numel(distances);
        mil_precisions(p) = nnz(distances2 < p) / numel(distances2);
        new_precisions(p) = nnz(distances3 < p) / numel(distances3);
    end
    %plot the precisions
    figure('NumberTitle','off', 'Name',['Precisions - ' title])
    plot(precisions, 'k-', 'LineWidth',2)
    hold on;
    plot(mil_precisions, 'g-', 'LineWidth',2);
    hold on;
    plot(new_precisions, 'r-', 'LineWidth',2);
    xlabel('Threshold'), ylabel('Precision')
end
```