IPOL IMAGE PROCESSING ON LINE

Published in Image Processing On Line on 2018–10–03.
Submitted on 2018–06–04, accepted on 2018–09–18.
ISSN 2105–1232 © 2018 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2018.229

An Analysis and Implementation of the Harris Corner Detector

Javier Sánchez[1], Nelson Monzón[2], Agustín Salgado[1]

[1] CTIM, Department of Computer Science, University of Las Palmas de Gran Canaria, Spain
({jsanchez, agustin.salgado}@ulpgc.es)
[2] CMLA, École Normale Supérieure , Université Paris-Saclay, France
(monzon@cmla.ens-cachan.fr)

Abstract

In this work, we present an implementation and thorough study of the Harris corner detector. This feature detector relies on the analysis of the eigenvalues of the autocorrelation matrix. The algorithm comprises seven steps, including several measures for the classification of corners, a generic non-maximum suppression method for selecting interest points, and the possibility to obtain the corners position with subpixel accuracy. We study each step in detail and propose several alternatives for improving the precision and speed. The experiments analyze the repeatability rate of the detector using different types of transformations.

Source Code

The reviewed source code and documentation for this algorithm are available from the web page of this article[1]. Compilation and usage instruction are included in the `README.txt` file of the archive.

Keywords: Harris corner; feature detector; interest point; autocorrelation matrix; non-maximum suppression

1 Introduction

The Harris corner detector [9] is a standard technique for locating interest points on an image. Despite the appearance of many feature detectors in the last decade [11, 1, 17, 24, 23], it continues to be a reference technique, which is typically used for camera calibration, image matching, tracking [21] or video stabilization [18].

The main idea is based on Moravec's detector [14], that relies on the autocorrelation function of the image for measuring the intensity differences between a patch and windows shifted in several directions. The success of the Harris detector resides in its simplicity and efficiency. It depends on the information of the *autocorrelation matrix*, and the analysis of its eigenvalues, in order to locate

[1] https://doi.org/10.5201/ipol.2018.229

points with strong intensity variations in a local neighborhood. This matrix, also called *structure tensor*, is the base for several image processing problems, such as the estimation of the optical flow between two images [19, 13, 12].

In this work, we propose an efficient implementation of the method, which comprises seven steps. The autocorrelation matrix depends on the gradient of the image and the convolution with a Gaussian function. We study the influence of several gradient masks and different Gaussian convolution methods. The aim is to understand the performance of each strategy, taking into account the runtime and precision.

From the eigenvalues of the autocorrelation matrix, it is possible to define several corner response functions, or measures, for which we implement the best known approaches. A non-maximum suppression algorithm is necessary for selecting the maxima of these functions, which represent the interest points. In the following step, the algorithm permits to sort the output corners according to their measures, select a subset of the most distinctive ones, or select corners equally distributed on the image. Finally, the location of the interest points can be refined in order to obtain subpixel accuracy, using quadratic interpolation.

We analyze the method following the same approach as in [20]. In particular, we rely on the *repeatability rate* measure to study the performance of our implementation with respect to several geometric transformations, such as rotations, scalings, and affinities, as well as illumination changes and noise.

The basics of the Harris corner detector are explained in Section 2. Then, we analyze the steps of the algorithm in depth and study several alternatives in each one. Section 3 describes details of implementation and the parameters of the online demo. The experiments in Section 4 show the performance of the method with respect to the repeatability of the detector and the speed of our implementation. Finally, the conclusions are given in Section 5.

2 The Harris Corner Detector

The idea behind the Harris method is to detect points based on the intensity variation in a local neighborhood: a small region around the feature should show a large intensity change when compared with windows shifted in any direction.

This idea can be expressed through the autocorrelation function as follows: let the image be a scalar function $I : \Omega \to \mathcal{R}$ and h a small increment around any position in the domain, $x \in \Omega$. Corners are defined as the points x that maximize the following functional for small shifts h,

$$E(h) = \sum w(x) \left(I(x+h) - I(x)\right)^2, \tag{1}$$

i.e. the maximum variation in any direction. The function $w(x)$ allows selecting the support region that is typically defined as a rectangular or Gaussian function. Taylor expansions can be used to linearize the expression $I(x+h)$ as $I(x+h) \simeq I(x) + \nabla I(x)^T h$, so that the right hand of (1) becomes

$$E(h) \simeq \sum w(x) \left(\nabla I(x) h\right)^2 dx = \sum w(x) \left(h^T \nabla I(x) \nabla I(x)^T h\right). \tag{2}$$

This last expression depends on the gradient of the image through the autocorrelation matrix, or structure tensor, which is given by

$$M = \sum w(x) \left(\nabla I(x) \nabla I(x)^T\right) = \begin{pmatrix} \sum w(x) I_x^2 & \sum w(x) I_x I_y \\ \sum w(x) I_x I_y & \sum w(x) I_y^2 \end{pmatrix}. \tag{3}$$

The maxima of (2) are found through the analysis of this matrix. The largest eigenvalue of M corresponds to the direction of largest intensity variation, while the second one corresponds to the

intensity variation in its orthogonal direction. Analyzing their values, we may find three possible situations:

- Both eigenvalues are small, $\lambda_1 \approx \lambda_2 \approx 0$, then the region is likely to be a homogeneous region with intensity variations due to the presence of noise.
- One of the eigenvalues is much larger than the other one, $\lambda_1 \gg \lambda_2 \approx 0$, then the region is likely to belong to an edge, with the largest eigenvalue corresponding to the edge orthogonal direction.
- Both eigenvalues are large, $\lambda_1 > \lambda_2 \gg 0$, then the region is likely to contain large intensity variations in the two orthogonal directions, therefore corresponding to a corner-like structure.

Based on these ideas, the Harris method can be implemented through Algorithm 1. In the first step, the image is convolved with a Gaussian function of small standard deviation, in order to reduce image noise and aliasing artifacts.

Algorithm 1: harris

input : I, measure, κ, σ_d, σ_i, τ, strategy, cells, N, subpixel
output: corners

$\widetilde{I} \leftarrow$ gaussian(I, σ_d) *// 1. Smoothing the image*

$(I_x, I_y) \leftarrow$ gradient($\widetilde{I}$) *// 2. Computing the gradient of the image*

$(\widetilde{A}, \widetilde{B}, \widetilde{C}) \leftarrow$ compute_autocorrelation_matrix(I_x, I_y, σ_i) *// 3. Computing autocorrelation matrix*

R $\leftarrow$ compute_corner_response($\widetilde{A}$, $\widetilde{B}$, $\widetilde{C}$, measure, κ) *// 4. Computing corner strength*

corners $\leftarrow$ non_maximum_suppression(R, τ, $2\sigma_i$) *// 5. Non-maximum suppression*

select_output_corners(corners, strategy, cells, N) *// 6. Selecting output corners*

if *subpixel* **then**
  compute_subpixel_accuracy(R, corners) *// 7. Calculating subpixel accuracy*

The autocorrelation matrix is calculated from the gradient of the image, and its eigenvalues are used to identify any of the previous situations. A non-maximum suppression process allows selecting a unique feature in each neighborhood. In this function, it is necessary to specify a threshold to discard regions with small values. This threshold depends on the corner strength function used and is related with the level of noise in the images. In the last two steps, the points can be selected in several ways and the precision of the corners can be improved by using quadratic interpolation. The following sections explain each of these steps in detail and analyze different alternatives.

In order to improve the stability of the response, we propose a simple scale space approach in Algorithm 2. The stability is measured by checking that the corner is still present after a zoom out. At each scale, we first zoom out the image by a factor of two and compute the Harris' corners. This is done in a recursive way, as many times as specified by the number of scales chosen ($NScales$). Then, we compute the corners at the current scale and check that they are present in both scales (through function `select_corners`). The algorithm stops at the coarsest scale ($NScales = 1$).

Note that the value of σ_i is also reduced by a factor of two at the coarse scales. This allows to preserve the same area of integration. Algorithm 3 selects the points at the finer scale for which there exists a corner at a distance less than σ_i. The *corner* position is divided by two inside the *distance* function.

Algorithm 2: harris_scale

input : I, NScales, measure, κ, σ_d, σ_i, τ, strategy, cells, N, subpixel
output: corners

if *NScales*≤ 1 **then**
    // Compute Harris' corners at coarsest scale
    corners$\leftarrow$ harris(I, measure, κ, σ_d, σ_i, τ, strategy, cells, N, subpixel)
else
    // Zoom out the image by a factor of two
    $I_z \leftarrow$ zoom_out(I)
    // Compute Harris' corners at the coarse scale (recursive)
    $corners_z \leftarrow$
        harris_scale(I_z, NScales-1, measure, κ, σ_d, $\sigma_i/2$, τ, strategy, cells, N, subpixel)
    // Compute Harris' corners at the current scale
    corners$\leftarrow$ harris(I, measure, κ, σ_d, σ_i, τ, strategy, cells, N, subpixel)
    // Select stable corners
    corners$\leftarrow$ select_corners(corners, $corners_z$, σ_i)

Algorithm 3: select_corners

input : $corners_1$, $corners_2$, σ_i
output: corners

foreach *corner in* $corners_1$ **do**
    j $\leftarrow 0$
    // Search the corresponding corner
    while $j < size(corners_2)$ ***and*** $distance^2(corner, corners_2(j)) > \sigma_i^2$ **do**
        j $\leftarrow$ j+1
    if $j < size(corners_2)$ **then**
        corners.insert(corner)
    return corners

Step 1. Smoothing the Image

The purpose of this step is to reduce image noise and aliasing artifacts through the convolution with a Gaussian function. This step was not included in the original proposal, but it improves the performance of the method, as shown in [20]. In that case, the authors used Deriche's recursive filter [5] for computing the derivatives of a Gaussian (with $\sigma = 1$), so that the first two steps (convolution with a Gaussian and gradient estimation) are calculated in a single step.

In order to convolve the image with a Gaussian function, we use the implementations given in [8]. Since we are interested in a fast implementation, we chose the method based on stacked integral images (SII) [2], which is the fastest approach.

In the experiments, we use the two images shown in Figure 1, of a building and a calibration pattern. We analyze the behavior of the feature detector using the repeatability rate measure as defined in [20] (see Section 4 for more details).

Figure 2 compares the repeatability rate using different Gaussian convolution implementations: 'No Gaussian' means that no smoothing was applied to the image, as in the original Harris method;

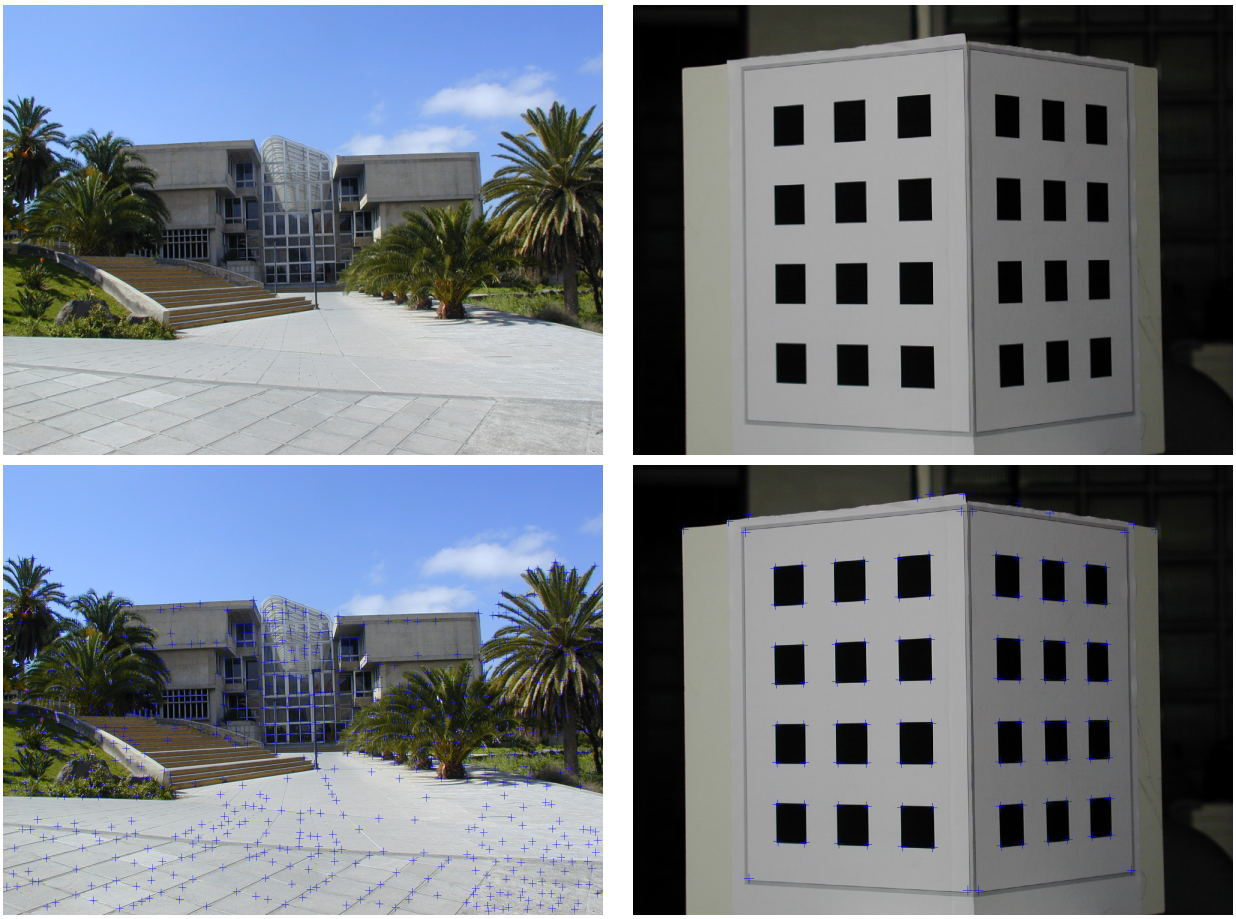


Figure 1: Test sequences used in the experiments and their Harris corners: on the left, the *building* sequence and, on the right, the *calibrator* sequence.

'Discrete Gaussian' stands for a convolution with a discrete Gaussian kernel; 'Fast Gaussian' uses the SII method, which is faster but less accurate than the previous one. A Gaussian convolution is also used for computing the autocorrelation matrix in Step 3. In each case, we used the same Gaussian filter and, for the 'No Gaussian' graphic, we chose the SII strategy. The repeatability rate is computed for rotation angles between 0^0 and 180^0. This detector is rotationally invariant, so we expect to obtain a high repeatability rate in general.

From these graphics, we conclude that the Gaussian convolution improves the repeatability rate, as shown in [20], and the precision is apparently similar regardless of the Gaussian implementation.

Figure 3 shows the repeatability rate with respect to different values of ϵ (see Section 4). At the top, we present the results for the discrete Gaussian filter —using *building* on the left and *calibrator* on the right— and, at the bottom, the results for the SII implementation.

Analyzing the results of the building, we may conclude the following: The precision is better for 0^0, 90^0 and 180^0, which has to do with the accuracy of the gradient in these orientations; although the results are similar, we observe a slightly better behavior of the discrete Gaussian convolution; the result for values smaller than $\epsilon = 1.5$ is definitely better for the discrete Gaussian.

In the case of *calibrator*, the results are similar in both cases. The repeatability rate is very high —almost 100%— for $\epsilon \geq 1.5$. This is due to the simplicity of this image, where corners are easily detected on the pattern.

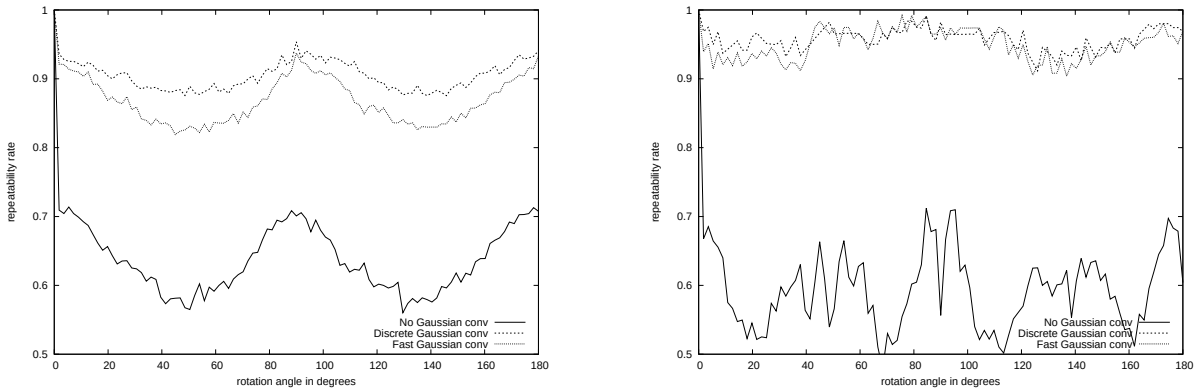


Figure 2: Comparison using several Gaussian filter implementations. We compare the performance of the method using a discrete Gaussian filter, stacked integral images (SII) [2] and no Gaussian convolution, in the first step. On the left, we show the result for the *building* sequence and, on the right, the result for the *calibrator* sequence. The precision is high in both cases, with nearly 100% precision for the calibration pattern series. The use of Gaussian convolutions is important for improving the accuracy. The parameters used for this test are: Harris measure, $\kappa = 0.06$, $\sigma_d = 1$, $\sigma_i = 2.5$, $\tau = 130$, subpixel accuracy, and $\epsilon = 1$.

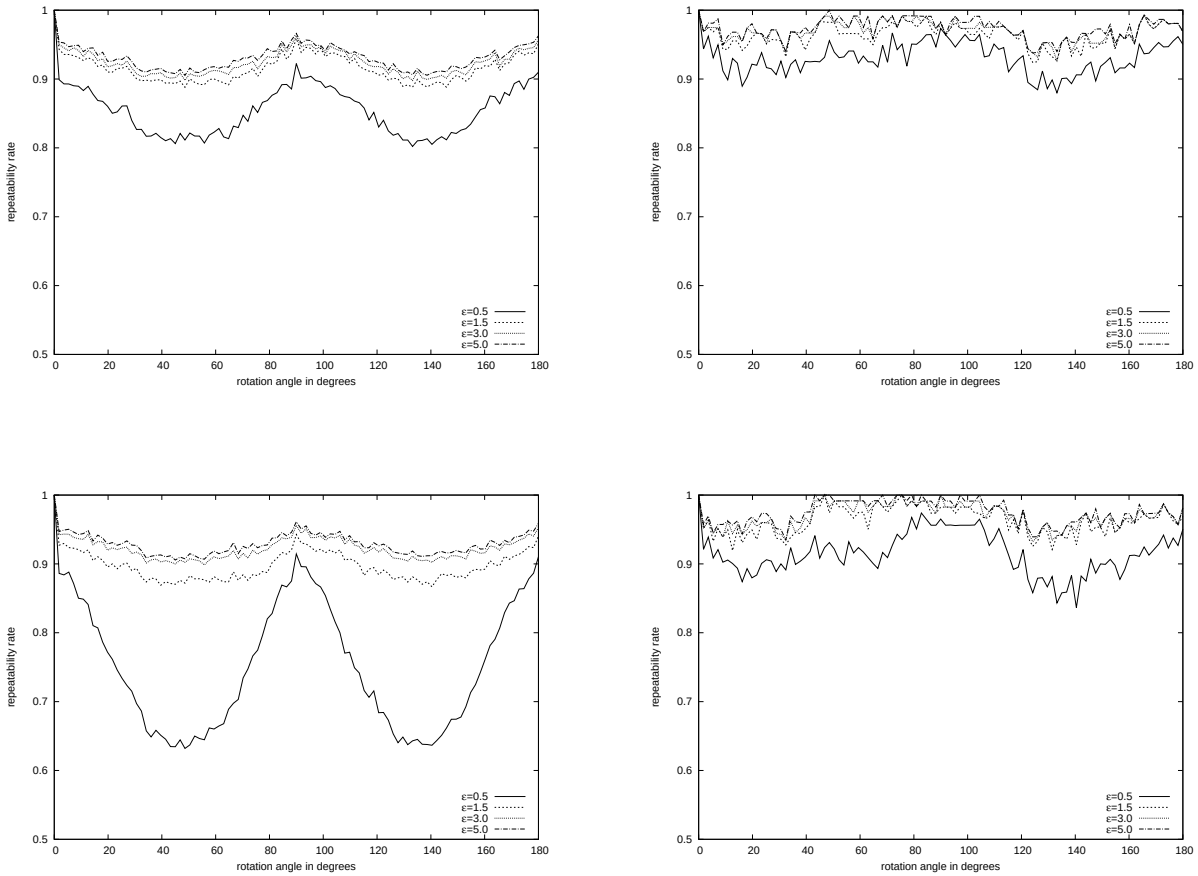


Figure 3: Comparison using two different Gaussian filter implementations. On the top row, we show the results for the discrete Gaussian with the *building* and *calibrator* images, respectively. On the bottom, the results for the SII convolutions. Using a more precise Gaussian implementation improves the repeatability rate, especially at orientations where the estimation of the gradient is less accurate. The parameters used for this test are: Harris measure, $\kappa = 0.06$, $\sigma_d = 1$, $\sigma_i = 2.5$, $\tau = 130$, and subpixel accuracy.

$\frac{1}{2}$ | -1 | 0 | 1 |

$\frac{1}{2}$ | 1 |
|---|
| 0 |
| -1 |

(a) Central differences

$\frac{1}{8}$	-1	0	1
-2	0	2
-1	0	1

$\frac{1}{8}$	1	2	1
0	0	0
-1	-2	-1

(b) Sobel operator

Table 1: Gradient masks: ∂_x and ∂_y masks for central differences and Sobel operator.

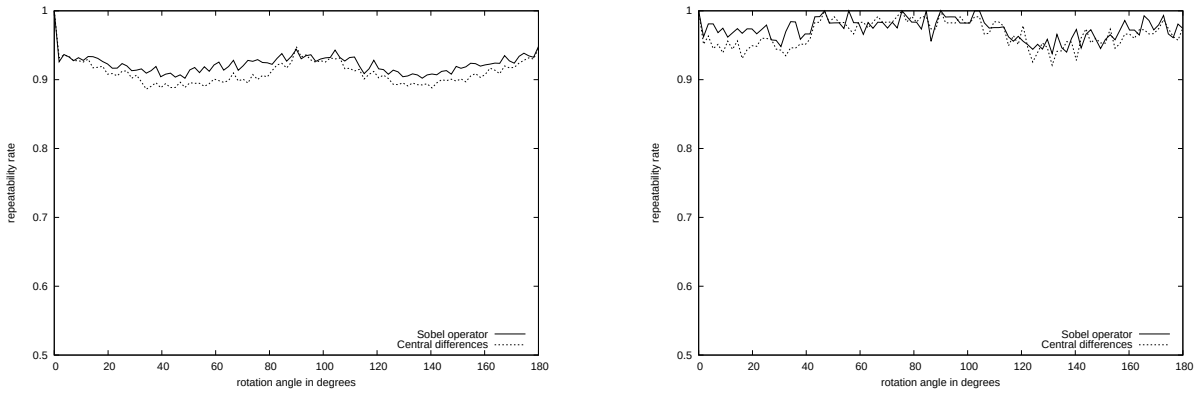


Figure 4: Comparison using two different gradient masks. The repeatability rate is in general high using any of the gradient masks. The result for the *building* sequence, on the left, is slightly better if we use the Sobel operator. The result for the *calibrator* sequence, on the right, is similar in both cases.

Step 2. Computing the Gradient of the Image

Hereinafter, we choose the SII filter for the convolution with a Gaussian function in the first and third steps. In order to study the influence of the gradient in the detector, we analyze the gradient masks given in Table 1.

Harris and Stephens [9] used central differences in their work. Figure 4 compares the repeatability rate for these gradient masks and rotations between 0^0 and 180^0. We observe that the precision is similar in all cases. The Sobel operator is slightly better than central differences for *building* although it is not relevant. Note that this mask introduces an additional smoothing, which is aggregated to the previous Gaussian convolution. The precision is better at 0^0, 90^0 and 180^0, where the gradient is more accurate. For the *calibrator* sequence, the results are very precise in both cases. We may conclude that the influence of the gradient mask is not meaningful.

Step 3. Computing the Autocorrelation Matrix

Algorithm 4 shows the steps for computing the autocorrelation matrix (3). The products of the derivatives are calculated at each position and the coefficients of the matrix are convolved with a Gaussian function. By default, we use the SII method. The standard deviation, σ_i, defines the region of integration.

This step is the slowest of the method because of the three Gaussian convolutions. The SII method is very fast and the execution time remains constant independently of the value of σ_i, except for border initializations. Typically, the runtime of the algorithm increases with the value of σ_i for other similar filters.

Algorithm 4: compute_autocorrelation_matrix

// Gradient of the image and standard deviation
input : I_x, I_y, σ_i
// Coefficients of the autocorrelation matrix
output: $\widetilde{A}$, $\widetilde{B}$, $\widetilde{C}$

// Compute coefficients of the autocorrelation matrix in each pixel
foreach *pixel at* (i, j) **do**
  A(i,j) $\leftarrow \mathrm{I}_x^2$(i,j)
  B(i,j) $\leftarrow \mathrm{I}_x$(i,j) I_y(i,j)
  C(i,j) $\leftarrow \mathrm{I}_y^2$(i,j)

// Convolve its elements with a Gaussian function
$\widetilde{A} \leftarrow$ gaussian(A, σ_i)
$\widetilde{B} \leftarrow$ gaussian(B, σ_i)
$\widetilde{C} \leftarrow$ gaussian(C, σ_i)

Step 4. Computing the Corner Strength Function

The autocorrelation matrix is symmetric and positive semidefinite, yielding two real non-negative eigenvalues. Analyzing these eigenvalues, we may define corner response functions that are invariant to in-plane rotations. We implement the following standard functions (see also Table 2):

Harris and Stephens [9]	$R_H = \lambda_1\lambda_2 - \kappa \cdot (\lambda_1 + \lambda_2)^2$
Shi and Tomasi [21]	$R_{ST} = \begin{cases} \lambda_{min} & \text{if } \lambda_{min} > \tau \\ 0 & \text{Otherwise} \end{cases}$
Harmonic mean [3]	$R_{HM} = \frac{2\lambda_1\lambda_2}{\lambda_1+\lambda_2}$

Table 2: Typical corner strength functions used in the literature. These are based on the analysis of the eigenvalues, λ_1 and λ_2, of the autocorrelation matrix (3); $\lambda_{min} = \min(\lambda_1, \lambda_2)$.

Harris measure [9]: The measure proposed by Harris and Stephens is given by

$$R_H = \lambda_1\lambda_2 - \kappa \cdot (\lambda_1 + \lambda_2)^2 = det(M) - \kappa \cdot trace(M)^2,$$

where κ is a value typically between 0.04 or 0.06. This function not only allows to calculate interest points but also to detect edges. Figure 5 depicts the regions of this detector in the $\lambda_1 - \lambda_2$ plane. When the value of R is negative, it means that one of the eigenvalues is much larger than the other one and the pixel is likely to belong to an edge. When R is positive and large, both eigenvalues are large, then it is likely to be a corner. If both eigenvalues are small, R is small and it is part of a flat region.

Shi and Tomasi [21]: This is based on the minimum eigenvalue of the autocorrelation matrix, which is given by

$$\lambda_{min} = \frac{\widetilde{A} + \widetilde{C} - \sqrt{(\widetilde{A} - \widetilde{C})^2 + 4\widetilde{B}^2}}{2}. \tag{4}$$

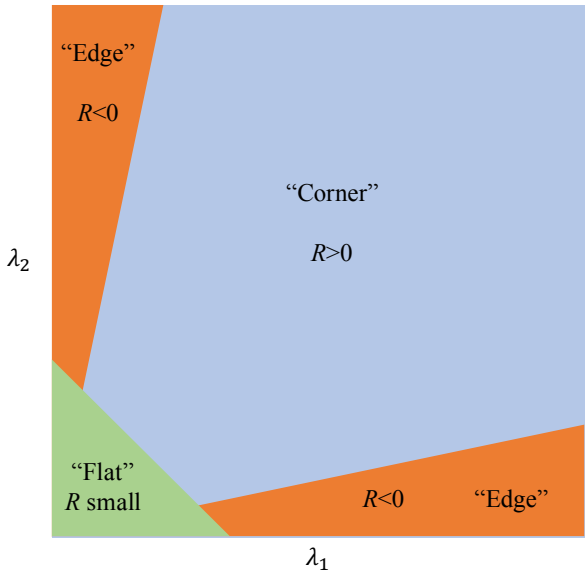


Figure 5: Harris measure. If both eigenvalues are large, then R is large and positive, providing a clue to detect a corner. If both eigenvalues are small, then R is also small and positive, which means that the point is probably part of a homogeneous region. Finally, if one of the eigenvalues is much larger than the other, R becomes negative, and the point belongs to an edge.

The corner response function is

$$R_{ST} = \begin{cases} \lambda_{min} & \text{if } \lambda_{min} > \tau \\ 0 & \text{otherwise} \end{cases}. \tag{5}$$

Although the discriminant function is more intuitive than the Harris measure, it requires more operations for computing the minimum eigenvalue.

Harmonic mean [3]: The shape of this function is similar to the Harris measure, with the benefit that it does not require an additional parameter. It is defined by

$$R_{HM} = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2} = \frac{det(M)}{trace(M)}.$$

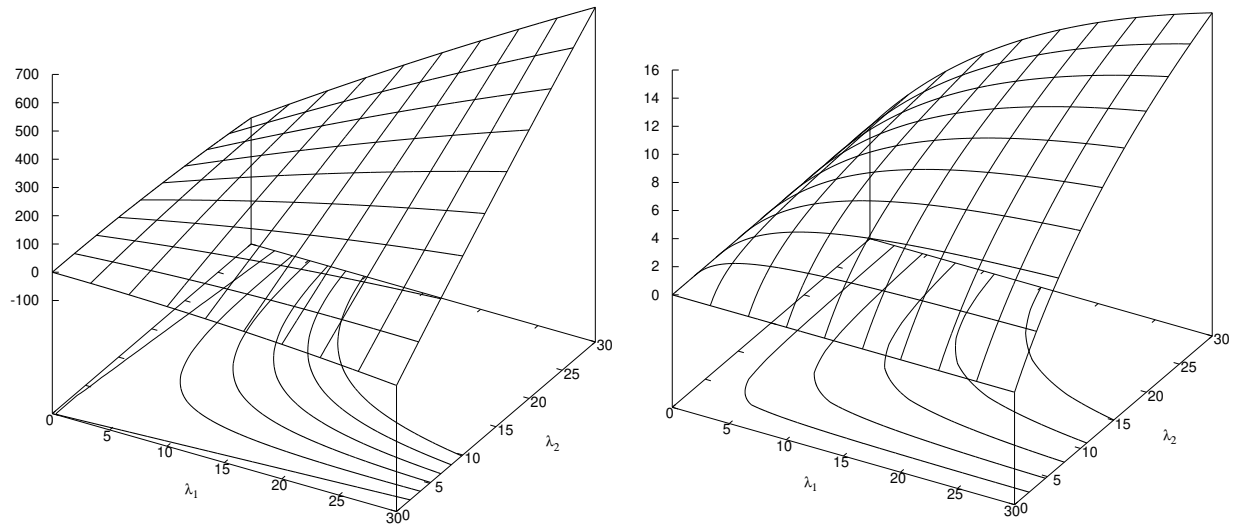
The Harris and harmonic mean functions are similar, as we can see in Figure 6. However, the Harris measure may have negative values, which allows to differentiate between corners and edges. On the other hand, the range of values of both functions is different —see the corresponding values for the level lines in each figure—. This means that the threshold used in the following step has to be adapted to each function.

Step 5. Non-Maximum Suppression

The purpose of the non-maximum suppression step is to find the best interest point in each local neighborhood. This technique is typically used for refining edge-like structures [4, 22], selecting points [10, 7, 15, 16], or discriminating among simultaneous object detections [6].

The maxima of the corner strength function contain the interest points. However, many of these points will be located close to each other, thus, it is necessary to select the best candidates.

In several works, the maxima are obtained from the 3×3 neighbors around the feature, as in [9, 20]. This means that two interest points can be separated by one pixel only. This can be



(a) Harris measure: plot of $R_H = \lambda_1\lambda_2 - \kappa \cdot (\lambda_1 + \lambda_2)^2$. The level lines in the $\lambda_1 - \lambda_2$ plane correspond to $R_H = \{0, 40, 80, 120, 160, 200\}$.

(b) Harmonic mean: plot of $R_{HM} = \frac{\lambda_1\lambda_2}{\lambda_1+\lambda_2}$. The level lines in the $\lambda_1 - \lambda_2$ plane correspond to $R_{HM} = \{2, 4, 6, 8, 10\}$.

Figure 6: Corner response functions. The shapes of the Harris and the harmonic mean measures are similar but the range of values is very different. Additionally, the Harris measure can have negative values, which can be used to detect edge points.

an acceptable distance for low resolution images, however, this is not convenient for high resolution ones.

For this reason, we chose to implement a generic algorithm that extracts points using a radius r. Intuitively, this radius must be related to the size of the integration kernel, σ_i, used in Step 3. A reasonable choice is to calculate the radius as $r = 2\sigma_i$. In this way, the features will be separated by the area of influence of the Gaussian function.

Computing the maximum of an array only requires one comparison per pixel, but finding the local maxima is much more cumbersome. A brute force algorithm has a computational cost of $O((2r+1)^2 N)$, with N the number of pixels. Current methods require around two comparisons per pixel on average [15, 16].

Algorithm 5 shows the steps of our method, which is similar in spirit to the method proposed in [16]. Since the number of local maxima is expected to be much smaller than the number of pixels, the key idea is to reject points as soon as a bigger value is detected —this is the reason to use so many conditional loops—.

At the beginning, we take into account a threshold to skip low values in the corner strength function. This threshold depends on the level of noise in the image and on the function selected. The Harris measure and the harmonic mean are similar but the threshold must be higher in the first function to approximately select the same number of points, as depicted in Figure 6. The threshold for the Shi-Tomasi function should be in general smaller than the previous ones.

For each line, we first reject border points. The next peak is found and compared with the pixels on the right and then on the left. We use an array, *skip*, that allows us to jump positions that have been already visited and are smaller than a neighbor. It is interesting to prioritize the non-visited neighbors first so that the number of comparisons are reduced and new positions are marked as skippable. If the selected point is the maximum in its line, it is then compared with the 2D regions, of size $(2r+1) \times r$, below and above the current line.

Some improvements can be made in this algorithm. The method proposed in [15], for instance, only requires two comparisons with the previous pixels in the same line, but it is more complex to

Algorithm 5: non_maximum_suppression

```
input  : R, τ, radius
output: corners
foreach pixel at (i, j) do
    if R(i, j) < τ then skip(i,j) ← true                    // apply threshold
    else skip(i,j) ← false                                  // initialize variable
for i ← radius to N_y − radius do
    j ← radius
    while j < N_x − radius and (skip(i, j) or R(i, j − 1) ≥ R(i, j)) do
        j ← j+1                                             // avoid the downhill at the beginning
    while j < N_x − radius do
        while j < N_x − radius and (skip(i, j) or R(i, j + 1) ≥ R(i, j)) do
            j ← j+1                                         // find the next peak
        if j < N_x − radius then
            p_1 ← j+2
            while p_1 ≤ j + radius and R(i, p_1) < R(i, j) do
                skip(i,p_1) ← true
                p_1 ← p_1+1                                 // find a bigger value on the right
            if p_1 > j + radius then
                p_2 ← j-1                                   // if not found
                while p_2 ≥ j − radius and R(i, p_2) ≤ R(i, j) do
                    p_2 ← p_2 − 1                           // find a bigger value on the left
                if p_2 < j − radius then
                    k ← i+radius                            // if not found, test the 2D region
                    found ← false
                    while not found and k > i do
                        l ← j+radius                        // first test the bottom region
                        while not found and l ≥ j − radius do
                            if R(k,l) > R(i,j) then found ← true
                            else skip(k,l) ← true
                            l ← l-1
                        k ← k-1
                    k ← i-radius
                    while not found and k < i do
                        l ← j-radius                        // then test the top region
                        while not found and l ≤ j + radius do
                            if R(k, l) ≥ R(i, j) then found ← true
                            l ← l+1
                        k ← k+1
                    if not found then
                        corners.add(i,j,R(i,j))             // a new local maximum detected
            j ← p_1
```

implement.

Step 6. Selecting Output Corners

We implemented the following strategies for selecting the output corners:

- The simplest one is to select all the corners detected. In this case, the sorting is given by the non-maximum suppression algorithm, i.e. sorted by rows and then by columns.

- In the second strategy, all the corners are sorted by their corner strength values in descending order. This is interesting for applications that need to process the more discriminant features first.

- Another alternative is to select a subset of the corners detected. The user specifies a number of corners to be found and the application returns the set with the highest discriminant values. The corners are also sorted in descending order. It is possible that the number specified by the user is bigger than the corners detected.

- Finally, we may also select a set of corners equally distributed on the image, which is interesting for several applications such as camera calibration, panorama stitching [3] or video stabilization [18]. In that case, the user specifies a number of cells and the total number of points to be detected. The algorithm tries to find the same amount of points in each cell. It is possible that no distinctive points are detected in some cells, so, in general, the number of features will be smaller than the target number of points specified by the user.

Another interesting alternative is the *adaptive non-maximum suppression* algorithm proposed in [3], which generates spatially well distributed features over the image.

Step 7. Calculating Subpixel Accuracy

Given any of the corner strength functions of Section 2, it is possible to refine the features with subpixel accuracy. For this, we need to estimate an approximate function in a local neighborhood around each feature and then find its maximum. We implemented two interpolation methods based on quadratic and quartic polynomials, respectively.

Quadratic Approximation

We follow the same approach as in [3]. The corner strength function is approximated by a quadratic polynomial around each feature, $\mathbf{x}_f = (x_f, y_f)$, as

$$P(\mathbf{x}) = R(\mathbf{x}_f) + \frac{\partial R(\mathbf{x}_f)^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T \frac{\partial^2 R(\mathbf{x}_f)}{\partial \mathbf{x}^2}\mathbf{x}. \tag{6}$$

The derivatives of the function are calculated as

$$\begin{aligned}
\frac{\partial R(\mathbf{x}_f)}{\partial x} &= \frac{(R(x_f+1,y_f)-R(x_f-1,y_f))}{2} \\
\frac{\partial R(\mathbf{x}_f)}{\partial y} &= \frac{R(x_f,y_f+1)-R(x_f,y_f-1)}{2} \\
\frac{\partial^2 R(\mathbf{x}_f)}{\partial^2 x} &= R(x_f+1,y_f)-2R(x_f,y_f)+R(x_f-1,y_f) \\
\frac{\partial^2 R(\mathbf{x}_f)}{\partial^2 y} &= R(x_f,y_f+1)-2R(x_f,y_f)+R(x_f,y_f-1) \\
\frac{\partial^2 R(\mathbf{x}_f)}{\partial x \partial y} &= \frac{R(x_f+1,y_f+1)+R(x_f-1,y_f-1)-R(x_f+1,y_f-1)-R(x_f-1,y_f+1)}{4}.
\end{aligned} \tag{7}$$

Deriving (6) to find its maximum, and shifting to the feature position, gives the subpixel location as

$$\mathbf{x}_{max} = \mathbf{x}_f - \frac{\partial^2 R(\mathbf{x}_f)}{\partial \mathbf{x}^2}^{-1} \frac{\partial R(\mathbf{x}_f)}{\partial \mathbf{x}}. \tag{8}$$

Quartic Interpolation

Another possibility is to calculate an interpolation polynomial using the nine points around the feature. This polynomial must have nine degrees of freedom and can be expressed as

$$P(x,y) = a_0x^2y^2 + a_1x^2y + a_2xy^2 + a_3x^2 + a_4y^2 + a_5xy + a_6x + a_7y + a_8. \tag{9}$$

The coefficients of this polynomial are easily obtained if we assume that it is centered at the feature position. Indeed, substituting $x \in [-1,1]$ and $y \in [-1,1]$ in the polynomial, we obtain the following system of equations

$$\begin{pmatrix}
1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 1 \\
1 & -1 & 1 & 1 & 1 & -1 & 1 & -1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
1 & 1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}
\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \end{pmatrix} =
\begin{pmatrix}
R(x_f-1,y_f-1) \\ R(x_f,y_f-1) \\ R(x_f+1,y_f-1) \\ R(x_f-1,y_f) \\ R(x_f,y_f) \\ R(x_f+1,y_f) \\ R(x_f-1,y_f+1) \\ R(x_f,y_f+1) \\ R(x_f+1,y_f+1)
\end{pmatrix}. \tag{10}$$

The solution is given by

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \\ a_8 \end{pmatrix} =
\begin{pmatrix}
R_{0,0} - \frac{R_{0,-1}+R_{0,1}+R_{-1,0}+R_{1,0}}{2} + \frac{R_{-1,-1}+R_{-1,1}+R_{1,-1}+R_{1,1}}{4} \\
\frac{R_{0,-1}-R_{0,1}}{2} + \frac{-R_{-1,-1}+R_{-1,1}-R_{1,-1}+R_{1,1}}{4} \\
\frac{R_{-1,0}-R_{1,0}}{2} + \frac{-R_{-1,-1}-R_{-1,1}+R_{1,-1}+R_{1,1}}{4} \\
\frac{R_{-1,0}+R_{1,0}}{2} - R_{0,0} \\
\frac{R_{0,-1}+R_{0,1}}{2} - R_{0,0} \\
\frac{R_{-1,-1}-R_{-1,1}-R_{1,-1}+R_{1,1}}{4} \\
\frac{R_{1,0}-R_{-1,0}}{2} \\
\frac{R_{0,1}-R_{0,-1}}{2} \\
R_{0,0}
\end{pmatrix}, \tag{11}$$

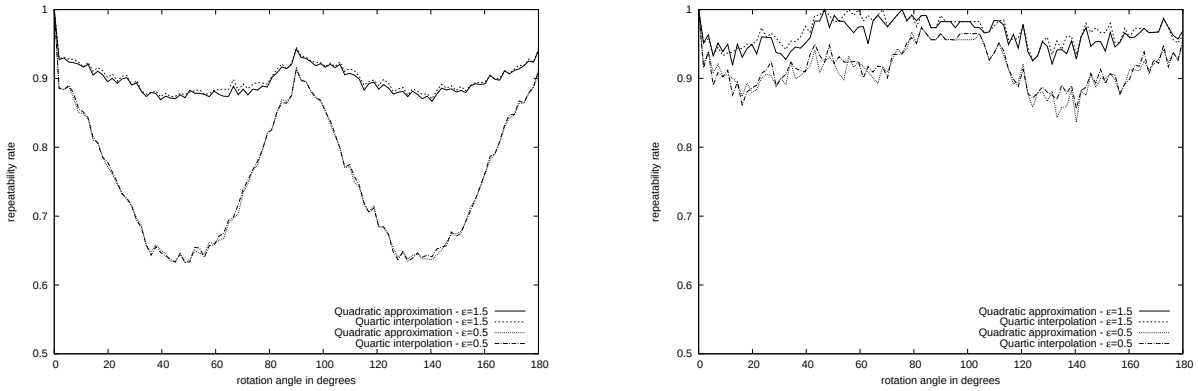


Figure 7: Comparison of subpixel accuracy strategies. These graphics compare the repeatability rate using quadratic and quartic interpolation for the *building* and *calibrator* test sequences, respectively. Both approaches provide similar results, although the quartic interpolation is slightly more accurate.

with subscripts denoting shifts around the feature. Since the degree of the polynomial is four, it may have several extrema. We implement the Newton method in order to find a maximum. The iterative scheme is given by

$$\mathbf{x}^{n+1} = \mathbf{x}^n - \frac{\partial^2 P(\mathbf{x}^n)}{\partial \mathbf{x}^2}^{-1} \frac{\partial R(\mathbf{x}^n)}{\partial \mathbf{x}}, \tag{12}$$

with $\mathbf{x}^0 = \mathbf{0}$. The final solution is obtained as

$$\mathbf{x}_{max} = \mathbf{x}_f + \mathbf{x}^{n+1}. \tag{13}$$

This algorithm is typically very fast and converges in a few iterations (3–5 on average). We define a maximum number of iterations in order to stop the process.

Figure 7 compares both approaches. We observe that the results of the quartic interpolation are better for both ϵ values, but the difference is not meaningful. The quadratic approach provides a closed-form solution, is simpler to implement and faster. The runtime of the quadratic approximation is between two and three times faster than the quartic interpolation on average. Note, however, that the time spent in this step is very small in comparison with other steps in the algorithm, as we can see in the following section.

3 Details of Implementation and Online Demo

The online demo allows selecting the options in each step. The user can choose between different Gaussian and gradient strategies. The rest of parameters of the demo are explained in Table 3.

Choosing the Discrete or Fast Gaussian strategies results in less features detected. They usually provide a similar number of features. The No Gaussian alternative produces many more features in general due to the effect of noise. The Sobel operator provides slightly fewer features than central differences, because of the additional smoothing.

The zoom out reduces the number of features by approximately a factor of z^2. If we want to obtain a similar number of features, we can divide σ_i by z. Increasing the value of σ_d has a behavior similar to the zoom factor, thus, we can again decrease the value of σ_i to maintain the same amount of corners. If we increase σ_i, the features tend to get farther from the true corners. Additionally, the selection area in the non-maximum suppression algorithm is bigger, therefore, it obtains less points. Each corner strength function needs a different τ for selecting the same amount of points.

Parameter	Description
Zoom	Zoom out the input image by a factor of 1, 2, 4, 8 and 16. Default value 1.
Scale check	Number of scales used in Algorithm 2 for checking corner stability. Default 1.
Gaussian	Determines the Gaussian convolution method to be used (Discrete Gaussian, Fast Gaussian – SII method – and No Gaussian). Default value 'Discrete Gaussian'.
Gradient	Choose between central differences or the Sobel operator for computing the gradient of the image. Default value 'central differences'.
σ_d	Standard deviation of the Gaussian used for smoothing the image (step 1). Default value 1.
σ_i	Standard deviation of the Gaussian used for computing the autocorrelation matrix (step 3). Default value 2.5.
Corner function	Specifies the corner strength function to be computed (Harris, Shi-Tomasi or Harmonic Mean, in step 4)
κ	Value of the Harris constant. Default value 0.006.
τ	Threshold used for rejecting small values (step 5). Default values are 130 for the Harris function, 10 for Shi-Tomasi and 15 for the Harmonic mean.
Output	Strategy for selecting the output corners (All, Sorted, N Sorted and N Distributed, in step 6)
Number of corners	Number of output corners in the third and fourth previous strategies.
Cells	Number of cells to distribute the selected points (3 means a mesh of 3×3 cells of equal size on the image).
Subpixel accuracy	Choose between no subpixel accuracy, quadratic or quartic interpolation (step 7).

Table 3: Parameters of the method

Table 4 shows the execution times for the *building* sequence, whose size is 1600x1200 pixels. The number of corners detected is 1722. The output corners are sorted.

Step	Time
1. Smoothing the image	9.51 ms
2. Computing the gradient	4.16 ms
3. Computing the autocorrelation matrix	50.03 ms
4. Computing corner strength function	2.83 ms
5. Non-maximum suppression	4.48 ms
6. Selecting output corners	0.08 ms
7. Calculating subpixel accuracy	0.13 ms
Total:	**71.22ms**

Table 4: Execution times for each step using the *building* sequence. The size of this image is 1600×1200 pixels and the number of features detected was 1722. We set default parameters.

The time of the last two steps is negligible in comparison with the previous steps. These depend on the detected corners and not on the image itself. The most expensive process is the estimation of the autocorrelation matrix, which depends on the convolution with a Gaussian function. To improve the runtime of the algorithm, it would be necessary to accelerate the Gaussian convolutions. Another alternative is to replace it with a box-filter. The non-maximum suppression algorithm is efficient: it is as fast as the computation of the gradient.

4 Experiments

In our experiments, we used the two images shown in Figure 1. We analyzed the behavior of the feature detector using the repeatability measure as defined in [20]. The main idea was to apply a known transformation to the input image and to study the rate of features that appear in both cases. This measure was computed from points that lie in all images.

First, we selected the interest points in each image by checking if, after applying the transformation, they lie inside the dimensions of the image, after having removed a margin of $radius = 2\sigma_i$. The transformation was synthetically generated as a homography. For the second image, we used the inverse homography. Then, the set with the smaller number of points was selected as the reference set. As in [20], the $\epsilon - repeatability$ was used to represent the ratio of points that are at a distance smaller than ϵ of their corresponding point. The set of point pairs $(\tilde{x}_1, \tilde{x}_i)$, which correspond within an ϵ-neighborhood, is defined by

$$R_i(\epsilon) = \{(\tilde{x}_1, \tilde{x}_i) | dist(H_{1,i}\tilde{x}_1, \tilde{x}_i) < \epsilon)\}.$$

The repeatability rate is defined as

$$r_i(\epsilon) = \frac{\|R_i(\epsilon)\|}{\min(n_1, n_i)}.$$

The type of transformations used in these experiments were rotations, scale changes, illumination variations and affine transformations. We also analyzed the behavior with respect to image noise. For a more realistic behavior, we added white Gaussian noise to the images after each transformation. For the first image, and after adding noise, we computed the reference set of features.

4.1 Repeatability with Respect to Geometric Transformations

The Harris detector is in principle invariant to in-plane rotations. Figure 9 shows the repeatability rate for different values of ϵ and rotations between 0^0 and 180^0 (see Figure 8). These are the same used in the graphics of Figure 3.

Figure 8: Several images of the rotation test.

The repeatability rate is especially higher for 0^0, 90^0 and 180^0, and, as explained above, this has to do with the accuracy of the gradient at these orientations.

Figure 11 shows the behavior of the Harris detector with respect to scale changes. We chose zoom factors between 0.2 and 4, as shown in Figure 10.

The maximum rate is obtained around 1, where the image has the same scale as the original image. The graphics show that this detector is not invariant to scale changes.

Finally, we examined its behavior with respect to affine transformations. Figure 12 depicts the images used in this experiment, where we have changed the skew parameter.

The graphics in Figure 13 show that the repeatability rate decreases fast with the increase of the skew parameter.

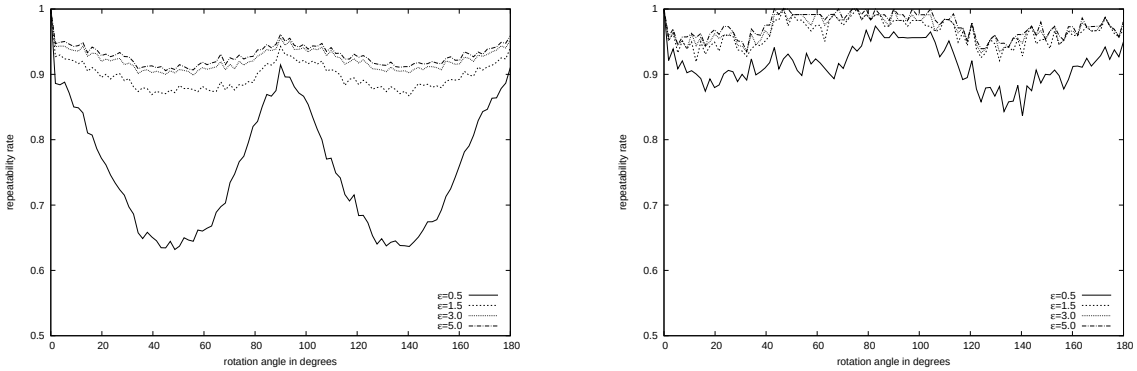


Figure 9: Repeatability rate for rotations. On the left, we show the repeatability rate for the *building* sequence and, on the right, for the *calibrator* sequence.

Figure 10: Several images of the scale test.

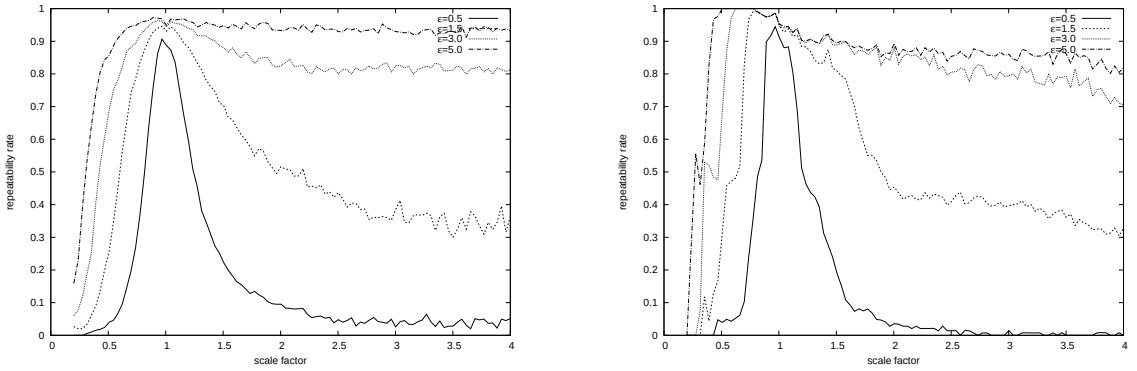


Figure 11: Repeatability rate for scalings. On the left, we show the repeatability rate for the *building* sequence and, on the right, for the *calibrator* sequence.

4.2 Repeatability with Respect to Changes in Illumination

Next, we changed the luminance conditions of the input image as $I' = \alpha I$, with $\alpha \in [0.2, 6]$. Figure 14 shows several images used in the test and Figure 15 depicts the results of the repeatability rate for the *building* and *calibrator* sequences, respectively.

The change in intensity has a direct relation with the magnitude of the eigenvalues of the auto-correlation matrix. The repeatability measure remains high as long as the chosen threshold is below the Harris measure of the selected corners. With a small α, the value of the corners will be below

Figure 12: Several images used for the affine transformation test.

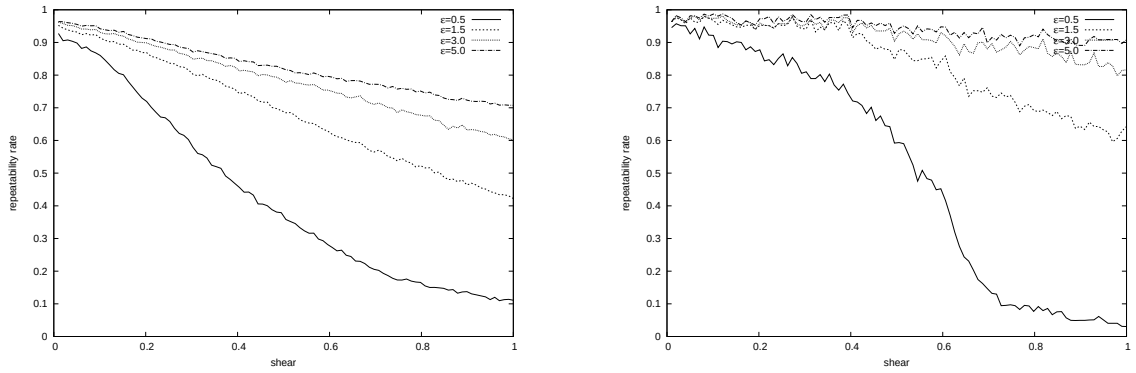


Figure 13: Repeatability rate for affinity transformations. On the left, we show the repeatability rate for the *building* sequence and, on the right, for the *calibrator* sequence.

Figure 14: Several images of the change in illumination test. These images are obtained as $I' = \alpha I$, with $\alpha \in [0.2, 6]$.

the threshold, and a large constant will saturate the image.

In the case of the *building* sequence, the repeatability measure decreases fast, with maximum coincidence around $\alpha = 1$. For the *calibrator*, this rate stays high because there is a good contrast at the corners of the square pattern.

4.3 Repeatability with Respect to Noise

In the last experiment, we studied the behavior of the detector with respect to noise. Figure 16 shows several images for this test. We added white Gaussian noise of increasing standard deviation ($\sigma \in [0, 30]$). The number of features augments with the level of noise.

Figure 17 depicts the evolution of the repeatability rate for increasing levels of noise. The graphic decreases very fast for the *building* sequence.

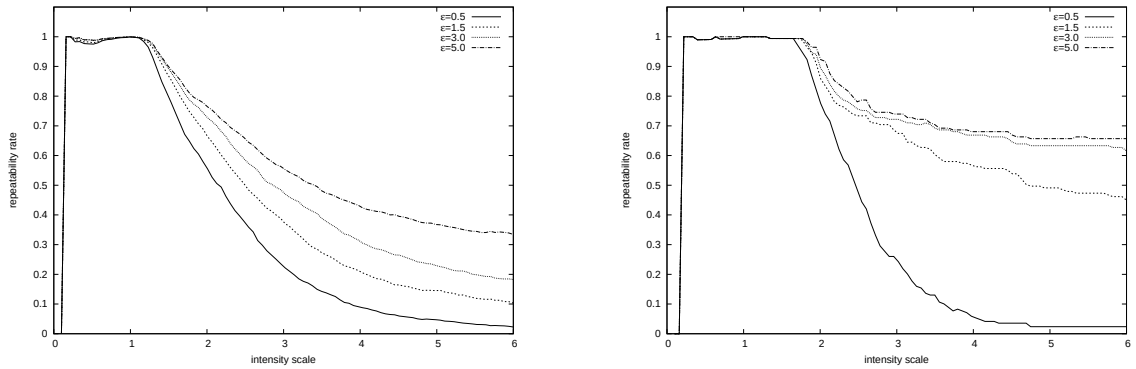


Figure 15: Repeatability rate for illumination changes. On the left, we show the repeatability rate for the *building* sequence and, on the right, for the *calibrator* sequence.

Figure 16: Several images of the noise test. We added white Gaussian noise of standard deviation $\sigma \in [0, 30]$.

This shows that the detector is not robust against noise. One way to tackle this problem is to adapt the values of σ_d, to reduce the level of noise, or σ_i, to integrate in larger regions. However, these strategies are limited, since they affect the structures of the objects or obtain features which represent larger regions of support, respectively.

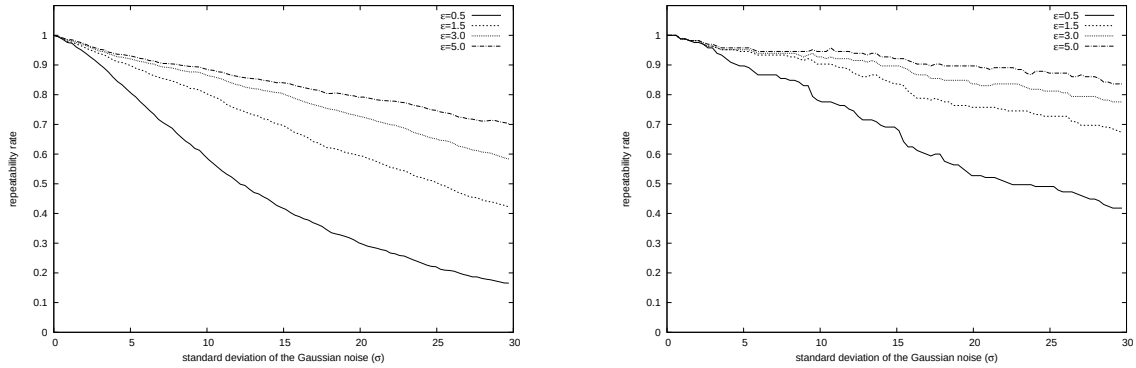


Figure 17: Repeatability rate for additive white Gaussian noise. On the left, we show the repeatability rate for the *building* sequence and, on the right, for the *calibrator* sequence.

4.4 Comparison with OpenCV

In this section, we compare the OpenCV 3.4.1 implementation with our own. We use the `cornerHarris` function, which is based on the Sobel operator, for computing the gradient of the image, box filtering, for calculating the autocorrelation matrix, and the Harris function. It does not use Gaussian convolutions, which makes it faster. If we compare it with our implementation, it only includes steps 1 through 4, until the computation of the corner strength function, and does not include the subsequent steps. Thus, for comparison purposes, we utilize as input the OpenCV estimation and then use our implementation for the last steps. We adapt the input parameters in order to obtain similar results.

Figures 18 and 19 compare the repeatability rate of both implementations. We selected the best 1500 features using the *building* sequence. The threshold is set to 0, so that it was always possible to select that number of features.

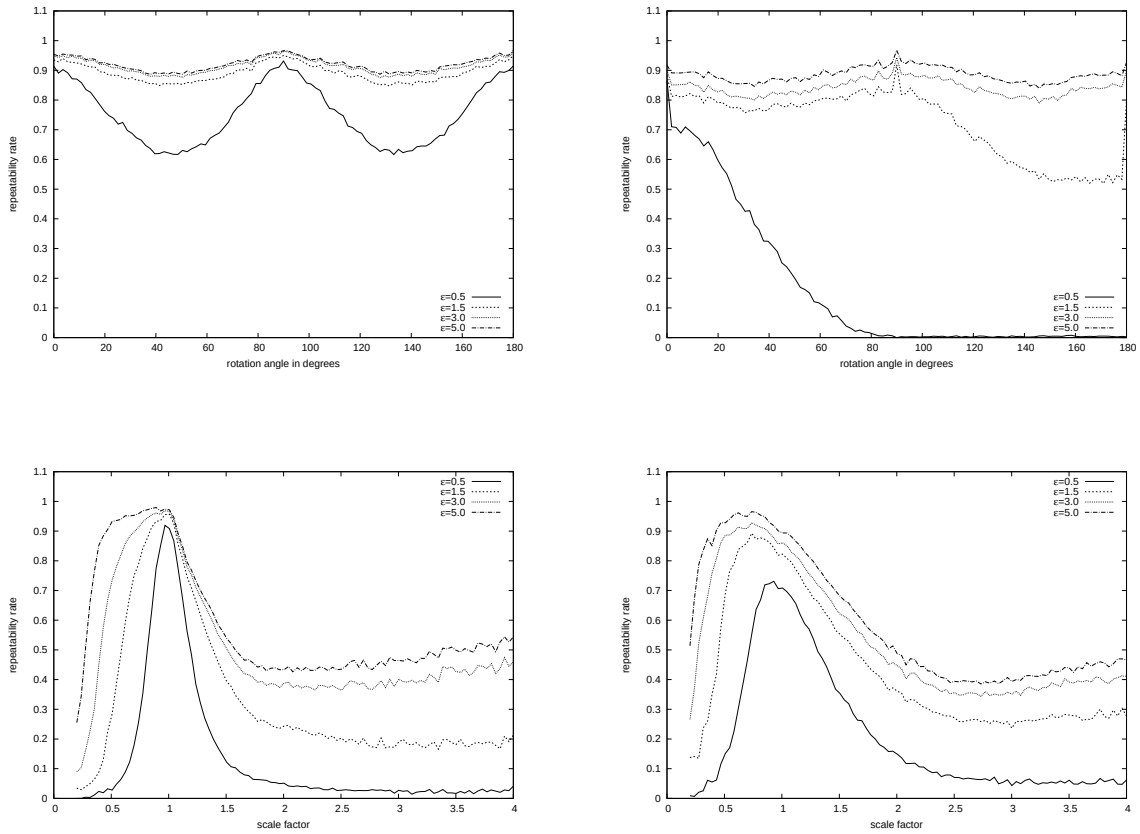


Figure 18: Comparison between the OpenCV implementation and ours. On the left column, we show the results using our implementation and the *building* sequence. On the right, we show the results of the OpenCV method. The graphics in the first row correspond to the evolution of the repeatability rate with respect to increasing rotations (from 0^0 to 180^0) and the second row contains the graphics for scale changes.

Looking at the first two rows of the first image, corresponding to the rotation and scale graphics, we observe that the behavior of the OpenCV implementation is poor. The repeatability rate for rotations is very low for $\epsilon < 1.5$. This is probably caused by the aliasing artifacts that are not reduced by the Sobel operator or the use of box filtering. Indeed, using Gaussian convolutions is key for improving the repeatability.

Our method also presents a better behavior under scale changes. We observe that the measure for the OpenCV implementation varies very fast around scale 1 (the initial image resolution). The

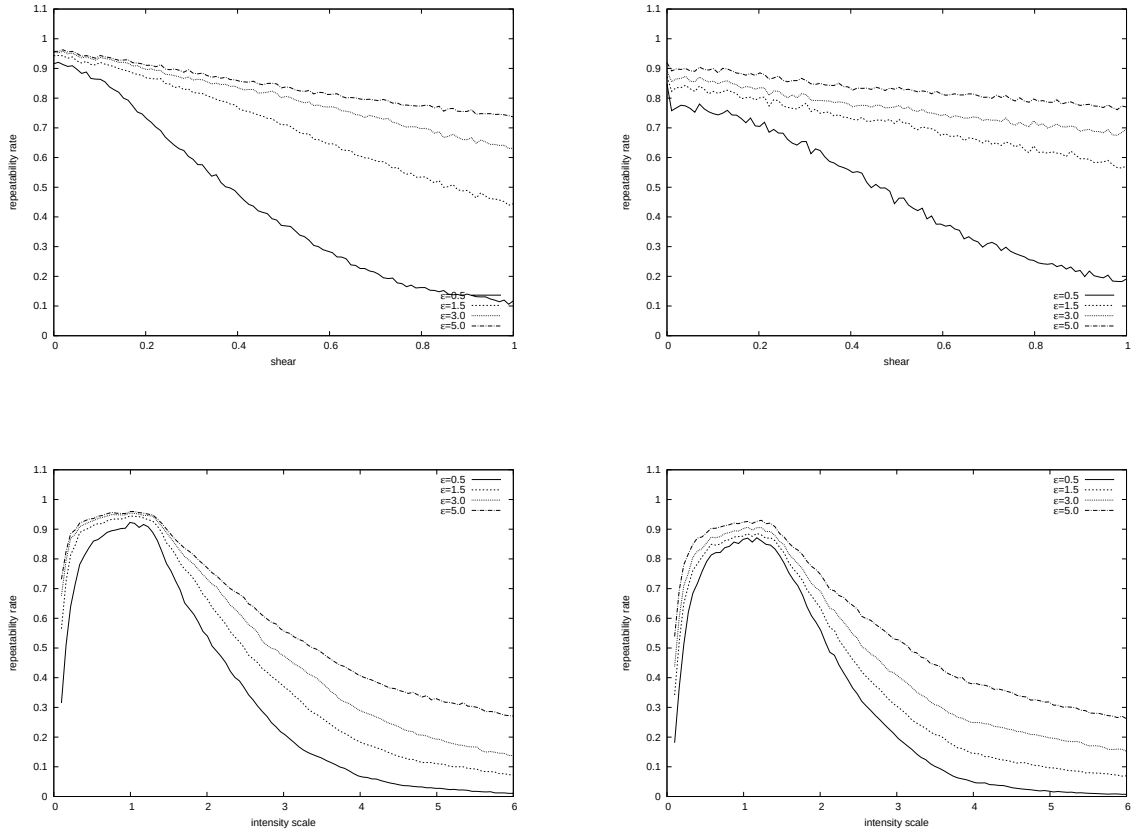


Figure 19: Comparison between the OpenCV implementation and ours. On the left column, we show the results using our implementation and the *building* sequence. On the right, we show the results of the OpenCV method. The graphics in the first row correspond to the evolution of the repeatability rate with respect to affine transformations (varying the skew parameter) and the second row shows the repeatability rate with respect to changes in illumination.

analysis with respect to affine transformations (first row in Figure 19) is also favorable to our approach for small skew changes, although the differences are not so noticeable for larger ones. The results corresponding to illumination changes (second row) are similar in both cases, with a slightly better accuracy in our implementation.

Table 5 compares the execution times for both approaches. Note that we have used the same values for Steps 5–7, so that the difference is related to the first steps. In this case, the OpenCV method is faster than our approach. The reason is that it does not convolve the image with a Gaussian function (Step 1) and the autocorrelation matrix is calculated through a box filtering, which is faster than Gaussian convolutions. On the other hand, the OpenCV implementation includes more code optimizations.

As a conclusion, the OpenCV implementation is interesting if speed is the most important issue for the application. However, the repeatability rate is in general not so good.

5 Conclusion

In this work, we presented an implementation of the Harris corner detector. We explained every step of the method and analyzed different alternatives for each one.

We found that, if we are interested in improving the repeatability rate measure, it is important to use Gaussian convolutions. Additionally, the use of an accurate Gaussian technique makes the

Step	Ours (milliseconds)	OpenCV
1. Smoothing the image	10.46 ms	–
2. Computing the gradient	4.84 ms	–
3. Computing the autocorrelation matrix	46.67 ms	–
4. Computing corner strength function	2.94 ms	23.06 ms
5. Non-maximum suppression	4.92 ms	4.92 ms
6. Selecting output corners	0.11 ms	0.11 ms
7. Calculating subpixel accuracy	0.15 ms	0.15 ms
Total:	**70,09ms**	**28.24ms**

Table 5: Comparison of the execution times between our implementation and OpenCV.

method more stable to the use of different gradient masks.

We implemented a generic non-maximum suppression algorithm that allows to select the prominent features on the image. We compared the use of quadratic and quartic interpolation in order to obtain interest points with subpixel accuracy. The quadratic approach is as accurate as the quartic strategy but much faster and simpler to implement.

To improve the speed of the method, it is necessary to implement a faster Gaussian convolution technique, or to replace it with a box filter, at the expense of an accuracy loss.

Acknowledgments

Work partly financed by Office of Naval research grant N00014-17-1-2552, DGA Astrid project "filmer la Terre" n^o ANR-17-ASTR-0013-01, DGA Defals challenge n^o ANR-16-DEFA-0004-01.

Image Credits

All the images in this work are provided by the authors.

References

[1] H. Bay, T. Tuytelaars, and L. Van Gool, *SURF: Speeded Up Robust Features*, in European Conference on Computer Vision (ECCV), Springer Berlin Heidelberg, 2006, pp. 404–417. https://doi.org/10.1007/11744023_32.

[2] A. Bhatia, W. E Snyder, and G. Bilbro, *Stacked integral image*, in IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2010, pp. 1530–1535. https://doi.org/10.1109/ROBOT.2010.5509400.

[3] M. Brown, R. Szeliski, and S. Winder, *Multi-image matching using multi-scale oriented patches*, in IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), vol. 1, IEEE, 2005, pp. 510–517. https://doi.org/10.1109/CVPR.2005.235.

[4] J. Canny, *A computational approach to edge detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence, (1986), pp. 679–698. https://doi.org/10.1109/TPAMI.1986.4767851.

[5] R. Deriche, *Recursively implementating the Gaussian and its derivatives*, Research Report RR-1893, INRIA, 1993. https://hal.inria.fr/inria-00074778/file/RR-1893.pdf.

[6] P.F. Felzenszwalb, R.B. Girshick, D. McAllester, and D. Ramanan, *Object detection with discriminatively trained part-based models*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 32 (2010), pp. 1627–1645. https://doi.org/10.1109/TPAMI.2009.167.

[7] W. Förstner and E. Gülch, *A fast operator for detection and precise location of distinct points, corners and centres of circular features*, in Proceedings of ISPRS Intercommission Conference on Fast Processing of Photogrammetric Data, 1987, pp. 281–305.

[8] P. Getreuer, *A Survey of Gaussian Convolution Algorithms*, Image Processing On Line, 3 (2013), pp. 286–310. https://doi.org/10.5201/ipol.2013.87.

[9] C. Harris and M. Stephens, *A combined corner and edge detector*, in Alvey Vision Conference, vol. 15, Manchester, UK, 1988, pp. 10–5244.

[10] L. Kitchen and A. Rosenfeld, *Gray-level corner detection*, Pattern Recognition Letters, 1 (1982), pp. 95–102. https://doi.org/10.1016/0167-8655(82)90020-4.

[11] D.G. Lowe, *Distinctive image features from scale-invariant keypoints*, International Journal of Computer Vision, 60 (2004), pp. 91–110. https://doi.org/10.1023/B:VISI.0000029664.99615.94.

[12] N. Monzón, A. Salgado, and J. Sánchez, *Regularization strategies for discontinuity-preserving optical flow methods*, IEEE Transactions on Image Processing, 25 (2016), pp. 1580–1591. https://doi.org/10.1109/TIP.2016.2526903.

[13] N. Monzón, A. Salgado, and J. Sánchez, *Robust Discontinuity Preserving Optical Flow Methods*, Image Processing On Line, 6 (2016), pp. 165–182. https://doi.org/10.5201/ipol.2016.172.

[14] H.P. Moravec, *Obstacle avoidance and navigation in the real world by a seeing robot rover*, tech. report, Stanford University (CA), Department of Computer Science, 1980.

[15] A. Neubeck and L. Van Gool, *Efficient non-maximum suppression*, in International Conference on Pattern Recognition (ICPR), vol. 3, 2006, pp. 850–855. https://doi.org/10.1109/ICPR.2006.479.

[16] T.Q Pham, *Non-maximum suppression using fewer than two comparisons per pixel*, in International Conference on Advanced Concepts for Intelligent Vision Systems, Springer, 2010, pp. 438–451. https://doi.org/10.1007/978-3-642-17688-3_41.

[17] E. Rosten and T. Drummond, *Machine learning for high-speed corner detection*, in European Conference on Computer Vision (ECCV), Springer Berlin Heidelberg, 2006, pp. 430–443. https://doi.org/10.1007/11744023_34.

[18] J. Sánchez, *Comparison of Motion Smoothing Strategies for Video Stabilization using Parametric Models*, Image Processing On Line, 7 (2017), pp. 309–346. https://doi.org/10.5201/ipol.2017.209.

[19] J. Sánchez, N. Monzón, and A. Salgado, *Robust Optical Flow Estimation*, Image Processing On Line, 3 (2013), pp. 252–270. https://doi.org/10.5201/ipol.2013.21.

[20] C. Schmid, R. Mohr, and C. Bauckhage, *Evaluation of interest point detectors*, International Journal of Computer Vision, 37 (2000), pp. 151–172. https://doi.org/10.1023/A:1008199403446.

[21] J. Shi and C. Tomasi, *Good features to track*, in IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 1994, pp. 593–600. https://doi.org/10.1109/CVPR.1994.323794.

[22] C. Sun and P. Vallotton, *Fast linear feature detection using multiple directional non-maximum suppression*, Journal of Microscopy, 234 (2009), pp. 147–157. https://doi.org/10.1109/ICPR.2006.548.

[23] Richard Szeliski, *Computer vision: algorithms and applications*, Springer Science & Business Media, 2010. ISBN 1848829345.

[24] T. Tuytelaars and K. Mikolajczyk, *Local invariant feature detectors: a survey*, Foundations and Trends® in Computer Graphics and Vision, 3 (2008), pp. 177–280.