HARRIS CORNER DETECTION
COMPUTER VISION

May 12, 2017

Partners: Evelyn Paiz & Nadile Nunes

Instructor: Alain Tremeau

OBJECTIVES

Learn how to detect corners in an image and understand the importance of the different steps involved in corner detection using the Harris algorithm.

INTRODUCTION

In this practical session, it was implemented the Harris corner detection. This method is based on the image gradient. A zone of interest such as a corner or an edge corresponds to a large change in appearance, impacting the value of the gradient. Harris corner detection applies a mathematical approach to this property in order to detect corners.

For each pixel, the "cornerness" can be estimated using the eigenvalues of the autocorrelation matrix M, defined by:

$$M = \begin{pmatrix} \sum W\langle {I_x}^2 \rangle & \sum W\langle I_x I_y \rangle \\ \sum \langle I_x I_y \rangle & \sum W\langle {I_y}^2 \rangle \end{pmatrix} \quad (1)$$

$$\langle {I_x}^2 \rangle = g \otimes {I_x}^2 \quad (2)$$

Where W is a 3×3 neighbourhood around the pixel, g a Gaussian filter and $\otimes$ the convolution operator.

The eigenvalues of $M(\lambda_1, \lambda_2)$ gives a classification of the image points. If λ_1 and λ_2 are small, the area is flat. If one of the eigenvalues is very small compared to the other, then the area corresponds to an edge. Finally, if λ_1 and λ_2 are both large and of similar magnitude, the area is a corner region.

Harris and Steffens suggested using the determinant and trace of M to detect corners, which removes the need to compute eigenvalues:

$$R = \det(M) - k \cdot tr(M)^2 \qquad (3)$$

Where k is a constant (with a typical value of 0.04), $\det(M) = \lambda_1 \lambda_2$ the determinant and $tr(M) = \lambda_1 + \lambda_2$ the trace of a square matrix.

R depends only on M's eigenvalues. R is large for a corner, R is negative with a large magnitude for an edge and $|R|$ is small for a flat region.

The implementation of the method is as follows:

- Compute the image gradient for each pixel in x and y directions: $I_x = \frac{\partial I}{\partial x}, I_y = \frac{\partial I}{\partial y}$.
- Compute ${I_x}^2$, ${I_y}^2$ and $I_x I_y$.
- Smooth the square image derivative with a Gaussian filter: $\langle {I_x}^2 \rangle = g \otimes {I_x}^2$.
- For each pixel, compute the autocorrelation matrix M defined in equation (1). Finally, compute R using equation (3).

If the matrix R is directly used for corner detection, each zone of interest might be detected several times. Redundancies are generally corrected using a "non-maximal suppression" algorithm (NMS), which consists of setting a pixel to zero if there is a higher value in its neighborhood. A threshold can be applied to improve the detection.

PRACTICAL SESSION

The Harris corner detection was implemented based on the method explained in the Introduction section. For the coding of all the method, the image 'chessboard00.png' was used.

Then similar images were used to compared the final result. The following results were achieved:

Part 1: Image derivatives and smooth filter

The first step was reading the image to be processed. Because it was possible to use different set of images, the following operations were done as a pre-processing step of the image:

- Verify if the image is RGB. If this is the case, the image is converted to grayscale.
- Increase the image contrast.
- Convert the image data onto values between $[0, 1]$.

Now, with the image pre-processed, the following steps were possible to be executed:

1. *Computation of image derivatives I_x and I_y:*

For each direction (x, y) a filter was used to compute the partial derivative. The convolution of the image with a gradient mask (dx, dy) gave us the partial derivative on the x and y direction (respectively). These values were saved as the I_x and I_y.

$$dx = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad dy = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad (4)$$

As seen from (4), the transpose of dx was dy, so only one definition was done of the filters.

2. *Gaussian filter:*

Here a Gaussian filter was necessary to be generated of size 3×3 and standard deviation of 2. We knew that a Gaussian filter is defined by

$$g = \frac{1}{2\pi\sigma} e^{\frac{-x^2}{2\sigma^2}} \qquad (5)$$

but for this case, matlab provided us the 'fspecial' function to generate the filter easier. This was applied to the set of images of $I_x \cdot I_x$, $I_y \cdot I_y$ and $I_x \cdot I_y$. Since filtering was also a convolution, after the operations mentioned a convolution was made between the Gaussian filter of the previous step and the result of each multiplication.

3. *Initial results:*

The results of this part are shown on figure 1. All the images displayed were converted to a grayscale image to be able to see the results correctly.

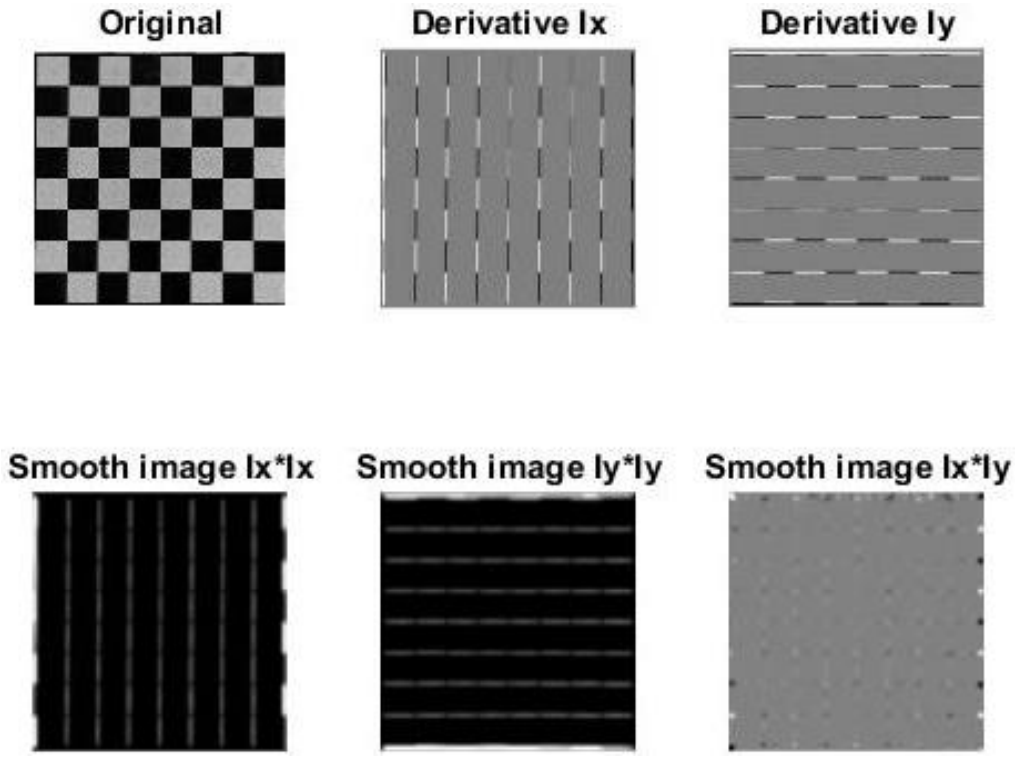

Figure 1: *Derivatives and smoothed images from part 1 for chessboard00.png*

Analyzing these results, on the derivative images and mostly on the images after smoothing operation it's possible to notice that the outer part of those images was more salient than other parts, being brighter than other regions. Even on the smooth image of $I_x \cdot I_y$, the difference is not so apparent but it is still possible to notice that the dots on the borders of the image are more visible than the others.

Depending of the way the getting of the derivative was implemented, the subsequent results may vary, with the results being visible for us to analyze. In the end this approach was the best from our point of view for this implementation.

Part 2: Matrix E

On this second part, the matrix E was calculated. This matrix basically finds the difference in intensity for a displacement of a (u, v) in all directions. It is based on the eigenvalues of the autocorrelation matrix M (equation 1) of each pixel. A corner is characterized then by a large variation of E in all direction of the vector (x, y).

$$E(u,v) \approx [u\ v]M\begin{bmatrix}u\\v\end{bmatrix} \qquad (6)$$

To fill the matrix E, it was considered a windows size of 3×3. For each pixel, the matrix M was computed with its eigenvalues, where the smallest eigenvalue went to E. Since the window size was 3×3, this meant that the outside part of the picture was not going to be considered. This means that the first and last row and column of E will be zero regardless of the other variables.

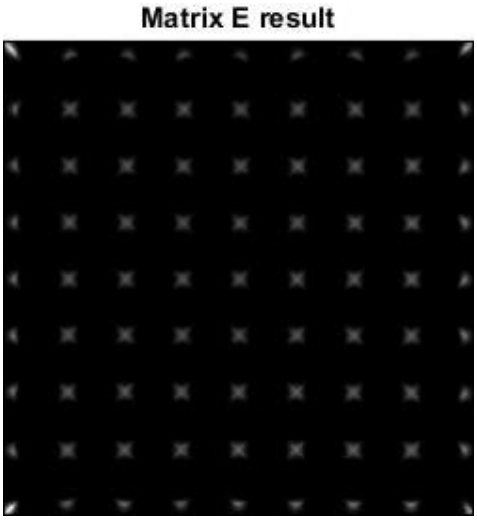

Figure 2: *Image result from part 2 for chessboard00.png*

As it shows on figure 2, the bright points represented the intersection between squares. For most of the chessboard images the results using the matrix E were quite similar and satisfactory but one particular image gave a different result (see figure 3).

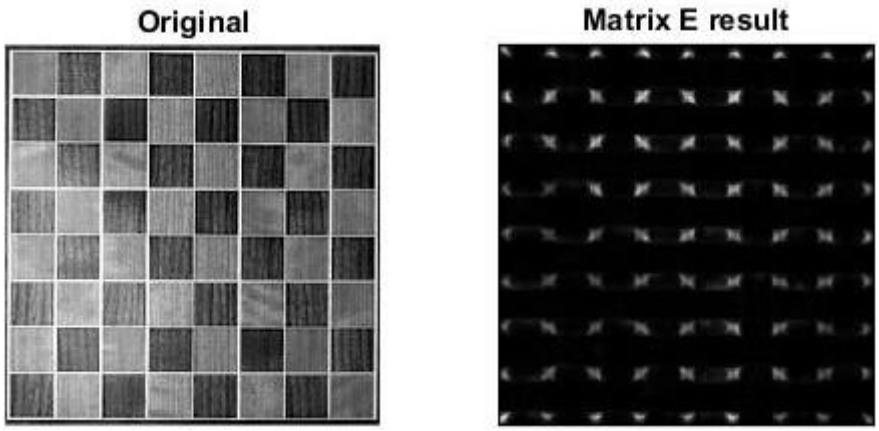

Figure 3: *Original image and matrix E result from chessboard03.png*

Comparing the matrices E from 'chessboard00' and 'chessboard03' it was noticeable that the points for the second matrix were less sharp and had some "connection" between the points. This might be related to how the squares were separated. For 'chessboard00' the black and white patches were aligned side by side meanwhile on 'chessboard03' there is a line between the patches. Those can be the reason why there was this "continuity" between the points.

Part 3: Matrix R

On the third part the matrix R was calculated. Like the matrix E, implemented on part 2, this matrix was used to detect the corners at the images. Uses the same autocorrelation matrix M, but instead of using the smallest eigenvalue, R was given by $\det(M) - k \cdot tr(M)$, where k was a constant (we used k equal to 0.04), $\det(M)$ and $tr(M)$ the respective determinant and trace of M. This matrix was proposed by Harris and Stephens when they noted that exact computation of the eigenvalues was computationally expensive, since it required the computation of the square root. They created then a score, basically an equation (equation 3), which determined if a window could contain a corner or not. Figure 4 show the result of the computation of matrix R.

Figure 4: *Image result of Part 3 for chessboard00.png*

The first noticeable difference between the results using matrix E and R was the fact that the image for R was opaquer. Like the operations on part 2, the matrix M is built with a 3×3 neighborhood window and because of this, the borders of image were not considered too. It was also noted that the borders of the image for matrix R were brighter than the inner area. These were the characteristics found for the simulation using the image chessboard.png, but the other images had some other distinct features (See from figure 5 to figure 8).

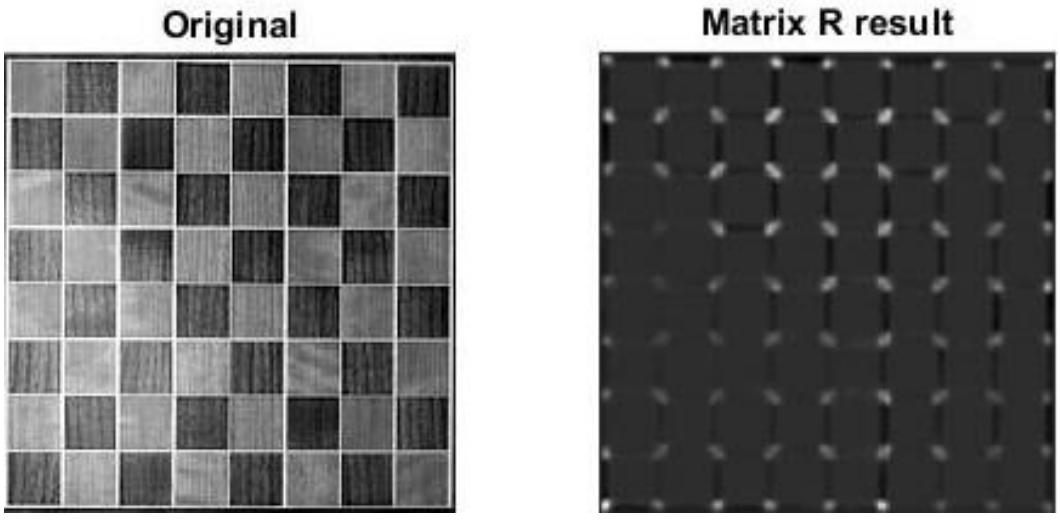

Figure 5: *Original image and matrix R result from chessboard03.png*

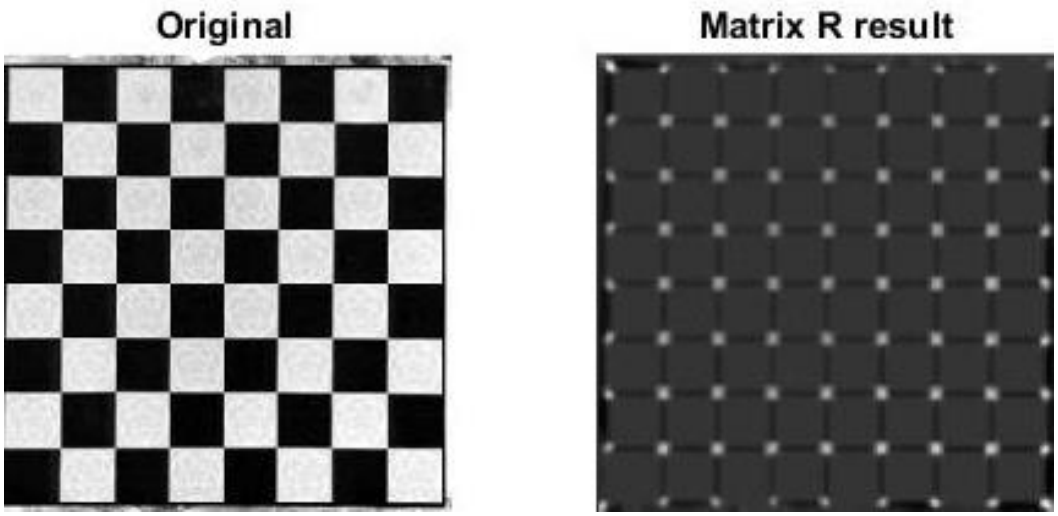

Figure 6: *Original image and matrix R result from chessboard04.png*

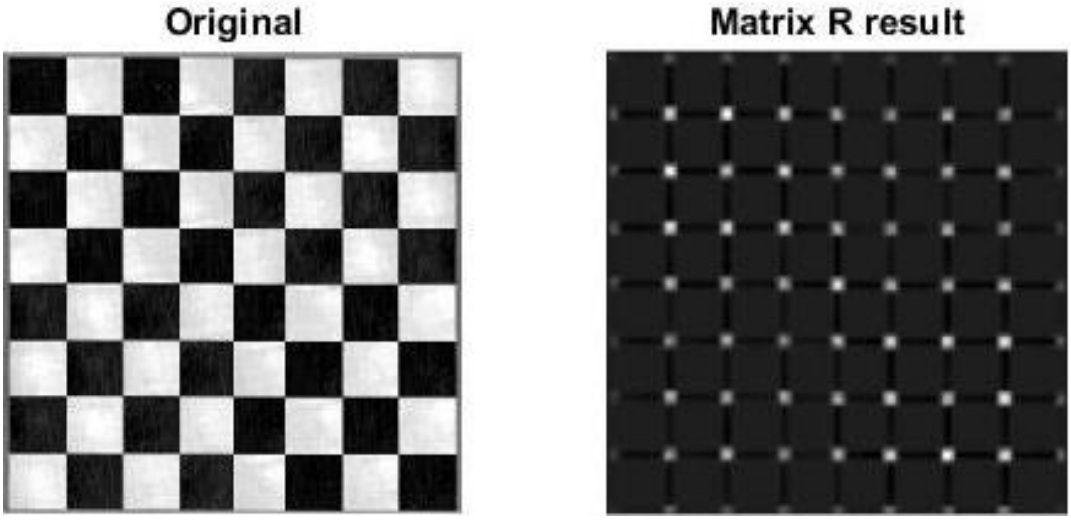

Figure 7: *Original image and matrix R result from chessboard05.png*

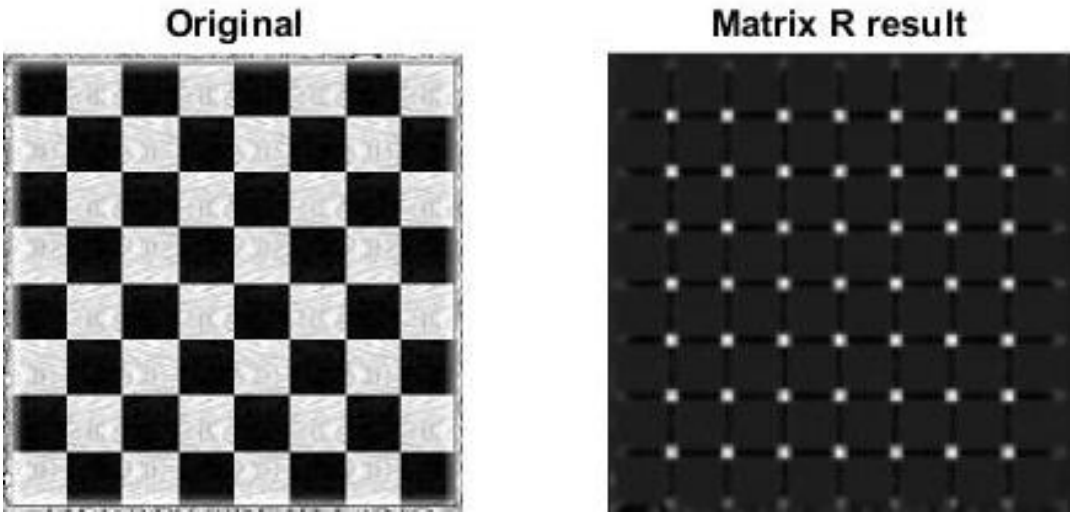

Figure 8: *Original image and matrix R result from chessboard06.png*

Comparing the results for the 'chessboard03' with the matrices E (Figure 3) and R (Figure 5), it seems that the second matrix had a better result because the continuity between the points that appears on the matrix E apparently was not present on matrix R. On the downside, the left side was now more fainted than the rest of the image. This was related to the illumination distribution on the original image and both matrices were affected but was more noticeable on matrix R.

Now, comparing the results between all the matrices R, unlike the result for the 'chessboard00', the 'chessboard05' and 'chessboard06' the points were brighter at the center and opaque on the surrounding. All the results had their points with different intensities distributed on the matrix and this could also be related to the different illumination and pattern on each original image. Also, for all the images tested the calculations for the matrix E was approximately twice faster than the computations to build the matrix R, with the average time of $0.28\,s$ and $0.56\,s$ for matrix E and R, respectively. As already mentioned, the computation time for R was supposed to be less than the E computation but because the function on matlab were used, the time was subject to the computation of the matlab function.

Part 4: Most prominent points

The function to extract the most prominent points, sorted the input matrix in a descendent order and returned the coordinates of the n^{th} first values. For these simulations, it was extracted the 81 most salient points and plotted them on the original image. Figure 9 shows the result for the image 'chessboard00.png'.

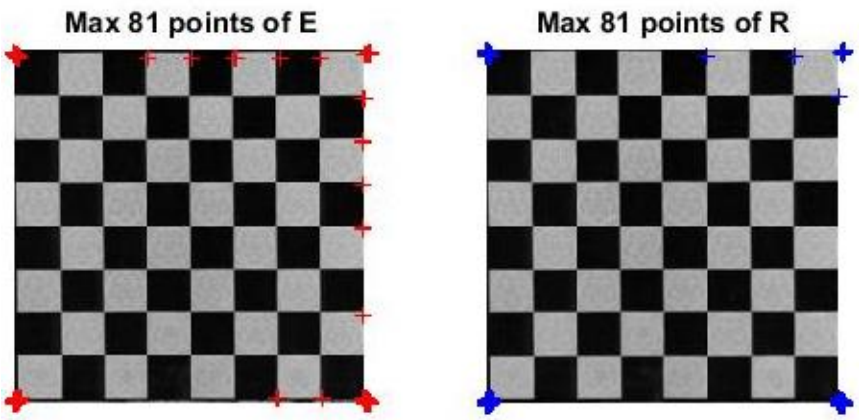

Figure 9: *Most prominent points localized on the matrices E and R for the image chessboard00.png*

Comparing these results with the matrices E (figure 2) and R (figure 4), it was mentioned before that both had bright spots on the surrounding part of the image. Specially the matrix R, which its inner part was opaque. These characteristics were reflected on the most salient points, were for both cases the points were located mostly on the extremities of the image. Next, from figure 10 to figure 13 it is showed the results for the other images that were also tested.

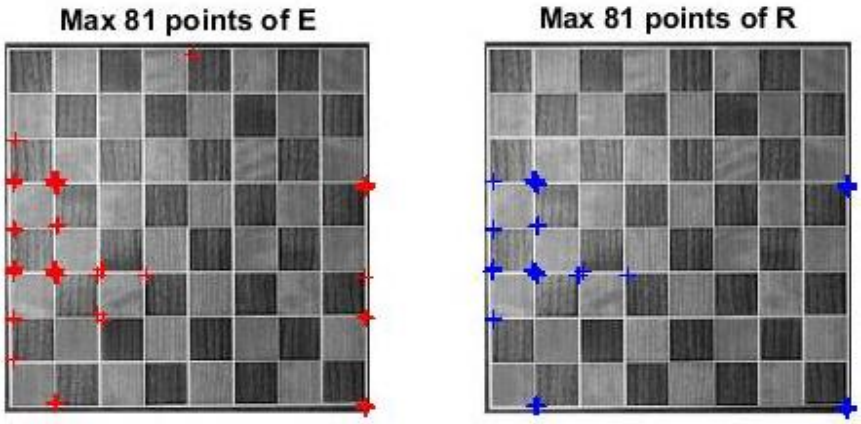

Figure 10: *Most prominent points localized on the matrices E and R for the image chessboard03.png*

For the image ‘chessboard03.png’ the continuity between the points on matrix E (figure 3) weren’t a problem, since there were no points on the line between patches, just corners. Other characteristic found was the fact the points were localized on the area that were a bit fainted on both matrices E and R (figure 5). Therefore, although visually they appear to be less prominent their values were higher than the other points.

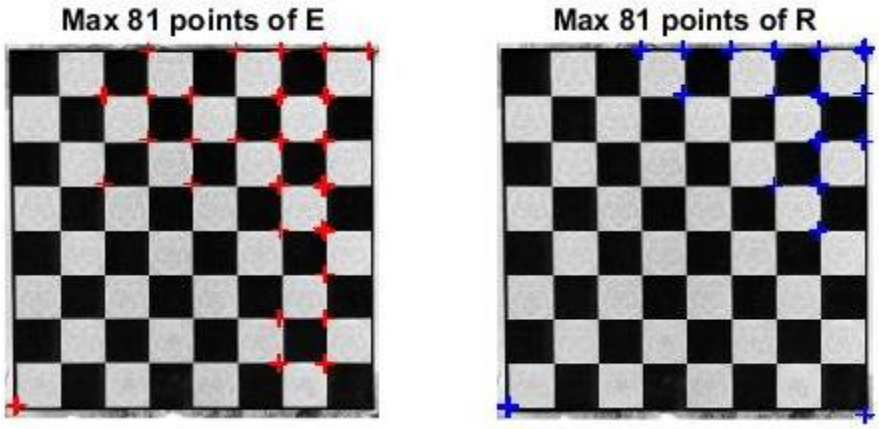

Figure 11: *Most prominent points localized on the matrices E and R for the image chessboard04.png*

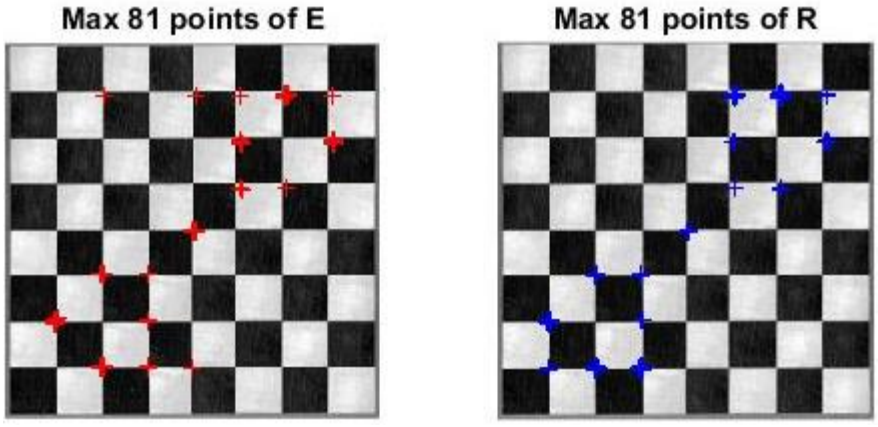

Figure 12: *Most prominent points localized on the matrices E and R for the image chessboard05.png*

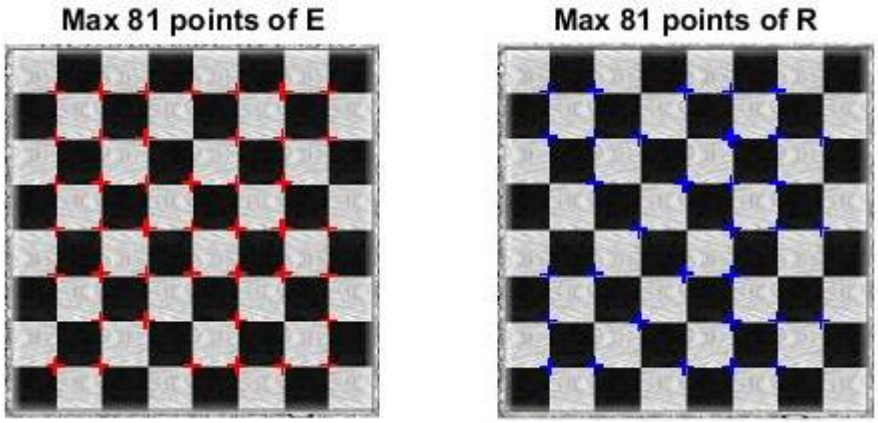

Figure 13: *Most prominent points localized on the matrices E and R for the image chessboard06.png*

For ‘chessboard04’, ‘chessboard05’ and ‘chessboard06’ they all had similar results for matrices E and R. As mentioned on part 3, their bright and fainted spots were randomly distributed and perhaps because of this distribution the most prominent points were also distributed along the image, most particularly for the ‘chessboard06’ (figure 13) that have the most distributed points between all the images tested.

Part 5: Non-maximum suppression algorithm

For the final part, a non-maximum suppression algorithm was applied on the matrixes E and R. Here a window with selected size was considered, for each pixel. If this pixel has the maximum value of this window, its value will be kept. If not, its value is set to zero. For the simulation tests, the window size used was 11×11 pixels. After applying the non-maximal suppression, the 81 most prominent points were localized once again.

To get the best result, a threshold was stablished for each matrix E and R respectively. This threshold was also considered as part of the selection of the maximum value. Figure 14 show the results for the image ‘chessboard00.png’.

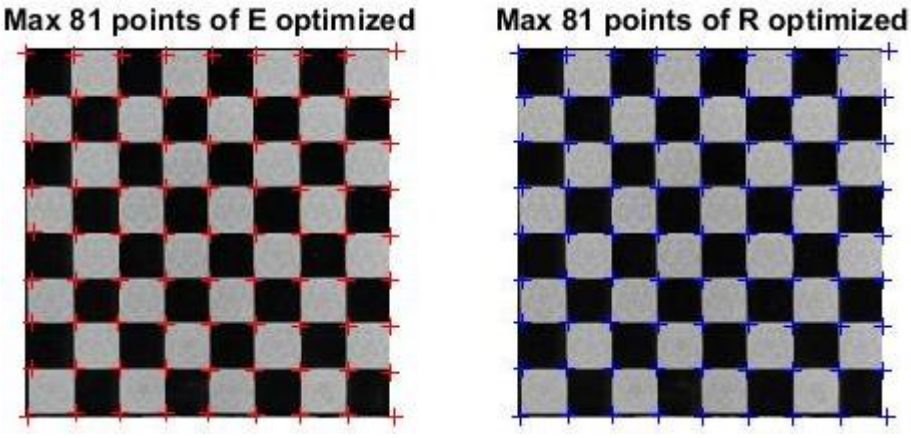

Figure 14: *Results of the non-maximal suppression on matrix E and R for the image chessboard00.png*

After the application of the suppression method, the improvement is quite impressive when is considered that for the previous part, only the points of the extremities on the image were localized. Although this method has given some satisfactory results, it still has some flaws, as for example, looking at the top part of the image, it's possible to observe that the points localized at the top of the image are not aligned with the division of the patches.

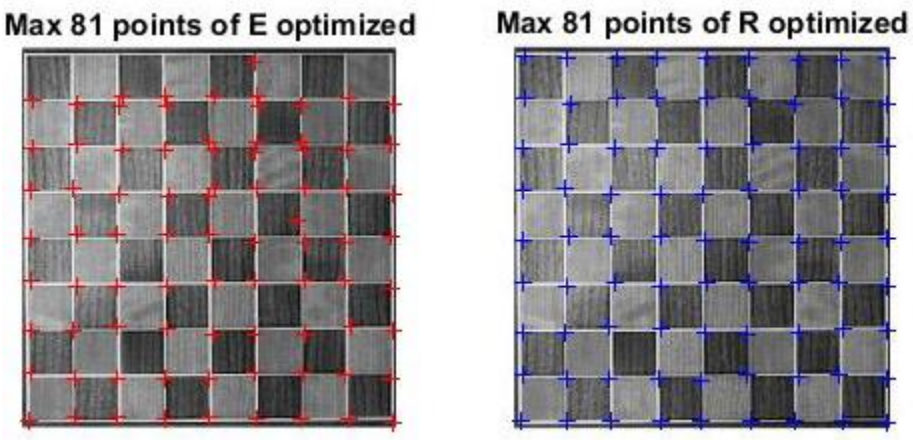

Figure 15: *Results of the non-maximal suppression on matrix E and R for the image chessboard03.png*

For the 'chessboard03' image the improvement from the previous method for this one is also truly noticeable. But, even though the points are more distributed, it still has some problems on the matrix E. On the upper side of the image result for the matrix E, basically there are no points

on the extremity of the chessboard. Therefore, for this image, the best solution to detect the corners is to use the matrix R followed the non-maximal suppression.

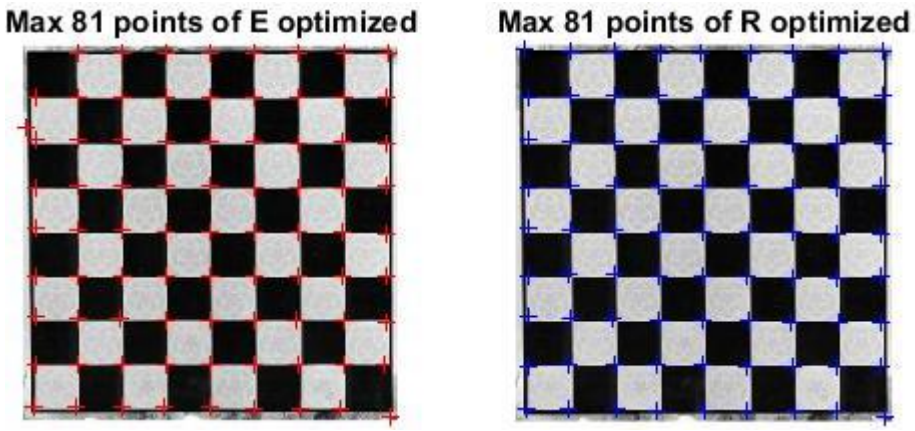

Figure 16: *Results of the non-maximal suppression on matrix E and R for the image chessboard04.png*

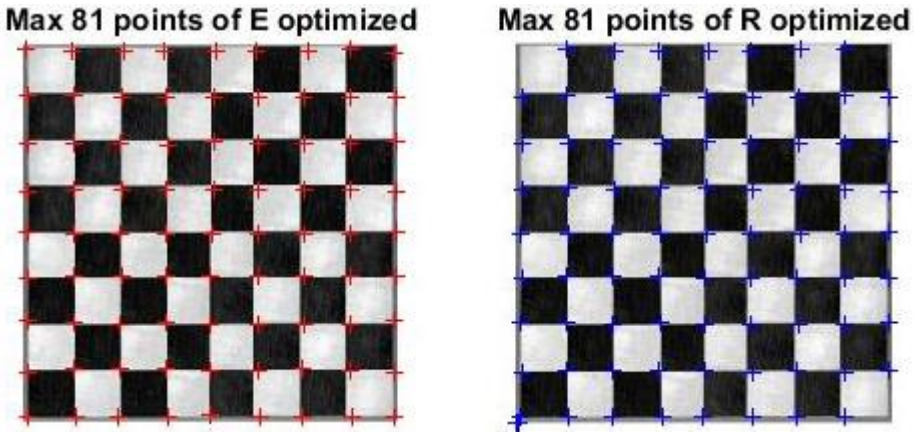

Figure 17: *Results of the non-maximal suppression on matrix E and R for the image chessboard05.png*

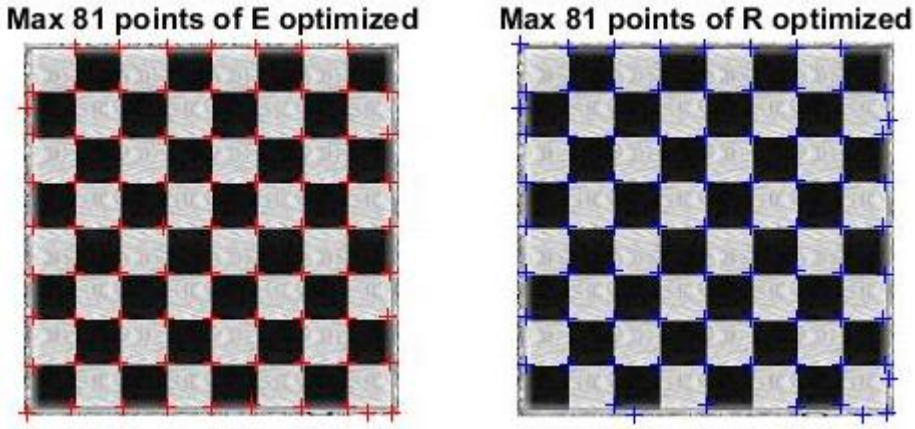

Figure 18: *Results of the non-maximal suppression on matrix E and R for the image chessboard06.png*

From figure 16 to figure 18 the results are quite similar with some particularities on each one of them:

- For the image ‘chessboard04’, the non-maximal suppression on the matrix R apparently was the one with the best results. The application on the matrix E was also good but the upper left corner didn’t have a mark and there is a mark on the line on the left side instead.
- For the image ‘chessboard05’ it was the opposite. The best results were shown on the matrix E. On the application for the matrix R the extremities of the image didn’t have marks and they were concentrated on the extreme bottom left corner.
- Finally, for the image ‘chessboard06’ both matrices returned similar results. They both had problems with the extremities of the image, not having marks on them but in the middle of a line instead.

CONCLUSIONS

The conclusion of the first lab session were the following:

- ✓ It was possible to implement the Harris corner detection algorithm using each step and analyzing them from the results of the ‘chessboard0X’ images.
- ✓ The results of the E and R matrixes depended on the image used. A pre-processing step of the image was a good idea to obtain better result of the corner detection.
- ✓ The time of computation for E and R were different. For R the time was almost the double than for E.
- ✓ Overall, the non-maximal suppression presented better results than the method presented on part 4 for all images tested, despite the small drawbacks detected.
- ✓ The value of the threshold was an important factor to locate the correct maximum value in the non-maximal suppression method.

REFERENCES

[1] Robert Collins, *Harris Corner Detector*, Penn State.