# Computer Vision Lab 1: Harris Corner Detection

Robin Khatri, M1 MLDM

March 2019

# Contents

# 1 Introduction

In Computer Vision, there are various tasks (such as stereo and motion estimation) in which we have to extract certain features from images. One such feature is "corners". Generally, corners are considered to be stable features. Corner detection provides a fast and easy method of feature extraction and inference from images. These corners can help in image classification and reconstructions. Computationally, corner detection is less consuming. During practical session 1, we worked on Harris Corner detection (Harris et al. [1988]).

During the practical session an implementation of Harris Corner Detection was done in MATLAB. We used Chessboard images (We were provided with 5 images - chessboard00.png, chessboard03.png, chessboard04.png, chessboard05.png, chessboard06.png). We experimented with some other images as well.

# 2 Harris Corner Detecion: Background

Harris Corner Detector provides a mathematical formulation for detection of corners in images. The detector is based on Moravec's corner detection. Moravec corner detection was one of the earliest corner detection approaches and it defines corners as points of self- similarity (Moravec [1980]). In Moravec's corner detection, 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 (Collins [2015]).

The basic idea is to find patches of image, otherwise known as window, which corresponds to greater variation when it moves around.
We want to measure the local changes in 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:

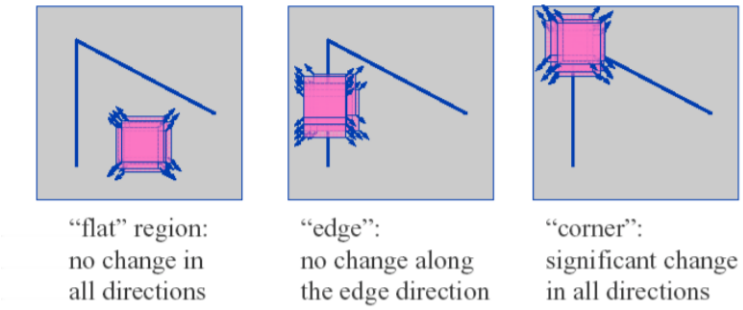


Figure 1: Harris Corner Detector: Basic idea

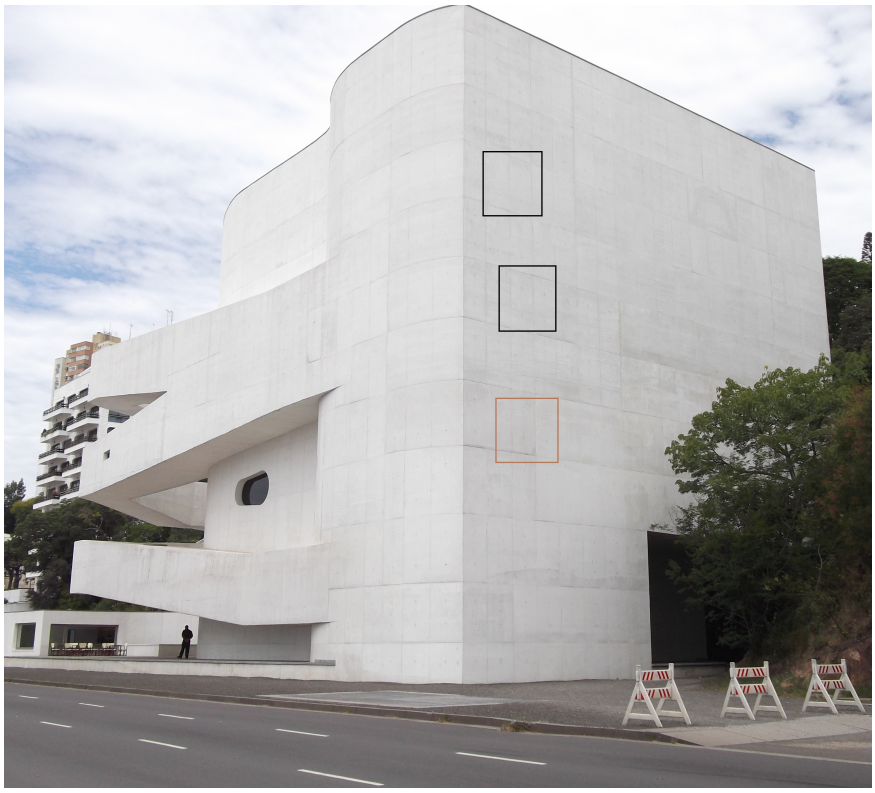

Figure 2: Problem of patch (Window) selection (Picture of building taken from `https://jooinn.com/white-building-10.html`)

Suppose in figure 2, we have selected the red square but when we move it around, it does not show a lot of variation. We want to select a window such that the difference between that patch of window and the image around it is high. Because if the variation is too low, we cannot be certain that the particular windows is a unique position in the image. In order to achieve correspondence between images, we need to establish feature set that are particular to that position in image. (Note: If we move the red square to the tree, the variation is high but then we'd be moving our window to a great distance which is not a good idea.)

Harris corner detector provides a mathematical formulation for selecting patches in image. The weighted sum of squared differences between two selected patches is given by $E$ which is defined as follows (Harris et al. [1988]):

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2 \tag{1}$$

In simple words, $E$ represents the variation between previous window and window after movement.

Here $w(x,y)$ is window function. It ensures that only a certain window is being used.

$I$ measures intensity. So, $I(x,y)$ is the intensity at original position, while $I(x+u, y+v)$ is the intensity at new moved position.

We want to maximize E.

The term $\sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2$ can be expanded using Taylor's expansion, and we obtain,

$$E \approx \sum_{x,y} [I(x,y) + uI_x + vI_y - I(x,y)]^2$$

$$\Rightarrow E \approx \sum_{x,y} (u^2 I_x^2 + v2I_y^2 + 2uvI_xI_y)$$

This equation can be written as,

$$E(u,v) \approx [u,v](\sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_{xy} \\ I_{xy} & I_y^2 \end{bmatrix}) \begin{bmatrix} u \\ v \end{bmatrix}$$

We define matrix M as,

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

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

M can be used to derive a measure of "cornerness" for every pixel. Depending on 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 curvature information of the local auto-correlation function as shown in Figure 3.

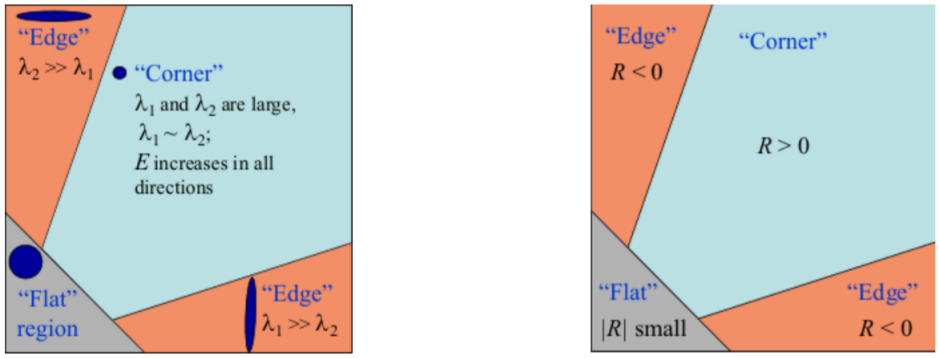


Figure 3: Detection of edges through M and R

By calculating determinant and eigenvalues of the matrix M, we can characterise window. Since eigen decomposition is computationally expensive, Harris purposes to utilize the determinant and the trace of M as the edge detector's components (Harris et al. [1988]).

Consider,

$$R = det(M) - k(trace(M))^2$$

Value of R helps us in selecting best windows as described in figure 3. R is also known as Harris response.

# 3 Image derivatives

During part 1 of the practical, we computed image derivatives and smoothed using a Gaussian filter. The gradient points to the direction of most rapid increase in intensity. Calculating derivatives of an image in x and y direction is essentially a convolution of image with a specific mask (In x-direction: [-1 0 1; -1 0 1; -1 0 1] , in y-direction: [-1 -1 -1; 0 0 0; 1 1 1]). Since, our task is to find the windows which help in distinguishing intensity levels and so to select best windows, we use image derivatives.

Noise in image may make neighbouring pixels much different than they really are, we need a way to deal with this noise. To deal with this problem, we use smoothing. Smoothing helps in forcing pixels (which seem different because of the noise) to be close to their true states. To smooth images, we used Gaussian filter. There is a certain trade off between dealing with noise (smoothing) and blurring of image (localization). The degree of smoothing is determined by standard deviation ($\sigma$) used for creating the Gaussian filter. During the practical, we used $\sigma = 2$. The Gaussian filter computes a 'weighted average' of each pixel's neighborhood. Gaussian filter is widely used in smoothing due to its frequency response. What it means is that by using Gaussian filter we can be fairly confident about what range of spatial frequencies are still present in the image after filtering (Witkin [1987]). We applied Gaussian filter to second order image derivatives $I_{x2}, I_{y2}$ and $I_{xy}$ , which we need to calculate E and R matrices. The results are displayed in figure 4 for image chessboard00.png.

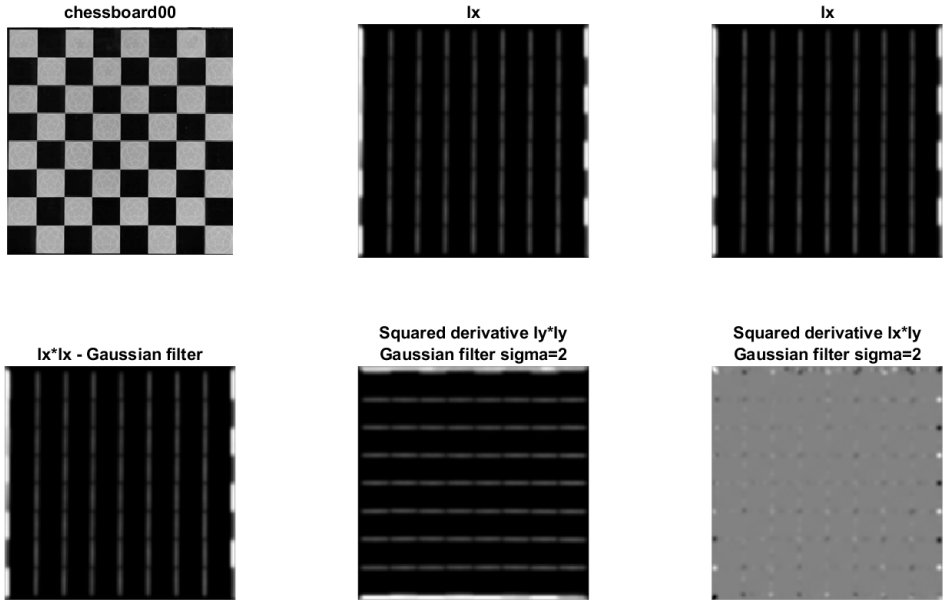


Figure 4: Image derivatives for chessboard00.png

# 4 Matrices E and R

In Part 2 of our practical, we computed matrix E defined by equation 1. Matrix E contains for every point the value of the smaller eigenvalue of auto correlation matrix M. As discussed in section 2, eigen values of M can be used to extract the curvatures information of the local autocorrelation function. To compute E in MATLAB, we used a filter which is a 3 by 3 matrix of 1's. It convolutes the second order derivatives of the images. For every pixel $(i, j)$, we stored the values of these derivatives in a 2 by 2 matrix M with convoluted $I_{xy}$ as M[1,2] and M[2,1] while the diagonal elements are the value of convoluted $I_x$ and $I_y$ on

that pixel $(i, j)$. Matrix E was then built by selecting minimum eigen value. Eigen values here measure the variances of points along the eigen vectors. Eigen vectors for a covariance matrix are same as the directions of gradients. As we know, the convoluted image derivatives are actually covariance matrices of gradients. Therefore This procedure based on eigen values is based on the property of covariance matrices where M is an unbiased estimator of the covariance matrix of gradients for pixels in a patch (window). The results on the image chessboard00.png are given below:

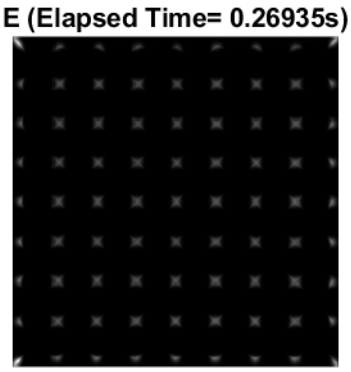


Figure 5: Matrix E for chessboard00.png

To compute E, it almost took 0.27 seconds for a small image. Since computation of eigen values can be large, we use an approximation by calculating matrix R containing Harris responses described in section 2. Both E and R try to achieve same task but there is clearly some difference in both computation and quality. The matrix E is essentially our main result while R is an approximation. On the same image, using formula defined in section 2, we obtained following R.


Figure 6: Matrix R for chessboard00.png

To compute R, it took around 0.30 seconds which is more than what was needed to compute E. However, it is due to the fact that code was run in a loop. After optimizing code to -

```
Ix2_c = conv2(Ix2, filter, 'same');
Iy2_c = conv2(Iy2, filter, 'same');
Ixy_c = conv2(Ixy, filter, 'same');

k = 0.04; % k is a constant
```

```
tic % Time start

ROptimzed = (Ix2_c.* Iy2_c+ Ixy_c.*Ixy_c) - k*(Ix2_c +Iy2_c).^2;
elapsed_time_R_optimized = toc;
```

We obtained much better computation time. The computation time was significantly faster.


Figure 7: Matrix R for chessboard00.png (Less Computation time)

Table 1 lists the computation times for chessboard images (00,03,04,05,06).

| **Image** | **Computation time for E** | **Computation time for R** |
|---|---|---|
| chessboard00.png | 0.26935 | 0.0011657 |
| chessboard03.png | 0.10377 | 0.0012628 |
| chessboard04.png | 0.12522 | 0.0010342 |
| chessboard05.png | 0.12898 | 0.0009956 |
| chessboard06.png | 0.12624 | 0.0012817 |

Table 1: Computation time E vs. R

Figure 8 presents matrices E and R for image chessboard04.png.

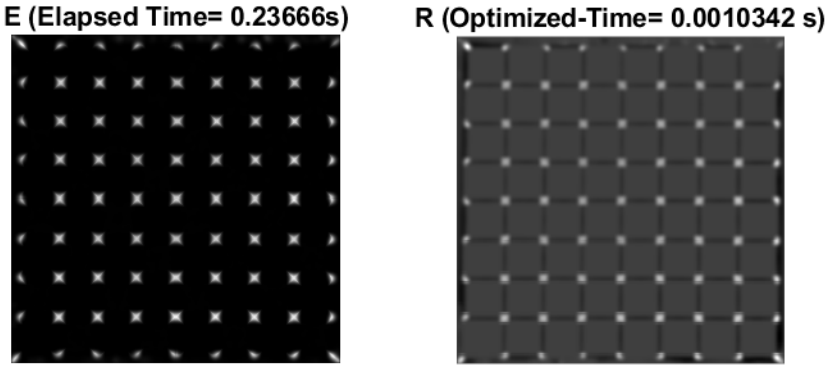


Figure 8: E and R for chessboard04.png

# 5 Selecting Most salient points

Salient points are defined as the points that have a locally unique image structure (Pedersen et al. [2007]). These points can be anywhere in the image and are generally sparsely distributed. We can select these points with help of sorted E and R scores. We selected the points for which E and R have highest values. On our images chessboard00.png and chessboard03.png, we obtained following results.

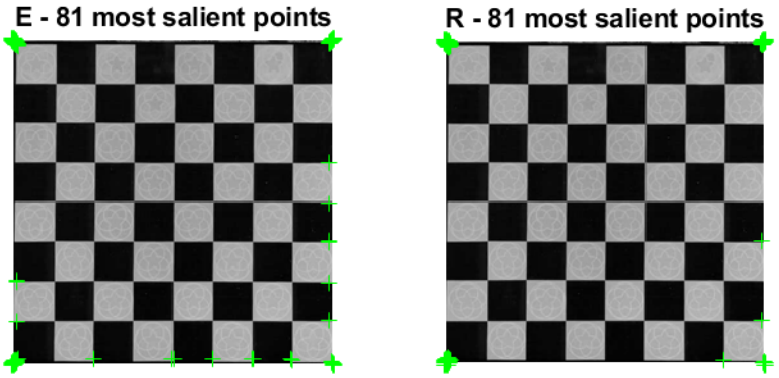


Figure 9: Salient points for chessboard00.png

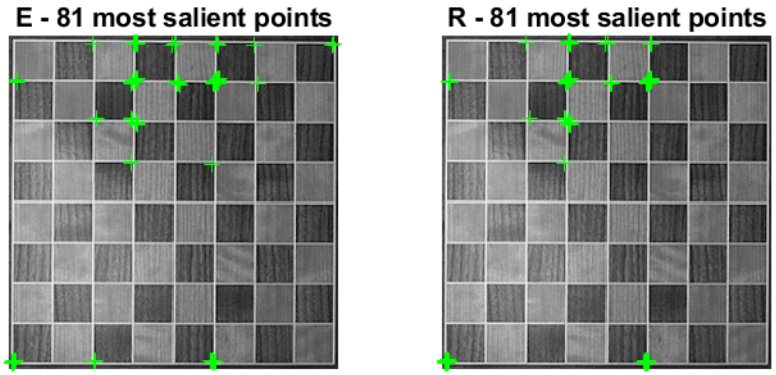


Figure 10: Salient points for chessboard03.png

It can be seen that the results are not that good. The corners are not detected well. This is to due to the reason that due to the multiple response, edge magnitude M(x, y) may contain wide ridges around the local maxima; one corner can be detected by many points resulting in overdetecting in one corner and underdetecting in others. This definitely causes a problem in detecting corners perfectly. To solve this problem Non-maximal supression algorithm can be used.

# 6 Non-maximal supression Algorithm

As we saw in figure 10, we could not detect corners properly using our method. This warrants a more robust technique which resolves the problem of overdetection of some corners. One such way is to implement non-maximal supression algorithm. To deal with the problem, non-maximal pixels are removed. This way the connectivity of the countours remains intact.

The algorithm can be described as:

**Algorithm 1:** Non Maximal Supression Algorithm

```
function NonMaximalSupression (E, points, neighborhood);
  Input  : Sort E
  Output: Counter ← 1
while Counter < points do
    neighbors ← neighbourhoodpixels;
    for all rows of E do
        Assign the neighbours of E to 0;
        Save the coordinates of this maximum;
        Counter ← +1;
        if Counter > points then
        end
    end
end
```

Non-maximal suppression algorithm works well in detecting corner because it eliminates the chance for some corners to be overdetected by a lot of points.

On 5 images that were included in the material for lab 1, non maximal suppression works well. It is able to detect corners pretty accurately as can be see by following pictures:

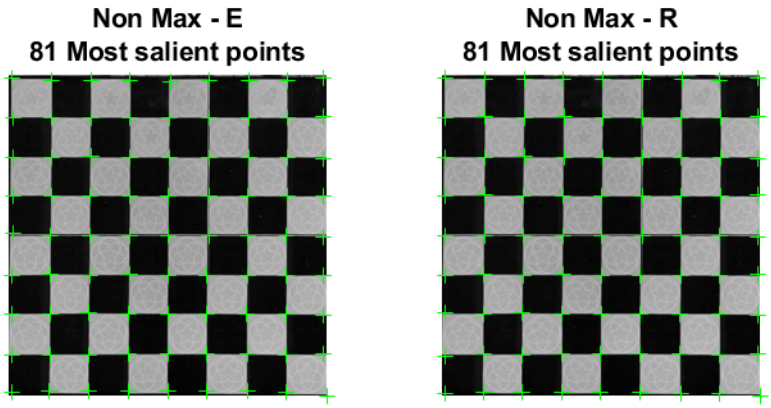


Figure 11: Non maximal suppression on Chessboard00.png

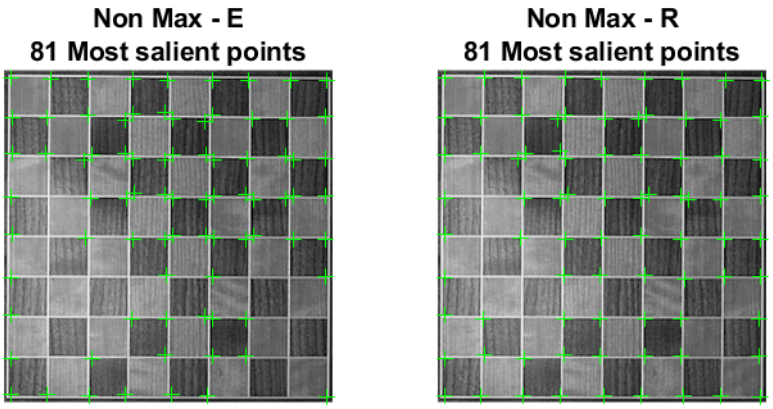


Figure 12: Non maximal suppression on Chessboard03.png

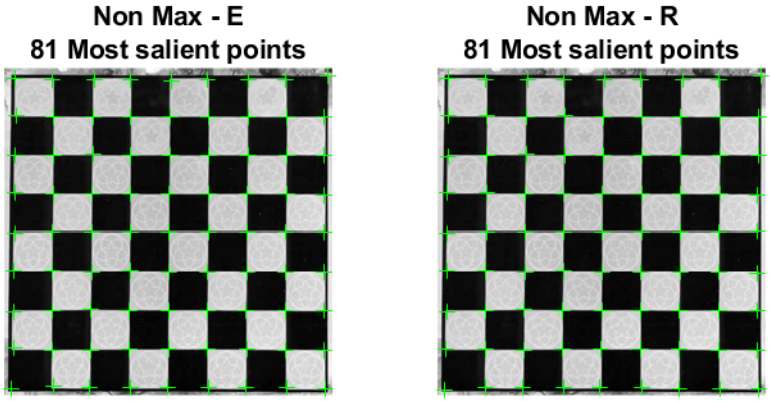


Figure 13: Non maximal suppression on Chessboard04.png

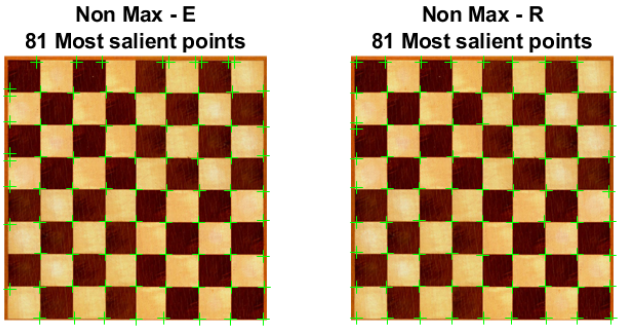


Figure 14: Non maximal suppression on Chessboard05.png

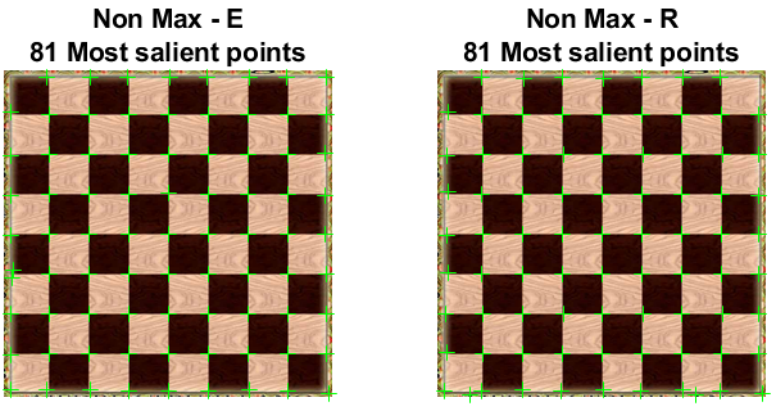


Figure 15: maximal suppression on Chessboard06.png

# 7 Illustration on other images

In this section, we demonstrate the results on some other images that we selected.

We took an image img-chessboard-01.png from `https://scm.gforge.inria.fr/anonscm/svn/visp/tags/ViSP_2_10_0/doc/image/`

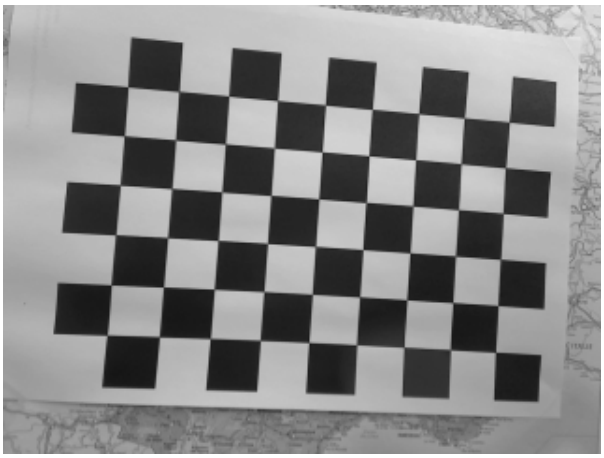

Figure 16: Image: img-chessboard-01.png (`https://scm.gforge.inria.fr/anonscm/svn/visp/tags/ViSP_2_10_0/doc/image/`)

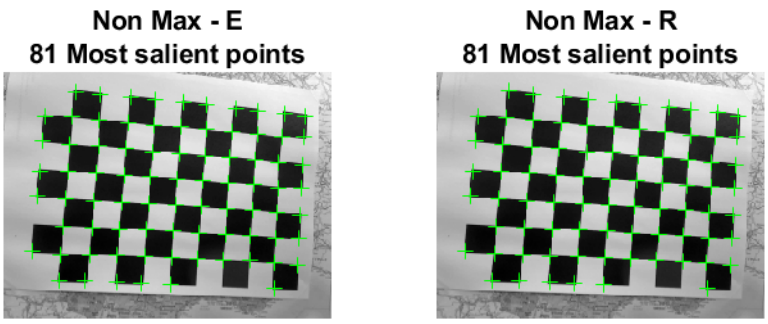


Figure 17: maximal suppression on img-Chessboard-01.png

As it is clear, the corner detection is good. This can further be improved by calibrating window size. To push the bar further, an image from a Cricket game was used (being a fan of cricket). Below is the original image.

Figure 18: An Australian cricketer during a game in Perth, Australia (Source: fox Sports, Australia)

On running the same MATLAB code that we ran on chessboard images, we obtained following result:


Figure 19: Corner Detection in cricket1.jpg

Obviously, this is not a good detection because it could not detect corners in ball and also on bat. We tested again with same code but with different window sizes in non-maximal suppression. We chose window sizes of 30 and 50. The results were as follows:

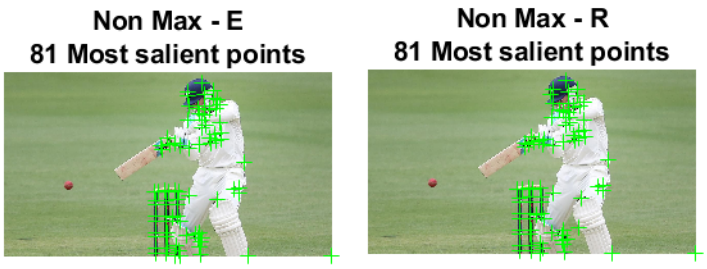


Figure 20: Corner detection: window size = 30 by 30

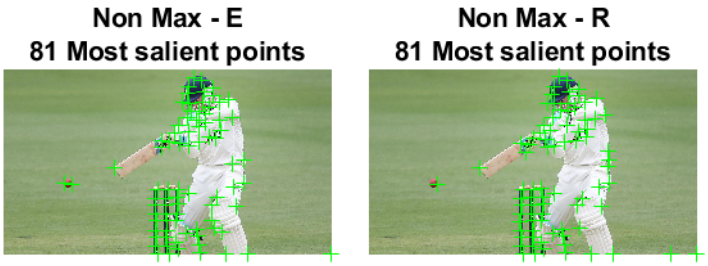


Figure 21: Corner detection: window size = 50 by 50

When the window size is 50, the results are pretty good as now the corners in ball are also detected.

# 8 Conclusion

During this practical, we learnt about the image derivatives and how we calculate them. We also saw the importance of selecting Gaussian filter to get convoluted images so as to reduce the noise and smooth the images. We learnt about Harris Corner Detection and successfully implemented it in MATLAB. The results vary among images selected but non-maximal suppression algorithm provided good results as it eliminates a key problem of over detection. The results from non-maximal suppression were good even on images that were taken outside the lab material and after calibrating the parameters such as window size used for non-maximal suppression algorithm, we successfully detected corners in a given image.

# References


Robert Collins. Harris corner detector. *University Lecture*, 2015.

Christopher G Harris, Mike Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

Hans P Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical report, STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1980.

Kim Steenstrup Pedersen, Marco Loog, and Pieter van Dorst. Salient point and scale detection by minimum likelihood. In *Gaussian Processes in Practice*, pages 59–72, 2007.

Andrew P Witkin. Scale-space filtering. In *Readings in Computer Vision*, pages 329–332. Elsevier, 1987.

# A Appendix 2: Image derivatives

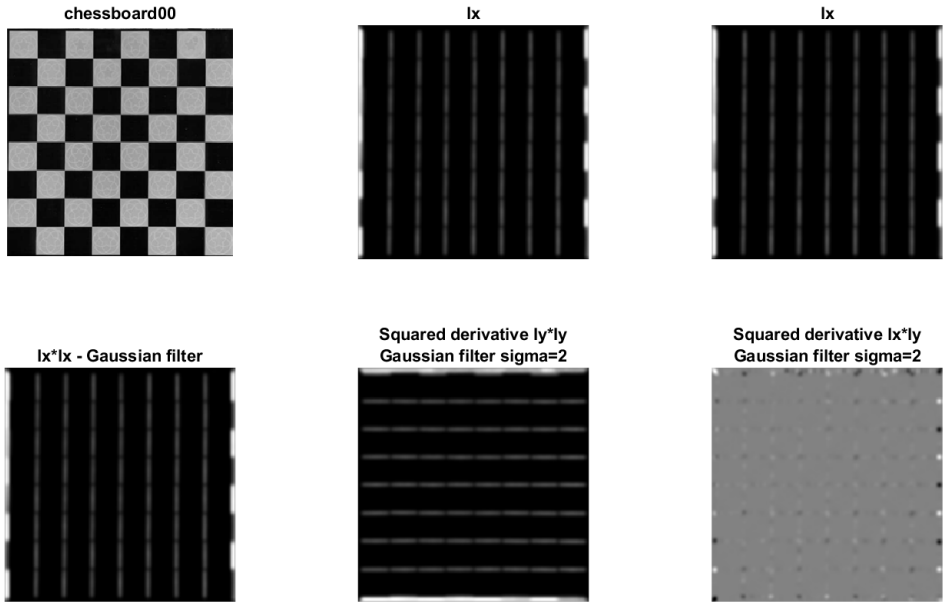


Figure 22: chessboard00.png

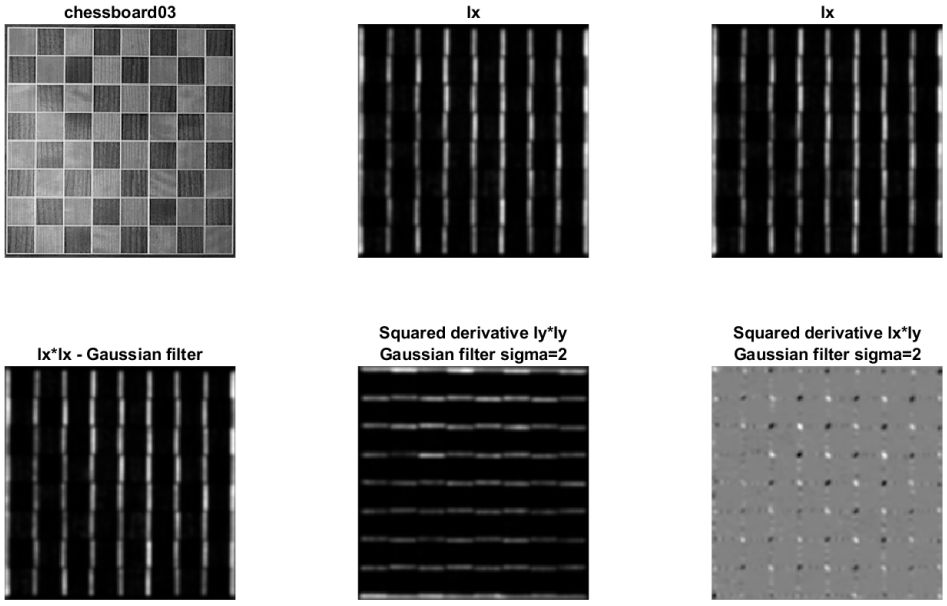


Figure 23: chessboard03.png

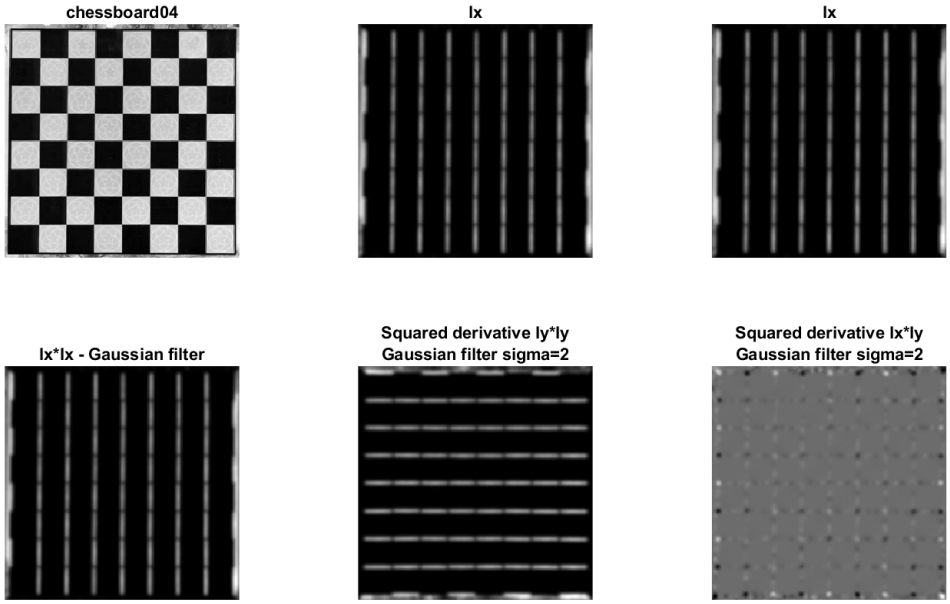


Figure 24: chessboard04.png

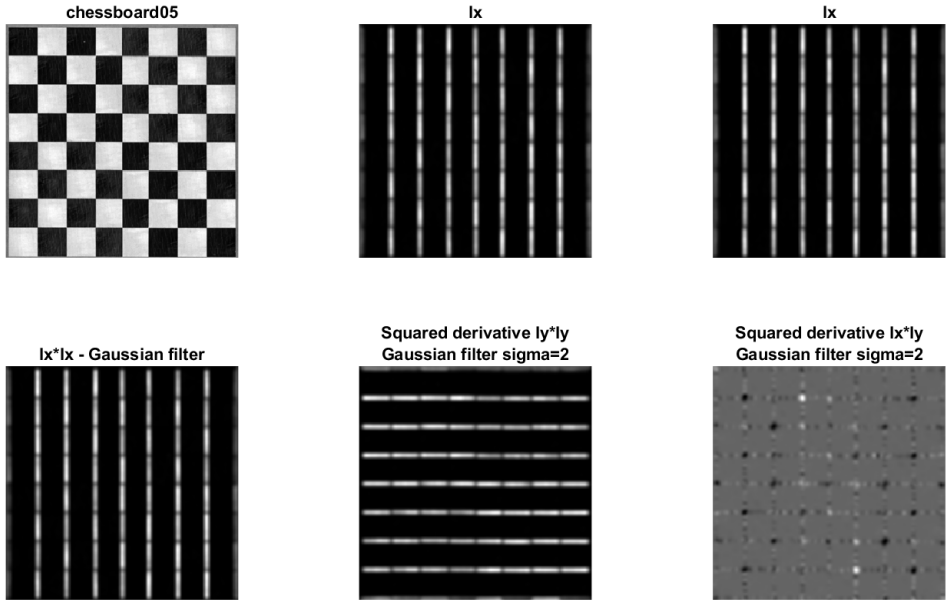


Figure 25: chessboard05.png

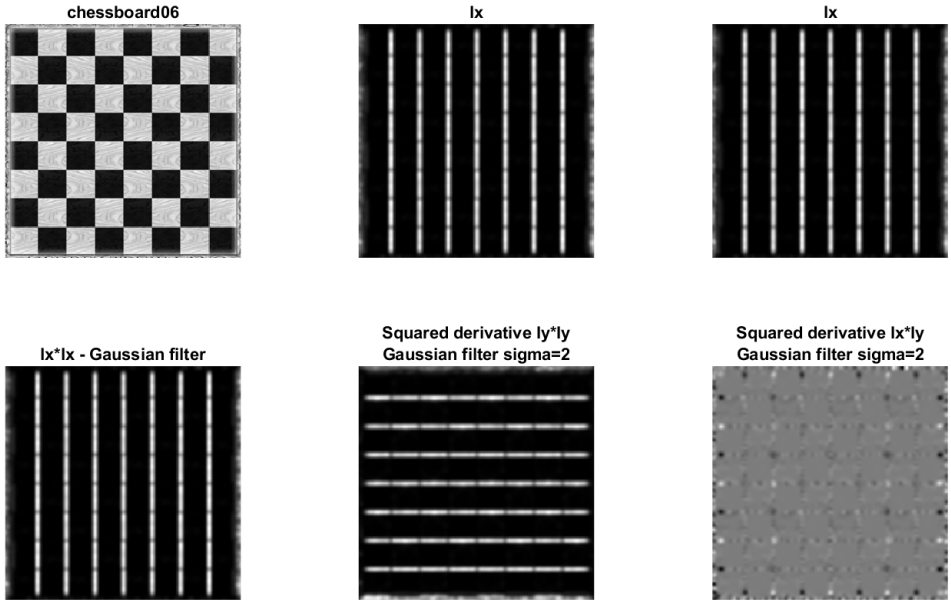


Figure 26: chessboard06.png

# B Appendix 2: Corners

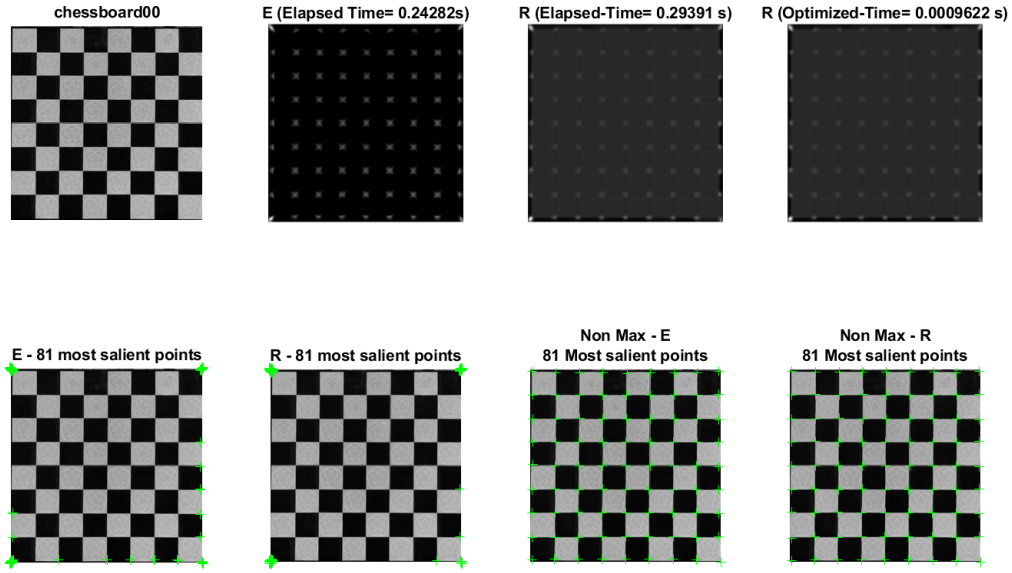


Figure 27: chessboard00.png

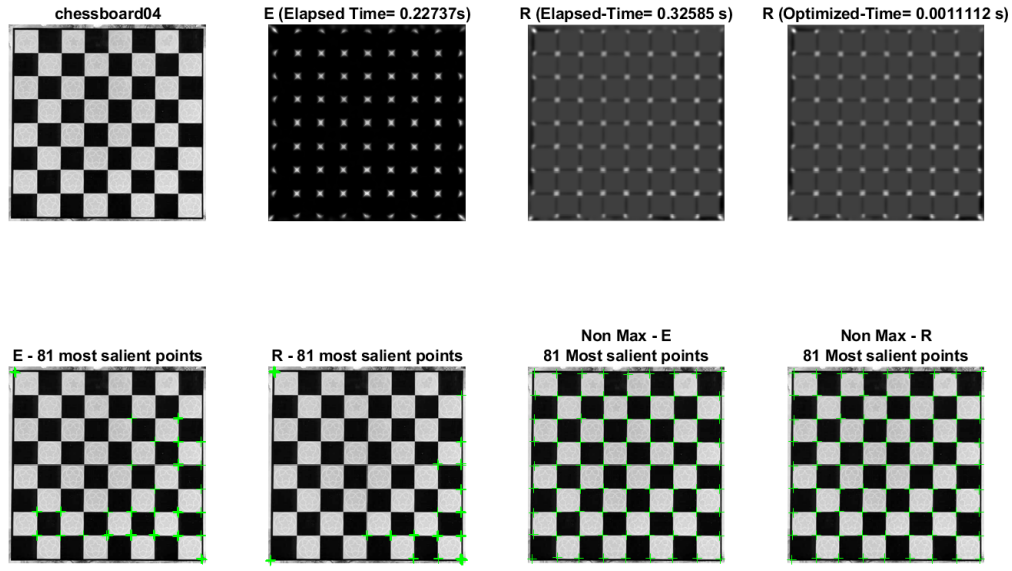


Figure 29: chessboard04.png

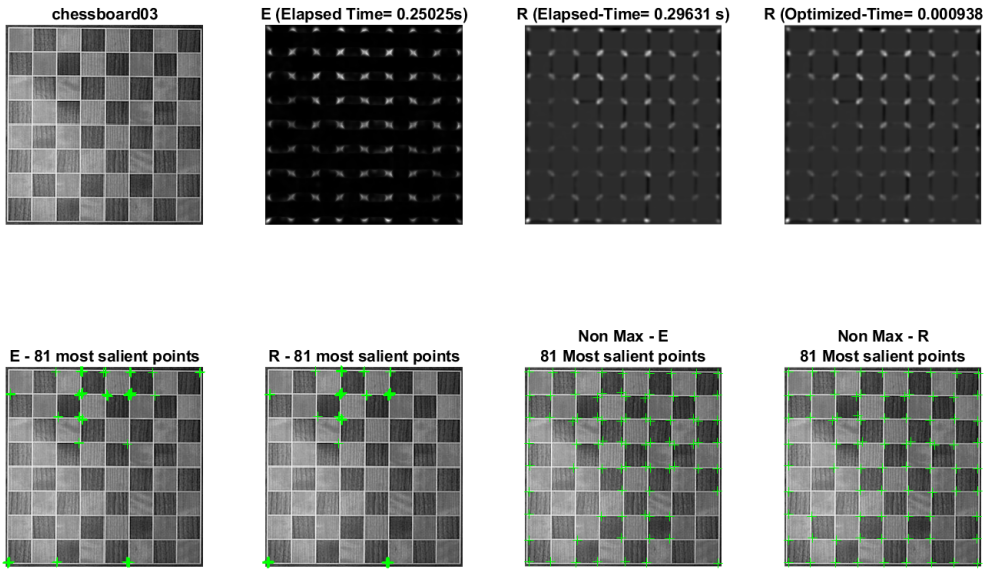


Figure 28: chessboard03.png

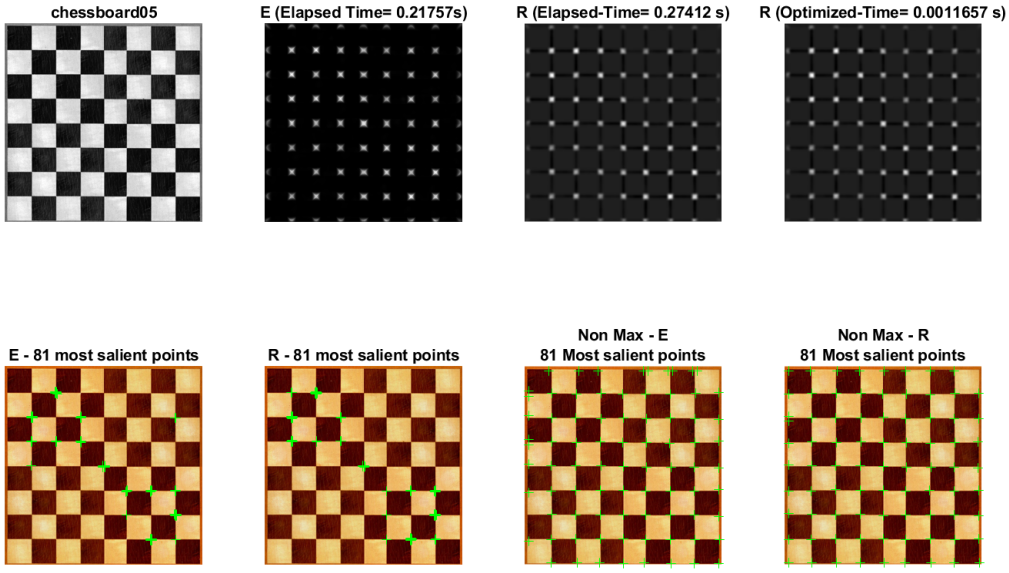


Figure 30: chessboard05.png

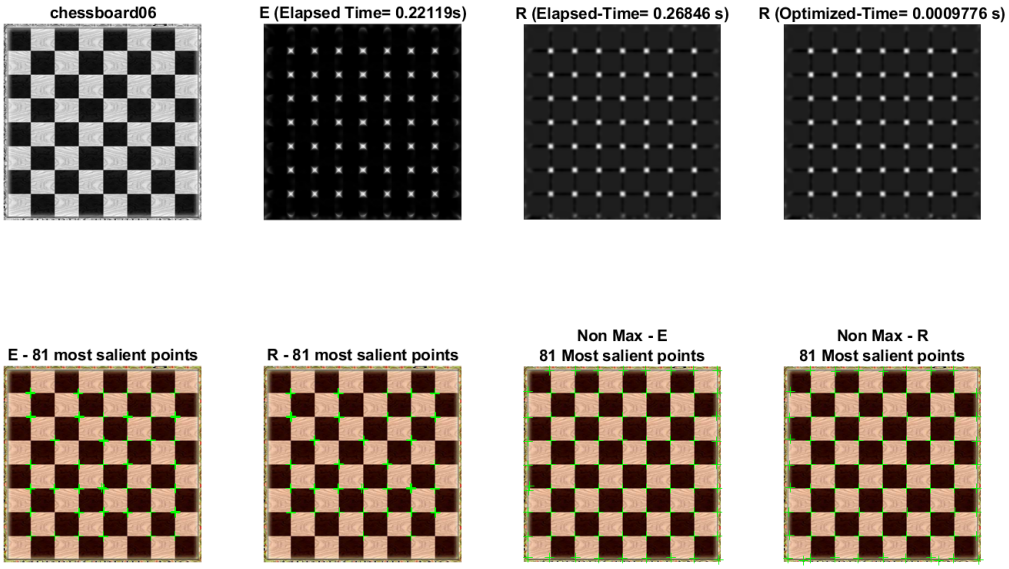


Figure 31: chessboard06.png

# C Appendix 3: Images from outside lab materials

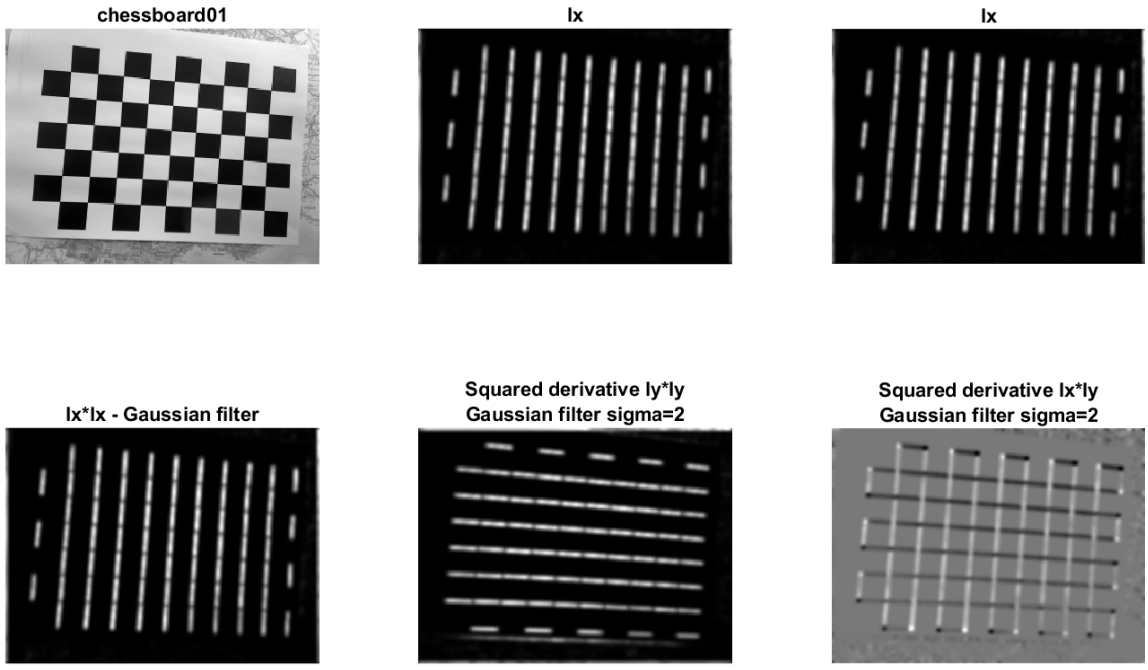


Figure 32: Derivatives: img-chessboard-01.png

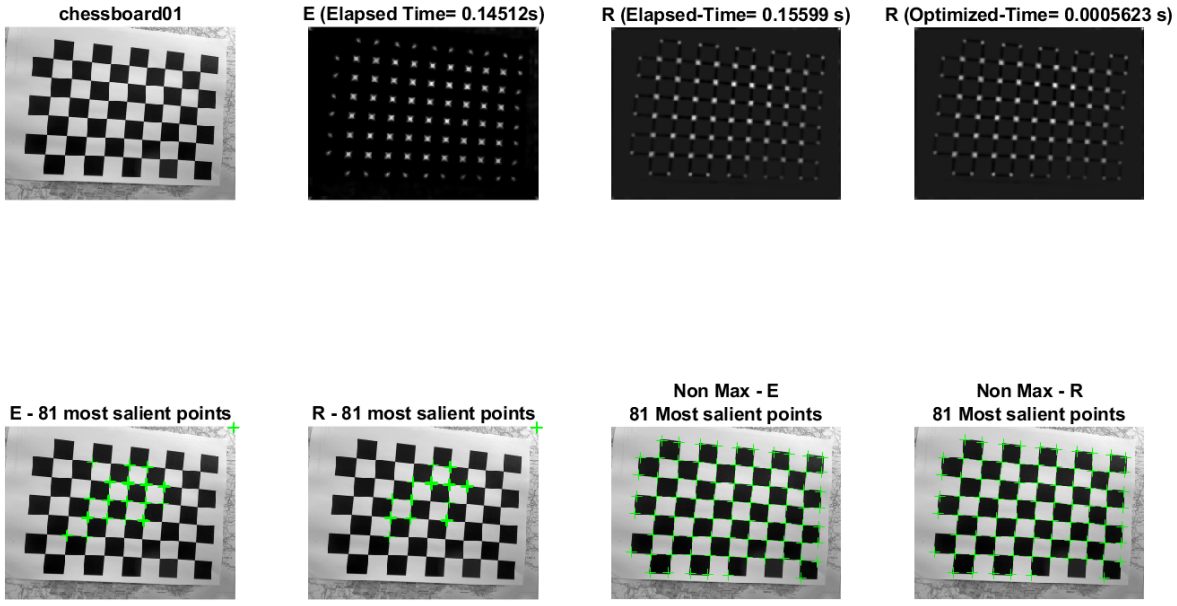


Figure 33: Corners: img-chessboard-01.png

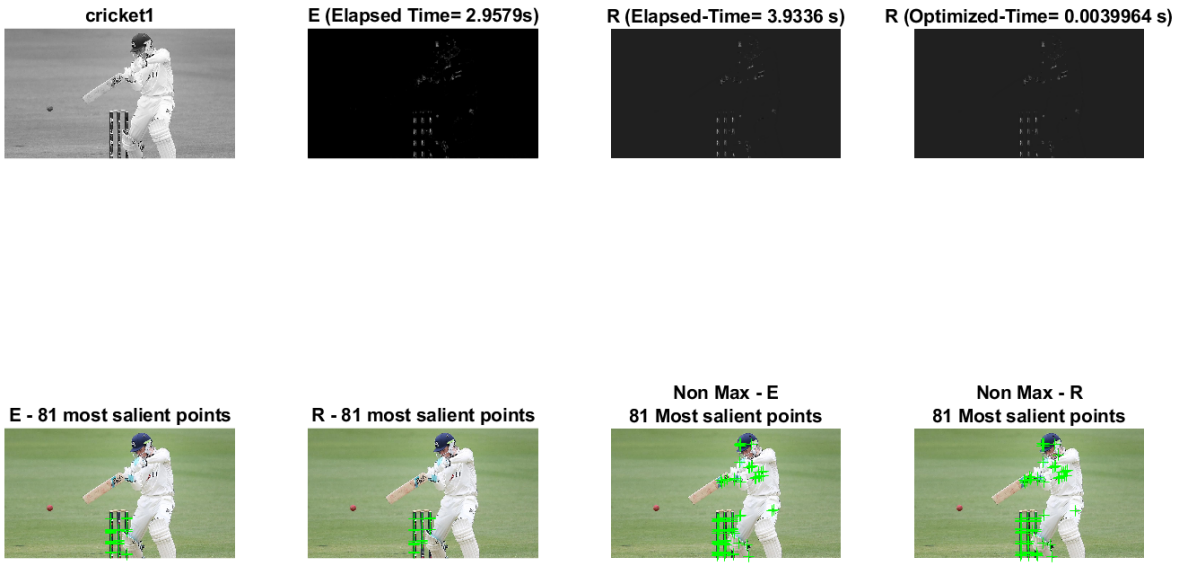


Figure 34: Corners: Window size 11

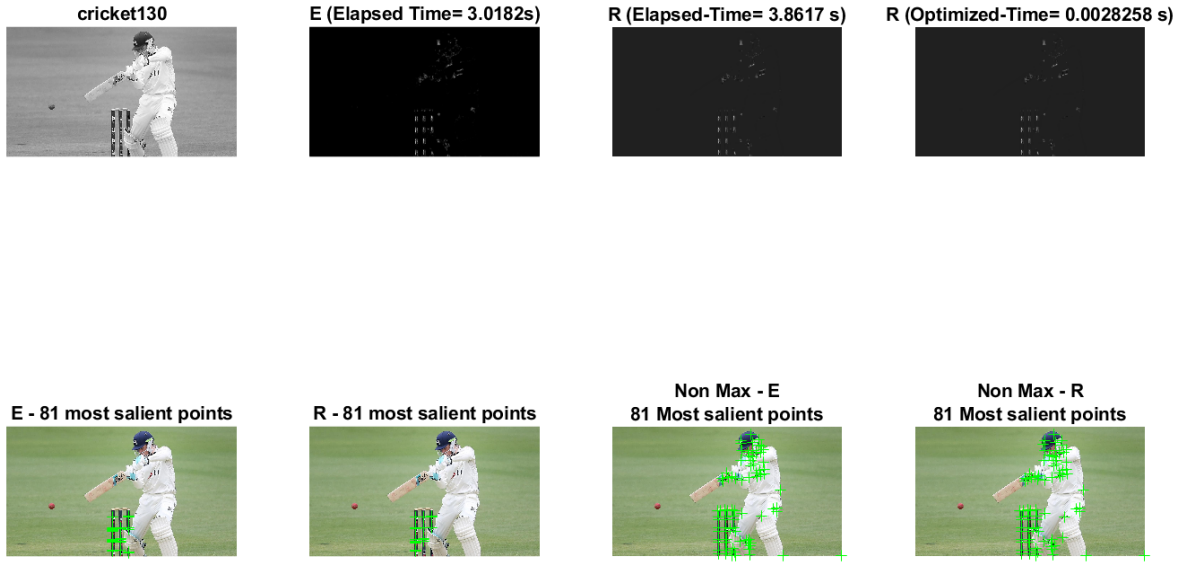


Figure 35: Corners: Window size 30

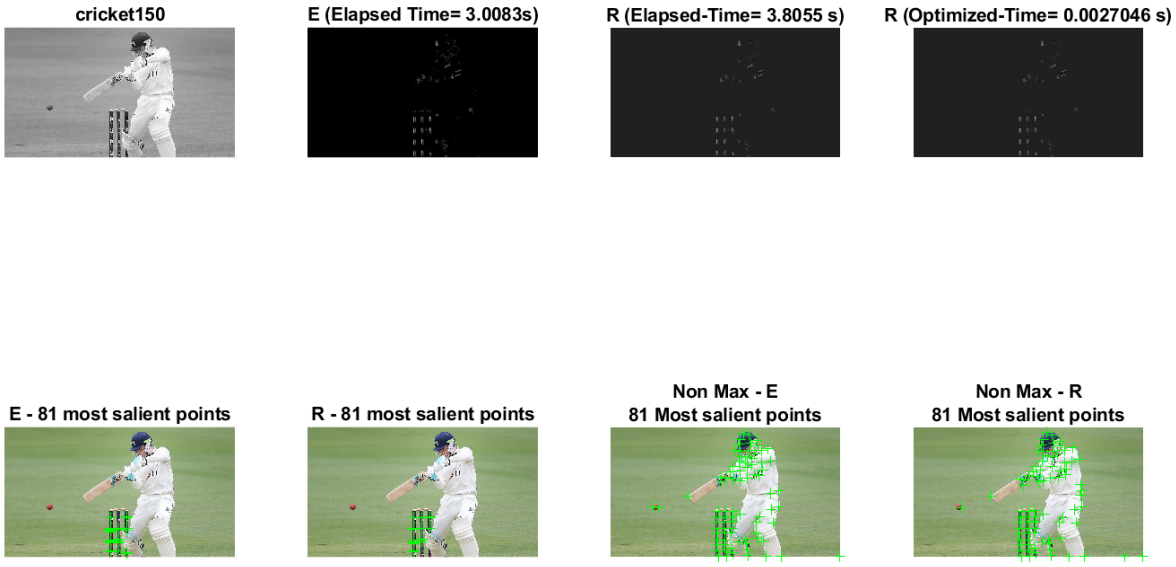


Figure 36: Corners: Window size 50