

Target Tracking

Yuhao Huang, Chengsheng Zou

Abstract

CM tracker is very efficient finding a translated copy of the target template, based on previous frames. However, it fails in occlusion cases, since it doesn't implement that functionality. In this paper, based on CM tracker algorithms, we improve this CM tracker by adding occlusion detection and occlusion recovery system. We use PSR to detect when occlusion happens, and then we use Hankel matrix to predict where the target should be based on previous state information, until we find the target again.

Occlusion Detection

The CM tracker which is provided in code right now, doesn't support occlusion at all. Basically, it always put image data into training set, despite if the target is occluded by something else. The consequence it generated is that it will put occlusion object into training data, and the original data will be totally wiped away when occlusion happens for a while. At the end, CM tracker will follow occlusion object instead of our target.

Thus, occlusion detection is necessary to avoid that consequence. Peak to Sidelobe Ratio (PSR) is used in our detection system. In a situation with no occlusion, the response of the filter will have a peak, indicating where is the best match of the target. However, when occlusion happens, the overall response of the filter is kind of flat, since there is no really good matching in the image. Thus, the PSR will become low when occlusion happens.

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

Where, G_{max} is the peak value of the response, μ and σ are the mean value and the standard deviation of the sidelobe, respectively.

In our code, we used two sets of images sequences, "tiger" and "girl". Depends on different target size, we set different PSR threshold and sidelobe size.

<i>Image sequence</i>	<i>PSR threshold</i>	<i>Sidelobe size</i>
tiger	6	9
girl	10.5	13

Occlusion Recovery

Once we detected occlusion, we will stop CM tracker data training, and attempt to hallucinate the target until it can detect the target again. The way we used in our code is Hankel matrix. Basically, we use the locations of the target in the past to predict where the target is now and where it should be in the next frame.

Hankel Matrix is the way to build a dynamic transmit matrix between past states and next state. In our case, we used 10 past states to predict the coming state, which is the coordinate position of the target center.

Basically, we will stop training data once occlusion happens. Instead, we will try to predict where the target should be, until the tracker find the target again with the previous training data.

Visualization for Comparison

In order to display the comparison clearly, first we output a file from our program, which contains the center location of image, and the window size. Once we get all coordinate positions from CM tracker, CM tracker with occlusion detection, and MIL tracker, then we fetch those locations data from local file, then display all of them on the video, to show the performance of each tracker system. Besides showing windows on videos, we also use `show_precision` method to calculate the result, and put all of them on a diagram, where we can clearly compare the performance of each tracker system.

Result Comparison

We tested our modified algorithm on two image sets that have obvious occlusions: *tiger1* and *girl*. We will present comparisons of the tracking windows of the three different algorithms: CM, MIL and our modified CM that considers occlusion. Besides, we will also compare the precisions of these three different algorithms.

1. Comparison of tracking windows

There are three rectangular windows of different colours. Red window represents our modified tracking window; green window represents CM tracking window; blue window represents the MIL tracking window. We will mainly discuss the results of the CM and modified CM algorithm because the MIL algorithm didn't follow the target well.

i. Image set *tiger1*: The following are five screenshots of the target tracking when occlusions happened. From these five images, we can see that when the tiger was initially occluded by the leaf, both CM and CM with occlusion could still track the target well. But as the target moved

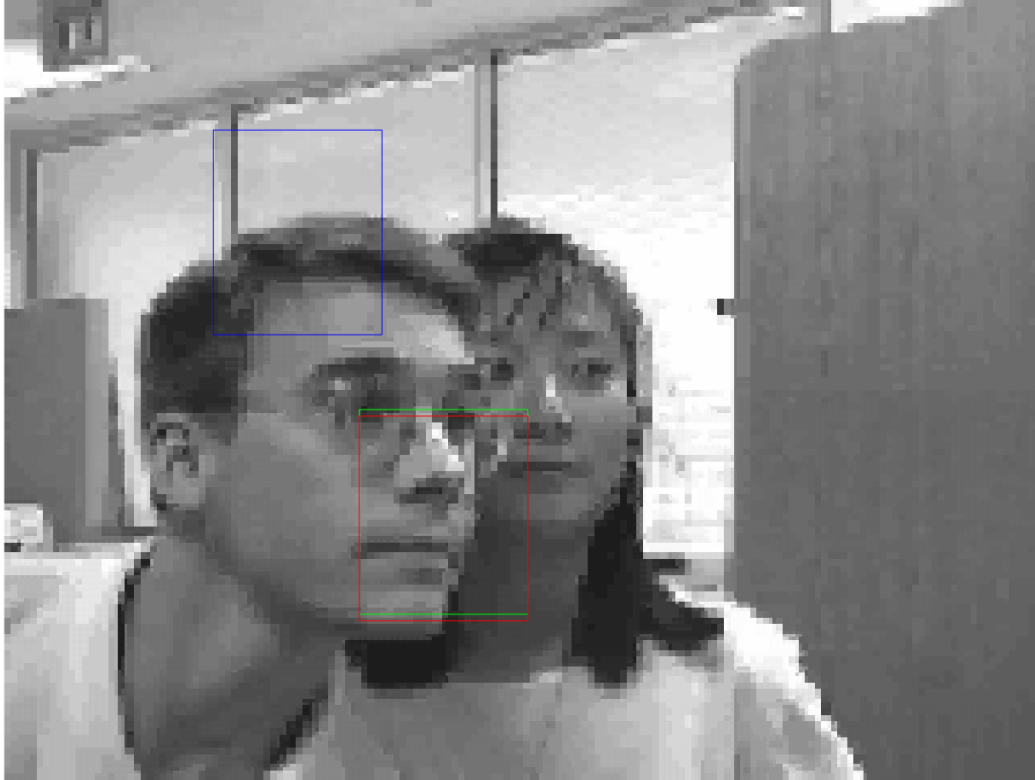
out of the back of the leaf, the CM tracking window(green one) was fixed on the leaf and the CM with occlusion tracking window(red one) could still track the target reasonably well.

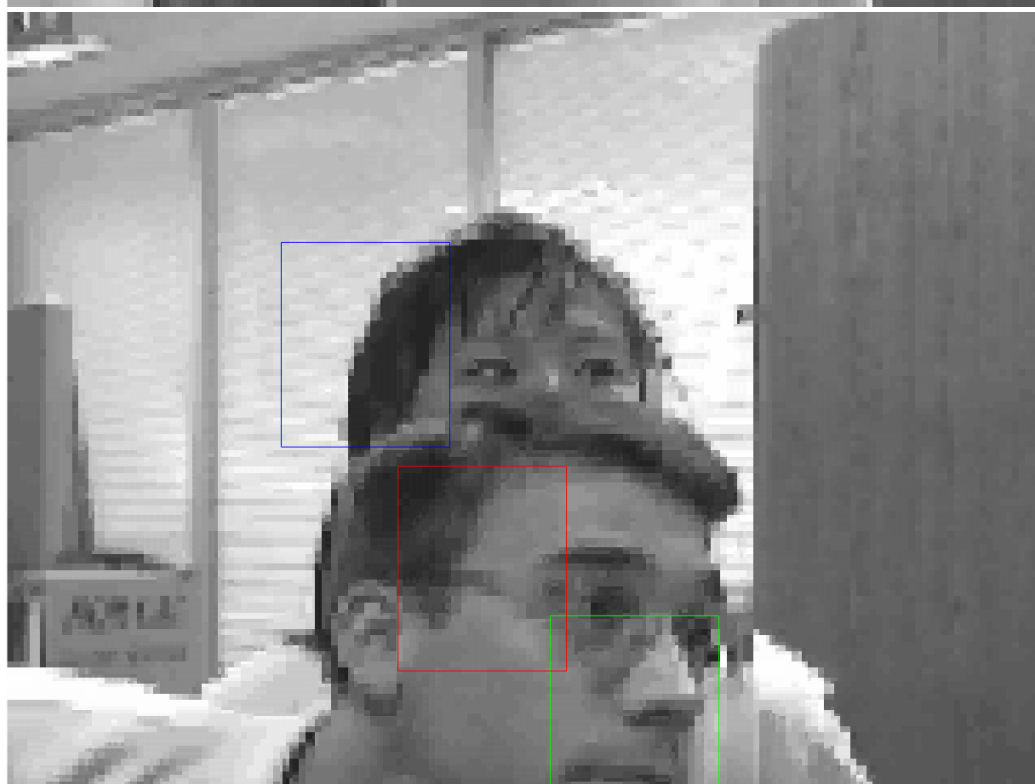


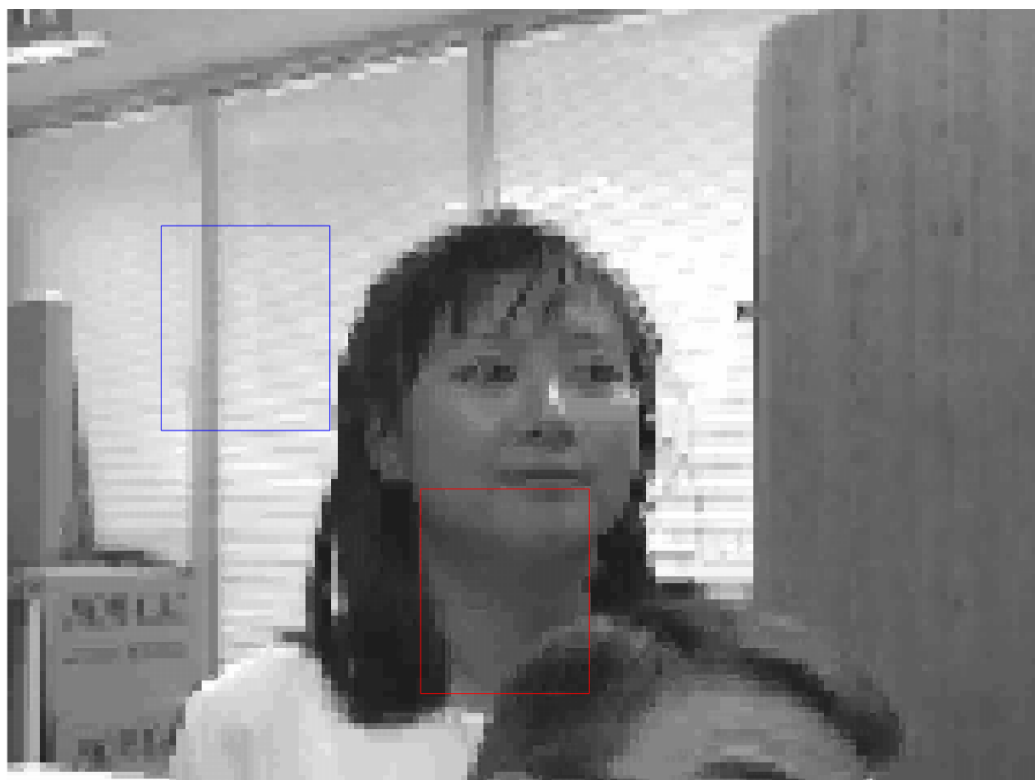




ii. Image set *girl*: There are six screenshots of the girl image set. At first, the man occluded the girl's face, but both CM and CM with occlusion tracked the target equally well(or bad). As the man occluded more of the girls face, the CM window(green one) started to follow the man's face and the CM with occlusion window(red one) kept at the same place. We can see as the man occluded even more of the girl's face and started to move out of the picture, the CM window followed the man and once disappeared from the picture. As the man came back, the CM window appeared again following the man's face while the CM with occlusion window still kept at the previous place.







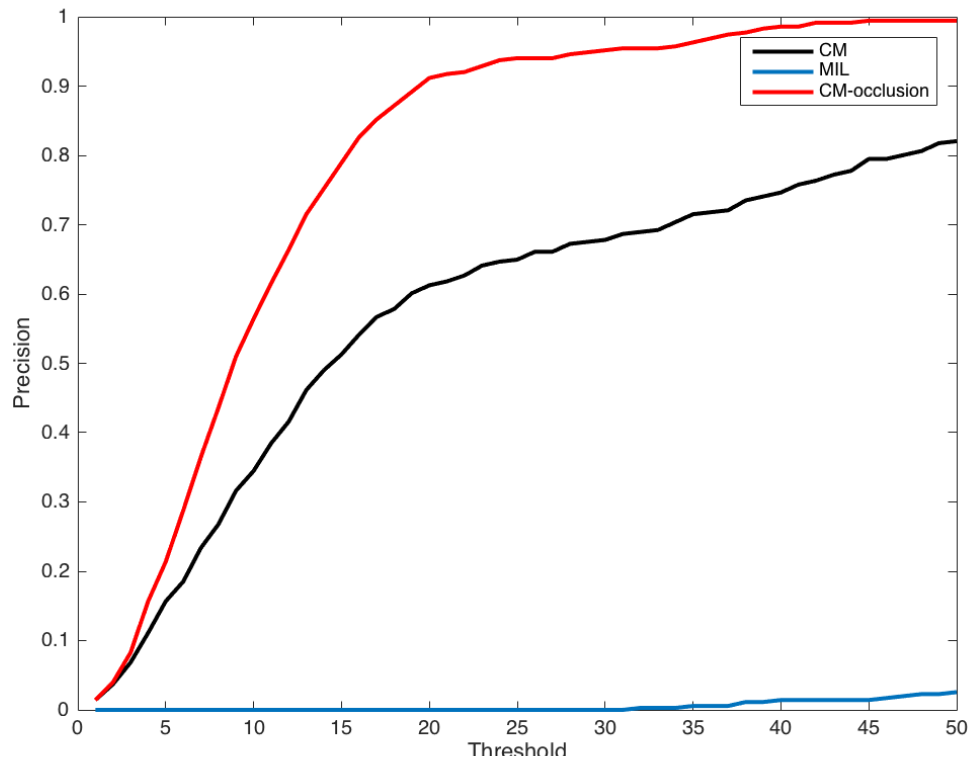


2. Comparison of precisions

There are three precision curves on the following image. The black curve represents the precision of the CM algorithm; the red curve represents the CM with occlusion algorithm; the blue curve represents the MIL algorithm. Our comparison focuses mainly on the comparison between the CM and CM with occlusion algorithms because the precision of MIL is very low.

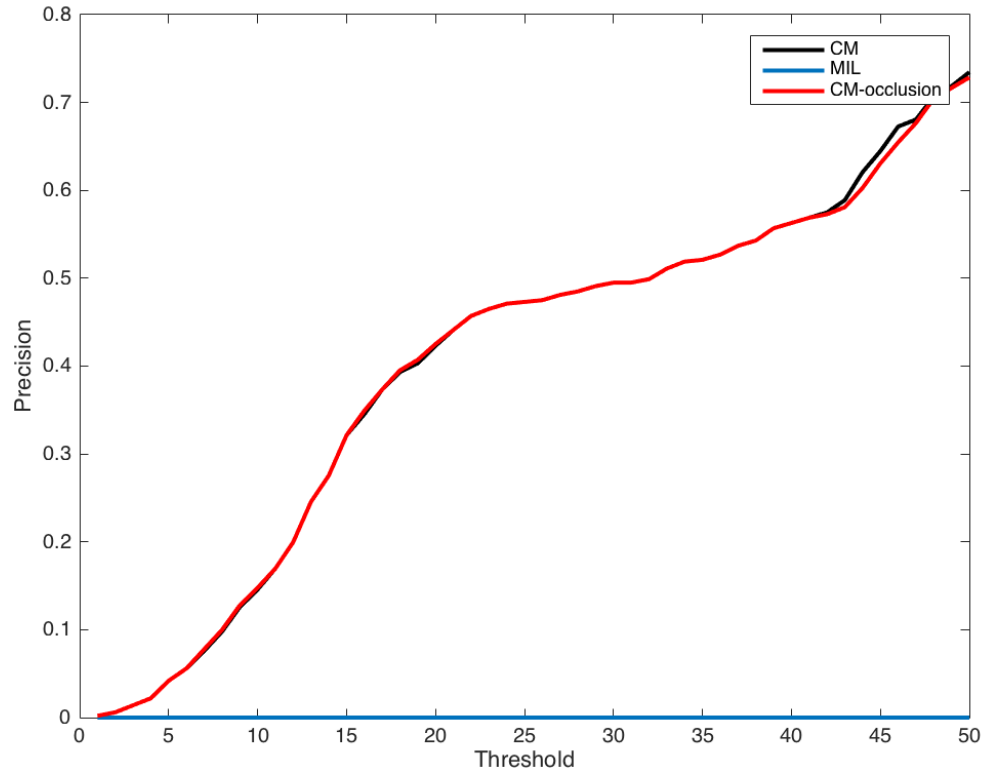
i. Image set *tiger1*

From the following figure we can see that the precision of the CM with occlusion is obviously higher than the CM, which means the former one does better job at tracking the target. This is what we expect from the tracking performance of the two methods.



ii. Image set *girl*

From the following figure we can see that CM and CM with occlusion basically performs equally well. But from the comparison of the tracking windows we can see that the latter method performs better than the former one. The same performance of the precisions is probably because even the CM with occlusion performs better, it doesn't completely follow the target and still has some deviation from the target, even though it's closer to the target compared to the CM window.



Conclusion

Overall speaking, we can clearly see that it really improve the precision of image matching, especially for tiger image set, since there are a lot of occlusion in that image set. CM tracker stay on that leaf once the tiger moves behind that leaf. However, with our occlusion detection, we will stop tracking that leaf, and predict where is the tiger should be, until we find tiger again. However, some quick moving blurring fools our detection system to indicate that is occlusion which is actually not. Generally speaking, with occlusion detection, it improve the ability to keep tracking target when it's really been covered, but it lose the performance to track the target when the target is not simply translated.

Appendix

Appendix 1: PSR

```
function PSR = getPSR(response, sidelobe)

%hide 11*11 matrix around [row,col]
gmax = max(response(:));
[row, col] = find(response == gmax, 1);

%set area around peak to 0
temp = (sidelobe - 1) / 2;
lt = max(1, row - temp);
lb = min(size(response,1), row + temp);
rt = max(1, col - temp);
rb = min(size(response, 2), row + temp);
response(lt:lb, rt:rb) = zeros(lb - lt + 1, rb - rt + 1);

%calculate the rest area
reconstructedRs = response(response~=0);
meanValue = mean(reconstructedRs);
sigmaValue = sqrt(var(reconstructedRs));
PSR = (gmax - meanValue)/sigmaValue;

end
```

Appendix 2: Hankel Matrix

```
function pos = hankelMatrix(hankelPos, lend)

n = min(5, floor(lend / 2 + 1));
hankelPos = hankelPos( lend - (2 * n - 3): lend, :);
A = zeros(2 * (n - 1), (n - 1));
b = zeros(2 * (n - 1), 1);
C = zeros(2, n - 1);

for row = 1: 2: 2 * (n - 1)
    for col = 1: n - 1
        index = (row - 1) / 2 + col;
        A(row:row+1, col) = hankelPos(index,:);
    end
end

for i = 1 :n - 1
    index = i + n - 1;
    C(:, i) = hankelPos(index,:);
    b(2 * i - 1: 2 * i, :) = hankelPos(index,:);
end
```

```

v = A \ b;
X = C * v;
pos = X';

```

```

end

```

Appendix 3: precision comparison and target comparison

```

% function rect_target(video_path, filePath, positions, figNum)
base_path = './tiger1';
video_path = choose_video(base_path);
figNum = 1;
file_CM = './tiger1/tiger_CM.txt';
file_MIL = './tiger1/tiger1_MIL_TR004.txt';
file_own = './tiger1/tiger_own.txt';

[img_files, ~, target_sz, ~, ground_truth, ~] = ...
    load_video_info(video_path);

color = ['g', 'r', 'b'];
[positions_CM, ~] = readingText(file_CM);
[positions_MIL, ~] = readingText(file_MIL);
[positions_own, ~] = readingText(file_own);

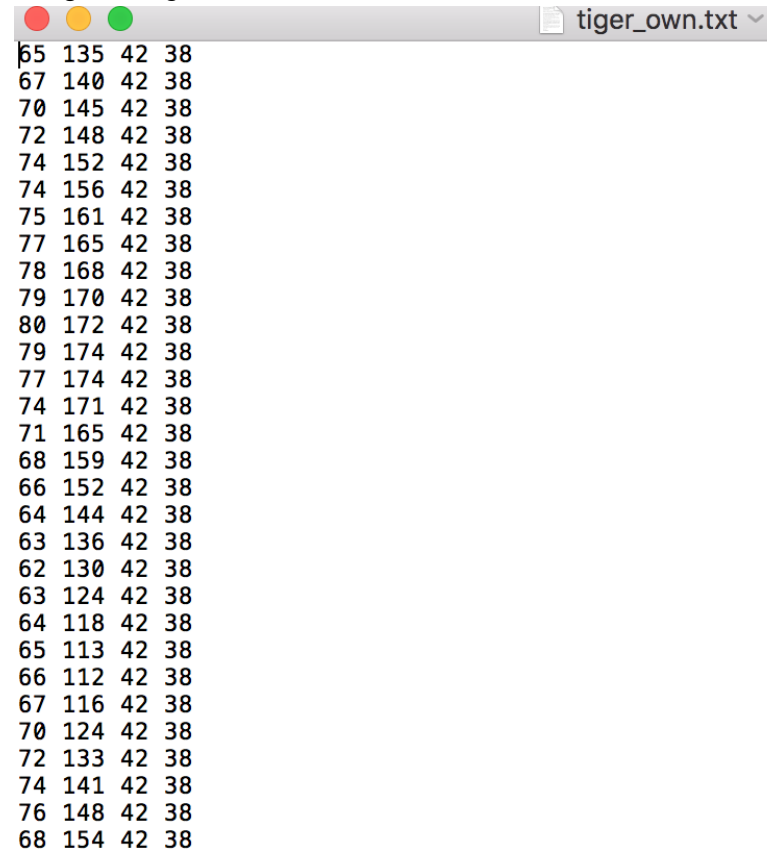
for frame = 1 : numel(img_files)
    pos1 = positions_CM(frame, :);
    pos2 = positions_own(frame, :);
    pos3 = positions_MIL(frame, :);
    im = imread([video_path img_files{frame}]);
    %visualization
    rect_position1 = [pos1([2,1]) - target_sz([2,1])/2, target_sz([2,1])];
    rect_position2 = [pos2([2,1]) - target_sz([2,1])/2, target_sz([2,1])];
    rect_position3 = [pos3([2,1]) - target_sz([2,1])/2, target_sz([2,1])];
    if frame == 1, %first frame, create GUI
        figure(2)
        im_handle = imshow(im, 'Border','tight', 'InitialMag',200);
        rect_handle1 = rectangle('Position',rect_position1, 'EdgeColor',color(1));
        rect_handle2 = rectangle('Position',rect_position2, 'EdgeColor',color(2));
        rect_handle3 = rectangle('Position',rect_position3, 'EdgeColor',color(3));
    else
        try %subsequent frames, update GUI
            set(im_handle, 'CData', im)
            set(rect_handle1, 'Position', rect_position1)
            set(rect_handle2, 'Position', rect_position2)
            set(rect_handle3, 'Position', rect_position3)
        catch %#ok, user has closed the window
            return
        end
    end
end
end

```

```
drawnow
% pause(0.05) %uncomment to run slower
end
```

Appendix 2: Output file samples:

Image set *tiger1*



65	135	42	38
67	140	42	38
70	145	42	38
72	148	42	38
74	152	42	38
74	156	42	38
75	161	42	38
77	165	42	38
78	168	42	38
79	170	42	38
80	172	42	38
79	174	42	38
77	174	42	38
74	171	42	38
71	165	42	38
68	159	42	38
66	152	42	38
64	144	42	38
63	136	42	38
62	130	42	38
63	124	42	38
64	118	42	38
65	113	42	38
66	112	42	38
67	116	42	38
70	124	42	38
72	133	42	38
74	141	42	38
76	148	42	38
68	154	42	38

Image set *girl*



```
108 180 63 52
108 180 63 52
108 182 63 52
108 182 63 52
108 186 63 52
106 192 63 52
102 202 63 52
98 208 63 52
96 216 63 52
98 222 63 52
100 230 63 52
100 236 63 52
102 240 63 52
104 240 63 52
108 232 63 52
112 214 63 52
106 196 63 52
106 180 63 52
102 172 63 52
102 158 63 52
102 142 63 52
106 132 63 52
106 116 63 52
108 100 63 52
108 92 63 52
108 84 63 52
112 82 63 52
112 86 63 52
110 88 63 52
110 88 63 52
```