Visual Perception
Corner Detection

Due on Monday, March 30, 2015

Pramita Winata

Contents

Lab 2 - Corner Detection
Pramita Winata

1 Introduction

Corners are the most reliable feature one can use to find the correspondence between images. This characteristic permits easy and fast identification in the image. Corner detection is an approach used within computer vision systems to extract certain kinds of features and infer the contents of an image. Corners detection allows to detect features with advantages to be more simple and computationally less expensive but also robust. In this lab, we will implement corner detector presented by Harris and Stephen [1].

2 Harris Corner Detection

Harris corner detector bases its detector on Moeavec's corner. Moravec determines that to find edges we need to observe the average changes in the intensity of the image that is the result from shifting the window in all directions by a small amount.

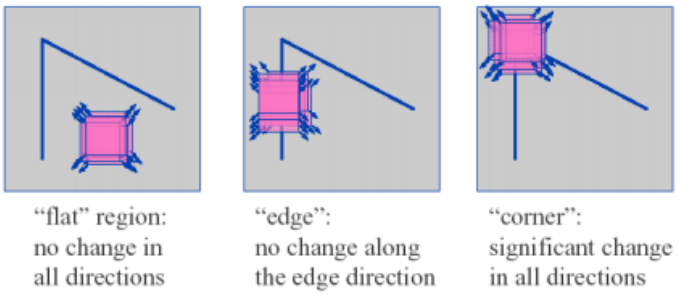


Figure 1: Harris Corner Detector - Basic Idea

The idea is to measure the local changes of the image. Using this information as starting point, if the window is moving in a constant area, the gradient is small. When the movement is over a line, the gradient is only in one direction, but when it is on an edge, it is clear that the gradient has a significant value in both axes. The formula can be formed as:

$$f(x,y) = \sum [w(x,y)[I(x+\Delta x, y+\Delta y) - I(x,y)]^2] \tag{1}$$

where,
$w(x,y)$ = window function
$I(x+\Delta x, y+\Delta y)$ = shifted intensity
$I(x,y)$ = intensity

We compute the gradient by calculating the derivates of the intensity in both directions, x and y.

$$I_x = \frac{\partial I}{\partial x}, I_y = \frac{\partial I}{\partial y}$$

Using Taylor expansion, we can approximate the shifted intensity, $[I(x + \Delta x, y + \Delta y)]$ as:

$$I(x + \Delta x, y + \Delta y) = I(x, y) + (I_x(x, y)I_y(x, y))(\Delta x \Delta y)^T \tag{2}$$

For small shifts $[\Delta x, \Delta y]$, rewritting in Eq. 1 using 2, we have a bilinear approximation:

$$f(x, y) = (\Delta x, \Delta y)M(\Delta x, \Delta y)^T \tag{3}$$

where M is a second moment matrix or autocorrelation matrix.

$$M = \sum_{x,y} \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} \tag{4}$$

To reduce the noise, we smooth the image derivatives by convolving them with a Gaussian filter g.

$$I_x = I_x * g, I_y = I_x * g \tag{5}$$

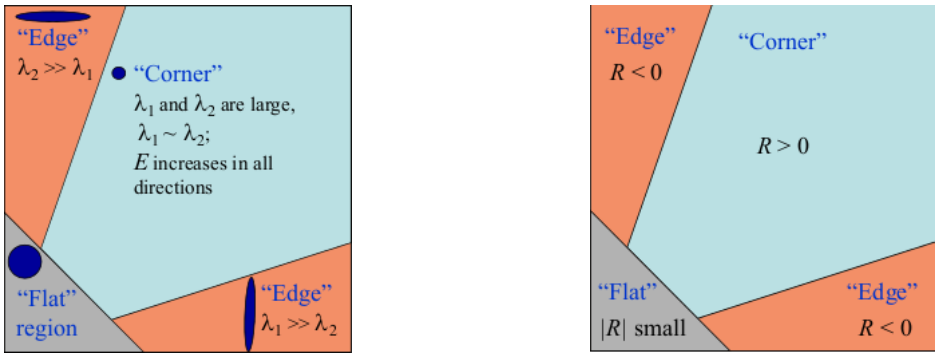
Considering an image with a point p and a neighborhood w. Corners are detected locally. Hence, for a window centered on the point p, the matrix M will be expressed as follow:

$$M = \sum_{x,y} w(x, y) \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} \tag{6}$$

$$M = \begin{pmatrix} \sum_w I_x^2 & \sum_w I_x I_y \\ \sum_w I_x I_y & \sum_w I_y^2 \end{pmatrix} \tag{7}$$

M can be used to derive a measure of "cornerness" for every pixel. Depending of the rank of M, the pixel belongs to an homogeneous region (rank M = 0), an edge (significant gradient in 1 direction, that is, rank M = 1), or a corner (significant gradients in both directions, thus, rank M = 2).

Eigenvalues from M can be used to extract the curvatures information of the local autocorrelation function as shown in Figure 2a.



(a) Eigenvalues (b) R

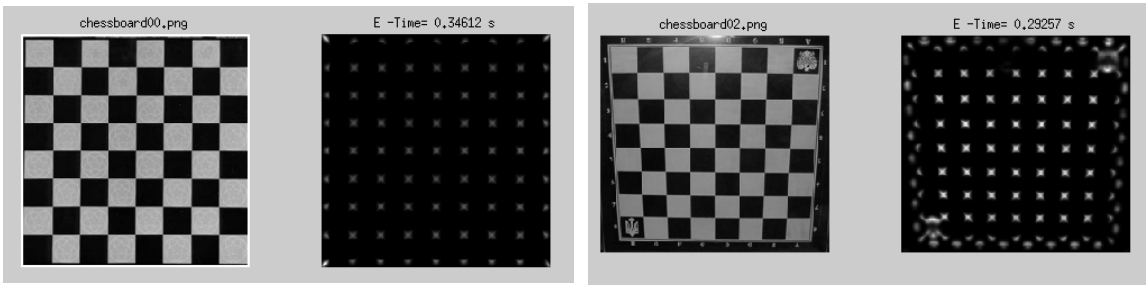
Figure 2: Edge detection

Since eigen decomposition is computationally expensive, Harris [1] purposes to utilize the determinant and the trace of M as the edge detector's components.

$$Det(M) = \lambda 1 * \lambda 2 = I_x^2 I_y^2 - I_{xy}^2$$

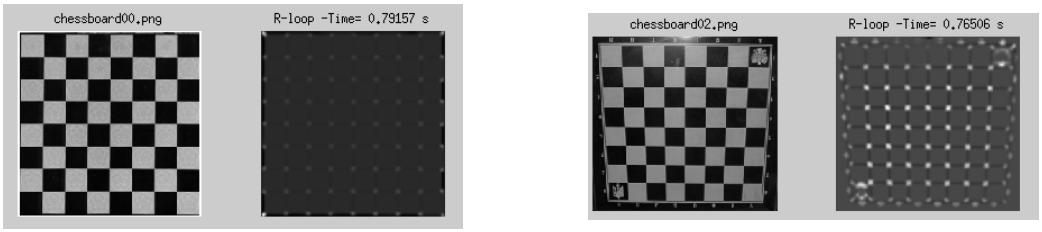
$$Trace(M) = \lambda 1 + \lambda 2 = I_x^2 I_y^2$$

$$R = Det(M) - k(Trace(M))^2 \tag{8}$$



(a) Chesboard 00 (b) Chessboard 02

Figure 3: E matrix



(a) Chesboard 00 (b) Chessboard 02

Figure 4: R matrix

Part one of the lab, deals with constructing the E matrix, which contains for every point the value of the smaller eigenvalue of M. In the next part, we compute the R matrix, based on the 8. Two examples results can be seen in Figure 3 and Figure 4. There is a clear difference in the resulting image of the matrices. The matrix E is the main result, and the matrix R is the approximation. However, it can be observed that they behave similiarly in detecting corners.

Since R matrix is purposed to reduce the computation time, it needs to be understood that the way of computing R matrix will have a huge impact. As shown in Table 1, calculating R matrix using an efficient coding, using Eq. 9, will have huge differences.The computation time is 99% faster.

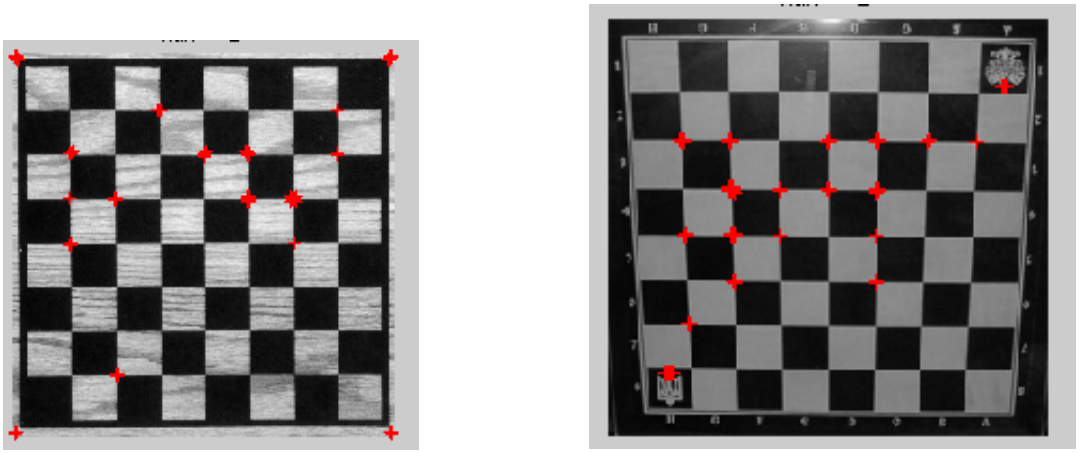
$$R = (I_{x2}. * I_{y2} + I_{xy}. * I_{xy}) - k * (I_{x2} + I_{y2}).^2 \tag{9}$$

Using E and R matrices, corner detection is done by selecting 81 first maximum values.The result for $Chessboard\_01$ and $Chessboard\_02$ are shown in Figure 5.

It can be seen that the results are not perfect. This is to due to the reason that one edge can be detected by many points resulting in overdetecting in one corner and underdetecting in others. One of the solution to overcome this is to non-maxima suppresion which will be discussed in Section 3.

Time (seconds)	**E**	**R**	**R-optimized**
Chessboard - 00	0.31648	0.70284	0.000367
Chessboard - 01	0.29083	0.63463	0.000275
Chessboard - 02	0.34614	0.76358	0.000367
Chessboard - 03	0.32033	0.71537	0.000395
Chessboard - 04	0.38334	0.81705	0.000379
Chessboard - 05	0.37506	0.8156	0.000679
Chessboard - 06	0.32794	0.71795	0.000391
Average	0.33715	0.73814	0.000407

Table 1: Computation time for matrix E and R



(a) Chesboard 01 Maxima suppresion (b) Chessboard 02 Maxima suppresion

Figure 5: Maxima suppresion

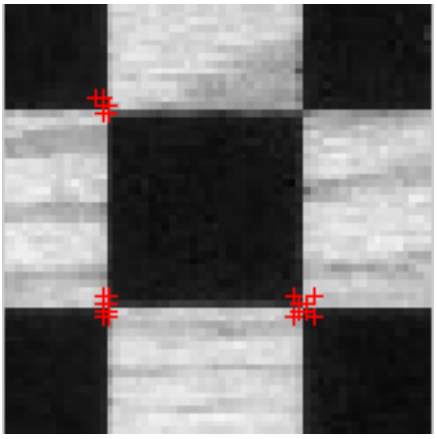


Figure 6: Overdetecting for one corner

3 Non Maxima Suppresion

Non-maximal suppresion is implemented to improve the detection. There are one parameter needs to be define in non maxima suppresion. The size of the neighborhood that wants to be considered, the window size. An optional parameter such as how many corners want to be detected also can be used. The algorithm of this is shown in Algorithm 1.

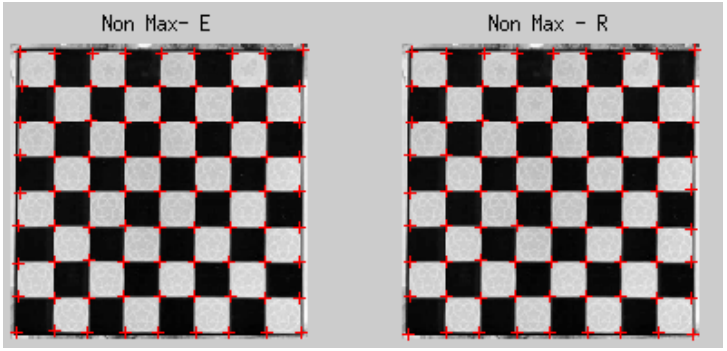


Figure 7: Non maxima suppresion Chessboard_04

Algorithm 1 Non maxima suppresion

```
procedure NON_MAXIMA_SUPPRESION(E, nbpints, neighborhood)
    [row_Esort, col_Esort] ← index of sorted E
    counter ← 1
    while counter < nbpoints do
        neighboors ← neighborhoodpixels
        for all row_E do
            Assign the neighboorsof E to 0
            Save the coordinates of this maximum
            Increase counter
            if counter > nbpoints then
                Break
            end if
        end for
    end while
end procedure
```

Perfect corner detection is obtained in Chessboard_04, Figure 7. However, not the case with Chessboard_01 and Chessboard_02, Figure 8 and Figure 9. This can be overcome by adjusting the window size. Figure 10 shows that if we increase the window size to become 35 the detection becoming better in Chessboard_02. Need to be understood that to get a better result, the window size chosen is highly depends on the nature of the image itself.

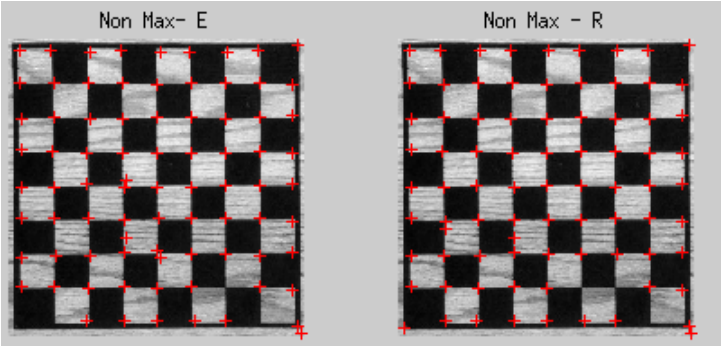


Figure 8: Non maxima suppresion Chessboard_01 - Window size 11

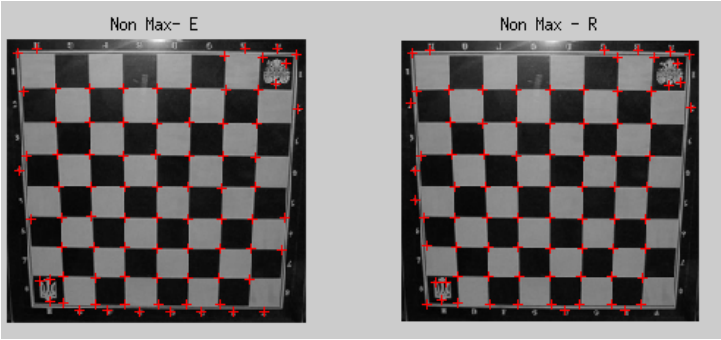


Figure 9: Non maxima suppresion Chessboard_02 - Window size 11

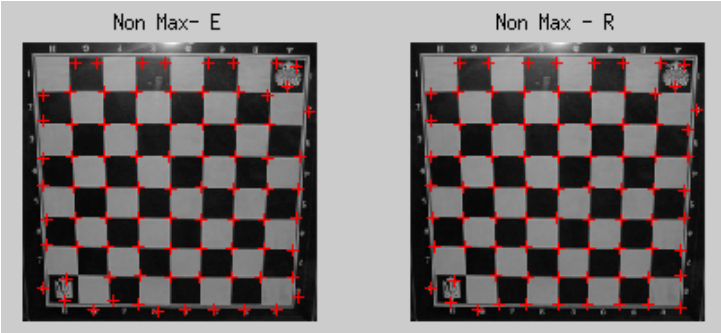


Figure 10: Non maxima suppresion Chessboard_02 - Window size 35

4 Subpixel Accuracy

Further improving the corner detection, we will apply subpixel accuracy in the detection process. We will based the method from [2]. In order to obtain the subpixel accuracy in the detection, the neighborhood of each corner is used to approximate a parabolic crossing all points. The derivatives of this will give the maximum of the parabolic (the corner).
General parabolic function is defined by :

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

We have six unknown parameters, so we need at least seven points to solve the equation. The 8 neighboors of the pixel is chosen to be these points. We have 9 known points, pixel evaluated and its 8-adjency points.

Applying least square method to solve the equation, the problem is written as:

$$AX = B$$

where:
A is the matrix containing the information of the coordinates from the equation.
X is the transformation matrix containing the coefficients that needs to be searched.
B is the pixel value of the coordinates.
The solution of X is given as:

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

Since we want to find the offset, we need to do partial derivation:

$$\frac{dp(x, y)}{dx} = 2ax + c + e = 0$$

$$\frac{dp(x, y)}{dy} = 2by + d + e = 0$$

Finding the corner or the maximum point, we need to make the derivatives to null.

$$\frac{dp(x, y)}{dx} = 2ax + c + e = 0$$

$$\frac{dp(x, y)}{dy} = 2by + d + e = 0$$

Then, we used least square method to compute the offset. Formulizing the problem into :

$$AX = B$$

where

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

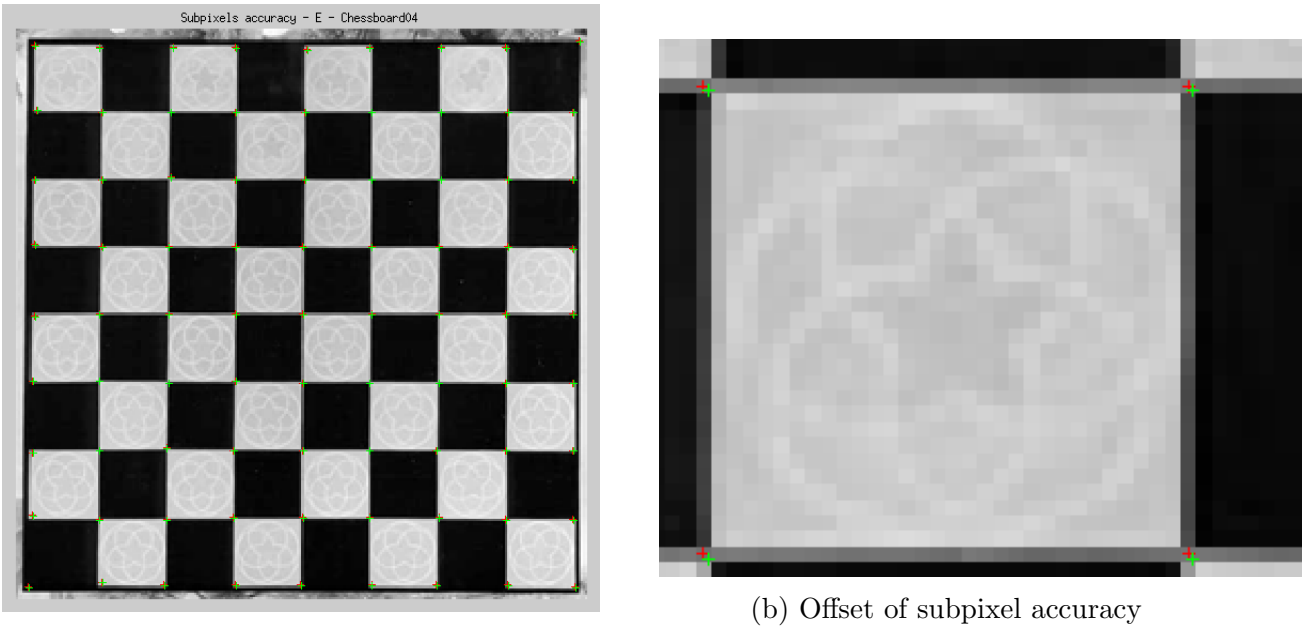
$$X = \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}$$

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

Solving the equation by :

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

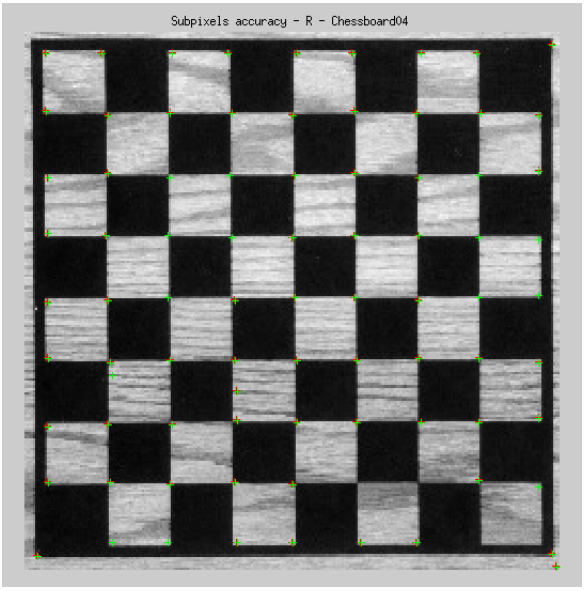
Resulting detection point using the pixel accuracy are shown in Figure 11 and Figure 12. It is noticed that there is a slight offset between the point with subpixel accuracy (green) and without subpixel accuracy (green).



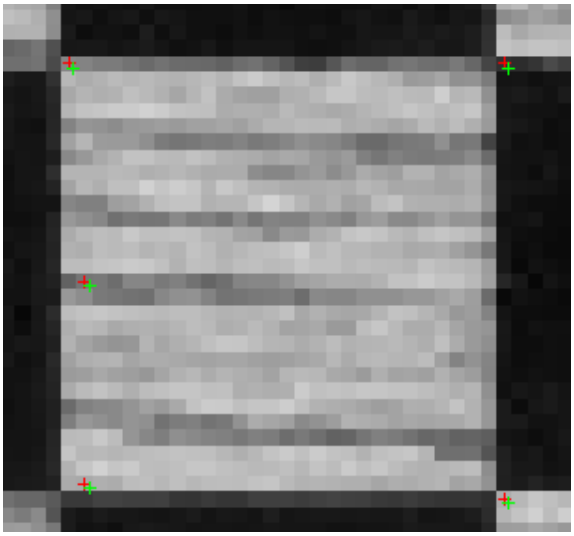
(a) Chessboard 4

(b) Offset of subpixel accuracy

Figure 11: Subpixel Accuracy with E matrix



(a) Chessboard 4



(b) Offset of subpixel accuracy

Figure 12: Subpixel Accuracy with R matrix

5 Conclusion

Harris corner detector is a powerful tool that permits identify corners in a fast and accurate form. We have elaborate the process on computing Harris corner detection and improved the result by applying non-maxima suppresion and subpixel accuracy.

6 Reference

References

[1] C. Harris and M. Stephens, "A combined corner and edge de- tection" *Proceedings of The Fourth Alvey Vision Conference, (1988)*, pp. pp. 147–151.

[2] Q Zhu, B Wu, N Wan, "A sub-pixel location method for interest points by means of the Harris interest strength" *The Photogrammetric Record, 22 (120) (2007)*, pp. 321–335.

Appendices

A Chessboard 00

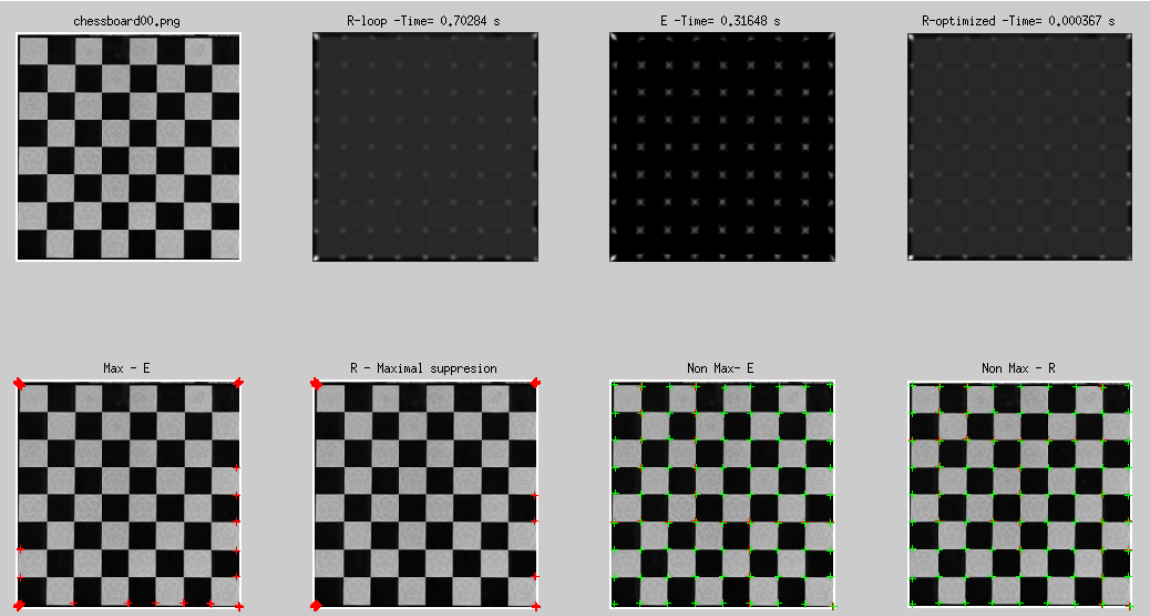


Figure 13: Chessboard 00 - Edge detection

B Chessboard 01

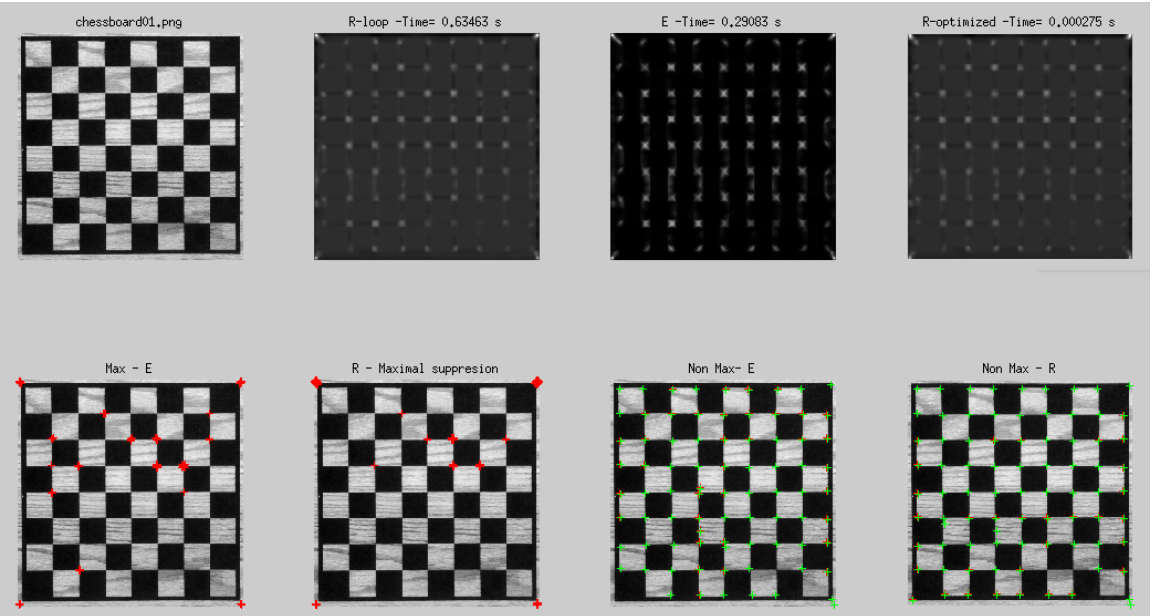


Figure 14: Chessboard 01 - Edge detection

C Chessboard 02

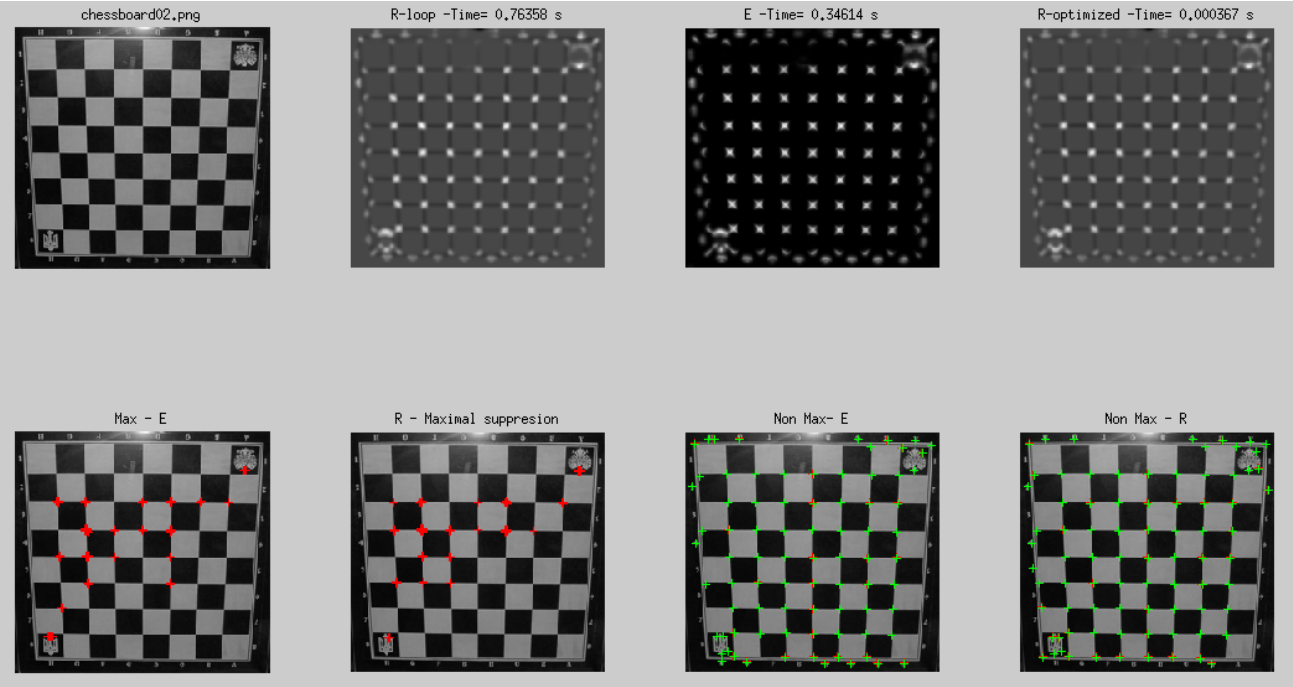


Figure 15: Chessboard 02 - Edge detection

D Chessboard 03

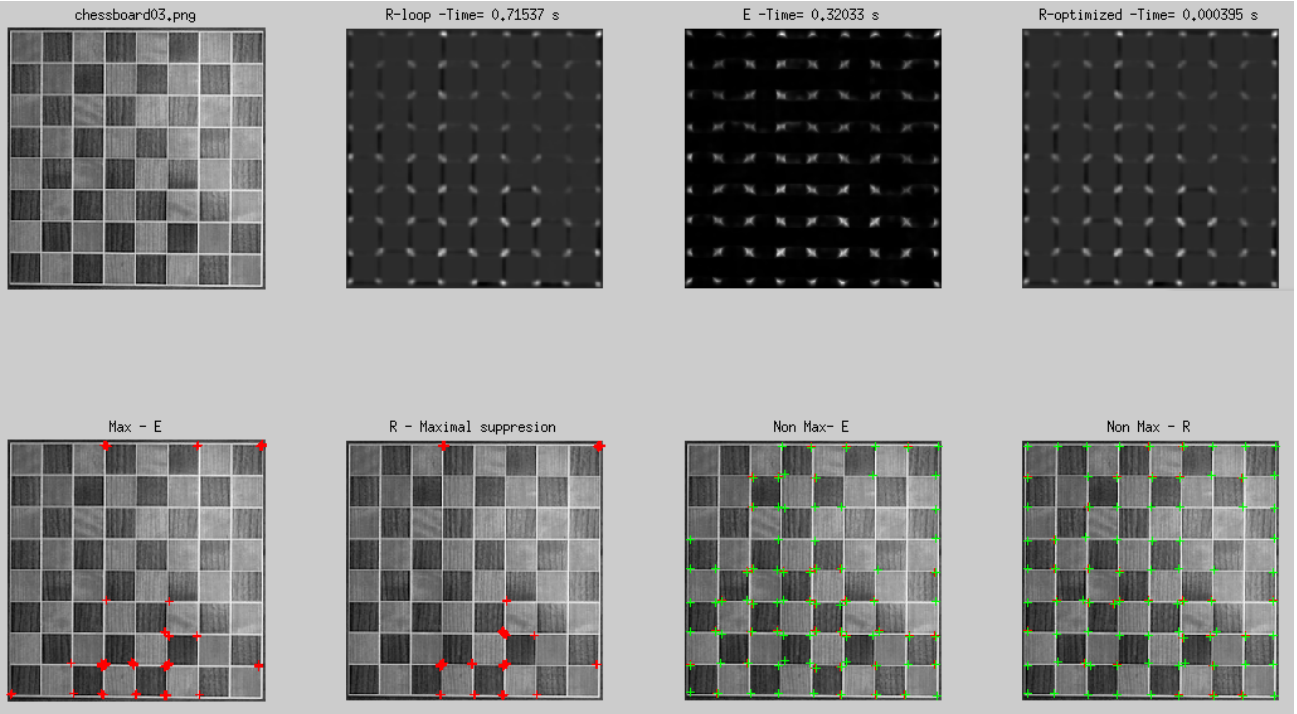


Figure 16: Chessboard 03 - Edge detection

E Chessboard 04

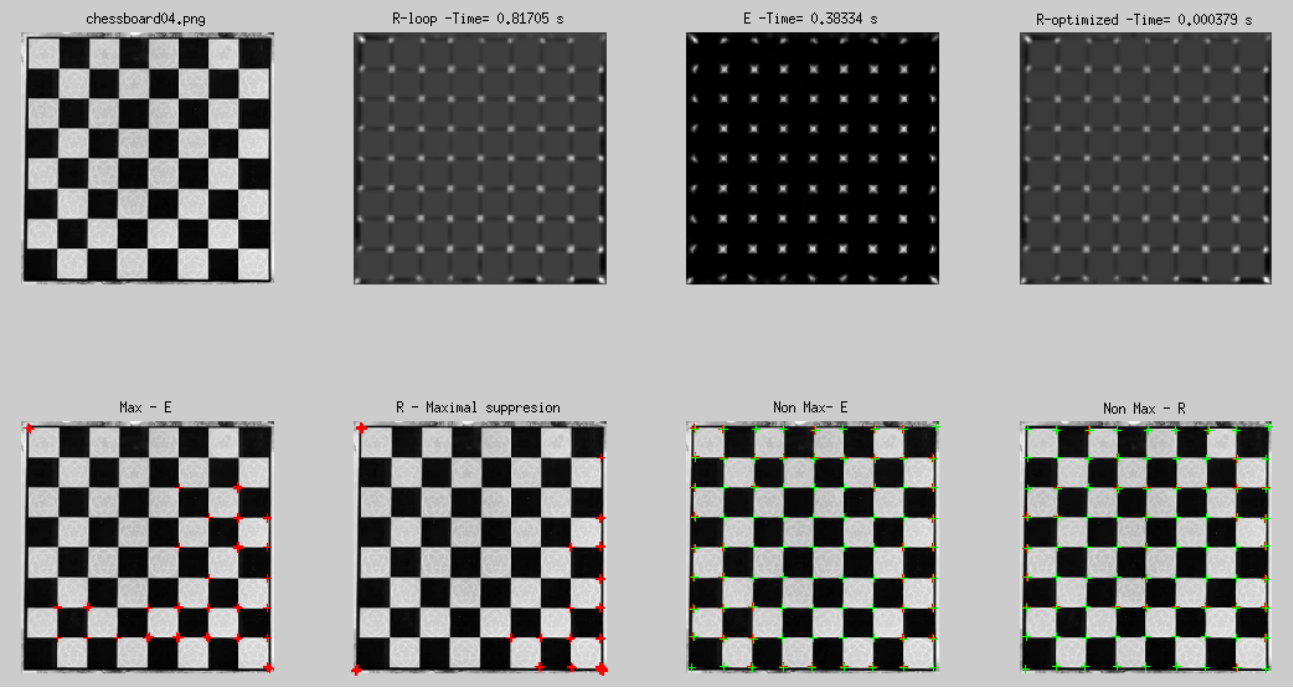


Figure 17: Chessboard 04 - Edge detection

F Chessboard 05

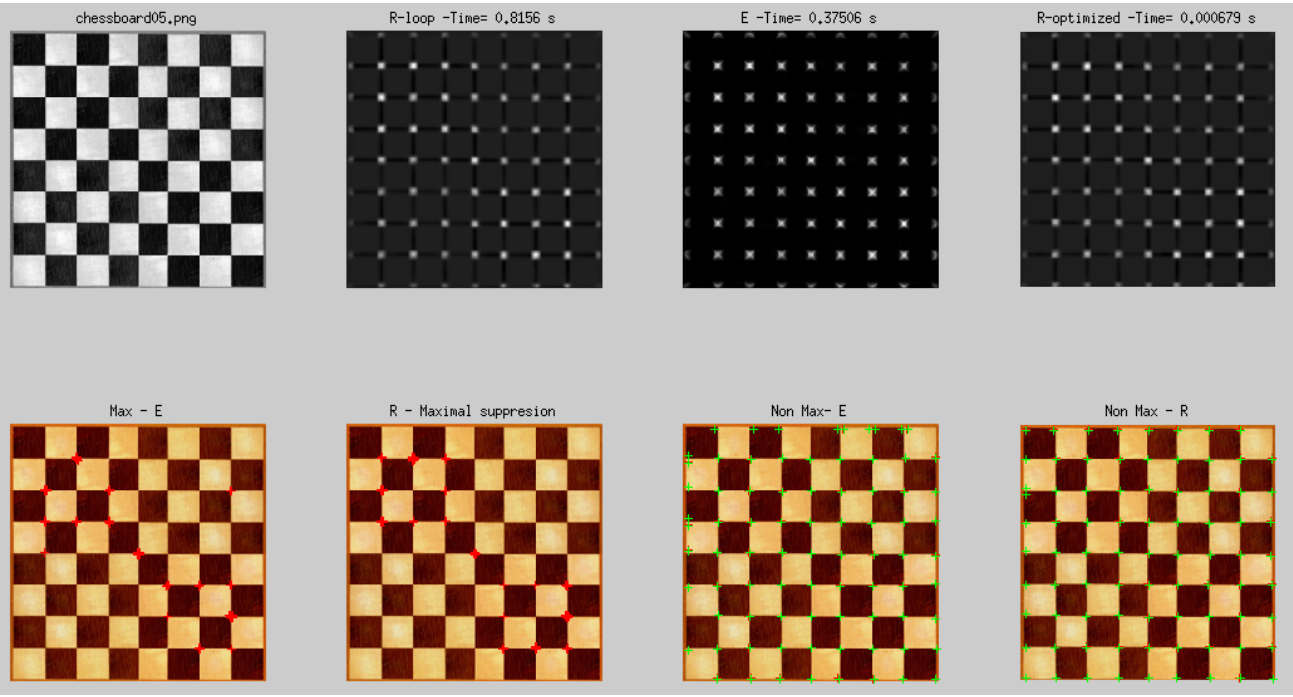


Figure 18: Chessboard 05 - Edge detection

G Chessboard 06

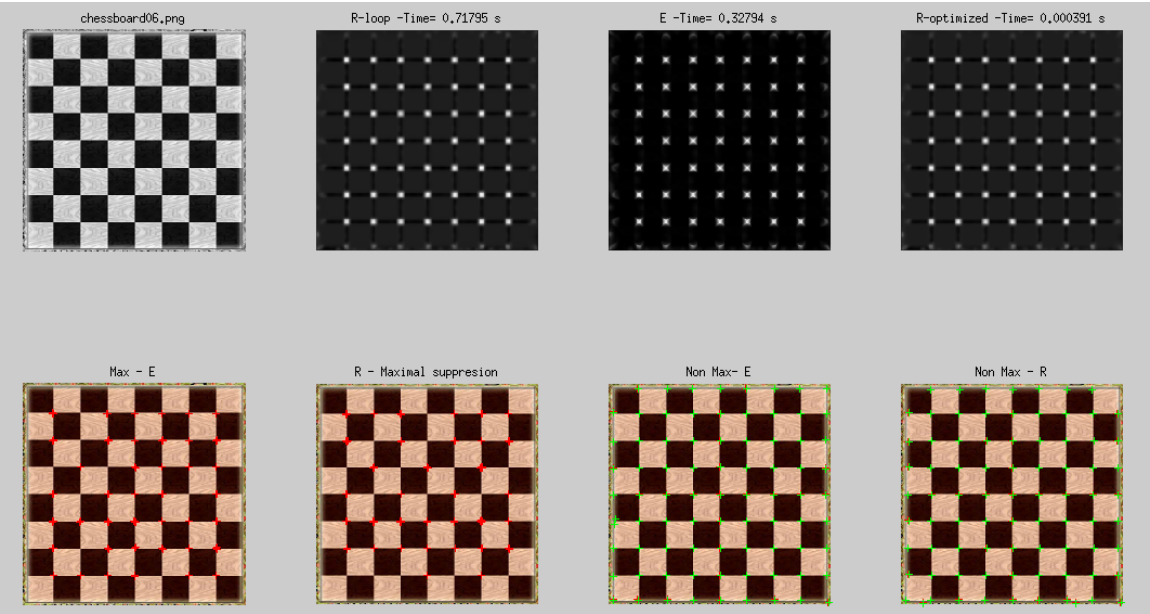


Figure 19: Chessboard 06 - Edge detection