Computer Vision

Locating Points of Interest and Exporting Features in Images

Kontopoulos Stefanos

1 Edge Detection in Gray Images

1.1 Create Input Images

In this first stage, we read our image as a gray image (Figure 1) and then add white Gaussian noise for 2 different values of PSNR (Peak-to-peak Signal to Noise Ratio) (Figure 2).

The noise added is Gaussian white with a mean value of zero and a standard deviation of 0.1 and 0.3162 respectively. The σ are chosen as the $I_{max} - I_{min}$ is equal to 1.

The PSNR is defined as:

$$PSNR = 20 \cdot \log_{10}(\frac{I_{max} - I_{min}}{\sigma_n})(dB),$$
$$I_{max} = \max_{x,y} I(x,y), \quad I_{min} = \min_{x,y} I(x,y) \tag{1}$$

Decreasing the PSNR corresponds to increasing the image noise levels.

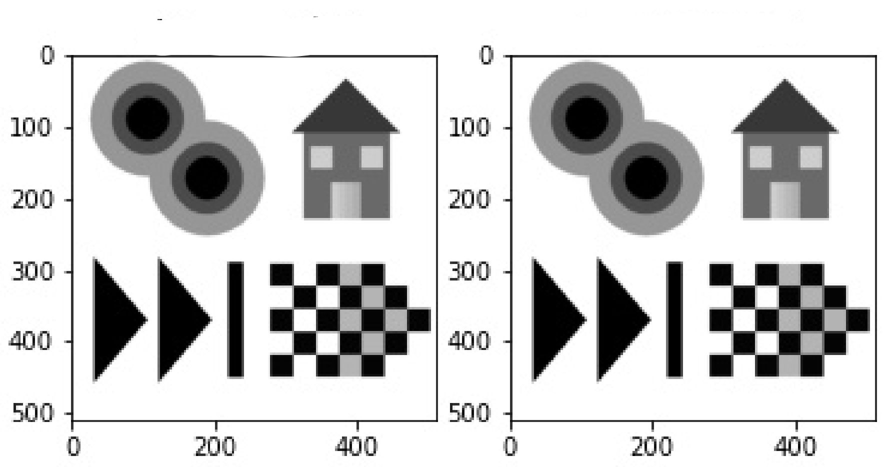


Figure 1: Original and gray image.

1.2 Implementation of Edge Detection Algorithms

The *Edge Detect* function is constructed, which accepts arguments for a gray image, the standard Gaussian deviation σ, the *theta_edge* parameter, and the Laplacian approach. Returns a binary image of edges with points selected as edges.

The function first implements the Gaussian and Laplacian of Gaussian filters. The Gaussian filter produces an image that at each point adds its neighbours to a radius proportional to Gaussian's σ. This helps to mitigate the effect of noise on the image, but also creates blur. High values of σ lead to a high degree of blurring and thus information is lost. Still, we can interpret σ as a scale parameter. Large values blur small objects, and so only large ones are preserved in the image. Both small and large items retain small values.

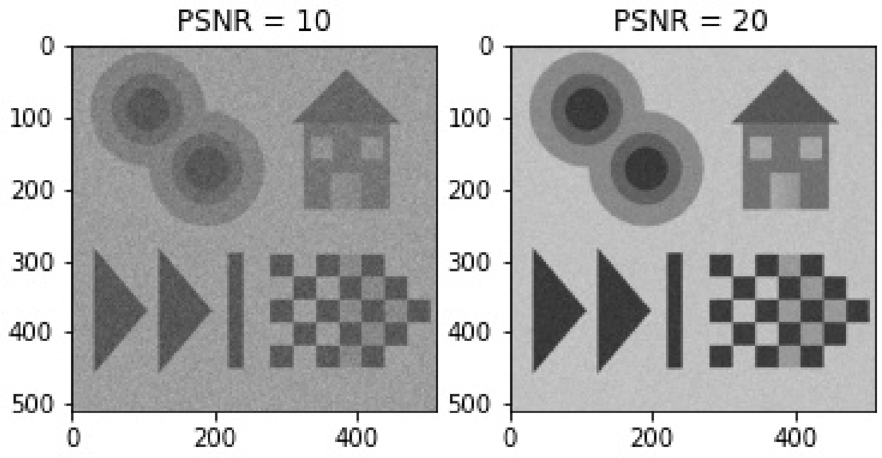


Figure 2: Images with Gaussian noise.

Next, we approach Laplacian in two ways:

- Linear: In the linear mode, we take advantage of the convergence of the convergence and apply the LoG filter to the original image. It is:

$$L_1 = \nabla^2 (G_\sigma * I) = (\nabla^2 G_\sigma) * I \tag{2}$$

- Non-Linear: In the non-linear way, we approach the Laplacian of the smoothed image based on the relation:

$$L_2 = I_\sigma \oplus B + I_\sigma \ominus B - 2 \cdot I_\sigma, \quad B = \begin{array}{|c|c|c|} \hline & \bullet & \\ \hline \bullet & \bullet & \bullet \\ \hline & \bullet & \\ \hline \end{array} \tag{3}$$

and $I_\sigma = G_\sigma * I(x, y)$.

Next, we calculate the Laplacian zero crossings. Zero crossings are points of local maximum of the first derivative of the smoothed image, i.e. points where the brightness changes rapidly in the original image. These points are possible edge points. Zero crossings are calculated by finding the outline in the Laplacian binary sign image, i.e. the image:

$$X = L \geq 0, \quad Y = (X \oplus B) - (X \ominus B) \approx \partial X \tag{4}$$

Finally, we do not take all the zero crossings we calculated, but only those where the brightness change is large enough in the original image. In other words, they satisfy the relationship:

$$Y[i,j] = 1 \quad and \quad \|\nabla I_\sigma[i,j]\| > \theta_{edge} \cdot \max_{x,y} \|\nabla I_\sigma\| \tag{5}$$

1.3 Evaluation of the Results of Edge Detection

Here we calculate the real edges using *cv2.morphologyEx*, which essentially finds the contours of the image that we practically get by subtracting the erosion of the original image from its dilation, for a given kernel (Figure 3).

In other words, it performs the following calculation:

$$M = (I_o \oplus B) - (I_o \ominus B) \tag{6}$$

We then apply an evaluation criterion to judge how effective the detection is.

$$C = \frac{[P_r(D|T) + P_r(T|D)]}{2} \tag{7}$$

, with $T = M > \theta_{realedge}$ and where $P_r(D|T)$ is the percentage of detected edges that are true (Precision) and $P_r(T|D)$ the percentage of true edges detected (Recall). If D and T are seen as distinct sets (with elements in

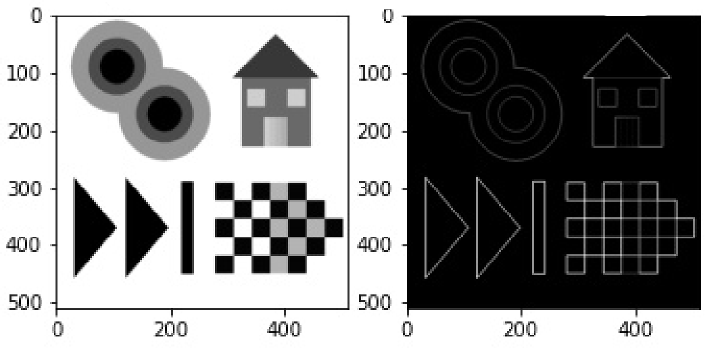


Figure 3: Real edges.

pixels in which the binary image has a value of 1), then:

$$P_r(T|D) = \frac{card(D \cap T)}{card(T)} \tag{8}$$

, where $card(.)$ gives the number of elements of a set.

Below are some experiments with different parameter values (Figures 5, 7).

We observe that as the standard Gaussian deviation increases, we have better image smoothing and noise removal. However, for large values of σ we have a large blur, and we lose information of the original image. Therefore, its high values group close edges. The parameter ϑ helps us to remove "soft" edges for which we do not have a rapid change in brightness.

We generally observe that we have better results for the non-linear Laplacian approach. Also, the noisier images give worse results and usually need higher values σ and ϑ as they have a higher noise rate.

In general, in computer vision, non-linear filters are used more often and

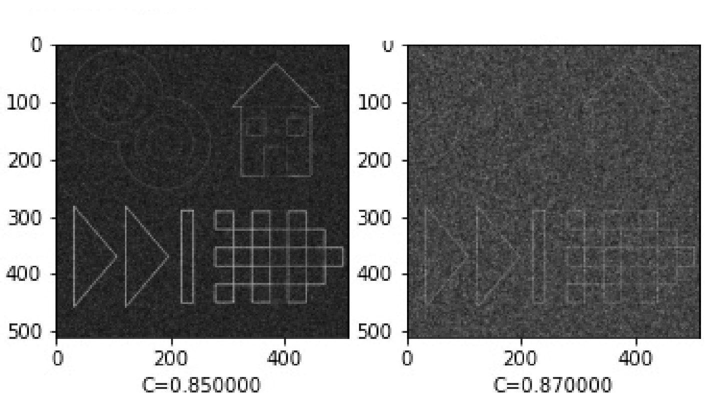


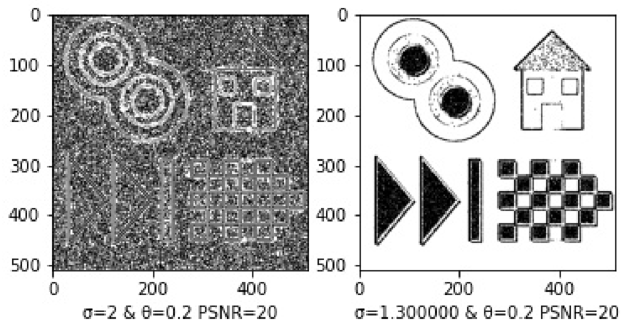
Figure 4: Real edges of noisy images.

give better results.

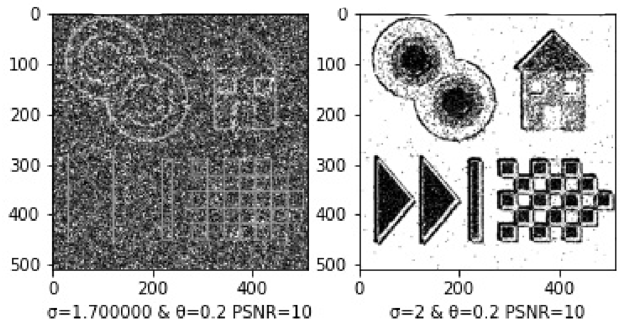
1.4 Application of Edge Detection Algorithms in Real Images

Finally, after applying the previous algorithms to our true image "*duomoedges.jpg*" (Figure 7), we observe that small values of σ give edges closer to each other while large values of it group close edges. The parameter ϑ determines how "thin" the edges we choose are. Large values of ϑ will reject edges that have mild changes in brightness, such as a transition from light to medium gray.

In conclusion, the parameter σ controls the scale at which we detect the edges, and the image that retains the most information is chosen to be the one with $\sigma = 1.9$ & $\theta = 0.3$.



(a) Edge detection for PSNR=20.



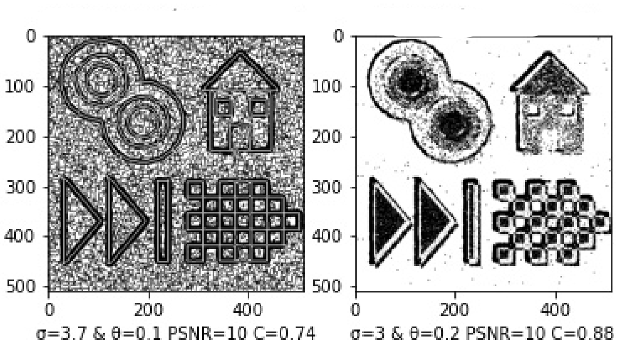
(b) Edge detection for PSNR=10.

Figure 5: Experiments on artificial image with Gaussian noise.

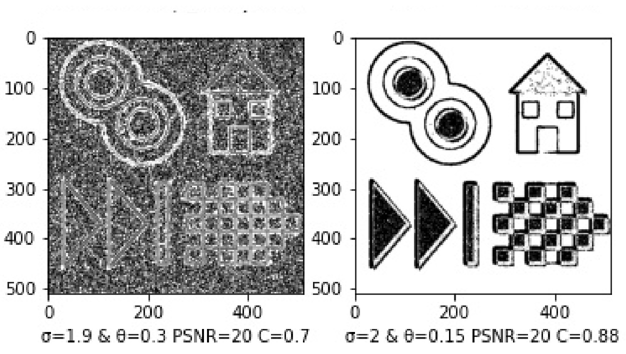
2 Detection of Points of Interest

2.1 Angle Detection

In Part 2 of this laboratory exercise, we will deal with the detection of points of interest in images. An intuitively simple category of points of interest is



(a) Edge detection with 74% & 88% success.

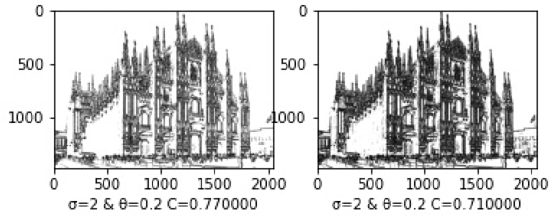


(b) Edge detection with 70% & 88% success.

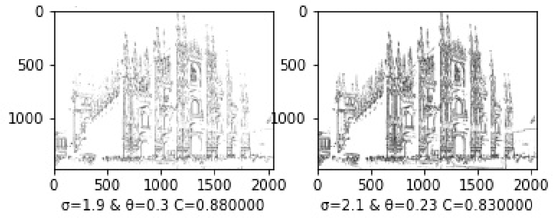
Figure 6: Optimal edge detection results.

the selection of points that correspond to corners of the image.

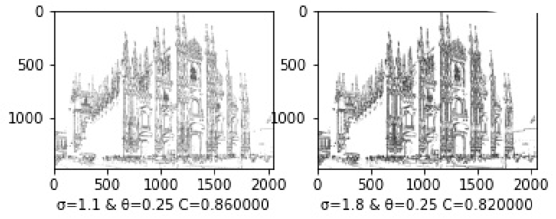
For the detection of angles, we will implement the classic Harris-Stephens method.



(a) Edge detection with 77% & 71% success.



(b) Edge detection with 88% & 83% success.



(c) Edge detection with 86% & 82% success.

Figure 7: Optimal edge detection results.

First we calculate the structural tensor J in each pixel of the image:

$$\begin{aligned} J_1(x,y) &= G_p * \left(\frac{\partial I_\sigma}{\partial x} \cdot \frac{\partial I_\sigma}{\partial x}\right)(x,y) \\ J_2(x,y) &= G_p * \left(\frac{\partial I_\sigma}{\partial x} \cdot \frac{\partial I_\sigma}{\partial y}\right)(x,y) \\ J_3(x,y) &= G_p * \left(\frac{\partial I_\sigma}{\partial y} \cdot \frac{\partial I_\sigma}{\partial y}\right)(x,y) \end{aligned} \tag{9}$$

, where $I_\sigma = G_\sigma * I$, and G_ρ, G_σ two-dimensional Gaussian normalization nuclei with standard deviations σ (difference scale) and ρ (integration scale) respectively.

Convergence with the σ kernel ensures that no objects appear in the tensor smaller than the scale σ. Also, convergence with the G_ρ (integration scale) kernel causes the tensor J to take into account the change in image over a wider size range of ρ around each point.

Then we calculate the eigenvalues of the tensor J. The corresponding eigenvectors show in the directions of maximum and minimum change of the image, and the eigenvalues give a measurement of these changes. Therefore, if both eigenvalues are large enough we have a large change in brightness in two directions, and we find an angle. Eigenvalues are calculated from the relation:

$$\lambda_\pm(x,y) = \frac{1}{2} \cdot \left(J_1 + J_3 \pm \sqrt{(J_1 - J_3)^2 + 4 \cdot J_2^2}\right) \tag{10}$$

In Figure 8. We can see the visualization of the eigenvalues $\lambda_\pm$. These images show the values taken by $\lambda_\pm$ for the image *duomo_edges*. We observe that we have angles in the points where we have a large λ_+ but also λ_-. Also, we notice that in the image, λ_+ the light areas correspond to edges. This makes sense as λ_+ indicates the maximum image change direction.

Based on the eigenvalues, we calculate the angle criterion:

$$R(x,y) = \lambda_- \cdot \lambda_+ - k \cdot (\lambda_- + \lambda_+) \tag{11}$$

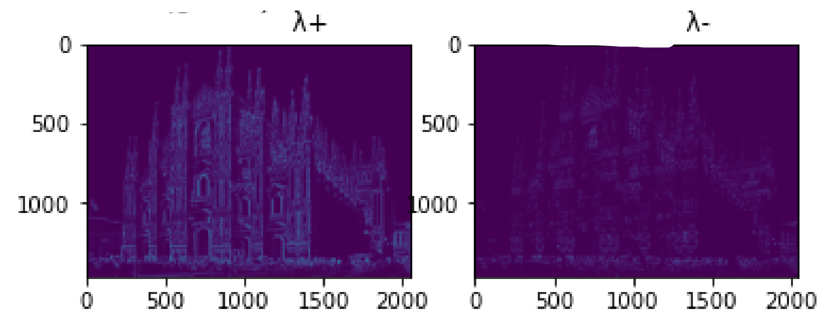


Figure 8: Eigenvalues $\lambda_{\pm}$.

, where k is a small positive constant. Finally, we choose as angles the pixels (x, y) which:

1. They are maxima of R within square windows that surround them, the size of which depends on the scale σ,

2. They correspond to a value of R greater than a percentage of the total maximum of R, i.e. $R(x, y) > \theta_{corn} \cdot R_{max}$, όπου θ_{corn} a properly selected threshold.

Results of the angle detection with the indicative parameter values, we see in the Figures 9,10,11.

We observe that although some angles are visible to the human eye, the algorithm is unable to detect them. This is due to the choice of parameters, which affects the performance of the algorithm depending on the input image, but also to the fact that absolute accuracy is a rather difficult achievement.

2.2 Multiscale Angle Detection

Here, we use the normalized LoG measure to be maximum on our scales.

$$|LoG(\mathbf{x}, \sigma_i)| = \sigma_i^2 \cdot |L_{xx}(\mathbf{x}, \sigma_i) + L_{xx}(\mathbf{x}, \sigma_i)|, \qquad i = 0, \ldots, N-1$$

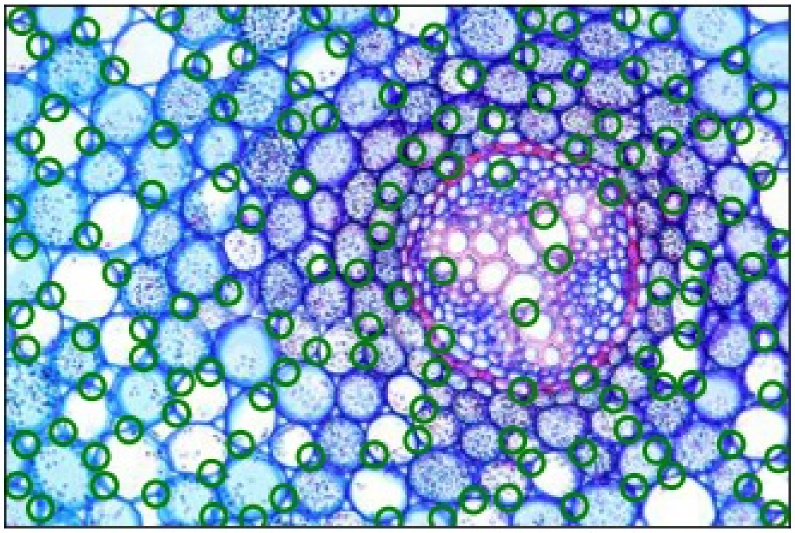


Figure 9: Angles in the cell image.

The σ differentiation scales are obtained from:

$$\sigma_0 + \sigma_1, \ldots, \sigma_{N-1} = s^0 \cdot \sigma_0 + s^1 \cdot \sigma_0, \ldots, s^{N-1} \cdot \sigma_0$$
$$\rho_0 + \rho_1, \ldots, \rho_{N-1} = s^0 \cdot \rho_0 + s^1 \cdot \rho_0, \ldots, s^{N-1} \cdot \rho_0$$

Results of the multiscale angle detection are shown in the Figures 12, 13.

We observe that for a constant σ and a large, rho the angles become thinner. This is because the tensor J (9) takes into account n points in a larger neighbourhood and chooses one of them at close angles. Furthermore, the value σ determines the scale at which we detect angles, as in the first part. Finally, in full accordance with the edge detection of the first part, large values of ϑ reject "soft" angles such as those in the image clouds.

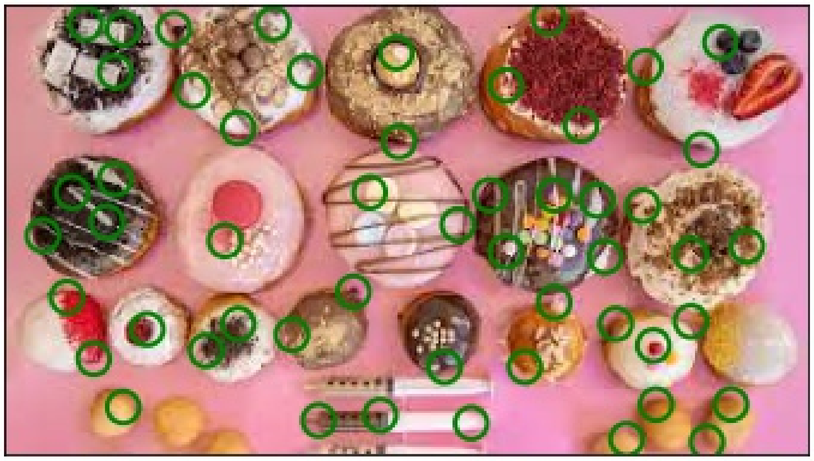


Figure 10: Corners in the picture with donuts.

2.3 Blobs Detection

One of the most important points of interest is based on the detection of $'Blobs'$, which are defined as areas of some homogeneity that differ significantly from their neighbourhood. some second-order derivatives of the image and in particular the Hessian table delimiter:

$$H = \begin{bmatrix} \frac{\partial^2 I_\sigma(x,y,\sigma)}{\partial x^2} & \frac{\partial^2 I_\sigma(x,y,\sigma)}{\partial x \partial y} \\ \frac{\partial^2 I_\sigma(x,y,\sigma)}{\partial y \partial x} & \frac{\partial^2 I_\sigma(x,y,\sigma)}{\partial y^2} \end{bmatrix} \tag{12}$$

, where $L_{xx} = \frac{\partial^2 I_\sigma(x,y,\sigma)}{\partial x^2}$, $L_{yy} = \frac{\partial^2 I_\sigma(x,y,\sigma)}{\partial y^2}$ and $L_{xy} = L_{yx} = \frac{\partial^2 I_\sigma(x,y,\sigma)}{\partial x \partial y}$.

The point selection process is similar to the previous question, i.e. we keep the points that are locally maximal and have a value greater than a

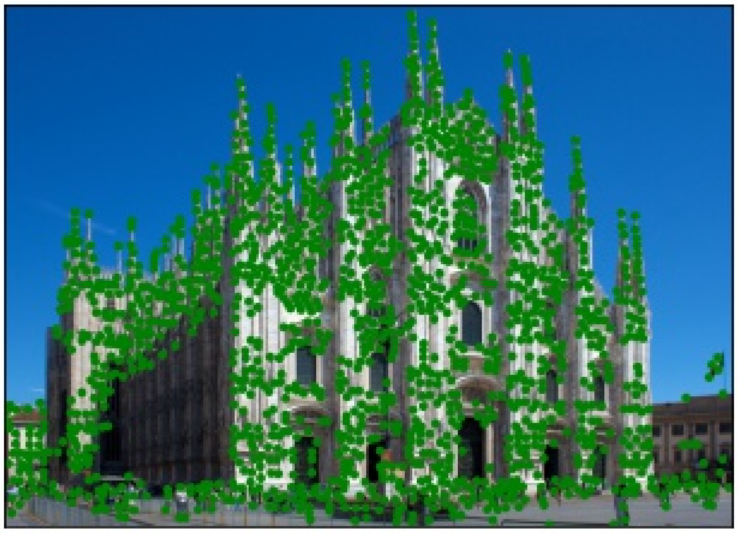


Figure 11: Corners in the image of the Duomo temple.

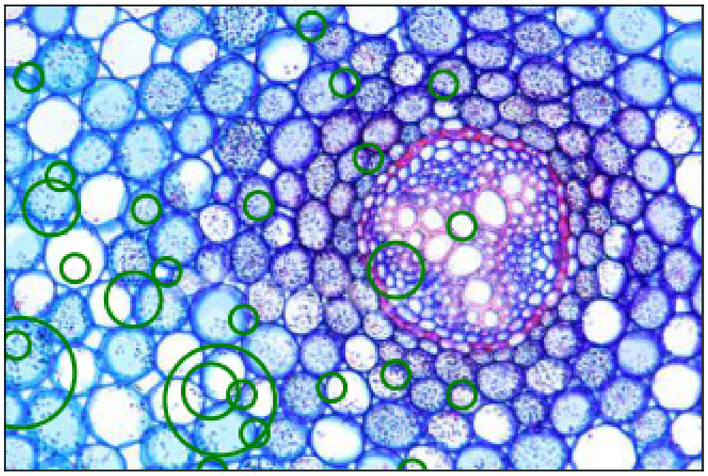


Figure 12: Angles in the cell image on many scales.

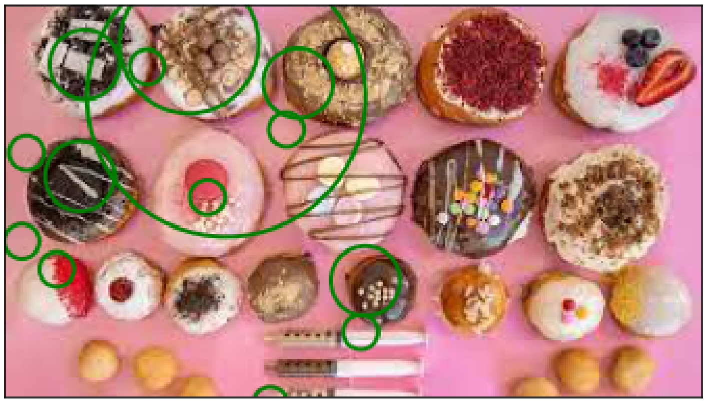


Figure 13: Corners in the picture with doughnuts on many scales.

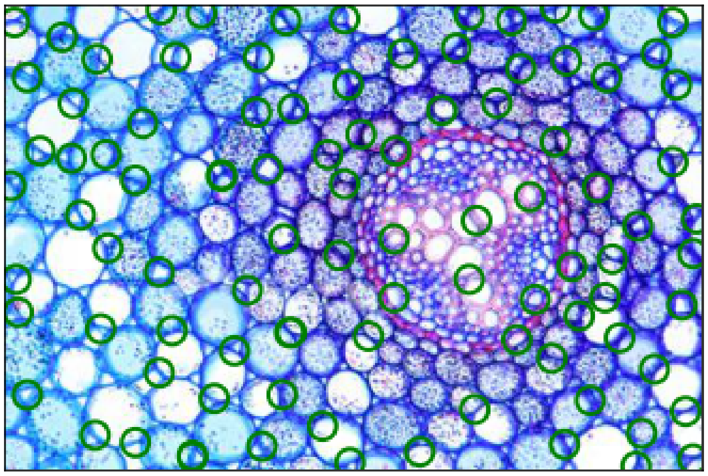


Figure 14: Blobs in the cell image.

properly defined threshold.

Results of the Blobs detection are shown in the Figures 14, 15.

2.4 Multiscale Blobs Detection

We repeat the process of multistage angle detection, adding a second step for the selection of points of interest that are at maximum scale (Hessian-

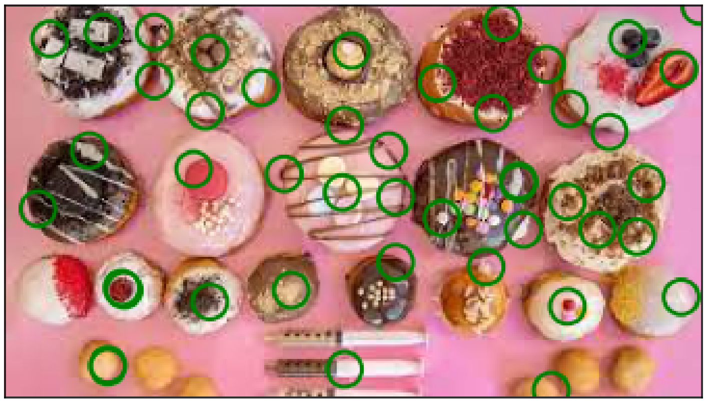


Figure 15: Blobs in the picture with doughnuts.

Laplace). The initial points of interest for each scale are selected according to the method of the previous Blobs detection query.

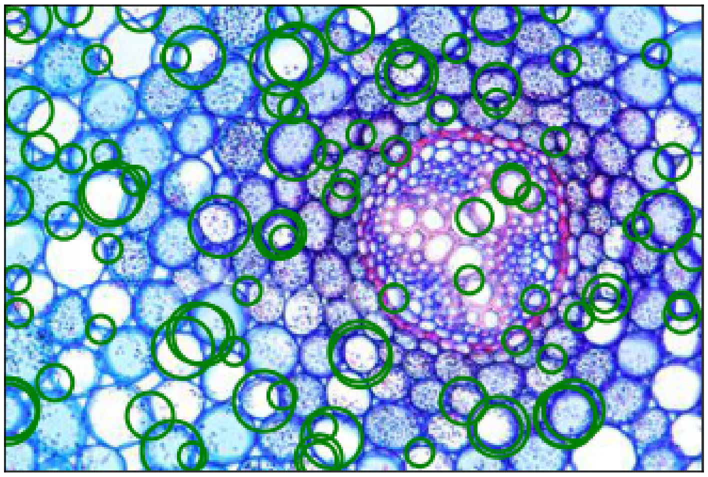


Figure 16: Blobs in the cell image on many scales.

As we can see, higher values of ϑ reject the milder Blobs such as those in the cell, while higher values of σ (scale) detect larger Blobs.

Multistep Blobs detection results are shown in the Figures 16, 17.

For images with strong information such as the one with the cell, small

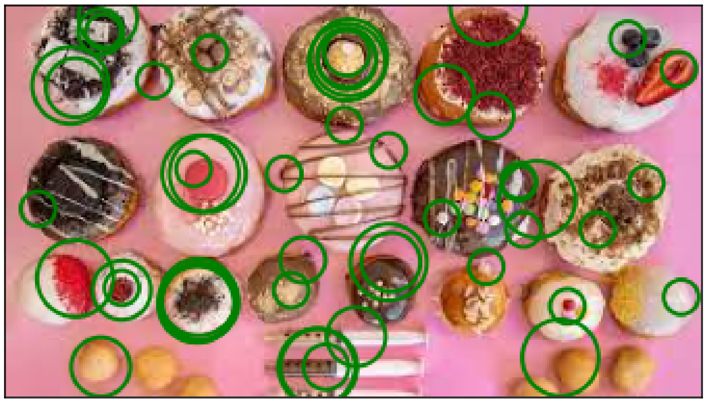


Figure 17: Blobs in the picture with doughnuts on many scales.

values of the parameter ϑ give numerous Blobs, something we may want to avoid. The differences between Blobs and corners are not very noticeable at first glance in the images we examine, but they are easily distinguished for large scale values.

For a large Blob will be the entire donut while the corner will be its edge. Even for the same values of ϑ and σ we observe fewer angles than Blobs in the same images. This highlights the accuracy of the Harris-Stephens angle finding algorithm compared to the simple algorithm we used to find Blobs.

2.5 Acceleration using Box Filters and Integral Images

Hessian's calculation for each scale corresponds to the convergence of the image with increasingly sized filters, which is a computationally expensive process. To accelerate it In this method, it was proposed to approach the 2^{nd} derivative filters with Box Filters, i.e. filters based on orthogonal area sums, as shown in Figure 18. The implementation of such sums is very effective using Integral Images.

Laplacian's computational algorithm makes convolutions with Gaussian scales filters σ and each convolution has a complexity of $O(NM\sigma^2)$, where N and M are the dimensions of the image. This is because Gaussian's area is proportional to σ^2. For large scale values, this calculation costs a lot of time. So we will approach Laplacian using Box Filters.

For this, we will calculate the complete image of our image, something that can be done in $O(NM)$. The value at each point in the complete image is equal to the sum of all the points to the left and above that point. Therefore, the sum of all values in each rectangular passage ABCD of the original image, where A is at the top left, is:

$$\sum \sum_{ABCD} I(i,j) = S_A + S_C - S_B - S_D \tag{13}$$

, where S_j the value of the complete image at point j. It is obvious that the complete picture can be calculated in time $O(NM)$. Yet, each sum of points within a rectangle can be calculated at a fixed time.

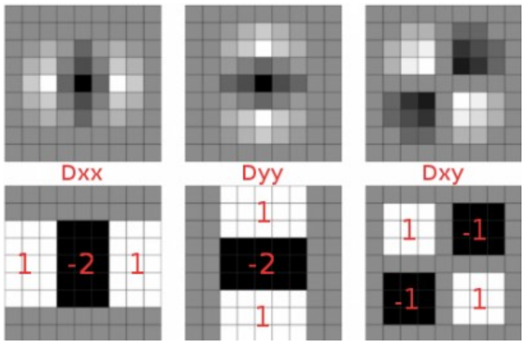


Figure 18: Visualization of the 2^{nd} derivative filter approach with Box Filters.

We calculate Laplacian as value differences of the original image inside a rectangle, the size of which depends on the scale p. These windows are called Box Filters and are as in Figure 18.

The size of each component of each window depends on the diameter n of Gaussian ($n = 2 \cdot ceil(3\sigma) + 1$) and is calculated from the following relations:

- D_{xx}: window height 4×floor(n/6)+1, window width 2×floor(n/6)+1
- D_{yy}: window height 2×floor(n/6)+1, window width 4×floor(n/6)+1

- D_{xy}: window height 2×floor(n/6)+1, window width 2×floor(n/6)+1

Finally, the image is multiplied with one of these Box Filters at one point at a fixed time, since it consists of rectangular windows. We can do this acceleration because the shape of the window is square (complete image) and because the coefficients for one part of the Box Filter are fixed (each Box Filter consists of three parts). These windows calculate approaches for L_{xx}, L_{yy}, L_{xy} respectively.

Next, we calculate the criterion for locating Blobs:

$$R(x,y) = L_{xx}(x,y) \cdot L_{yy}(x,y) - (0.9 \cdot L_{xy}(x,y))^2 \tag{14}$$

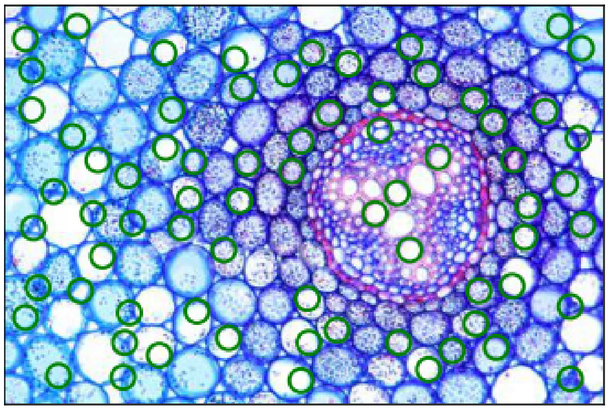


Figure 19: Box Filters use in the cell image.

Next, we visualize the Hessian criterion of blobs detection (Figure 23) and the above (Figure 24). We notice that for small ones the approach is good. But as it grows, the image becomes distorted and blurred, something that does not happen in Hessian. Also, because the calculation of Box Filters was done with the assumption that there are no blobs at the "edges" of the image, the larger it gets, the more points are ignored.

In the same way we calculate local maximums, values that exceed the maximum value of the threshold parameter, and local maximums in the scale space in the case of multistage detection. This approach has some implications for the quality of the criterion.

Results of the Blobs detection using Box Filters are shown in the Figures 19, 20.

We notice that Box Filters give some good results but can not reach the accuracy of the normal method.

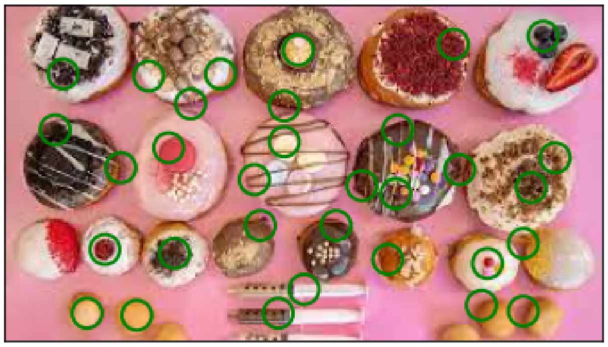


Figure 20: Box Filters use in the image with donuts.

Multi-Scale Blobs Detection Results with Box Filters are shown in Figures 21, 22.

The results agree with our observations above. In these calculations we used $\vartheta = 0.05$ and $\vartheta = 0.005$ for the images with cells and doughnuts respectively. This is because the image with the cells contains richer information.

In all three of the above methods, the image was first converted to greyscale and then the algorithms were applied.

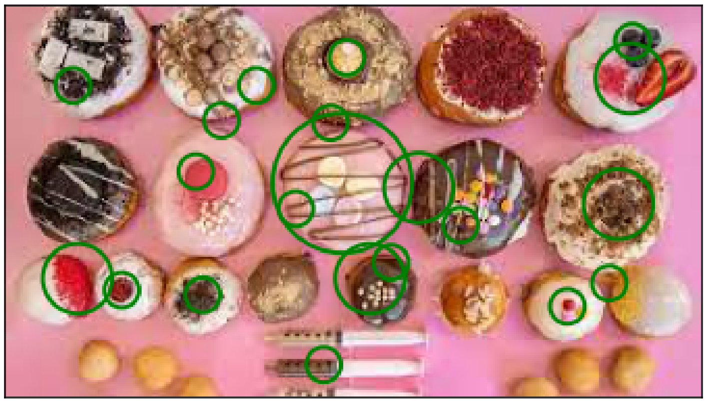


Figure 21: Use of multiscale Box Filters in the image with donuts.

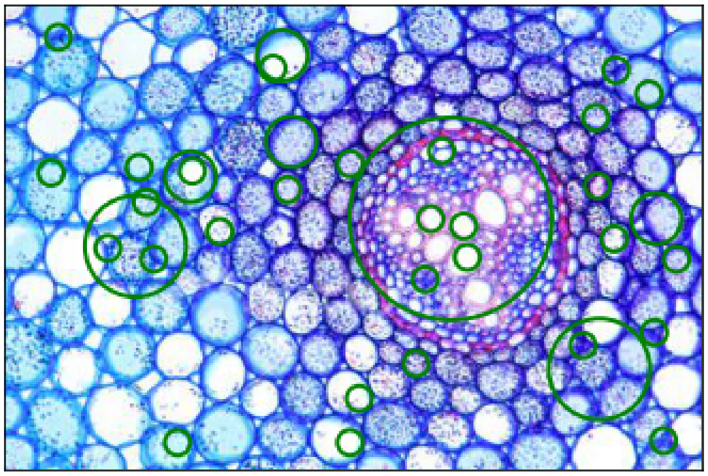


Figure 22: Using Multi-Scale Box Filters in the Cell Image.

3 Applications in Matching and Categorizing Images Using Local Descriptors to Points of Interest

In this section, we will look at the ability to find rotation and scale for the point detectors of the second part and the descriptors described above.

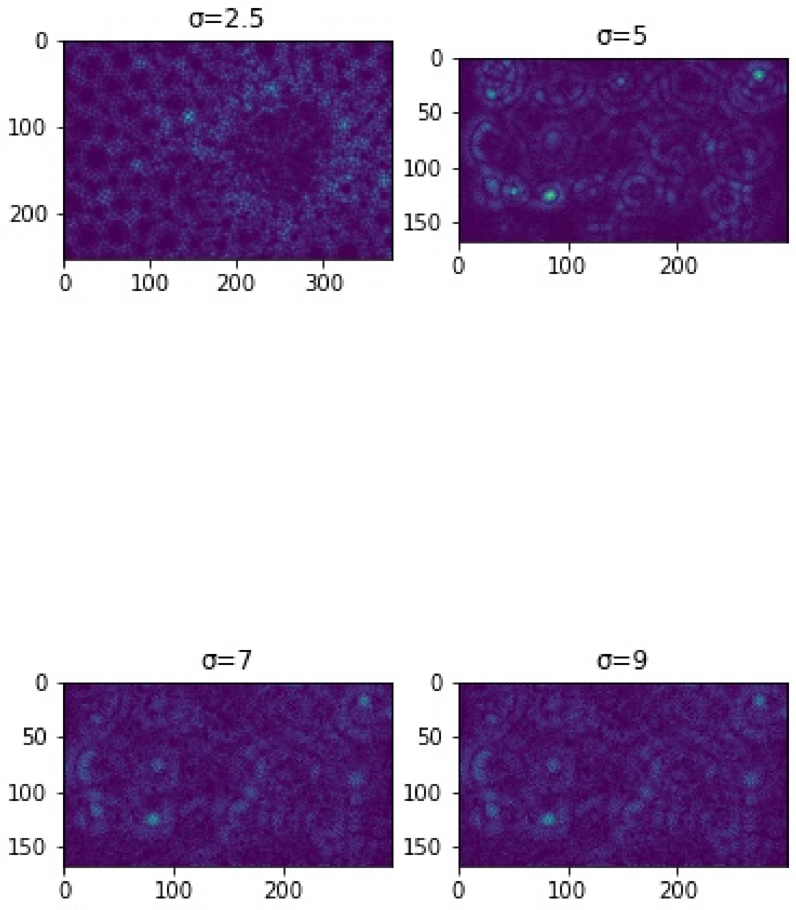


Figure 23: Blobs criterion visualization.

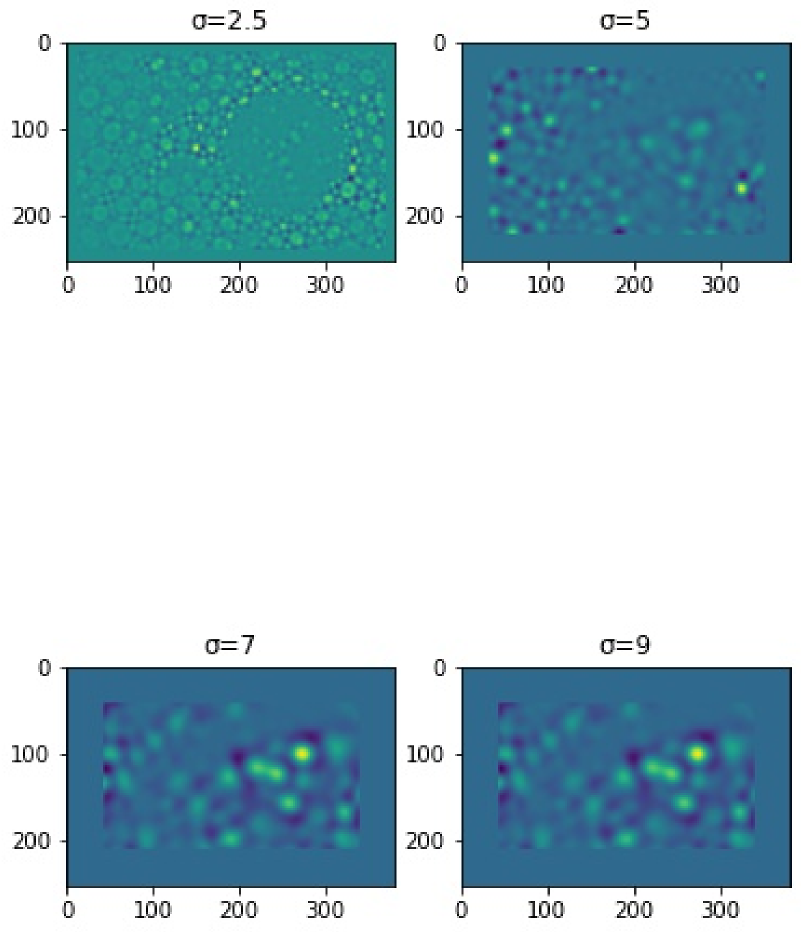


Figure 24: Box filter criterion visualization.

Using an image as a reference image, we will try to match its descriptors with those of other images.

From each point of interest in an image, we extract a local descriptor that encodes a neighbourhood around it. We will use two types of local descriptors, SURF and HOG.

The HOG descriptor is calculated by the following procedure. First, we calculate the first directional derivative for the original image. This is achieved by concatenating the image with the cores $h_x = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$ ανδ $h_y = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}^T$ for the horizontal and vertical derivative, respectively.

Then we calculate the measure of the derivative, but also the angle for each point of the image:

$$m(x,y) = \sqrt{I_x^2(x,y) + I_y^2(x,y)}, \quad \theta(x,y) = \arctan \frac{I_x(x,y)}{I_y(x,y)} \tag{15}$$

In the second step, we spatially divide the image into uniform cells to export local descriptors. For each cell, we extract a histogram that encodes the edge concentration at predetermined angles. The histogram is implemented as follows: First, we divide the range of angles $0° - 180°$ in 9 pieces. Each pixel votes for the angle range to which it belongs (i.e. $\theta(x,y)$) with a voting power commensurate with the derivative measure at this point $m(x,y)$. Also, we group cells into blocks and normalize each block with L_2 norm. Finally, the overall descriptor of the image is the aggregation of all local descriptors (normalized blocks).

SURF descriptors are calculated by the following procedure. In the first stage, we find the directional derivatives using Haar Wavelets. At each point, we find the direction in an area around it to export descriptors that do not depend on rotation. The calculation of the main direction is done in a range of angles that maximizes (in meters) the vector resulting from the first derivatives I_x and I_y.

Finally, we calculate a vector of characteristic 64 dimensions for a square window with a size that depends on the scale and direction we calculated. The (directed) window is divided into 4x4 blocks and in each block we calculate the vector u:

$$u = (\sum I_x, \sum I_y, \sum |I_x|, \sum |I_y|) \tag{16}$$

, where I_x and I_y the values of the derivatives in each pixel within the block.

The final feature vector is the joining of the 16 u vectors for the window. It is worth noting that the SURF descriptor is rotationally and stepwise independent of HOG.

3.1 Match Images under Rotation and Scale Change

In this section, we will examine the ability to find rotation and scale using the points of interest detectors we implemented and the above local descriptors. Specifically for this experiment, 3 images have been used, which we have deformed by rotating them (5 rotations: $-20°, -10°, 0°, 10°, 20°$) and changing their size (4 scales: 0.6, 0.8, 1.0, 1.2).

Therefore, we have a total of 20 distortions (similarity transformations) and using one of the images as a reference image, we will try to match its local descriptors with those of the other distorted images. This process is called matching. By choosing a set of point mapping - local descriptors, we can estimate the similarity transformation between the images and consequently the rotation and various scale.

It would not make sense to make this match with some metric distance between the descriptors, as there are points that are not interesting, but also because this criterion does not respect the arrangement of points of interest (e.g. in a picture of a hand the thumb is not corresponds to the index).

We know that a match is made in images that resulted in a similarity transformation. Also, the optimal match does not have to contain all the points of interest. These two observations lead us to the RANSAC (Random

Sample Consensus) algorithm. RANSAC finds the largest possible subset of pairs of points that satisfy the above geometric conditions. The algorithm works iteratively, approaching the parameters of the transformation. The point pairs are selected so that if we apply the transformation to the first set of points, we get the second. Therefore, we will use a transformation calculation algorithm given a point mapping, such as Direct Linear Transformation (DLT). We usually select the minimum number of matches to determine the transformation (5 in the case of similarity conversion).

In summary, the RANSAC algorithm randomly extracts the minimum set of mappings in each iteration, calculates the transformation with DLT, and checks how many other pairs of points satisfy this transformation. It has been shown that this approach most likely leads to a good solution for an appropriate number of repetitions.

Matching results are shown in the table below 1:

Descriptor	Detector	Avg. Scale Error			Avg. Theta Error		
SURF	Corner Detection Harris-Stephen	0.003	0.002	0.097	1.968	0.318	12.909
HOG		0.186	0.351	0.285	22.619	19.199	23.699
SURF	Multiscale Corner Detection Harris-Laplace	0.469	0.521	0.414	37.124	42.490	33.705
SURF	Blob Detection	0.049	0.001	0.041	2.099	0.066	1.610
HOG		0.121	0.272	0.128	15.471	14.185	24.956
SURF	Multiscale Blob Detection	0.001	0.002	0.002	0.171	0.094	0.088
SURF	Box Filters	0.003	0.032	0.008	0.334	7.356	2.296

Table 1: Matching results.

The results show clear superiority for the SURF descriptor for all feature detectors. This is to be expected since SURF is unchanged in scale and rotation while HOG is not. As we saw in the description of the implementation of descriptors, SURF calculates the descriptor in a block whose size depends on the scale while its orientation depends on the orientation of the derivatives in that block. This means that the descriptor we export depends neither on the scale nor on the rotation.

In contrast, HOG calculates the descriptor in blocks of constant size and fixed orientation (cells are always in the form of a chessboard). Thus, the HOG depends on the rotation and scaling of the image. The result of these two observations is that SURF gives much better matching performance with errors of an order of magnitude smaller for all point detectors. Also, for the SURF descriptor, we observe that the multistage version of the detectors gives much better results. This is due to the fact that for multistage point detection we extract better information about the image at more scales, and thus we have better information about the matching between the images. Also, there is no significant difference between the different types of points of interest.