Implémentation du jeu Hex via la Programmation Orientée Objet et stratégies de décision en C++

1st Jair Anderson Vasquez Torres
Ingénieur Degree Programme STIC
ENSTA Paris
Paris, France
jair-anderson.vasquez@ensta.fr

2nd Santiago Florido Gomez
Ingénieur Degree Programme STIC
ENSTA Paris
Paris, France
santiago.florido@ensta-paris.fr

***Abstract*—Ce projet implémente en C++ un système complet pour le jeu Hex, conçu principalement comme un exercice de Programmation Orientée Objet. La solution organise le domaine du jeu au moyen de classes qui encapsulent le plateau, l'état, les coordonnées et les règles, permettant la génération de coups légaux et la vérification de victoire au moyen d'un parcours BFS sur les connexions des pions. Sur cette base s'intègrent des stratégies de décision (p. ex., Negamax avec hachage et table de transposition) et une évaluation de positions découplée du moteur, qui peut être heuristique ou s'appuyer sur un modèle neuronal de valeur intégré à l'exécutable (exportable en TorchScript), sans lier la conception à une architecture spécifique. De plus, une interface graphique (GUI) en SFML a été développée pour faciliter l'interaction, la visualisation du plateau et les tests du comportement des agents.**

***Index Terms*—Hex game, C++, Object-Oriented Programming, Negamax, Transposition Table, TorchScript, SFML.**

I. Image Formats and Convolutions

A discrete two-dimensional convolution is a linear operation that produces an output image by sliding a small matrix, the kernel K of size $(2a+1)\times(2b+1)$, over every pixel of an input image I and computing a weighted sum of the neighborhood, as defined in Eq. (1):

$$(I * K)(y,x) = \sum_{j=-a}^{a} \sum_{i=-b}^{b} I(y+j,\, x+i)\; K(j,i). \qquad (1)$$

This operation is the computational basis for a range of image-processing tasks such as smoothing, sharpening, edge detection, and feature extraction, all depending on the choice of kernel [1].

A. OpenCV and Matplotlib Functions

The provided code in `Convolutions.py` relies on several OpenCV and Matplotlib functions whose behavior needs to be understood to interpret the results correctly.

The image is loaded with `cv2.imread(path, 0)`, where the flag `0` forces grayscale loading as a single-channel 8-bit array, and the result is cast to `float64` via `np.float64()` so that subsequent convolution operations can produce values outside $[0, 255]$ without overflow. For copying the image, the code calls `cv2.copyMakeBorder (img,0,0,0,0,cv2.BORDER_REPLICATE)` with zero padding on every side, which in practice just creates a deep copy; the `BORDER_REPLICATE` flag would replicate edge pixels if positive padding were specified. The optimized convolution is done via `cv2.filter2D(img, -1, kernel)`, where -1 means the output keeps the same depth as the input; internally, this delegates to OpenCV's C++ backend, which can use SIMD and multi-threading [2].

B. Direct Computation vs OpenCV filter2D

The code compares two ways of computing the convolution of the grayscale image `FlowerGarden2.png` (240×360 pixels) with the 3×3 sharpening kernel. The first is a direct method thaimitationt uses a nested Python loop over all interior pixels (y, x) with $1 \le y \le h-2$, $1 \le x \le w-2$, computing explicitly:

$$v = 5\,I(y,x) - I(y-1,x) - I(y,x-1) - I(y+1,x) - I(y,x+1), \qquad (2)$$

followed by a clamp $I_{\text{out}}(y,x) = \min(\max(v,\,0),\,255)$. The second is a single call to `cv2.filter2D`, which does the same operation but through OpenCV's compiled C++ backend.

Given the nature of these two implementations, the hypothesis is straightforward: the direct Python loop should be dramatically slower than `filter2D`, because every pixel iteration goes through the Python interpreter with dynamic type checking, whereas the C++ path processes contiguous memory with compiled instructions and can leverage SIMD vectorization and multi-threading. We would also expect that, for a small image like this one, enabling or disabling multi-threading in OpenCV should not make a large difference, since the overhead of thread synchronization could offset the parallelism gains when the workload per thread is small.

To test this, each method was benchmarked over multiple independent runs (5 for the slow direct method, 20 for the fast `filter2D` variants), and the median execution time was retained to mitigate warm-up and scheduling noise. A single-thread variant of `filter2D` was measured separately by calling `cv2.setNumThreads(1)` before execution.

As reported in Table I and illustrated in Fig. 1, the direct Python loop takes about $0.70\,\text{s}$ while `filter2D` completes in

TABLE I
MEDIAN EXECUTION TIME FOR THE CONVOLUTION OF A 240×360 GRAYSCALE IMAGE WITH A 3×3 KERNEL.

Method	Median time	Runs
Direct (Python loop)	0.6987 s	5
`filter2D` (multi-thread)	0.329 ms	20
`filter2D` (1-thread)	0.348 ms	20

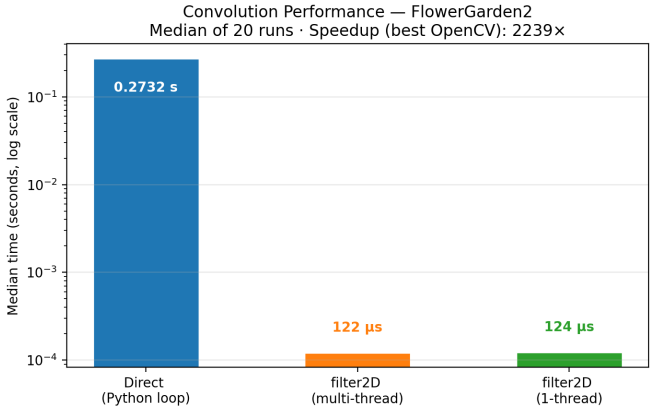

Fig. 1. Median execution time comparison (log scale) for the three convolution methods on a 240×360 image.

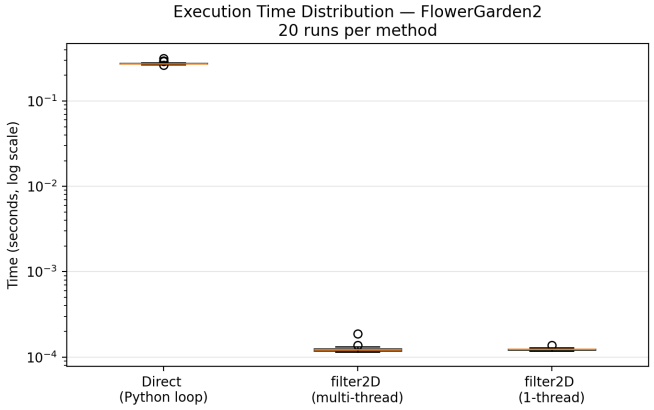

Fig. 2. Distribution of execution times across all benchmark runs for each method.

roughly 0.33 ms, giving a speedup of approximately $2\,100\times$. This is consistent with the hypothesis: the per-pixel overhead of the Python interpreter is enormous compared to the compiled, vectorized C++ implementation. The difference between the multi-thread and single-thread variants of `filter2D` is almost negligible (0.329 ms vs 0.348 ms), which also confirms our expectation that thread synchronization cost offsets any parallelism benefit at this image size. Moreover, Fig. 2 shows that the `filter2D` measurements have very low variance, reflecting the stability of a compiled implementation, while the direct method exhibits more dispersion—likely caused by Python's garbage collector and OS scheduling interrupting the long-running loop.

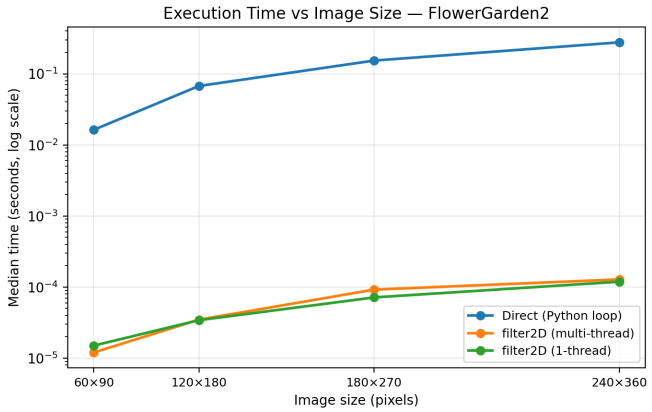

Fig. 3. Execution time as a function of image resolution (25 %, 50 %, 75 %, and 100 % of the original 240×360 image).

To see how these results scale, the benchmark was repeated on resized versions of the image at 25 %, 50 %, 75 %, and 100 % of the original resolution. The hypothesis here is that the direct method should grow roughly quadratically with the number of pixels (since it visits every pixel once in a doubly-nested loop), whereas `filter2D` should remain nearly flat at these small sizes because the actual convolution time is dwarfed by the fixed overhead of the function call. As shown in Fig. 3, this is exactly what happens: the direct method curve grows steeply as the resolution increases, consistent with $\mathcal{O}(H \cdot W \cdot k^2)$ complexity, while both `filter2D` variants stay close to the bottom of the plot across all tested resolutions.

C. Visual and Numerical Comparison

Since both methods are computing the same mathematical convolution, the hypothesis is that their outputs should be identical in the interior of the image, and that any differences should appear only at the borders due to the different ways each method handles boundary pixels. The direct loop simply skips the border (it only iterates from $y = 1$ to $h-2$) and leaves the original values there via the initial copy, whereas `filter2D` applies its own default border extrapolation (`BORDER_REFLECT_101`) before computing the convolution, so the border values will differ.

The results in Fig. 4 confirm this. A pixel-wise comparison shows that $98.9\,\%$ of the pixels are numerically identical between the two outputs, with a mean absolute difference of only 0.575 gray levels. The maximum difference reaches 153 gray levels, but as visible in the amplified difference heatmap (bottom right of Fig. 4), these large differences are concentrated exclusively at the image borders. In the interior, both methods produce the same result, which makes sense because the convolution formula is the same—the only divergence comes from boundary handling, not from the computation itself. This confirms that the performance gap discussed in the previous subsection is purely an implementation-level phenomenon and does not affect the mathematical result.

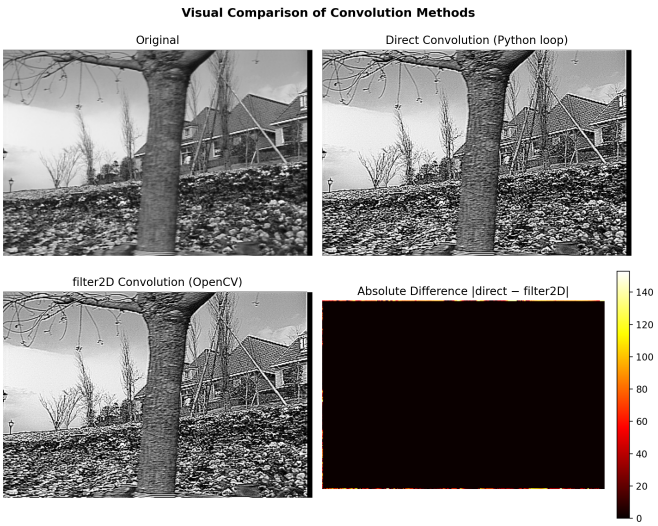

Fig. 4. Visual comparison: original image (top left), direct method result (top right), `filter2D` result (bottom left), and absolute difference amplified $\times 10$ (bottom right).

D. Contrast Enhancement by Unsharp Masking

The goal of this subsection is to explain *why* the 3×3 convolution kernel used in the TP code enhances contrast. The kernel is:

$$K = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}. \tag{3}$$

At first glance the coefficients may seem arbitrary, but they have a precise algebraic structure: K can be decomposed as the identity minus a discrete Laplacian. This decomposition reveals that K implements *unsharp masking*, a classical technique that preserves low-frequency content (uniform regions) while amplifying high-frequency content (edges).

a) Step 1 — The discrete Laplacian.: The continuous Laplacian of a function $f(x, y)$ is $\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$. On a discrete pixel grid, each second derivative is approximated by a centered finite difference (e.g. $\frac{\partial^2 f}{\partial x^2} \approx f[i, j+1] - 2f[i, j]+f[i, j-1]$), and the sum of both gives the 4-connected Laplacian stencil:

$$L_4 = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \qquad (f \star L_4)[i, j] \approx \Delta f(i, j). \tag{4}$$

Two properties of L_4 are important: its coefficients sum to zero (so it has no DC response: a constant image gives zero output), and its output is *signed* (positive where the pixel is darker than its neighbors, negative where it is brighter).

b) Step 2 — Decomposing K as identity minus Laplacian.: Let δ denote the identity kernel $\delta = \left(\begin{smallmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{smallmatrix}\right)$. Subtracting L_4 from δ coefficient by coefficient:

$$\delta - L_4 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} = K. \tag{5}$$

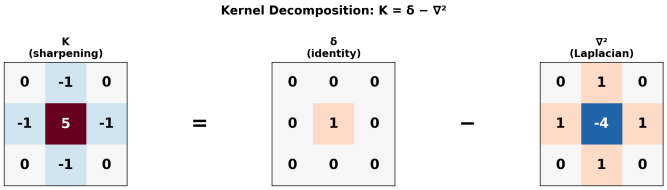

Fig. 5. Kernel decomposition: $K = \delta - L_4$ (4-neighborhood Laplacian).

This is verified numerically in Fig. 5. Since convolution is linear, the output of filtering with K can be split into two terms:

$$f_{\text{out}} = f \star K = f \star (\delta - L_4) = \underbrace{f \star \delta}_{=f} - \underbrace{f \star L_4}_{\approx \Delta f} = f - (f \star L_4). \tag{6}$$

This is exactly the *Laplacian-based unsharp masking* formula $f_{\text{out}} = f - \gamma\, \Delta f$ with gain $\gamma = 1$.

c) Step 3 — Why does this enhance contrast? (pixel-level view).: To understand the effect intuitively, we can rewrite Eq. (6) in terms of the local neighborhood. Writing out the Laplacian convolution at pixel $[i, j]$:

$$(f \star L_4)[i, j] = f[i+1, j]+f[i-1, j]+f[i, j+1]+f[i, j-1]-4f[i, j]. \tag{7}$$

If we define the mean of the four direct neighbors as

$$\overline{f}_{N_4}[i, j] = \tfrac{1}{4}(f[i+1, j] + f[i-1, j] + f[i, j+1] + f[i, j-1]), \tag{8}$$

then the Laplacian simplifies to $(f \star L_4)[i, j] = 4\big(\overline{f}_{N_4}[i, j] - f[i, j]\big)$. Substituting into Eq. (6):

$$f_{\text{out}}[i, j] = f[i, j] \; + \; 4\big(f[i, j] - \overline{f}_{N_4}[i, j]\big). \tag{9}$$

Equation (9) is the key: the output equals the original value plus a correction proportional to how much the pixel *differs from its local mean*. Three cases arise:

- **Smooth region** ($f[i, j] \approx \overline{f}_{N_4}$): the correction is nearly zero, so $f_{\text{out}} \approx f$. Uniform areas are preserved.
- **Locally bright pixel** ($f[i, j] > \overline{f}_{N_4}$): the correction is positive, pushing the pixel *brighter*.
- **Locally dark pixel** ($f[i, j] < \overline{f}_{N_4}$): the correction is negative, pushing the pixel *darker*.

In short, pixels are pushed *away* from their local mean. At an edge, the bright side becomes brighter and the dark side becomes darker, which is exactly what "enhancing contrast" means.

d) Experimental results.: Fig. 6 shows the decomposition applied to the test image: the Laplacian response (panel b) is signed and concentrated at edges, while the sharpened result (panel d) shows visibly crisper contours.

e) Dynamic-range considerations.: Since the filter pushes pixels away from their local mean (Eq. (9)), the output distribution necessarily becomes wider than the input. Near strong edges, the Laplacian response can be large, causing f_{out} to exceed the $[0, 255]$ range. Table II quantifies this: before clipping, values range from -609 to 877, and the standard

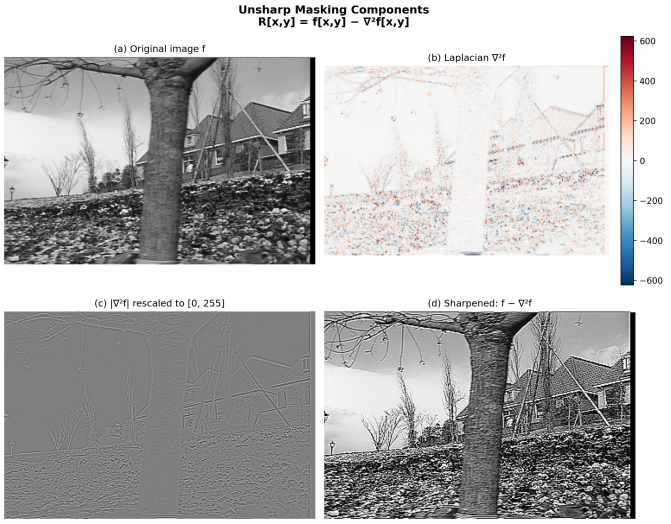

Fig. 6. Unsharp masking components: (a) original f, (b) Laplacian $f \star L_4$ (signed, divergent colormap), (c) $|f \star L_4|$ rescaled to $[0, 255]$, (d) sharpened result $f - (f \star L_4)$ after clipping.

TABLE II
PIXEL INTENSITY STATISTICS BEFORE AND AFTER SHARPENING.

	Original	Sharp (float)	Sharp (clip)
Min	0.0	−609.0	0.0
Max	255.0	877.0	255.0
Mean	127.14	126.99	129.68
Std	65.90	136.08	89.58

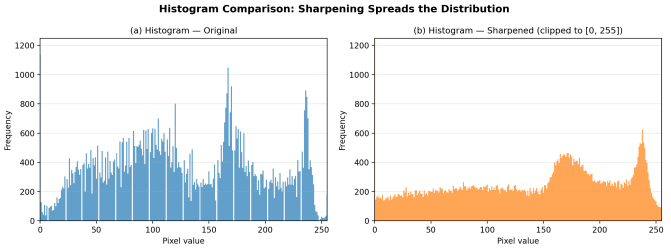

Fig. 7. Histogram comparison: (a) original image, (b) sharpened image after clipping to $[0, 255]$.

deviation doubles from 65.90 to 136.08. After clipping to $[0, 255]$, the extreme values accumulate at the boundaries (visible as a saturation peak at bin 255 in Fig. 7b), and the mean shifts slightly upward (127.14 → 129.68) because clipping is asymmetric. The DC-gain-equals-one property ($\widehat{K}(0, 0) = 1$) guarantees that the mean is exactly preserved in the float domain (127.14 → 126.99, the residual 0.15 being a boundary effect); it is only the clipping step that introduces a bias.

f) Takeaway.: K enhances contrast because it is algebraically equivalent to the identity minus the discrete Laplacian ($K = \delta - L_4$). At the pixel level, this pushes each value away from its local mean, amplifying edges while leaving flat regions unchanged. The practical limitation is that strong edges can push values outside $[0, 255]$, requiring either floating-point storage or clipping with the associated saturation artifacts.

E. Gradient Filters (Sobel)

In the previous parts, we applied convolution kernels to modify the image (enhancement using the Laplacian). Here we keep the same tool—convolution—but change the goal: instead of sharpening, we want to estimate local intensity variations by computing two derivative-like images, I_x and I_y. These two maps can then be combined into a single edge-strength image through the gradient magnitude $\|\nabla I\|$.

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

These two quantities form the gradient vector

$$\nabla I = (I_x,\, I_y),$$

which points toward the direction of maximum increase of intensity. Its magnitude,

$$\|\nabla I\| = \sqrt{I_x^2 + I_y^2},$$

is a simple measure of local contrast: it becomes large around edges and sharp transitions.

1) From derivatives to discrete filters: In a digital image we only have samples $I[i, j]$ (pixels), so derivatives must be approximated. A basic idea is finite differences (e.g. $[-1 \;\; 1]$), but this is very sensitive to noise because it uses only two neighboring pixels.

A common improvement is to combine:

- a *derivative* in one direction,
- with a *smoothing* (low-pass) in the orthogonal direction.

This is exactly what the Sobel operator does. We use the standard 3×3 kernels:

$$h_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \qquad h_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}. \quad (10)$$

They are separable (up to a constant factor), for example

$$h_x \propto \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}, \qquad h_y = h_x^\top.$$

So we can read Sobel as: *smooth with* $[1, 2, 1]$ in the perpendicular direction, then *differentiate with* $[-1, 0, 1]$. In practice this produces gradient maps that are less noisy than raw finite differences, while still reacting strongly at edges.

2) Gradient components and magnitude: We compute the two components by filtering the image:

$$I_x[i, j] = (I * h_x)[i, j], \qquad I_y[i, j] = (I * h_y)[i, j], \quad (11)$$

and then the gradient magnitude:

$$\|\nabla I[i, j]\| = \sqrt{I_x[i, j]^2 + I_y[i, j]^2}. \quad (12)$$

In OpenCV, the filtering function behaves like a correlation (the kernel is not flipped). In this TP context, this does not change the usefulness of the result: it mainly affects the *sign convention* of I_x and I_y depending on the chosen kernel orientation. What matters for edges is that strong transitions create large responses (and the magnitude $\|\nabla I\|$ is independent of the sign).

3) Why floating point matters (and how display can go wrong): A practical detail is that I_x and I_y are **signed**: they contain both positive and negative values depending on whether the transition is dark-to-bright or bright-to-dark. Because of that, we should avoid computing the convolution in `uint8`. Otherwise, negative values are clipped to 0, and half of the information disappears.

In our implementation we therefore compute the convolutions in floating point:

- either by converting the image to `float64` before filtering,
- or by explicitly forcing the output depth in `cv2.filter2D` (e.g. `cv2.CV_64F`).

This guarantees that negative responses are preserved and that $\|\nabla I\|$ can exceed the usual $[0, 255]$ range.

4) Visualization of signed components: Since I_x and I_y can be negative, their visualization needs a mapping that does not destroy the sign. Typical options are:

1) **Incorrect: naive clipping to** $[0, 255]$. Negative values become 0, so one edge polarity vanishes.
2) **Absolute value**. Displays edge strength but removes the sign (direction of transition).
3) **Shift + rescale**. Map $[\min, \max]$ linearly to $[0, 255]$ so that 0 becomes mid-gray.
4) **Diverging colormap centered at 0 (recommended for interpretation)**. Use a symmetric range

$$m = \max\left(|\min(I_x)|,\ |\max(I_x)|\right),$$

$$\text{and display } I_x \text{ in } [-m, +m],$$

(and similarly for I_y). This makes positive/negative values comparable and avoids a misleading color dominance.

For the magnitude $\|\nabla I\|$, values are non-negative, so a grayscale colormap is enough. However, since the maximum can be much larger than 255, we typically normalize *only for display*. The raw values should be kept if we plan to compare gradients across images or apply consistent thresholds.

Fig. 8 illustrates the gradient decomposition on the test image. As introduced in [14], the gradient vector ∇I encodes two fundamental pieces of local geometry: its *norm* $\|\nabla I\|$ measures the local contrast (the rate of intensity change), while its *argument* $\arg \nabla I$ gives the direction of steepest ascent, orthogonal to the isophote.

These properties are directly visible in the figure. In panel (b), the horizontal component I_x highlights **vertical edges**: positive values (red) appear where intensity increases from left to right, and negative values (blue) mark the opposite transition. Panel (c) shows I_y, which correspondingly highlights **horizontal edges**. For instance, the strong blue and red bands along the flower petals in panel (b) trace the vertical contours of those structures, while the horizontal top-to-bottom transitions that appear in the fence and the grass are captured in panel (c).

Panel (d) combines both components into the scalar magnitude $\|\nabla I\|$: every edge appears bright regardless of its orientation, confirming that the norm acts as a direction-independent measure of local contrast. Panel (e) displays the gradient direction $\theta = \text{atan2}(I_y, I_x)$ using a circular (HSV) colormap; it directly corresponds to the quantity $\arg \nabla I$ from the course [14], and one can verify that edges in the same physical direction share the same hue. Finally, panel (f) shows a simple binary edge map obtained by keeping only pixels where $\|\nabla I\| > \mu + \sigma$ (a heuristic threshold); this illustrates how the gradient magnitude can be used as a starting point for edge detection—a topic further developed in the course through the zero-crossings of the Laplacian and the Canny detector.

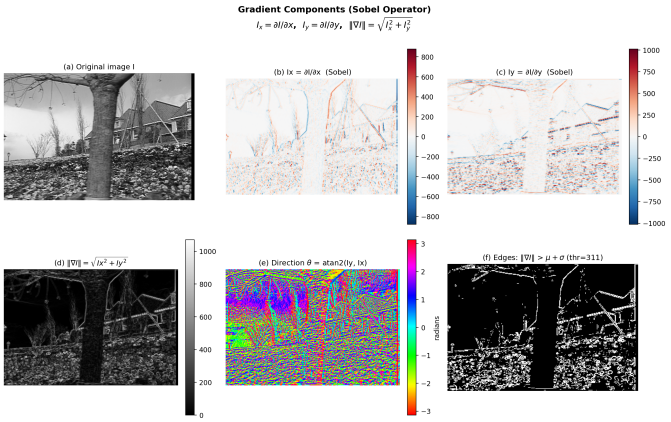

Fig. 8. Gradient decomposition: (a) original image, (b) I_x and (c) I_y shown with a diverging colormap (red = positive, blue = negative, white = zero), (d) gradient magnitude $\|\nabla I\|$, (e) gradient direction $\theta = \arg \nabla I$, (f) binary edge map ($\|\nabla I\| > \mu + \sigma$).

II. Descriptors and pairing

In computer vision, a descriptor is understood as a numerical representation, generally vectorial, that is used for the summarized representation in features of the neighborhood of an entity in an image, which is specifically conceived in order to be able to perform matching or comparison operations of those same entities with other images even when there are events such as changes of scale, rotations, illuminations, or noise that can interfere with the normal matching process [3]; a descriptor can then be understood, therefore, as the result of mapping into a vector a local neighborhood in an image, guaranteeing robustness understood as stability under image changes, viewpoints, geometric distortion and occlusions, the discrimination of different objects given the vector, and efficiency, in such a way that it will be easy to compute and compact in memory, that is, it is a feature vector corresponding to the key points.

On the other hand, a detector is an algorithm whose function is to find in the image a set of points, or keypoints, that are repeatable and well localizable [4]; generally, detectors operate from the definition of a measure $r(x, y)$ or $R(x, y, \omega)$ that is used for the detection of corners, blobs, or stable regions in the domain mainly of local multi-scale characterization, seeking local maxima or minima of these functions and filtering unsta-

ble candidates, focusing on repeatability, localization precision of these keypoints in the figure, and stability in detection [5].

A. Oriented FAST and Rotated BRIEF

ORB is a local feature method that outputs keypoints and a binary descriptor; it can be conceptually understood as a pipeline that combines a FAST-family detector and a BRIEF-family descriptor. Considering that since FAST is not scale-invariant, ORB detects FAST keypoints on multiple rescaled versions of the image, on an image pyramid [6].

The FAST method works by considering a circle of generally 16 pixels in the neighborhood of a candidate pixel p and classifies the pixel p as a corner in the case where there exists a group of pixels on that circle (the neighborhood of p) that have sufficient intensity in comparison with p using a threshold value t, as defined in Eq. (13):

$$S_{p\to x} = \begin{cases} d, & I_{p\to x} \le I_p - t \\ s, & I_p - t < I_{p\to x} < I_p + t \\ b, & I_p + t \le I_{p\to x} \end{cases} \tag{13}$$

where $S_{p\to x}$ is the "label" of pixel x on the circle.

- d (*dark*): the circle pixel is at least t darker than the center.
- b (*bright*): the circle pixel is at least t brighter than the center.
- s (*similar*): it is inside the band $(I_p - t,\ I_p + t)$, that is, it does not differ enough.

FAST *declares* that p is a corner if there exists a set of contiguous pixels on the circle such that all of them are *bright* or all of them are *dark*. It is a very fast method because it can be implemented via process optimizations such as the use of a learned decision tree to select which positions to evaluate first [7].

By itself, the FAST segment test constitutes only a binary classification; however, FAST introduces a score value so that non-maximum suppression can be performed and only local maxima are kept [7]. One (efficient) definition given is in Eq. (14):

$$V = \max\left(\sum_{x\in S_{\text{bright}}} \left(|I_{p\to x} - I_p| - t\right), \sum_{x\in S_{\text{dark}}} \left(|I_p - I_{p\to x}| - t\right) \right). \tag{14}$$

In ORB, the FAST threshold is set sufficiently low so as to obtain more than N candidates that can then be evaluated using a Harris corner measure, and to keep the top N values [6].

A standard Harris measure uses the second-moment (structure tensor) matrix H and response in Eq. (15):

$$C = |H| - k\big(\operatorname{trace}(H)\big)^2. \tag{15}$$

The selection between a score type, more stable with Harris, or faster but slightly less stable with FAST_SCORE, is made available by OpenCV [8].

Given the nature of FAST, it does not naturally produce an orientation; this is why ORB needs to add an orientation step, and for that it implements the centroid idea. ORB improves stability by computing moments only within a circular region of radius r, with raw moments defined in Eq. (16).

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y) \tag{16}$$

$$C = \left(\frac{m_{10}}{m_{00}},\ \frac{m_{01}}{m_{00}} \right) \tag{17}$$

Then the orientation is the angle of the vector from patch center O to centroid, as in Eq. (18):

$$\theta = \operatorname{atan2}(m_{01}, m_{10}) \tag{18}$$

On the other hand, BRIEF is responsible for representing a patch p by means of a bitstring that is produced from simple intensity comparisons, as shown in Eq. (19).

$$\tau(p; x, y) = \begin{cases} 1, & p(x) < p(y) \\ 0, & p(x) \ge p(y) \end{cases} \tag{19}$$

An n-bit descriptor can then be built as in Eq. (20).

$$f_n(p) = \sum_{i=1}^{n} 2^{i-1}\, \tau(p; x_i, y_i) \tag{20}$$

It focuses on binary strings given the inherent ease of comparing them using Hamming distance rather than L_2 distances in vectors [6]. Plain BRIEF is very sensitive to rotation; in response to this it suggests the concept of steered BRIEF, which rotates the sampling pattern as a function of a discretized angle θ. Let the test locations be encoded in a matrix as in Eq. (21).

$$S = \begin{pmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \end{pmatrix} \tag{21}$$

These locations are rotated according to Eq. (22).

$$S_\theta = R_\theta S \tag{22}$$

Finally, the descriptor is computed using the rotated test coordinates, as expressed in Eq. (23).

$$g_n(p, \theta) := f_n(p)\big|_{(x_i, y_i)\in S_\theta} \tag{23}$$

ORB then analyzes a subtle but critical issue: when you orient BRIEF consistently, the bit statistics change—the means move away from 0.5 and the tests become less discriminative and more correlated.

The final descriptor used in ORB is an rBRIEF constructed by generating tests with a mean close to 0.5 and high variance, which also preserve low correlation with the tests already selected. The procedure can be summarized as follows: (i) enumerate all pairs of subwindows (then remove overlapping tests), yielding candidate tests; (ii) run each test over all training patches; (iii) sort tests by the distance of their mean from 0.5 (best first); and (iv) apply a greedy selection, keeping a test only if its absolute correlation with all selected tests

is below a threshold. This produces a final descriptor that remains binary and fast (Hamming), but with bits that are more informative and less redundant than a naive "steered BRIEF".

For binary descriptors $d_1, d_2 \in \{0,1\}^{256}$, matching uses the Hamming distance, as defined in Eq. (24):

$$\mathrm{Ham}(d_1, d_2) = \sum_{i=1}^{256} \mathbf{1}[d_{1,i} \neq d_{2,i}]\,. \tag{24}$$

This can be computed efficiently as in Eq. (25):

$$\mathrm{Ham}(d_1, d_2) = \mathrm{popcount}(d_1 \oplus d_2). \tag{25}$$

BRIEF emphasizes this efficiency, and ORB notes SSE popcount optimizations in their matching implementation [9].

Finally, in ORB an image pyramid of scales is constructed and, for each scale, FAST corners are detected using a test threshold and are assigned a score using FAST_SCORE or Harris; the strongest features are retained, generally using Harris; the orientation θ is computed via the intensity centroid moments; and finally a descriptor is computed using a smoothed patch, a rotated sampling pattern, and an rBRIEF learned test set, in order to be able to perform descriptor matching using Hamming distance computed via popcount. It is worth highlighting that the way ORB is constructed allows it to handle in-plane rotation, which is addressed through the centroid-based orientation and the steered sampling pattern. Additionally, it can handle image scale by implementing pyramidal detection; however, it remains not fully affine-invariant to viewpoint changes, because projective changes can still break patch appearance.

B. KAZE

As in the case of ORB, KAZE is an algorithm for the detection and description of keypoints, but unlike ORB it constructs the scale space with nonlinear, edge-preserving diffusion; it detects points with a Hessian-type detector and describes them with a (M-)SURF-type descriptor over that nonlinear scale space [10].

KAZE starts with the construction of the nonlinear scale space; for that, a family of images $L(x, y, t)$ is defined, where t plays the role of scale or diffusion time, and it is modeled using the PDE given in Eq. (26):

$$\frac{\partial L}{\partial t} = \mathrm{div}\big(c(x, y, t)\,\nabla L\big). \tag{26}$$

Here, ∇L points toward where the image changes the fastest (edges = large gradient). The diffusion "flow" can be seen as Eq. (27):

$$\mathbf{J} = -c\,\nabla L, \tag{27}$$

then $\mathrm{div}(\mathbf{J})$ measures how much flow "accumulates" or "leaves" a point, resulting in a definition of brightness propagation analogous to heat propagation, but with a conductivity c that is a controllable parameter [10]. This is precisely the key, because c depends on the gradient, as expressed in Eq. (28):

$$c(x, y, t) = g(\|\nabla L(x, y, t)\|)\,. \tag{28}$$

It should be noted that this gradient is not the raw gradient of the image, but rather the gradient computed on a Gaussian-smoothed version.

If $|\nabla L_\sigma|$ is small, it corresponds to a flat region in the image, and what is sought is to increase diffusion; however, in the opposite case, if $|\nabla L_\sigma|$ is large, it corresponds to a strong edge in the image where the main intention is not to cross that edge [11]. This is ensured by the two typical forms of g given in Eq. (29) and Eq. (30):

$$g_1(s) = \exp\left(-\frac{s^2}{k^2}\right), \tag{29}$$

$$g_2(s) = \frac{1}{1 + \frac{s^2}{k^2}}. \tag{30}$$

In both definitions, a fundamental element that emerges is k, which is a contrast threshold used as a separation between variations that are considered small for the image, such as noise or textures, and those that are considered large, such as an edge. When k is small, many gradients are considered as edges, which implies that diffusion is stopped in many parts of the image, that is, it is smoothed less; whereas if k is large, only very large gradients are considered as edges, so there is more diffusion and the image is smoothed more.

Since this problem does not have a closed-form analytic solution, KAZE uses numerical schemes that are semi-implicit and that use additive operator splitting in order to construct the scale space with stability [10].

At each level or scale of KAZE, the Hessian response is computed through its determinant and maxima are searched both in position and in scale, as given in Eq. (31):

$$L_{\text{Hessian}} = \sigma^2 \left(L_{xx} L_{yy} - L_{xy}^2\right). \tag{31}$$

Here, L_{xx}, L_{yy}, and L_{xy} are the second derivatives (local curvature) measured on L, and the factor σ^2 is a normalization so that the response is comparable across scales (because derivatives "shrink" as scale increases).

Then, KAZE searches for extrema in spatial neighborhoods and across scales, and estimates with precision the position of the maximum that was found with the Hessian response in Eq. (31). KAZE also computes a SURF-type orientation: it takes first-order derivatives in a circular neighborhood with a radius proportional to ω, weights them with a Gaussian, and searches—through a sliding angular window—for a dominant angle.

For the descriptor, KAZE makes use of an M-SURF descriptor, as already mentioned, adapted to the nonlinear scale space, integrating gradient-type responses over sub-patches; the objective of this type of descriptor is to capture how intensity changes around the point in a way that is robust to noise [12].

For a keypoint at scale σ_i, it computes derivatives L_x and L_y at that scale. It builds a 4×4 grid of subregions around

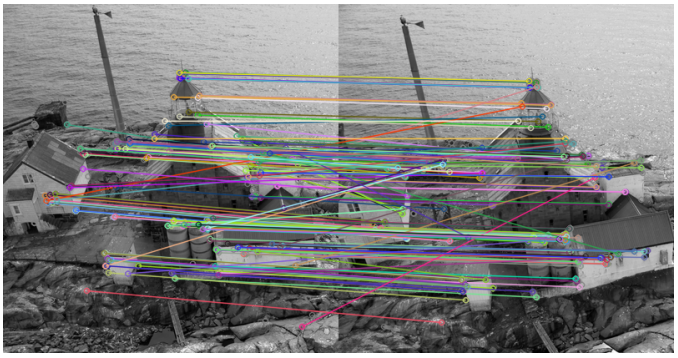

Fig. 9. Cross-check matching results using ORB.

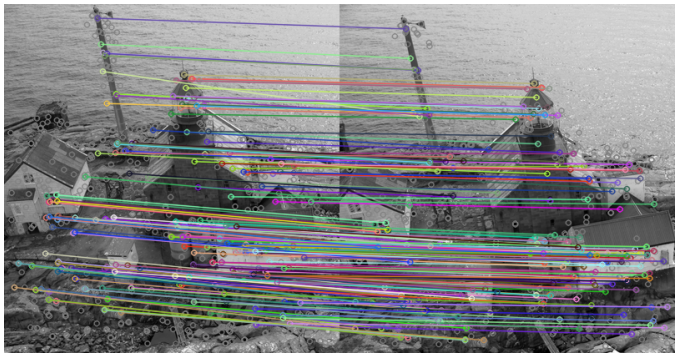

Fig. 10. Cross-check matching results using KAZE.

the keypoint [10]. In each subregion it sums a vector of the form shown in Eq. (32):

$$d_v = \left(\sum L_x,\ \sum L_y,\ \sum |L_x|,\ \sum |L_y| \right). \qquad (32)$$

It then concatenates all subregion vectors to obtain a typical 64-dimensional descriptor, and finally normalizes it. If orientation is used, the sampling is rotated and the derivatives are also computed in that orientation.

This allows, finally, each KAZE keypoint to be described as in Eq. (33), together with a descriptor (64D or extended depending on the implementation).

$$(x, y, \sigma, \theta) \qquad (33)$$

Given the implementation of extrema detection in nonlinear scale spaces constructed by diffusion, the detector is scale-invariant. The computation of the dominant orientation of the keypoint in order to build the descriptor makes it rotation-invariant. If the upright mode of OpenCV is used, although the algorithm becomes faster, it loses that invariance property by not computing the dominant orientation. It can be said that, due to the normalization inherent to the M-SURF descriptor, this algorithm also obtains descriptors that are approximately contrast-invariant. Finally, KAZE typically behaves well for moderate viewpoints, but it is not "affine-invariant" in the strong sense.

C. Qualitative Performance of descriptors-pairing

As a preliminary stage, it is proposed to qualitatively analyze the results obtained by the descriptor and pairing for matching, FLANN and CROSSCHECK, with and without ratio test, for the ORB and KAZE descriptors on two images for which the second is the result of a transformation of the first.

In Figs. 9–10 the results of the implementation of the detectors and matching using ORB and KAZE, respectively, are plotted with cross-check matching for a unique value, which consists of taking the descriptors with the smallest distance in the two images and verifying that the correspondence is mutual; if it is, it is identified as a match. In both cases, the best 200 matches between the two images are plotted with lines. From the comparison of the use of both pipelines, results emerge that are immediately striking for the analysis.

It was mentioned in the description of the ORB algorithm that, by combining FAST with steered BRIEF, it is very fast and rotation-invariant, and if it uses a pyramid as is the case, it can handle scale reasonably well; however, including a change in the point of view of the image, which is the case between the compared images, causes anisotropy that makes the pointwise intensity comparisons of the bits performed by BRIEF reduce the rate of correct matches by comparing different entities.

The effect of viewpoint on pairing using cross-check is visible for ORB, since under a slight variation in viewpoint one would expect vectors between the identified pairs with similar directions and lengths; however, in this case, a considerable number of vectors with directions that are not very coherent with the viewpoint change becomes evident. There is also clear evidence of some matches connecting structures that are semantically distinct in the sense of the image, and there is also an accumulation of matches on repeated similar structures that generate ambiguity.

Additionally, in the case of the ORB algorithm, selecting a number n of features—in this case 500—causes regions with strong textures, such as the central area of the image, to occupy most of the keypoints identified in the figure, mainly due to the very fast corner response that is accentuated in structures that are mostly repetitive.

On the other hand, in the case of the KAZE implementation with cross-checking, a much more uniform behavior of the vectors between the paired keypoints identified in the images is observed, with few outliers among the 200 best matches. It is important to highlight that this is favored because, although KAZE is also not constructed to be invariant to point-of-view modifications, the fact that KAZE operates on a nonlinear scale space, achieved through a diffusion process, in some sense reduces localization precision and distinctiveness, so that under slight viewpoint variations it can still deliver repeatable results in the presence of small geometric deformations. Additionally, the form of the descriptor, which uses the notion of gradients in KAZE, is by definition more tolerant to small geometric deformations (such as those in the images under study) than a binary identifier based on pointwise comparisons, because it

TABLE III
MEASURED RUNTIME FOR DETECTION/DESCRIPTION AND MATCHING CROSS-CHECK.

Detector	Detect+Describe (s)	Matching (s)
ORB	0.011849542	0.001364103
KAZE	0.195190992	0.003179951

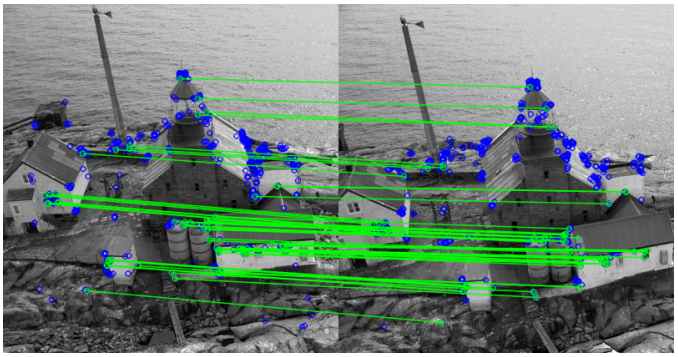

Fig. 11. Ratio-test matching results using ORB.

is integrated over areas rather than exact points, which favors this improved response.

It is important to highlight that the KAZE detector is based on extrema of the normalized Hessian determinant in its nonlinear scale space, so in textures/structures it can produce quite a few valid extrema, even in outer regions. Thus, areas that in the ORB case appeared as a concentration zone due to the presence of repetitive textures, in KAZE tend to yield keypoints that are more spatially distributed. In the image, it can be observed how the keypoints identified with the KAZE algorithm are more spread across the scene, even leading some of them to form matches between figures.

We also report the ratio-test matching results for the same pair. Fig. 11 shows ORB with ratio test, and Fig. 12 shows KAZE with ratio test.

In terms of keypoint detection and description time, the result is consistent with the implemented algorithms. In the case of ORB, since it is based on FAST with simple comparisons and rBRIEF with binary operations, it is computationally cheaper and takes a much shorter time for detection and description. In contrast, KAZE, which constructs scale spaces by nonlinear diffusion, implies solving a diffusion equation at different levels, multiple iterations, and Hessian-type derivative operations; thus it is much heavier and therefore takes significantly more time to execute detection and description on the image. KAZE matching is also slower, mainly because ORB uses binary operations based on Hamming distance, whereas KAZE uses floating-point operations with L_2 distances, implying higher computational complexity, as is evident in Table III.

We also include the FLANN matching results for visual comparison. Fig. 13 shows ORB with FLANN, and Fig. 14 shows KAZE with FLANN.

When analyzing the results, mainly the matching between the two images using ORB and cross-check matching, it is proposed to modify the way the match results are selected between the images, with the main objective of obtaining

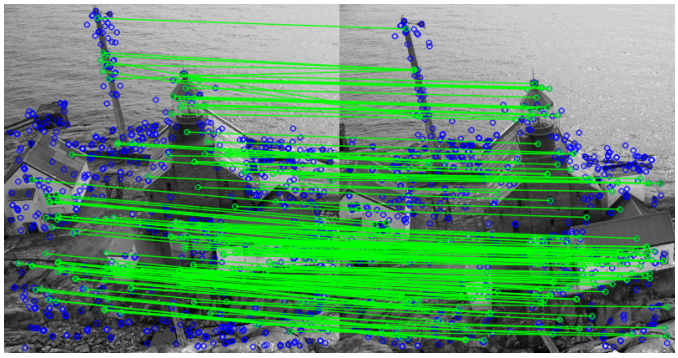

Fig. 12. Ratio-test matching results using KAZE.

TABLE IV
MEASURED RUNTIME FOR RATIO-TEST MATCHING.

Detector	Detect+Describe (s)	Matching (s)
ORB	0.019036635	0.003361369
KAZE	0.181437621	0.001895746

TABLE V
MEASURED RUNTIME FOR FLANN MATCHING.

Detector	Detect+Describe (s)	Matching (s)
ORB	0.018674926	0.002685916
KAZE	0.205246737	0.015027942

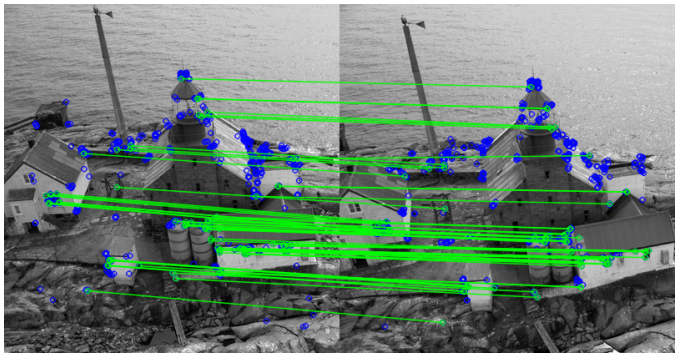

Fig. 13. FLANN matching results using ORB.

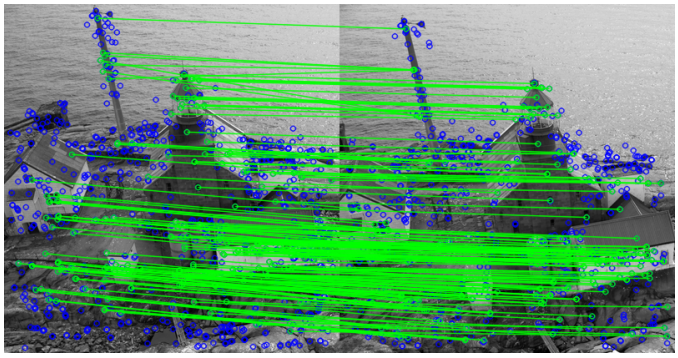

Fig. 14. FLANN matching results using KAZE.

final values that are more reliable in terms of the quality of the identified matches. For this reason, the ratio test logic is employed to validate the relationship between the paired candidates. In this way, after the detection and description stages described previously, a modification is proposed in the matching algorithm: for each keypoint it identifies the two nearest neighbors and then filters those neighbors, keeping only those for which the best match is at most 70% of the distance of the second-best match, mainly to avoid ambiguities.

As shown in Figs. 11 and 12, the results for ORB when implementing this modification in match selection become evident, since the trajectories of the vectors between pairs in the two images are more uniform than with the simple cross-checking implementation; however, since the implemented detection and description algorithm is the same, the observations regarding the effect of the viewpoint change on pairing quality remain evident, as does the difference with the KAZE results for the same images under the same pair-selection algorithm, given that KAZE can identify approximately three times more correspondences than ORB that satisfy the ratio-test criterion with a threshold of 0.7.

When analyzing the time taken by the algorithms in pairing, a striking result is found: the KAZE time decreases with respect to the cross-checking evaluation. This is explained because, in cross-checking, the algorithm must evaluate L_2 distances in both directions, whereas when cross-checking is set to false and only the two nearest neighbors are kept, the cost decreases because the L_2 computation is done in only one direction; moreover, the increase in execution time due to validating the threshold condition is smaller than the time required by the matching algorithm to perform the second mutual validation, which is why the total time decreases. In contrast, in the case of ORB, since the cost of validating in both directions is minimal because it is a binary operation, the additional computational cost of comparing the two nearest neighbors and applying the threshold causes the time spent by this algorithm in matching to be higher than in cross-checking.

Finally, the implementation of a FLANN algorithm is proposed for checking, enabling fast nearest-neighbor search through an indexed data structure and an approximate k-NN search. The results in terms of pair identification between the images are very similar to the match-ratio case without FLANN; however, the execution times of both algorithms are lower due to the optimized search for the nearest neighbors, which can make it attractive when compared with the simple ratio test. It is important to highlight that Hamming-based cross-checking for ORB remains the fastest in execution, even with FLANN optimizations. It is also important to note that, in this case, the approximate relation of about three times more pairs satisfying the ratio test for KAZE than for ORB is maintained, mainly due to the effects already discussed of nonlinear diffusion and how it favors invariance under small geometric changes such as a small change of point of view.

Although the computation of the distance in each one of the implemented algorithms has already been detailed in the introduction to their functioning, given the effects on execution time mainly in the matching stage, it is convenient to revisit this aspect in greater depth. As was already described, ORB is a binary descriptor that uses an rBRIEF-type descriptor where each component is the result of an intensity comparison, as shown in Eq. (34):

$$b_i = \mathbf{1}[I(x_i) < I(y_i)] \in \{0, 1\}. \tag{34}$$

Then the complete descriptor is given by Eq. (35):

$$\mathbf{b} = (b_1, \ldots, b_n) \in \{0, 1\}^n. \tag{35}$$

Thus, the natural way to compare is to identify how many bits change between one descriptor and another, and this measure is precisely the Hamming distance, which is an operation that, as was evidenced computationally, is very efficient because it can be expressed via an XOR operation and a counting of ones. In contrast, in the case of the KAZE descriptor, it is a real vector of 64 components built from sums of derivatives; it is computed over a 4×4 grid and concatenated, yielding a vector as in Eq. (36), which is then normalized:

$$\mathbf{d} \in \mathbb{R}^{64}. \tag{36}$$

Therefore, in its case the distance employed for comparison that makes the most sense is the L_2 distance. Here the components are real (floats) and represent integrated gradient magnitudes. For this type of descriptors, the natural notion of similarity is "how close they are as vectors" in a Euclidean space, as defined in Eq. (37):

$$d_2(\mathbf{d}, \mathbf{d}') = \|\mathbf{d} - \mathbf{d}'\|_2 = \sqrt{\sum_{i=1}^{n} (d_i - d'_i)^2}. \tag{37}$$

D. Quantitative Performance of descriptors-pairing

It is proposed to evaluate the response of the descriptor and pairing algorithms by analyzing their performance under known geometric transformations applied to the images. For this, it is proposed to analyze the geometric transformations for which the methods are invariant, in order to make evident the pairing dynamics under modifications in rotation and scale. Additionally, it is also proposed to include a transformation that resembles a change of point of view in the image, in order to observe how the pairing precision behaves as the change becomes more significant. For this purpose, the definition of a 2D transformation matrix in OpenCV is used, and then, with that matrix and `warpAffine`, the transformation is applied to the image.

On the side of the implemented algorithm for the evaluation, it is proposed to use the ORB and KAZE detectors for the identification of a set of keypoints in each image, and afterwards, with these, to apply the transformation matrix to the positions of the keypoints of the original image and use the already defined transformation matrix to map the expected position of those keypoints in the transformed image. This

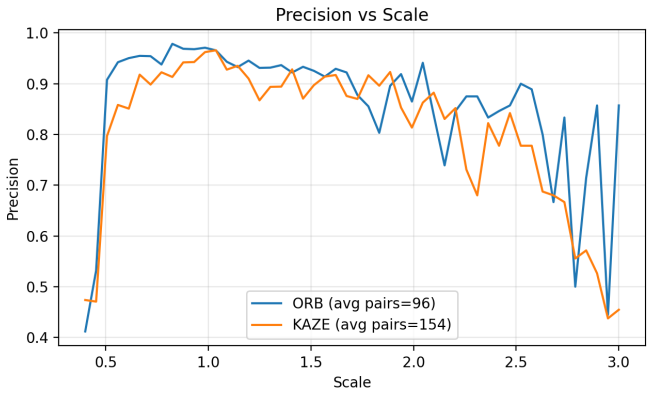

Fig. 15. Precision as a function of scale for ORB and KAZE.

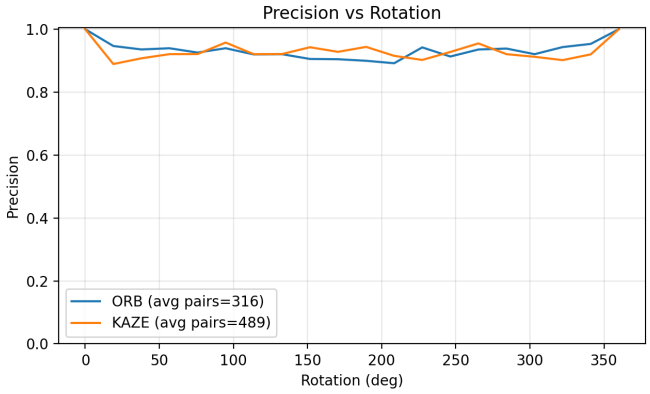

Fig. 16. Precision as a function of rotation for ORB and KAZE.

is then compared with the position of the nearest keypoint found with `ratioTest` and FLANN; the L_2 distance is measured between the expected position and the detected keypoint position, and a threshold value is proposed to identify the pairing as an error, delivering as a result of the estimation the percentage of correct pairings over the reviewed pairs.

To validate the functioning of the implemented descriptors, mainly in terms of their invariance with respect to the scale factor, and following the proposed methodology, it was proposed to analyze the behavior of pairing precision under scale variations from $\times 0.3$ to $\times 3.0$. It is observed that, under scale-only changes, the descriptor behaves as expected, being able to handle the effect of the geometric modification in feature extraction through the specific mechanisms of each algorithm; however, we note that when scale changes are below 50% of the original figure or above 200%, the quality of the results and the pairing becomes less stable and less precise, as shown in Fig. 15.

In accordance with the mathematical conception of the implemented descriptors, it is observed that when applying rotation changes to the second figure and performing pairing with the ORB and KAZE algorithms, the result is stable and additionally of high precision, as shown in Fig. 16. This

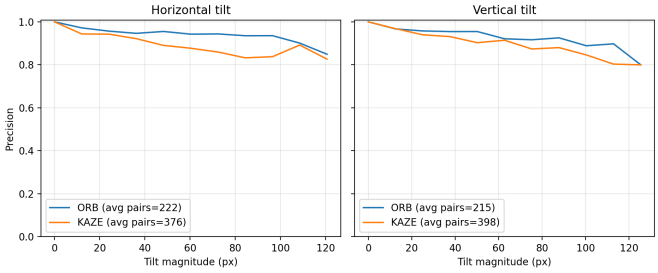

Fig. 17. Precision under viewpoint changes for ORB and KAZE (horizontal and vertical tilt).

result was expected, mainly due to the inclusion of a rotation-angle sampling method in the discrete domain with rBRIEF in the case of ORB, and in the case of KAZE through the computation and inclusion of the dominant orientation in the image.

Finally, it is proposed to evaluate the precision of the models under the modification for which, qualitatively, the largest difference in performance between the employed algorithms was observed; therefore, the implementation of variable points of view is proposed in order to evidence the capacity of the descriptors, especially the one based on nonlinear diffusion, to handle these geometric modifications for which it is not invariant. We evaluated descriptor matching robustness under perspective changes by generating synthetic viewpoint tilts via homographies. Two families of transforms were considered: a horizontal tilt (trapezoidal warping with a shorter top edge and longer bottom edge) and a vertical tilt (asymmetric compression/expansion of the left edge relative to the right). For each tilt magnitude, the original image was warped using the corresponding homography, and matching precision was computed for ORB and KAZE. Precision was measured by projecting keypoints from the original image with the same homography and counting matches whose reprojection error fell below a fixed pixel threshold. The resulting curves show how matching accuracy degrades as the viewpoint distortion increases, enabling a direct comparison of ORB and KAZE under perspective changes.

As evidenced in the results in Fig. 17, although the dynamics are similar in terms of model precision—in both cases, as the tilt magnitude in pixels in the image increases from the simulated edge and point of view, precision decreases—the phenomenon observed for these geometric modifications and previously evidenced qualitatively is maintained: given the nature of the KAZE descriptor, the number of paired keypoints identified is, on average, noticeably higher than the number of keypoints obtained under ORB. Taking into account that, although both models are capable of handling to some extent the geometric change implied by the viewpoint change, KAZE adapts better to it due to the way diffusion is performed and how the descriptor is constructed; this does not necessarily appear as a different dynamic in terms of precision as tilt increases, but it does appear as a higher average number of pairs found in the image.

REFERENCES

[1] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. Pearson, 2018.

[2] OpenCV, "cv::filter2D," *OpenCV Documentation* (OpenCV 4.x), accessed Feb. 17, 2026. [Online]. Available: https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html

[3] A. Huamán, "Feature Description," *OpenCV Documentation* (OpenCV 4.14.0-pre), accessed Feb. 6, 2026. [Online]. Available: https://docs.opencv.org/4.x/d5/dde/tutorial_feature_description.html

[4] A. Huamán, "Feature Detection," *OpenCV Documentation* (OpenCV 4.14.0-pre), accessed Feb. 6, 2026. [Online]. Available: https://docs.opencv.org/4.x/d7/d66/tutorial_feature_detection.html

[5] T. Tuytelaars and K. Mikolajczyk, "Local Invariant Feature Detectors: A Survey," *Foundations and Trends in Computer Graphics and Vision*, vol. 3, no. 3, pp. 177–280, 2007, accessed Feb. 6, 2026. [Online]. Available: https://lvelho.impa.br/ip08/reading/features.pdf

[6] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: an efficient alternative to SIFT or SURF," in *Proc. IEEE International Conference on Computer Vision (ICCV)*, Nov. 2011, doi: 10.1109/ICCV.2011.6126544, accessed Feb. 6, 2026. [Online]. Available: https://sites.cc.gatech.edu/classes/AY2024/cs4475_summer/images/ORB_an_efficient_alternative_to_SIFT_or_SURF.pdf

[7] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision – ECCV 2006* (Lecture Notes in Computer Science), vol. 3951, pp. 430–443, 2006, accessed Feb. 6, 2026. [Online]. Available: https://www.edwardrosten.com/work/rosten_2006_machine.pdf

[8] OpenCV, "cv::ORB Class Reference," *OpenCV Documentation* (OpenCV 3.4), accessed Feb. 6, 2026. [Online]. Available: https://docs.opencv.org/3.4/db/d95/classcv_1_1ORB.html

[9] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary Robust Independent Elementary Features," in *Proc. European Conference on Computer Vision (ECCV)*, 2010, accessed Feb. 6, 2026. [Online]. Available: https://www.cs.ubc.ca/~lowe/525/papers/calonder_eccv10.pdf

[10] P. F. Alcantarilla, A. Bartoli, and A. J. Davison, "KAZE Features," in *Proc. European Conference on Computer Vision (ECCV)*, 2012, accessed Feb. 6, 2026. [Online]. Available: https://www.doc.ic.ac.uk/~ajd/Publications/alcantarilla_etal_eccv2012.pdf

[11] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 7, pp. 629–639, 1990, accessed Feb. 6, 2026. [Online]. Available: https://www.sci.utah.edu/ gerig/CS7960-S2010/materials/Perona-Malik/PeronaMalik-PAMI-1990.pdf

[12] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "SURF: Speeded Up Robust Features," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008, accessed Feb. 6, 2026. [Online]. Available: https://www.cs.jhu.edu/ misha/ReadingSeminar/Papers/Bay08.pdf

[13] A. Manzanera, "Cours n°1 : Introduction — Modèles et outils fondamentaux," *Cours CSC_4MI04 — Reconnaissance d'Images*, ENSTA Paris, 2025–2026.

[14] A. Manzanera, "Cours n°2 : Caractéristiques multi-échelles," *Cours CSC_4MI04 — Reconnaissance d'Images*, ENSTA Paris, 2025–2026.