Visual Perception: Corner detection

Guillaume Lemaître
Heriot-Watt University, Universitat de Girona, Université de Bourgogne
g.lemaitre58@gmail.com

I. Introduction

In order to estimate 3-D positions of objects performing by triangulation method, critical points is to detect features present in two different views points of images. The actual and the most popular method is SIFT. However, corners detection allows to detect features with advantages to be more simple and computationally less expensive but nether more less robust. In this paper, we will introduce an implementation of the corner detector presented by Harris and Stephen [1]. We will present step by step, the way to detect corners.

II. Gradient Images

A corner is a point having low self similarity. Hence, the sum of the difference of intensities of the corner and his neighbourhood has to be important. Gradient images I_x and I_y characterize the difference of intensities. I_x describes an intensity variations in the x axis while I_y describes an intensity variations in the y axis. Initially, Harris and Stephen used the first gradients [1] as follow:

$$I_x = I * [-1, 0, 1]$$
$$I_y = I * [-1, 0, 1]^T$$

where I is an image, and $*$ is the operator of convolution. The problem of using this operator is that it is very sensitive to the noise. In this paper, Sobel and Prewitt operators have been implemented to correct this problem.

A. Sobel operator

1) Theory: Sobel operator is defined as follow:

1	0	-1
2	0	-2
1	0	-1

Table I
G_x: Sobel - Gradient in x

-1	-2	-1
0	0	0
1	2	1

Table II
G_y: Sobel - Gradient in y

Then, the following operations allow to compute the gradient images:

$$I_x = I * G_x$$
$$I_y = I * G_y$$

where I is an image, and $*$ is the operator of convolution and G_x and G_y are the operators defined previously.

2) Results: This function is implemented in the appendix A-B. Figure 1 presents an example of gradient images.

B. Prewitt operator

1) Theory: Prewitt operator is defined as follow:

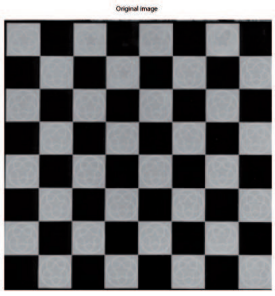
1	0	-1
1	0	-1
1	0	-1

Table III
G_x: Prewitt - Gradient in x

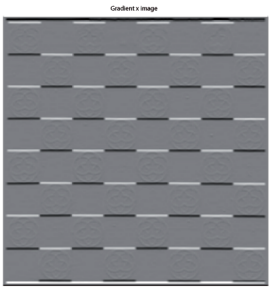
-1	-1	-1
0	0	0
1	1	1

Table IV
G_y: Prewitt - Gradient in y

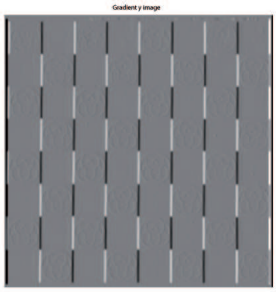
Then, the following operations allow to compute the gradient images:



(a) Original Image

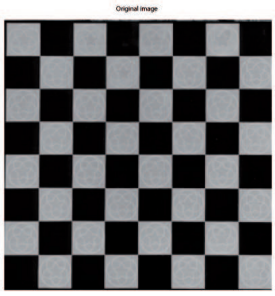


(b) Sobel: Gradient x

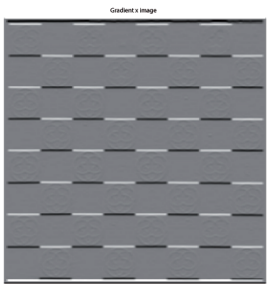


(c) Sobel: Gradient y

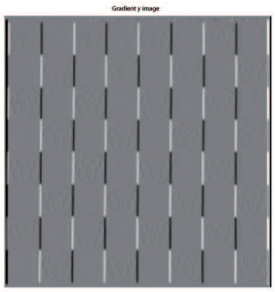
Figure 1. Gradient image using Sobel operator



(a) Original Image



(b) Prewitt: Gradient x



(c) Prewitt: Gradient y

Figure 2. Gradient image using Prewitt operator

$$I_x = I * G_x$$
$$I_y = I * G_y$$

where I is an image, and $*$ is the operator of convolution and G_x and G_y are the operators defined previously.

2) Results: This function is implemented in the appendix A-B. Figure 2 presents an example of gradient images.

III. Corner detection

A. Autocorrelation matrix

In the previous part, a corner was defined like a point having low self similarity. Similarity can be computed using the sum of squared difference. The sum of squared difference is defined as follow:

$$f(x,y) = \sum_{x_i,y_i[I(x_i,y_i)-I(x_i+\Delta x,y_i+\Delta y)]^2}$$

Using Taylor approximation we can expressed:

$$I(x_i+\Delta x, y_i+\Delta y) \equiv$$
$$I(x_i,y_i) + (I_x(x_i,y_i)I_y(x_i,y_i))(\Delta x \Delta y)^T$$

where $I_x(x_i,y_i)$ and $I_y(x_i,y_i)$ are the gradient images defined previously. Hence, the f function can be rewrite as:

$$f(x,y) = [\Delta x \Delta y] M [\Delta x \Delta y]^T$$

where:

$$M = \sum_{x,y} \begin{pmatrix} I_x^2(x_i,y_i) & I_{yx}(x_i,y_i) \\ I_{xy}(x_i,y_i) & I_y^2(x_i,y_i) \end{pmatrix}$$

In order to remove the noise, Harris and Stephen propose to use a smooth circular windows like a Gaussian [1]. Hence, the matrix M can be expressed as:

$$M = \sum_{x,y} w(x,y) \begin{pmatrix} I_x^2(x_i,y_i) & I_{yx}(x_i,y_i) \\ I_{xy}(x_i,y_i) & I_y^2(x_i,y_i) \end{pmatrix}$$

where $w(x,y)$ is the window smooth function.

Now, consider an image, a point p, a neighborhood w. Corners are detected locally. Hence, for a window centered on the point p, the matrix M will be expressed as follow:

$$M = \begin{pmatrix} \sum_w I_x^2 & \sum_w I_{yx} \\ \sum_w I_{xy} & \sum_w I_y^2 \end{pmatrix}$$

This matrix will be used to compute the response in order to detect corners. It will be computed for each pixel of the image.

B. Eigenvalues

1) Theory: This M matrix characterizes the structure of the grey level intensities. Eigen values can be used to describe the matrix M. The M matrix is a 2 by 2 matrix. Hence, the computation of Eigen values is defined by:

$$\lambda_{1,2} = \frac{1}{2}[Tra(M) \pm \sqrt{(Tra(M))^2 - 4(Det(M))}]$$
$$\lambda_{1,2} = \frac{1}{2}[I_x^2 + I_y^2 \pm \sqrt{(I_x^2 + I_y^2)^2 - 4(I_x^2 I_y^2 - I_{xy}^2)}]$$

Eigen values encode edge strength.

Considering a uniform image, no edge are detected, so λ_1 and λ_2 will be null.

Considering an image composed of a black and white step which is a perfect edge. One of the two λ will be null while the other λ will be superior to zero.

Considering an image composed of a perfect comer. Both λ will be superior to zero and have an important values.

To conclude, a corner is represented by two Eigen values superior to zero with the smaller Eigen value between λ_1 and λ_2 is large enough.

In the implementation, an R-image is created where for each pixel , M matrix is computed and the value of each pixel of the response is as follow:

$$p(x, y) = min(\lambda_1, \lambda_2)$$

2) Result: This function is implemented in the appendix A-F. We computed the response with an image of size 250x253 and a PC with an Intel Pentium Core Duo 1.5 GHz. The time of computation was **0.1935** seconds. Harris and Stephen introduce an approximation function that avoid to compute the Eigen values [1]. This method is faster but less accurate.

The R-image response is showing on figure 3.

C. Harris and Stephen corner/edge response function

Harris and Stephen, to avoid the computation of the Eigen values, purpose a function to approximate the previous function using a linear function based on the determinant and the trace of the matrix M. Hence,

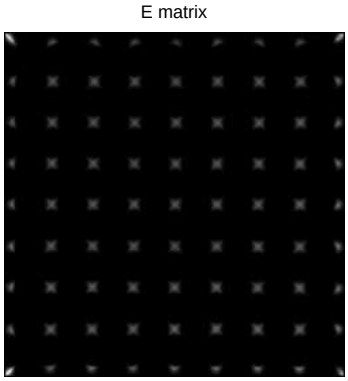


Figure 3. R-image response using Eigen values

$$Det(M) = \lambda_1 \lambda_2 = I_x^2 I_y^2 - I_{xy}^2$$
$$Tra(M) = \lambda_1 + \lambda_2 = I_x^2 + I_y^2$$

The formulation for the corner detection proposed by Harris and Stephen was:

$$R = Det(A) - k(Tra(A))^2$$

1) Results: This function is implemented in the appendix A-G. We computed the response with an image of size 250x253 and a PC with an Intel Pentium Core Duo 1.5 GHz. The time of computation was **0.0183** seconds. This method is 10 times faster than the previous one.

The R-image response is showing on figure 4.

IV. Non-maxima suppression

A. Presentation of the problem

After computed R-images E and R, corner detection is not performing. Figure 5 shows the results obtained using only the eighty first maximum of R-images.

It can happen that several maximum are detected near of the same corner. Figure 6 shows a representation of this phenomena.

In order to resolve this problem, an algorithm of non-maxima suppression has to be implemented. This algorithm has to remove all values around a corner and keep only the highest value of this region.

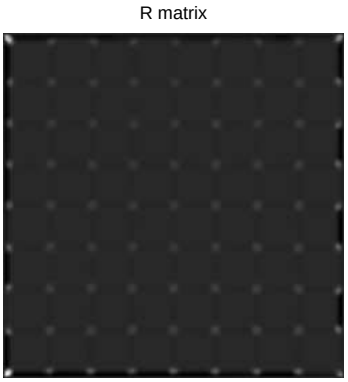
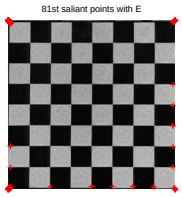
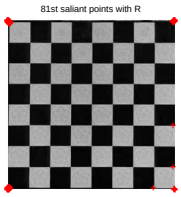


Figure 4. R-image response using Harris and Stephen approximation



(a) Corner detection using eighty first maximum of E R-image



(b) Corner detection using eighty first maximum of R R-image

Figure 5. Corner detection using the eighty first maximum of R-images

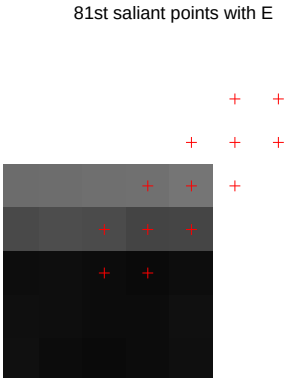


Figure 6. Detection of several points around the same corner

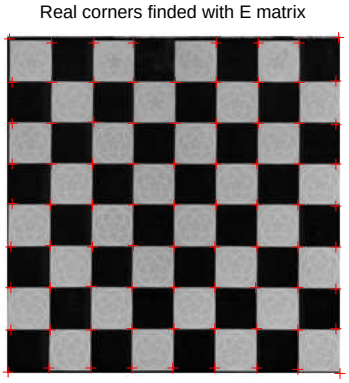


Figure 7. Corner detection after applied non-maxima suppression using the neighborhood

B. Non-maxima suppression using neighborhood

1) Theory: The first method implemented is an iterative method. The user put in parameter the number of points that he wishes detected. He has to define the size of the neighborhood where the algorithm will perform the suppression. Then the iterative processing is the following:

for (number of points defined) {

Detect the maximum of the R-image
Assign the neighborhood of this maximum to 0
Save the co-ordinates of this maximum
Assign the maximum to 0

}

2) Results: This function is implemented in the appendix A-I. We computed the response with an image of size 250x253 and a PC with an Intel Pentium Core Duo 1.5 GHz. The time of computation was **0.5213** seconds. Figure 7 presents results of the non-maxima suppression.

C. Non-maxima suppression using morphological transformation

1) Theory: The second method implemented is a method based on morphological transformation. This method is proposed by Peter Kosevi. The user has to define the size of the radius to operate the morphological operation. Then the iterative processing is the following:

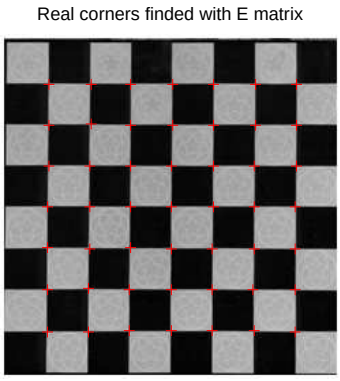


Figure 8. Corner detection after applied non-maxima suppression using morphological operation

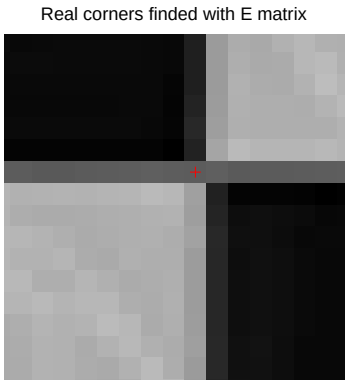


Figure 9. Corner detected without subpixel suppression

{

Apply a dilatation on the R-image using
the parameters given by the user
Create a border image to remove corner
near of the border
Create an image keeping only common point
between dilated image and R-image
Make *AND* operation between the
border image and the previous image

}

2) Results: This function is implemented in the appendix A-J. We computed the response with an image of size 250x253 and a PC with an Intel Pentium Core Duo 1.5 GHz. The time of computation was **0.9781** seconds. Figure 8 presents results of the non-maxima suppression.

V. Subpixel Accuracy

The last step after detected the good corners is to try to find the exact position of the corner. Figure 9 shows the result of a corner after non-maxima suppression. The algorithm of subpixel accuracy will permit to put the corner at the real position.

A. Theory

In order to correct the position of the corner, the neighborhood of each corner is used to approximate a parabolic crossing all points. Then, the derivative of this function gives the maximum of the parabolic function which is the real corner.

1) Approximation of the parabolic function: The general equation of a parabolic function is as follow:

$$p(x,y) = ax^2 + by^2 + cx + dy + exy + f$$

Least square method is used to resolve this equation. The problem can be formalized as follow:

$$AX = B$$

where:

$$X = [abcdef]^T$$

$$A = \begin{pmatrix} 1 & 1 & -1 & -1 & 1 & 1 \\ 0 & 1 & 0 & -1 & 0 & 1 \\ 1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & -1 & 1 & -1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$B = \begin{pmatrix} E(y-1,x-1) \\ E(y-1,x) \\ E(y-1,x+1) \\ E(y,x-1) \\ E(y,x) \\ E(y,x+1) \\ E(y+1,x-1) \\ E(y+1,x) \\ E(y+1,x+1) \end{pmatrix}$$

The vector X have to be computed, so the solution to find X is given by:

$$X = (A^T A)^{-1} A^T B$$

In order to find the offset for x and y axes, the partial derivative have to be computed:

$$\begin{array}{l} \frac{dp(x,y)}{dx} = 2ax + c + e \\ \frac{dp(x,y)}{dy} = 2by + d + e \end{array}$$

To find the corner, the maximum has to be find. Hence the partial derivative is null:

$$\begin{array}{l} \frac{dp(x,y)}{dx} = 2ax + c + e = 0 \\ \frac{dp(x,y)}{dy} = 2by + d + e = 0 \end{array}$$

$$\begin{array}{l} 2ax + e = -c \\ 2by + e = -d \end{array}$$

Hence, least square method is used to find the offset:

$$AX = B$$

where

$$A = \begin{pmatrix} 2x + e \\ 2y + e \end{pmatrix}$$

$$X = \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}$$

$$B = \begin{pmatrix} -c \\ -d \end{pmatrix}$$

Hence, to obtain the offset:

$$X = (A^T A)^{-1} A^T B$$

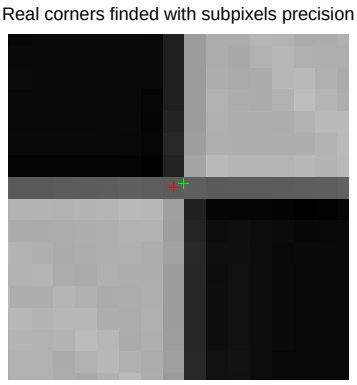


Figure 10. Correction using subpixel accuracy - green point: without subpixel precision, red point: without subpixel position

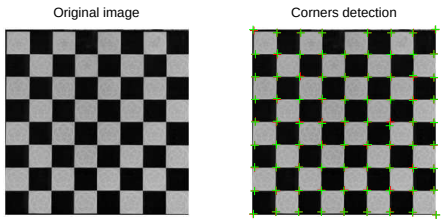


Figure 11. Correction using subpixel accuracy - green point: without subpixel precision, red point: without subpixel position

B. Results

This function is implemented in the appendix A-K. We computed the response with an image of size 250x253 and a PC with an Intel Pentium Core Duo 1.5 GHz. The time of computation was **0.0175** seconds. Figure 10 and 11 presents results of the correction with subpixel accuracy.

VI. Conclusion

In this paper, we presented an implementation of Harris & Stephen corner detector. We explain methods to find with more accuracy corners using first an algorithm of non-maxima suppression and secondly an algorithm to interpolate the subpixel accuracy.

REFERENCES

[1] C. Harris and M. Stephens, “A combined corner and edge detection,” in *Proceedings of The Fourth Alvey Vision Conference*, 1988, pp. 147–151.

APPENDIX A
CODE

A. Main function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Main file
%%% Harris corners detector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Matlab parameters
clc; clear all; close all;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Parameters

%%% Gaussian filter
window_size_gaussian_filter = 9;
sigma_gaussian_filter = 2;

%%% Prewitt Kernel
prewittx = [1 1 1; 0 0 0; -1 -1 -1];
prewitty = prewittx';

%%% Sobel Kernel
sobelx = [1 2 1; 0 0 0; -1 -2 -1];
sobely = sobelx';

%%% M Matrix
window_size_m_matrix = 3;

%%% R parameters
k_param = 0.04;

%%% Number of points to detect
nb_points = 81;

%%% Non-Maxima suppression
win_size_sub = 11;

%%% Radius morphological operation
radius = 8;

%%% Quadratic subpixels
win = 3;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% PART 0 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
tic
%%% Open an image
im_in = imread('chessboard01.png');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Parameters image
[height_im, width_im, depth_im] = size(im_in);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Initialisation
%%% Convert in gray level if necessary
if (depth_im ≠ 1)
    im_work = im2double(rgb2gray(im_in));
```

```
    disp('Conversion in grayscale')
else
    im_work = im2double(im_in);
end

figure;
subplot(1,3,1);
imshow(im_work);
title('Original image');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Step 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Compute image derivatives Ix Iy

[Iy,Ix] = derivativeImages(im_work,sobely,sobelx);
subplot(1,3,2);
imshow(mat2gray(Ix));
title('Gradient x image');
subplot(1,3,3);
imshow(mat2gray(Iy));
title('Gradient y image');

[Iysq_ns,Ixsq_ns,Iyx_ns,Ixy_ns] = productImages(Iy,Ix);

%%% Show derivative images
figure;
subplot(2,2,1);
imshow(mat2gray(Ixsq_ns));
title('Ixx image');
subplot(2,2,2);
imshow(mat2gray(Iysq_ns));
title('Iyy image');
subplot(2,2,3);
imshow(mat2gray(Ixy_ns));
title('Ixy image');
subplot(2,2,4);
imshow(mat2gray(Iyx_ns));
title('Iyx image');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Step 2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Generate Gaussian filter
g = fspecial('Gaussian', [window_size_gaussian_filter window_size_gaussian_filter]
, sigma_gaussian_filter);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% Step 3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[Iysq,Ixsq,Iyx,Ixy] = smoothProductImages(Iysq_ns,Ixsq_ns,Ixy_ns,g);
disp('Smooth applied')
%%% Show derivative images
figure;
subplot(2,2,1);
imshow(mat2gray(Ixsq));
title('Ixx image');
subplot(2,2,2);
imshow(mat2gray(Iysq));
title('Iyy image');
subplot(2,2,3);
imshow(mat2gray(Ixy));
title('Ixy image');
subplot(2,2,4);
imshow(mat2gray(Iyx));
title('Iyx image');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% PART 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
tic;

[Iysqsum,Ixsqsum,Iyxsum,Ixysum] = sumImages(Iysq,Ixsq,Iyx,window_size_m_matrix);

disp('Compute E matrix')
E = computeEmatrix(Iysqsum,Ixsqsum,Iyxsum,Ixysum,window_size_m_matrix);
disp('E matrix computed')

timeE_matrix = toc

figure;
subplot(2,1,1);
imshow(mat2gray(E));
title('E matrix');

tic;


disp('Compute R matrix')
R = computeRmatrix(Iysqsum,Ixsqsum,Iyxsum,Ixysum,window_size_m_matrix,k_param);
disp('R matrix computed')

timeR_matrix = toc

subplot(2,1,2);
imshow(mat2gray(R));
title('R matrix');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% PART 3 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

tic

[Evecsort.val,Evecsort.p_y,Evecsort.p_x] = findSalient(E,nb_points);

timeSaliantE_matrix = toc

%%% Display result
figure;
subplot(2,1,1);
imshow(im_work); hold on;
for i=1:nb_points
plot(Evecsort.p_x(i), Evecsort.p_y(i), 'r+');
end
title('81st saliant points with E');

tic

[Evecsort.val,Evecsort.p_y,Evecsort.p_x] = findSalient(R,nb_points);

timeSaliantR_matrix = toc

%%% Display result
subplot(2,1,2);
imshow(im_work); hold on;
for i=1:nb_points
plot(Evecsort.p_x(i), Evecsort.p_y(i), 'r+');
end
title('81st saliant points with R');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% PART 4 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Suppression of non-maxima
```

```
tic;

[Evecsort.val,Evecsort.p_y,Evecsort.p_x] = suppNonMaxima(E,nb_points,win_size_sub);

timeSup_matrix = toc

%%% Display results
figure;
imshow(im_work); hold on;
for i=1:nb_points
plot(Evecsort.p_x(i), Evecsort.p_y(i), 'r+');
end
title('Real corners finded with E matrix');

% tic;
%
% [Evecsort.val,Evecsort.p_y,Evecsort.p_x] = suppNonMaxima(R,nb_points,win_size_sub);
%
% timeSupR_matrix = toc
%
% %%% Display results
% figure;
% imshow(im_work); hold on;
% for i=1:nb_points
% plot(Evecsort.p_x(i), Evecsort.p_y(i), 'r+');
% end
% title('Real corners finded with R matrix');
%
% tic;
%
% [Evecsort.p_y,Evecsort.p_x] = suppNonMaxima2(E,radius);
%
% timeSup2E_matrix = toc
%
% %%% Display results
% figure;
% imshow(im_work); hold on;
% for i=1:size(Evecsort.p_x)
% plot(Evecsort.p_x(i), Evecsort.p_y(i), 'r+');
% end
% title('Real corners finded with E matrix');
%
% tic;
%
% [Evecsort.p_y,Evecsort.p_x] = suppNonMaxima2(R,radius);
%
% timeSup2R_matrix = toc
%
% %%% Display results
% figure;
% imshow(im_work); hold on;
% for i=1:size(Evecsort.p_x)
% plot(Evecsort.p_x(i), Evecsort.p_y(i), 'r+');
% end
% title('Real corners finded with R matrix');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% PART 5 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Find subpixels precision
%%% Compute coefficients

[Evecsort2.p_y,Evecsort2.p_x] = subPixelsAccuracy(R,Evecsort.p_y,Evecsort.p_x,win,nb_points);

timesubPixel = toc

%%% Display results
```

```
for i=1:nb_points
plot(Evecsort2.p_x(i), Evecsort2.p_y(i), 'g+');
end
title('Real corners finded with subpixels precision');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

B. derivateImages function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function to compute derivative images
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Iy, Ix] = derivativeImages(in_im, filtery, filterx)

disp('Compute derivative images')
Ix = conv2(in_im,filterx,'same');
Iy = conv2(in_im,filtery,'same');
disp('Derivative images computed')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

C. productImages function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function compute products image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Iysq, Ixsq, Iyx, Ixy] = productImages(Iy, Ix)

%%% Compute Ix x Ix non-smooth
Ixsq = Ix.*Ix;
%%% Compute Iy x Iy non-smooth
Iysq = Iy.*Iy;
%%% Compute Ix x Iy non-smooth
Ixy = Ix.*Iy;
%%% Compute Iy x Ix non-smooth
Iyx = Ixy;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

D. smoothProductImages function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function to smooth products images
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Iysq,Ixsq,Iyx,Ixy] = smoothProductImages(Iysq_ns,Ixsq_ns,Ixy_ns,filter)


%%% Compute Ix x Ix smooth
Ixsq = imfilter(Ixsq_ns,filter);
%%% Compute Iy x Iy smooth
Iysq = imfilter(Iysq_ns,filter);
%%% Compute Ix x Iy smooth
Ixy = imfilter(Ixy_ns,filter);
%%% Compute Iy x Ix smooth
Iyx = Ixy;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

E. sumImages function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%% Function to compute the sum images
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [Iysqsum,Ixsqsum,Iyxsum,Ixysum] = sumImages(Iysq,Ixsq,Iyx,window_size_m_matrix)

Ixsqsum = conv2(Ixsq,ones(window_size_m_matrix),'same');
Iysqsum = conv2(Iysq,ones(window_size_m_matrix),'same');
Iyxsum = conv2(Iyx,ones(window_size_m_matrix),'same');
Ixysum = Iyxsum;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

F. computeEmatrix function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function to compute the E matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [E] = computeEmatrix(Iysqsum,Ixsqsum,Iyxsum,Ixysum,window_size_m_matrix)

[height_dev,width_dev] = size(Iysqsum);

%%% Preallocation matrices
E = zeros(size(Iysqsum));
M = zeros(2);

for i = ((window_size_m_matrix - 1) /2)+1:height_dev-((window_size_m_matrix - 1) /2)
    for j = ((window_size_m_matrix - 1) /2)+1:width_dev-((window_size_m_matrix - 1) /2)
        M(1,1) = Ixsqsum(i,j);
        M(1,2) = Ixysum(i,j);
        M(2,1) = Iyxsum(i,j);
        M(2,2) = Iysqsum(i,j);

        landa1 = 0.5*(M(1,1) + M(2,2) + sqrt((M(1,1)+M(2,2))^2 - 4*(M(1,1)*M(2,2) - M(2,1)^2)));
        landa2 = 0.5*(M(1,1) + M(2,2) - sqrt((M(1,1)+M(2,2))^2 - 4*(M(1,1)*M(2,2) - M(2,1)^2)));
        eigv = [landa1 landa2];
        %eigv = eig(M);
        E(i,j) = min(eigv);
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

G. computeRmatrix function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function to compute R matrix
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [R] = computeRmatrix(Iysqsum,Ixsqsum,Iyxsum,Ixysum,window_size_m_matrix,k_param)

[height_dev,width_dev] = size(Iysqsum);

%%% Preallocate matrices
R = zeros(size(Iysqsum));
M = zeros(2);

for i = ((window_size_m_matrix - 1) /2)+1:height_dev-((window_size_m_matrix - 1) /2)
    for j = ((window_size_m_matrix - 1) /2)+1:width_dev-((window_size_m_matrix - 1) /2)
        M(1,1) = Ixsqsum(i,j);
        M(1,2) = Ixysum(i,j);
        M(2,1) = Iyxsum(i,j);
        M(2,2) = Iysqsum(i,j);

        detM = M(1,1)*M(2,2) - M(1,2)*M(2,1);
        traceM = M(1,1)+M(2,2);
```

```
        R(i,j) = detM - k_param*(traceM^2);
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

H. findSalient function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function to find salient point
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [val,p_y,p_x] = findSalient(matrix,nb_points)

%%% Select the best 81 saliant points
[height_dev,width_dev] = size(matrix);

%%% Convert matrix to vector to sort it
vector = matrix(:);

%%% Allocation of coordinates
Evecsort.val = zeros(nb_points,1);
Evecsort.p_x = zeros(nb_points,1);
Evecsort.p_y = zeros(nb_points,1);

[val, temp] = sort(vector,'descend');
[p_y,p_x] = ind2sub([height_dev,width_dev],temp);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

I. suppNonMaxima function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function to remove non-maxima using only the neighbourhood
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [val,p_y,p_x] = suppNonMaxima(matrix,nb_points,win_size_sub)


[height_dev,width_dev] = size(matrix);

vector = matrix(:);

%%% Allocation of coordinates
val = zeros(nb_points,1);
p_x = zeros(nb_points,1);
p_y = zeros(nb_points,1);

i = 1;
while (i < nb_points)

    %%% Maximum detection and co ordinates correspondances
    [maxnb,ind] = max(vector);
    [ind_y,ind_x] = ind2sub([height_dev,width_dev],ind);

    %%% New maximum
    val(i) = maxnb;
    p_y(i) = ind_y;
    p_x(i) = ind_x;

    %%% Remove neighbourhood
    for j = ind_y - (win_size_sub-1)/2:ind_y + (win_size_sub-1)/2
        for k = ind_x - (win_size_sub-1)/2:ind_x + (win_size_sub-1)/2
            if (j > 1)&&(j < height_dev)&&(k > 1)&&(k < width_dev)
                vector(sub2ind([height_dev,width_dev],j,k)) = 0;
```

```
            end
        end
    end
    %%% Remove the maximum
    vector(ind) = 0;
    i = i + 1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

J. suppNonMaxima2 function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function to remove non-maxima using morphologicql operation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [p_y,p_x] = suppNonMaxima2(matrix,radius)

sze = 2*radius+1;
morpho = ordfilt2(matrix,sze^2,ones(sze));

figure,imshow(mat2gray(morpho));
figure;

% Make mask to exclude points within radius of the image boundary.
bordermask = zeros(size(matrix));
bordermask(radius+1:end-radius, radius+1:end-radius) = 1;

% Find only maximum
matrix = (matrix==morpho) & bordermask;

[p_y,p_x] = find(matrix);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

K. subPixelsAccuracy function

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Function to compute corners with subpixels accuracy
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [p_y,p_x] = subPixelsAccuracy(matrix,p_y_in,p_x_in,win,nb_points)

[height_dev,width_dev] = size(matrix);

%%% Allocation of features
points = zeros(win*win,1);
coeff_matrix = zeros(win*win,6);
p_y = p_y_in;
p_x = p_x_in;

for i = 1:nb_points
    %%% Compute the neighbourhood
    neigh_ind = 1;
    for j = p_y_in(i) - (win-1)/2:p_y_in(i) + (win-1)/2
        for k = p_x_in(i) - (win-1)/2:p_x_in(i) + (win-1)/2
            if (j > 1)&&(j < height_dev)&&(k > 1)&&(k < width_dev)
                points(neigh_ind) = matrix(j,k);
                neigh_ind = neigh_ind + 1;
            end
        end
    end
    coeff_matrix(1,:) = [1,1,-1,-1,1,1];
    coeff_matrix(2,:) = [0,1,0,-1,0,1];
    coeff_matrix(3,:) = [1,1,1,-1,-1,1];
```

```
    coeff_matrix(4,:) = [1,0,-1,0,0,1];
    coeff_matrix(5,:) = [0,0,0,0,0,1];
    coeff_matrix(6,:) = [1,0,1,0,0,1];
    coeff_matrix(7,:) = [1,1,-1,1,-1,1];
    coeff_matrix(8,:) = [0,1,0,1,0,1];
    coeff_matrix(9,:) = [1,1,1,1,1,1];

    %%% Compute the coefficient
    coeff = (coeff_matrix'*coeff_matrix)^-1*coeff_matrix'*points;

    %%% Compute the shift position
    dev_1_mat = [2*coeff(1), coeff(5) ; coeff(5), 2*coeff(2)];
    dev_2_mat = [-coeff(3) ; -coeff(4)];

    shift = (dev_1_mat'*dev_1_mat)^-1*dev_1_mat*dev_2_mat;

    p_x(i) = p_x_in(i) + shift(1);
    p_y(i) = p_y_in(i) + shift(2);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

APPENDIX B
RESULTS

A. *Non-maxima suppression with neighbourhood*

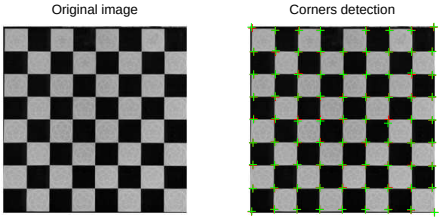


Figure 12. Results for *chessboard00.png*

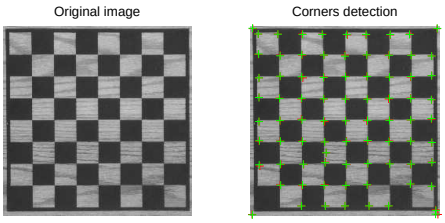


Figure 13. Results for *chessboard01.png*

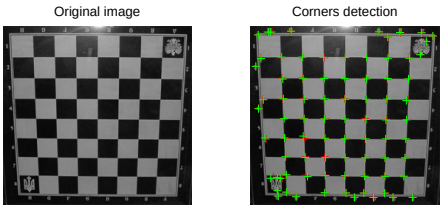


Figure 14. Results for *chessboard02.png*

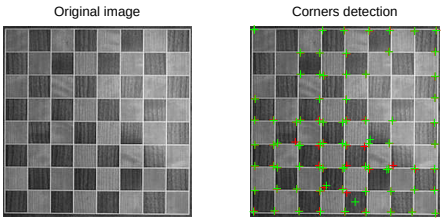


Figure 15. Results for *chessboard03.png*

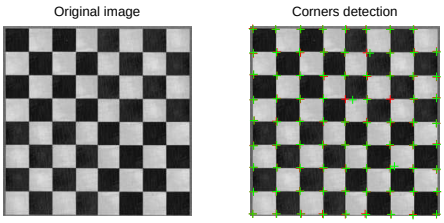


Figure 16. Results for *chessboard05.png*

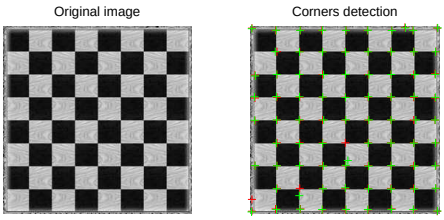


Figure 17. Results for *chessboard06.png*

B. Non-maxima suppression with morphological method

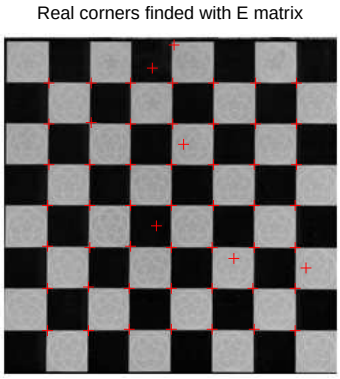


Figure 18. Results for *chessboard00.png*

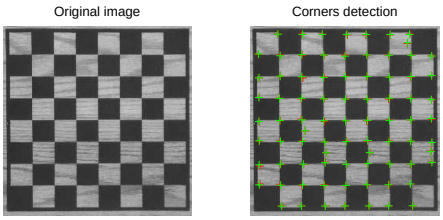


Figure 19. Results for *chessboard01.png*

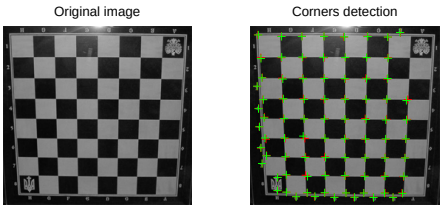


Figure 20. Results for *chessboard02.png*

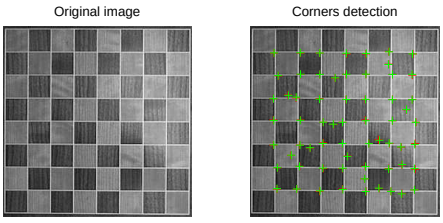


Figure 21. Results for *chessboard03.png*

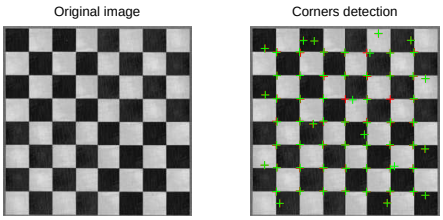


Figure 22. Results for *chessboard05.png*

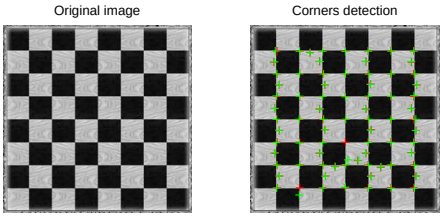


Figure 23. Results for *chessboard06.png*