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 wide range of image-processing tasks, including smoothing, sharpening, edge detection, and feature extraction, all of which can be expressed by choosing an appropriate kernel [1].

A. The Unsharp Masking Kernel

The convolution kernel provided in the TP code is shown in Eq. (2):

$$K = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}. \qquad (2)$$

This kernel can be decomposed as the sum of the identity kernel δ (which leaves the image unchanged) and the negative discrete Laplacian $-\nabla^2$, as shown in Eq. (3):

$$K = \underbrace{\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}}_{\delta} + \underbrace{\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}}_{-\nabla^2}. \qquad (3)$$

The result of the convolution with K can therefore be written as in Eq. (4):

$$I_{\text{out}} = I * K = I - \nabla^2 I. \qquad (4)$$

The Laplacian $\nabla^2 I$ captures the second-order intensity variation at each pixel: it is positive in regions where the pixel is darker than its neighbors and negative where it is brighter. Subtracting the Laplacian from the original image means that, at an edge where the intensity transitions from low to high, the dark side is made darker and the bright side is made brighter; in flat regions the Laplacian is close to zero and the pixel barely changes. This mechanism is known as *unsharp masking* and it effectively increases the local contrast around edges and fine details without modifying the global intensity distribution of the image [1].

B. OpenCV and Matplotlib Functions

The provided code in `Convolutions.py` uses a set of OpenCV and Matplotlib functions whose behavior is relevant for interpreting the results.

a) Image reading.: `cv2.imread(path, 0)` reads the image at `path` and returns it as a NumPy array; the second argument `0` corresponds to the flag `cv2.IMREAD_GRAYSCALE`, which forces loading as a single-channel 8-bit image regardless of the original format. The result is then cast to `float64` via `np.float64()` so that convolution operations can produce values outside the $[0, 255]$ range without overflow or truncation.

b) Image copy.: `cv2.copyMakeBorder(img,0,0,0,0,cv2.B` is called with zero padding on all four sides, which in practice creates a deep copy of the image. The flag `BORDER_REPLICATE` specifies that, if padding were nonzero, the border pixels would be replicated from the nearest edge pixel rather than being set to zero or reflected.

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)	0.7097 s	5
`filter2D` (multi-thread)	0.266 ms	20
`filter2D` (1-thread)	0.274 ms	20

c) Optimized convolution.: `cv2.filter2D(img, -1, kernel)` applies the 2D convolution of `img` with `kernel`; the second argument `-1` instructs OpenCV to produce an output with the same depth as the input. Internally, this function is implemented in C++ and may use SIMD instructions and multi-threading to accelerate the computation [2].

d) Display conventions.: There is an important difference between the conventions of `cv2.imshow` and `plt.imshow`:

- `cv2.imshow` with integer arrays interprets pixel values in $\{0, \ldots, 255\}$, while with floating-point arrays it expects values in $[0, 1]$; this is why the code explicitly divides by 255 when displaying the `filter2D` result.
- `plt.imshow`, by default, *normalizes* the full range of the array to the colormap. To ensure consistent display, the code specifies `vmin=0.0, vmax=255.0` and `cmap='gray'`.

C. Direct Computation vs OpenCV *filter2D*

Two approaches are compared for computing the convolution of the grayscale test image `FlowerGarden2.png` (240×360 pixels) with the kernel K.

a) Direct method.: A nested Python loop iterates over all interior pixels (y, x) with $1 \le y \le h-2$ and $1 \le x \le w-2$. For each pixel, the value is computed explicitly as in Eq. (5):

$$v = 5\,I(y,x) - I(y-1,x) - I(y,x-1) - I(y+1,x) - I(y,x+1), \tag{5}$$

followed by clamping: $I_{\text{out}}(y,x) = \min(\max(v, 0), 255)$. Because each iteration invokes the Python interpreter and involves NumPy scalar operations, this approach carries substantial per-pixel overhead.

b) filter2D method.: A single call to `cv2.filter2D` delegates the entire computation to OpenCV's C++ backend, which applies the kernel using optimized memory access patterns, SIMD vectorization, and, when enabled, multi-threaded parallelism over image rows. This makes the computation orders of magnitude faster for the same result.

To quantify the difference rigorously, each method was benchmarked over multiple independent runs, retaining the median execution time to mitigate warm-up and scheduling effects. Two variants of `filter2D` were measured: the default multi-threaded mode and a single-thread mode obtained by calling `cv2.setNumThreads(1)` before execution. Table I reports the results.

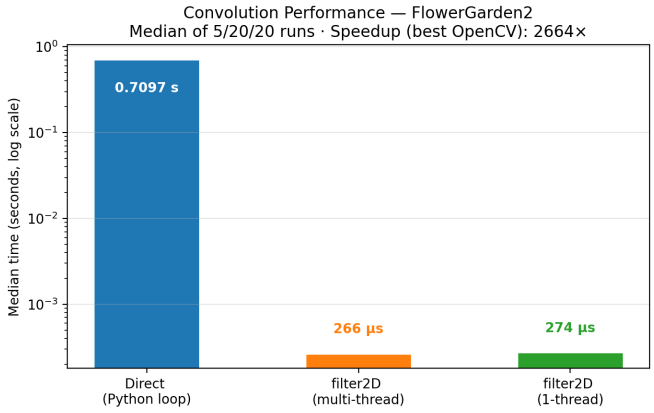

Fig. 1. Median execution time comparison (log scale) for the three convolution methods on a 240×360 image.

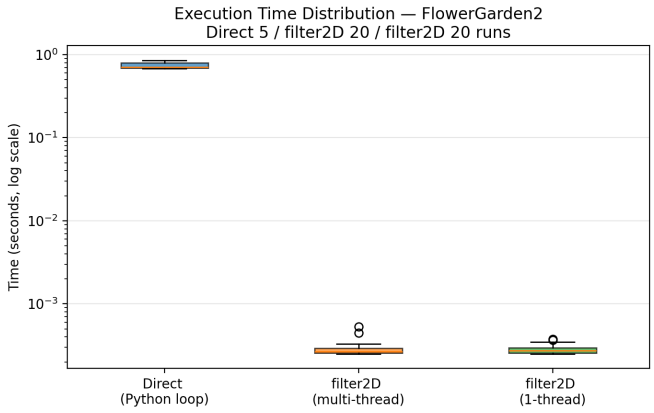

Fig. 2. Distribution of execution times across all benchmark runs for each method.

The direct Python loop is approximately $2\,664\times$ slower than the fastest OpenCV call. This gap is explained by three main factors: (i) the Python interpreter loop incurs function-call and dynamic-type-checking overhead on every pixel, whereas the C++ implementation operates on contiguous memory with compiled instructions; (ii) OpenCV can apply SIMD instructions (SSE/AVX) that process multiple pixels per clock cycle; and (iii) multi-threading distributes the work across CPU cores, although for a small image such as this one the difference between multi-thread and single-thread is marginal (the synchronization overhead nearly offsets the parallelism gain).

Fig. 1 presents the median times on a logarithmic scale, making the magnitude of the speedup visually apparent. Fig. 2 shows the distribution of all measured times as a box plot; the low variance of the `filter2D` measurements confirms the stability of the compiled implementation, while the direct method exhibits more variability due to Python's garbage collector and OS scheduling.

To further characterize the scaling behavior, the benchmark was repeated on resized versions of the image at 25 %, 50 %,

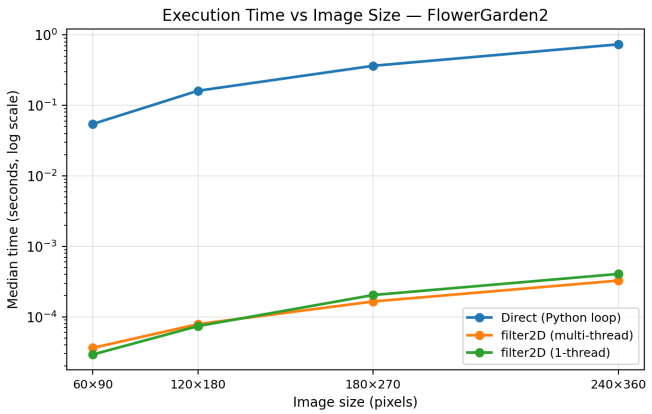

Fig. 3. Execution time as a function of image resolution (25 %, 50 %, 75 %, and 100 % of the original 240 × 360 image).

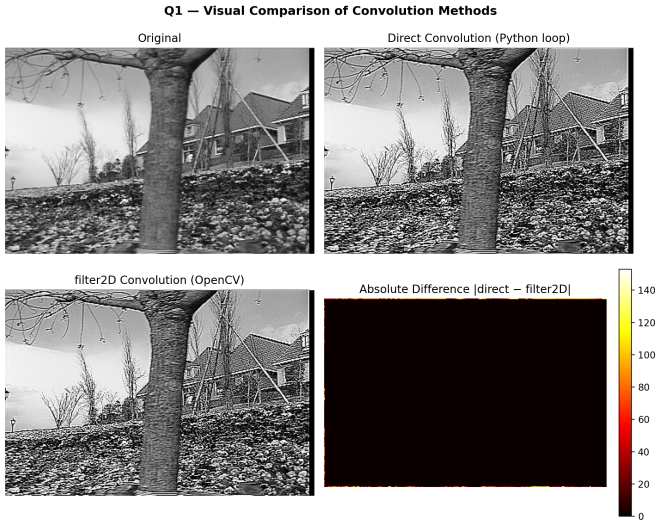

Fig. 4. Visual comparison of the convolution results: original image (top left), direct method (top right), `filter2D` (bottom left), and absolute difference amplified ×10 (bottom right).

75 %, and 100 % of the original resolution. As shown in Fig. 3, the direct method exhibits an approximately quadratic growth with the number of pixels, consistent with its $\mathcal{O}(H \cdot W \cdot k^2)$ complexity dominated by the Python loop. The `filter2D` methods remain nearly constant at these resolutions, since the image fits within the CPU cache and the constant overhead of the function call dominates the actual computation time for images of this size.

D. Visual and Numerical Comparison

Fig. 4 shows the original image alongside the results of both methods and a heatmap of the absolute pixel-wise difference. A quantitative analysis reveals that 98.9 % of the pixels are numerically identical between the two outputs, with a mean absolute difference of 0.575 and a maximum difference of 153 gray levels. The non-zero differences are concentrated at the image borders and are caused by the different padding strategies: the direct method only processes interior pixels ($1 \le y \le h-2$, $1 \le x \le w-2$) and replicates the original border values, whereas `filter2D` applies its own default border extrapolation (`BORDER_REFLECT_101`) before convolution, which produces different values at the edges. In the interior of the image, both methods produce identical results, confirming that the mathematical operation is the same and that the performance difference is purely an implementation-level phenomenon.

The sharpening effect of the kernel is clearly visible in both outputs: edges such as the flower contours and the fence structures appear with higher local contrast than in the original image, while flat regions remain largely unaffected, which is consistent with the unsharp masking interpretation derived in Eq. (4).

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 unstable 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. (6):

$$S_{p\to x} = \begin{cases} d, & I_{p\to x} \le I_p - t \\ s, & I_p - t < I_{p\to x} < I_p + t \\ b, & I_p + t \le I_{p\to x} \end{cases} \tag{6}$$

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. (7):

$$V = \max\left(\sum_{x\in S_{\text{bright}}} \left(|I_{p\to x} - I_p| - t\right), \sum_{x\in S_{\text{dark}}} \left(|I_p - I_{p\to x}| - t\right) \right). \tag{7}$$

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. (8):

$$C = |H| - k\big(\operatorname{trace}(H)\big)^2. \tag{8}$$

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. (9).

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y) \tag{9}$$

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

Then the orientation is the angle of the vector from patch center O to centroid, as in Eq. (11):

$$\theta = \operatorname{atan2}(m_{01}, m_{10}) \tag{11}$$

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. (12).

$$\tau(p; x, y) = \begin{cases} 1, & p(x) < p(y) \\ 0, & p(x) \ge p(y) \end{cases} \tag{12}$$

An n-bit descriptor can then be built as in Eq. (13).

$$f_n(p) = \sum_{i=1}^{n} 2^{i-1}\, \tau(p; x_i, y_i) \tag{13}$$

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. (14).

$$S = \begin{pmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \end{pmatrix} \tag{14}$$

These locations are rotated according to Eq. (15).

$$S_\theta = R_\theta S \tag{15}$$

Finally, the descriptor is computed using the rotated test coordinates, as expressed in Eq. (16).

$$g_n(p, \theta) := f_n(p)\big|_{(x_i, y_i)\in S_\theta} \tag{16}$$

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. (17):

$$\operatorname{Ham}(d_1, d_2) = \sum_{i=1}^{256} \mathbf{1}[d_{1,i} \ne d_{2,i}]\,. \tag{17}$$

This can be computed efficiently as in Eq. (18):

$$\operatorname{Ham}(d_1, d_2) = \operatorname{popcount}(d_1 \oplus d_2). \tag{18}$$

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. (19):

$$\frac{\partial L}{\partial t} = \operatorname{div}\big(c(x, y, t)\, \nabla L\big). \tag{19}$$

Here, ∇L points toward where the image changes the fastest (edges = large gradient). The diffusion "flow" can be seen as Eq. (20):

$$\mathbf{J} = -c\, \nabla L, \tag{20}$$

then $\operatorname{div}(\mathbf{J})$ 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. (21):

$$c(x, y, t) = g(\|\nabla L(x, y, t)\|)\,. \tag{21}$$

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. (22) and Eq. (23):

$$g_1(s) = \exp\left(-\frac{s^2}{k^2}\right), \tag{22}$$

$$g_2(s) = \frac{1}{1 + \frac{s^2}{k^2}}. \tag{23}$$

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. (24):

$$L_{\text{Hessian}} = \sigma^2 \left(L_{xx} L_{yy} - L_{xy}^2\right). \tag{24}$$

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. (24). 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 the keypoint [10]. In each subregion it sums a vector of the form shown in Eq. (25):

$$d_v = \left(\sum L_x,\ \sum L_y,\ \sum |L_x|,\ \sum |L_y|\right). \tag{25}$$

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. (26), together with a descriptor (64D or extended depending on the implementation).

$$(x, y, \sigma, \theta) \tag{26}$$

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

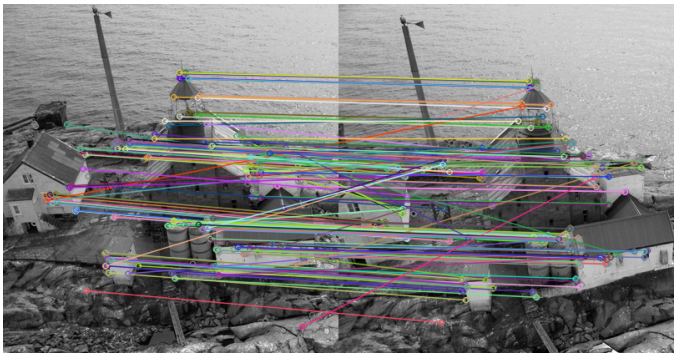

Fig. 5. Cross-check matching results using ORB.

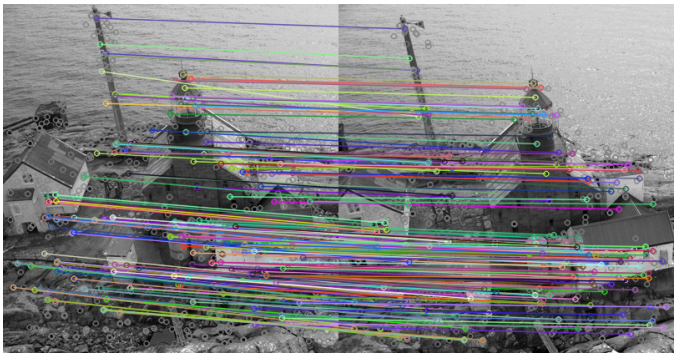

Fig. 6. Cross-check matching results using KAZE.

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. 5–6 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 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. 7 shows ORB with ratio test, and Fig. 8 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

TABLE II
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

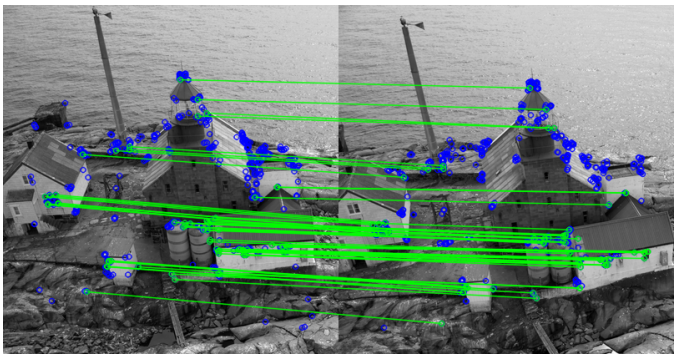

Fig. 7. Ratio-test matching results using ORB.

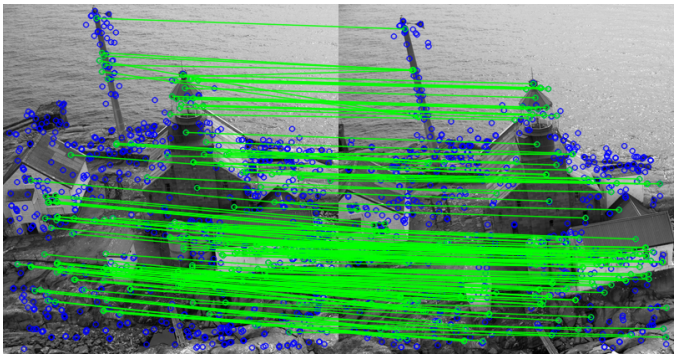

Fig. 8. Ratio-test matching results using KAZE.

TABLE III
MEASURED RUNTIME FOR RATIO-TEST MATCHING.

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

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 II.

We also include the FLANN matching results for visual comparison. Fig. 9 shows ORB with FLANN, and Fig. 10 shows KAZE with FLANN.

When analyzing the results, mainly the matching between

TABLE IV
MEASURED RUNTIME FOR FLANN MATCHING.

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

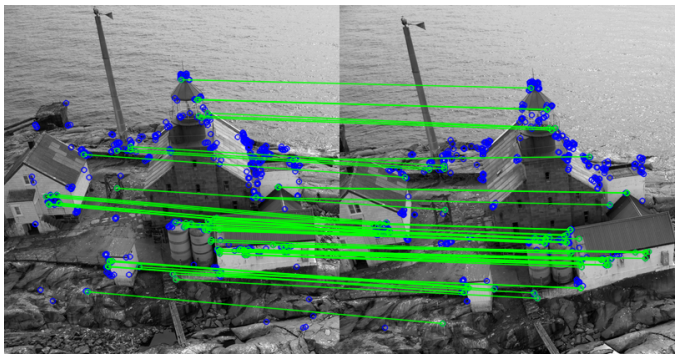

Fig. 9. FLANN matching results using ORB.

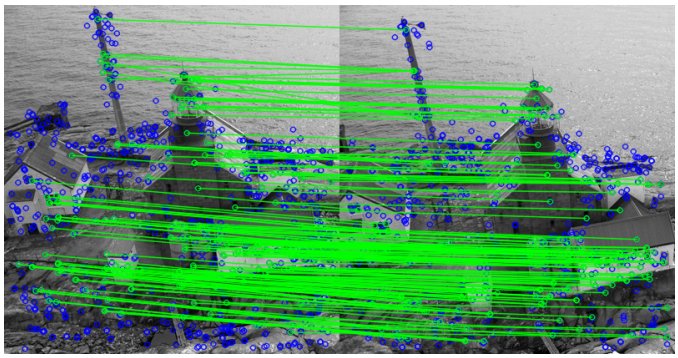

Fig. 10. FLANN matching results using KAZE.

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 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. 7 and 8, 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. (27):

$$b_i = \mathbf{1}[I(x_i) < I(y_i)] \in \{0, 1\}. \tag{27}$$

Then the complete descriptor is given by Eq. (28):

$$\mathbf{b} = (b_1, \ldots, b_n) \in \{0, 1\}^n. \tag{28}$$

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. (29), which is then normalized:

$$\mathbf{d} \in \mathbb{R}^{64}. \tag{29}$$

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. (30):

$$d_2(\mathbf{d}, \mathbf{d}') = \|\mathbf{d} - \mathbf{d}'\|_2 = \sqrt{\sum_{i=1}^{n} (d_i - d'_i)^2}. \tag{30}$$

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 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. 11.

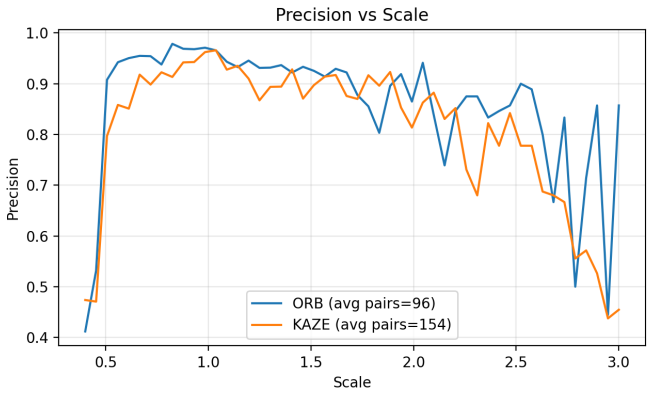

Fig. 11. Precision as a function of scale for ORB and KAZE.

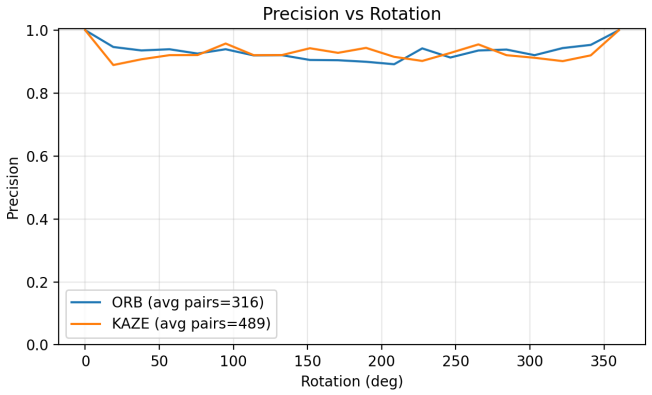

Fig. 12. Precision as a function of rotation for ORB and KAZE.

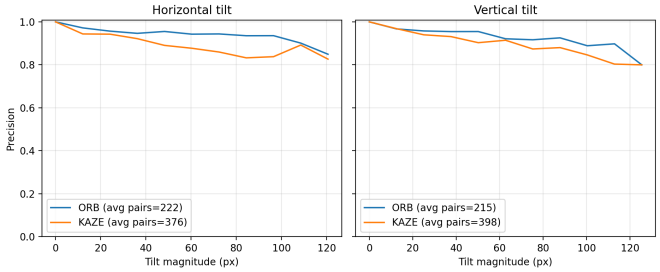

Fig. 13. Precision under viewpoint changes for ORB and KAZE (horizontal and vertical tilt).

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. 12. This 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. 13, 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#ga27c049795ce870216ddfb366086b5a04

[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