Homework 3 :Harris Corner Detection and Matching

Kaiyun Xue

In the folder submitted, there are 6 function files and 1 main MATLAB code file. Every function has an algorithm corresponding. I will talk about every function's detail by the steps in the instruction.

1. Compute the image derivatives I_x and I_y for each pixel of image separately. Use $\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ to compute I_x and its transpose to compute I_y. Use these first order derivaties to compute I_x^2, I_y^2 and I_{xy} at each pixel.

In this step, I create a function:

```
function [ Derivatives ] = ComputeDerivatives( Img , operator)
%UNTITLED5 Summary of this function goes here
%   OPERATOR HAS TO BE 3*3
[x,y] = size(Img);
Img = double(Img);
operator = double(operator);
for i = 2:x-1
    for j = 2:y-1
        Derivatives(i,j) = operator(1,1)*Img(i-1,j-1)+operator(1,2)*Img(i-1,j)+operator(1,3)*Img(i-1,j+1)+...
                           operator(2,1)*Img(i,j-1)+operator(2,2)*Img(i,j)+operator(2,3)*Img(i,j+1)+...
                           operator(3,1)*Img(i+1,j-1)+operator(3,2)*Img(i+1,j)+operator(3,3)*Img(i+1,j+1);
    end
end
end
```

The 'Img' value is the input imagine. And the 'operator' is the operator temple matrix. Then the function will out put the derivatives of the pixel.

I also create the 'HarrisCornerDetectior' function.

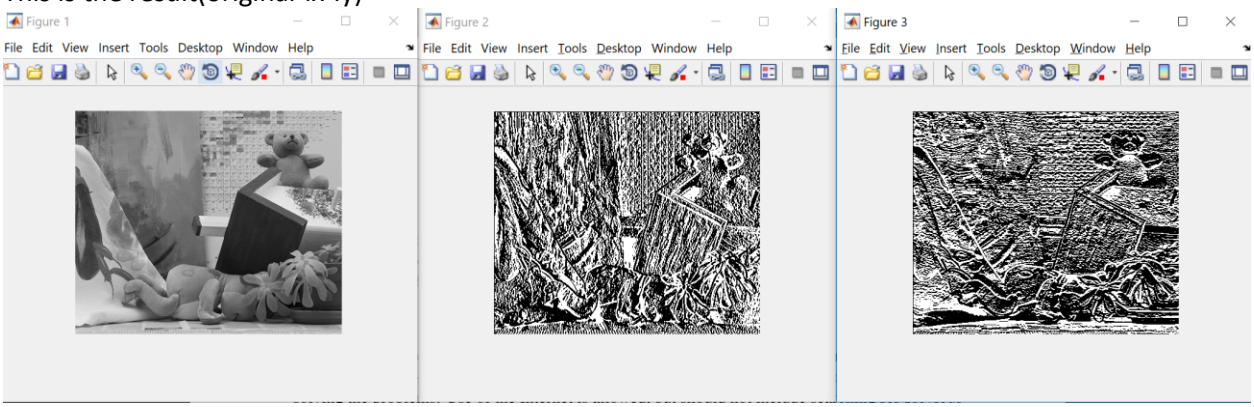
```
function [R_s, R ,M, NumOfCorner] = HarrisCornerDetectior( I,k,threshold)
```

'I' is the imagine, and k is the coefficient when calculating the R matrix. Threshold is the threshold of the corner detection. This function will output the information of corners.

For the first step, we use the front function and get the Ix and Iy.

```
figure;
imshow(I);
Ix = ComputeDerivatives(I,[-1 0 1;-1 0 1;-1 0 1;]);
Iy = ComputeDerivatives(I,[-1 -1 -1;0 0 0;1 1 1;]);
% figure;
% imshow(Ix);
% figure;
% imshow(Iy);
```

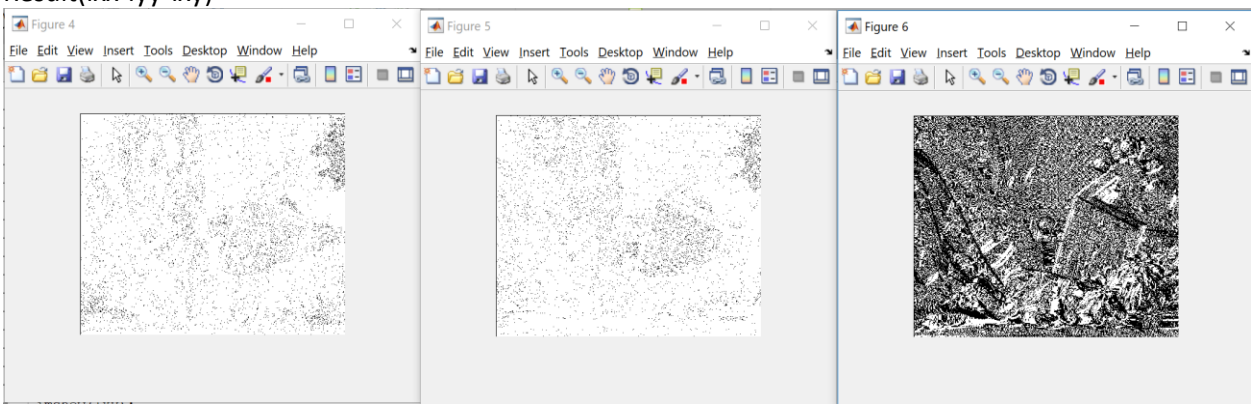
This is the result(original-Ix-Iy)



Then we get the Ixx, Ixy, Iyy from the Ix and Iy:

```
Ixx = Ix.*Ix;
Iyy = Iy.*Iy;
Ixy = Ix.*Iy;
% figure;
% imshow(Ixx);
% figure;
% imshow(Iyy);
% figure;
% imshow(Ixy);
```

Result(Ixx-Iyy-Ixy)



2. Apply Gaussian smoothing to the derivatives using the 5× 5 filter shown at `http://homepages.inf.ed.ac.uk/rbf/HIPR2/gsmooth.htm`. If for some reason, you are unable to do this, average the derivative values in 5× 5 windows centered at each pixel. Averaging will be penalized, but a filtering step is absolutely necessary to proceed.

In this part, I use the imfilter() function directly:

```
H = [1 4 7 4 1; 4 16 26 16 4; 7 26 41 26 41; 4 16 26 16 4; 1 4 7 4 1];
Sxx = imfilter(Ixx,H);
Syy = imfilter(Iyy,H);
Sxy = imfilter(Ixy,H);
```

3. Compute the Harris operator response function. Pick as corners all pixels above a threshold so that you select about 300-500 corners per image after the next step. 2000 is a good starting value for the threshold.

4. Apply *non-maximum suppression* on the responses of the Harris operator in 3×3 windows. This means that if a pixel does not have the maximum response in its 3×3 neighborhood, then it should not be included in the output. Make sure that the order you process pixels does not affect the output of this step.

Here is the new function:

```
function [R_s, R ,M,N] = GetHArrisMatrix( Sxx,Syy,Sxy,k,threshold)
```

Sxx, Syy,Sxy are the value we have got in the last step. K and threshold are the input value. In this part, the function output the M matrix cell, the number of the corners, and R matrix and R_s, the non-maximum suppression of R.

This is how I get the original R:

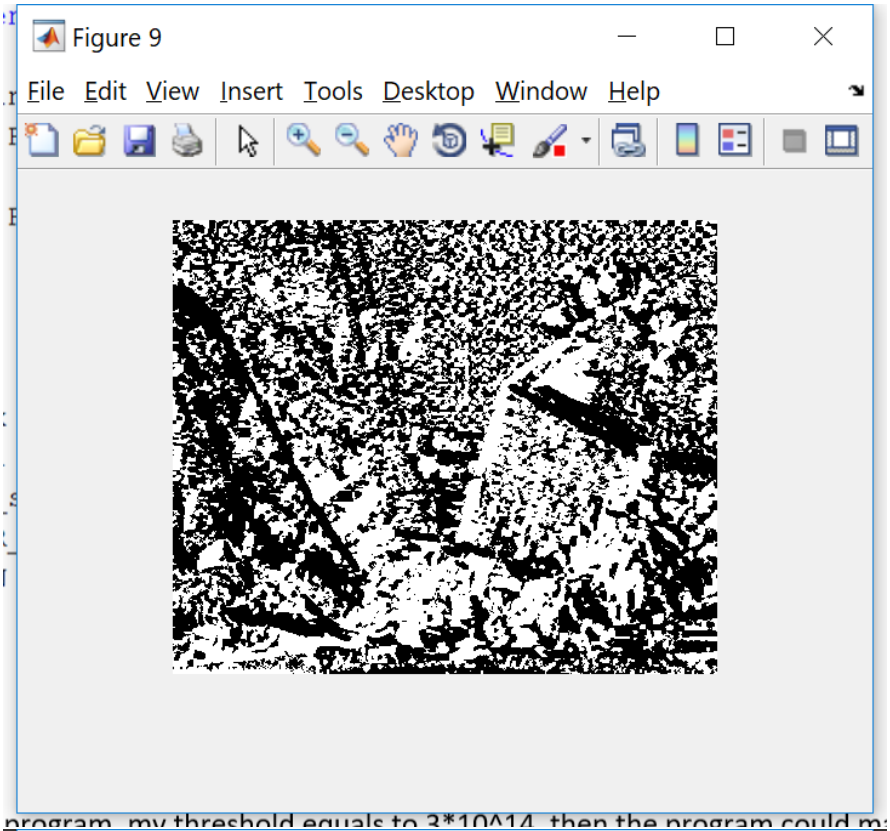
```
for i = 2 : x - 1
    for j = 2 : y - 1
        xx = 0; yy = 0; xy = 0;
        for wi = 1 : 3
            for wj = 1 : 3
                xx = double(xx) + double(Sxx(i-(2-wi),j-(2-wj)));
                xy = double(xy) + double(Sxy(i-(2-wi),j-(2-wj)));
                yy = double(yy) + double(Syy(i-(2-wi),j-(2-wj)));
            end
        end
        W(1,1) = xx; W(1,2) = xy; W(2,1) = xy; W(2,2) = yy;
        M{i,j} = W;
        R(i,j) = det(double(W))-(k*trace(double(W)))^2;
    end
end
```

This is the process we make the threshold and the maxium suppression:

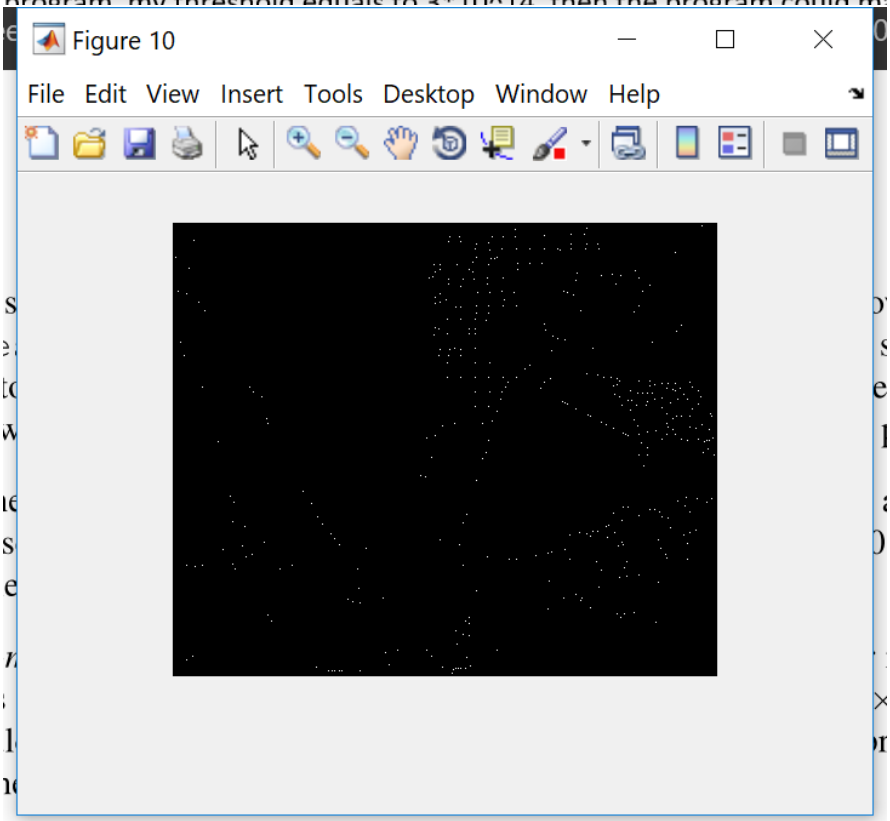
```
for i = 2 : x - 1
    for j = 2 : y - 1
        for wi = 1 : 3
            for wj = 1 : 3
                Win(wi,wj) = R(i-2+wi,j-2+wj);
            end
        end
        if Win(2,2)~=max(max(Win))
             R_s(i,j)=0;
        else
             R_s(i,j) = R(i,j);
        end
    end
end
N = x*y;
for i = 1 : x
    for j = 1 : y
        if R_s(i,j) < threshold
            R_s(i,j) = 0;
            N = N-1;
        end
    end
end
end
```

By the way, in this program, my threshold equals to 3*10^14, then the program could make a great result which the number of the corners will keeps in 300-500.

Result:



program, my threshold equals to 3*10^14, then the program could m



5. Compute the distance between every corner of the left image to every corner of the right image. IGNORE the epipolar constraint and use SAD in 3×3 windows for computing distances. If there are N corners in the left image and M corners in the right image there should be NM potential correspondences for which you should compute distance values.

In this part, I will talk about the main program.

Use the front function and calculate the corners of the left and the right imagine.

```
im_l = imread('teddyL.pgm');
im_r = imread('teddyR.pgm');
im_d = imread('disp2.pgm');
[Rs_l,R_l,M_l,n_l] = HarrisCornerDetectior(im_l,0.06,3*10^14);
[Rs_r,R_r,M_r,n_r] = HarrisCornerDetectior(im_r,0.06,3*10^14);
```

To record the corner information, such as number of the corners, positions.

```
for i = 1 : x
    for j = 1 : y
        if Rs_l(i,j) > 0
            vl{il,1} = [i,j];
            il=il+1;
        end
        if Rs_r(i,j) > 0
            vr{ir,1} = [i,j];
            ir=ir+1;
        end
    end
end
```

Now we could know the number of the corners in the left and the right imagine: il, ir. So we build a il*ir matrix to show their corresponding. Then calculate the SAD of each corresponding pair:

```
for i = 1 : n_l
    for j = 1 : n_r
        V{i,j} = vr{j,1}-vl{i,1};
        l = vl{i,1};
        r = vr{j,1};
        D(i,j) = (l(1,1)-r(1,1))^2 + (l(1,2)-r(1,2))^2;
        SADs(i,j) = GetSAD(Rs_l,vl{i,1},Rs_r,vr{j,1});
    end
end
```

In this part, I use the GetSAD function:

```
function [ sad ] = GetSAD( l,pl,r,pr )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
sad = 0;
for i = 1 : 3
    for j = 1 : 3
        sad = sad + abs(l(pl(1,1)-2+i,pl(1,2)-2+j) - r(pr(1,1)-2+i,pr(1,2)-2+j));
    end
end
end
```

6. Sort the distance values and select the top 5% most likely correspondences. Report the number of correct and incorrect correspondences included in this set. Use the ground truth disparity map and count small errors up to 1 pixel as correct. Repeat for the top 10% all the way to 100% in 5% increments.

In the step 6, I create the following function:

```
function [ g,b ] = CornerMatch( SADs,SADs_sorted,dstns,listL,GT, choosenRate)
%UNTITLED8 Summary of this function goes here
%   Detailed explanation goes here
g = 0;
b = 0;
[n_l,n_r] = size(SADs);
dsp = SADs_sorted(:,1:choosenRate*n_r);
[aa,bb] = size(dsp);
for i = 1 : aa
    for j = 1 : bb
        v = dsp(i,j);
        L = SADs(i,:);
        p = find(L == v);
        dis = sqrt(dstns(i,p));
        position = listL{i,1};
        disTruth = GT(position(1,1),position(1,2));
        if abs(dis - disTruth/4) < 1
            g = g + 1;
        else
            b = b + 1;
        end
    end
end
end
```

In this part of the code, I sort the sad and find the position to calculate the distance. And then match the distance with the ground truth. If the error is less then 1, then it is a good point, if not, it's a bad point.

In the main part of the project, I loop the CornerMatch for 20 time:

```
for i = 1 : 20
    [gp(i,1),bp(i,1)] = CornerMatch(SADs,SADs_sorted,D,vl,im_d,0.05*i);
    correctRate(i,1) = gp(i,1)/(gp(i,1)+bp(i,1));
end
```

Result:

Top percent(%)	Correct	Incorrect	Correct rate
5	515	8011	0.0604034717335210
10	993	16465	0.0568793676251575
15	1373	24611	0.0528402093596059
20	1730	33186	0.0495474853935159
25	2051	41797	0.0467752234993614
30	2347	50027	0.0448123114522473
35	2629	58677	0.0428832414445568
40	2923	67315	0.0416156496483385
45	3157	75607	0.0400817632420903
50	3369	84327	0.0384168035030104
55	3580	93048	0.0370493024796125
60	3745	101409	0.0356144321661563
65	3950	110136	0.0346230036989639
70	4137	118881	0.0336292249914647
75	4278	127266	0.0325214376938515
80	4489	135987	0.0319556365500157
85	4713	144695	0.0315444956093382
90	4977	152957	0.0315131637266200
95	5231	161635	0.0313485071854063
100	5541	170257	0.0315191299104654

From the Result, we could get that higher the top percent is, the lower the correct rate is. The correct correspondences are in the top of the SAD sequence mostly.