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

1st Santiago Florido Gomez
Ingénieur Degree Programme STIC
ENSTA Paris
Paris, France
santiago.florido@ensta.fr

2nd Javier Andres Tarazona Jimenez
Ingénieur Degree Programme STIC
ENSTA Paris
Paris, France
javier-andres.tarazona@ensta.fr

3rd Santiago Florido Gomez
Ingénieur Degree Programme STIC
ENSTA Paris
Paris, France
santiago.florido@ensta.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. 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 [1]; 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 [2]; 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 [3].

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 [4].

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. (1):

$$S_{p\to x} = \begin{cases} d, & I_{p\to x} \le I_p - t \\ s, & I_p - t < I_{p\to x} < I_p + t \\ b, & I_p + t \le I_{p\to x} \end{cases} \tag{1}$$

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 [5].

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 [5]. One (efficient) definition given is in Eq. (2):

$$V = \max\left(\sum_{x\in S_{\text{bright}}} \left(|I_{p\to x} - I_p| - t\right), \sum_{x\in S_{\text{dark}}} \left(|I_p - I_{p\to x}| - t\right)\right). \tag{2}$$

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 [4].

A standard Harris measure uses the second-moment (structure tensor) matrix H and response in Eq. (3):

$$C = |H| - k\big(\mathrm{trace}(H)\big)^2. \tag{3}$$

The selection between a score type, more stable with Harris, or faster but slightly less stable with FAST_SCORE, is made available by OpenCV [6].

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. (4).

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y) \tag{4}$$

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}\right) \tag{5}$$

Then the orientation is the angle of the vector from patch center O to centroid, as in Eq. (6):

$$\theta = \mathrm{atan2}(m_{01}, m_{10}) \tag{6}$$

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. (7).

$$\tau(p; x, y) = \begin{cases} 1, & p(x) < p(y) \\ 0, & p(x) \geq p(y) \end{cases} \tag{7}$$

An n-bit descriptor can then be built as in Eq. (8).

$$f_n(p) = \sum_{i=1}^{n} 2^{i-1}\, \tau(p; x_i, y_i) \tag{8}$$

It focuses on binary strings given the inherent ease of comparing them using Hamming distance rather than L_2 distances in vectors [4]. 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. (9).

$$S = \begin{pmatrix} x_1 & \cdots & x_n \\ y_1 & \cdots & y_n \end{pmatrix} \tag{9}$$

These locations are rotated according to Eq. (10).

$$S_\theta = R_\theta S \tag{10}$$

Finally, the descriptor is computed using the rotated test coordinates, as expressed in Eq. (11).

$$g_n(p, \theta) := f_n(p)\big|_{(x_i, y_i) \in S_\theta} \tag{11}$$

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. (12):

$$\mathrm{Ham}(d_1, d_2) = \sum_{i=1}^{256} \mathbf{1}[d_{1,i} \neq d_{2,i}]\,. \tag{12}$$

This can be computed efficiently as in Eq. (13):

$$\mathrm{Ham}(d_1, d_2) = \mathrm{popcount}(d_1 \oplus d_2). \tag{13}$$

BRIEF emphasizes this efficiency, and ORB notes SSE popcount optimizations in their matching implementation [7].

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 [8].

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. (14):

$$\frac{\partial L}{\partial t} = \operatorname{div}\big(c(x,y,t)\,\nabla L\big). \tag{14}$$

Here, ∇L points toward where the image changes the fastest (edges = large gradient). The diffusion "flow" can be seen as Eq. (15):

$$\mathbf{J} = -c\,\nabla L, \tag{15}$$

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 [8]. This is precisely the key, because c depends on the gradient, as expressed in Eq. (16):

$$c(x,y,t) = g(\|\nabla L(x,y,t)\|)\,. \tag{16}$$

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 [9]. This is ensured by the two typical forms of g given in Eq. (17) and Eq. (18):

$$g_1(s) = \exp\!\left(-\frac{s^2}{k^2}\right), \tag{17}$$

$$g_2(s) = \frac{1}{1+\frac{s^2}{k^2}}. \tag{18}$$

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 [8].

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. (19):

$$L_{\text{Hessian}} = \sigma^2\left(L_{xx}L_{yy} - L_{xy}^2\right). \tag{19}$$

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. (19). 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 [10].

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 [8]. In each subregion it sums a vector of the form shown in Eq. (20):

$$d_v = \left(\sum L_x,\ \sum L_y,\ \sum |L_x|,\ \sum |L_y|\right). \tag{20}$$

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

$$(x, y, \sigma, \theta) \tag{21}$$

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. 1–2 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

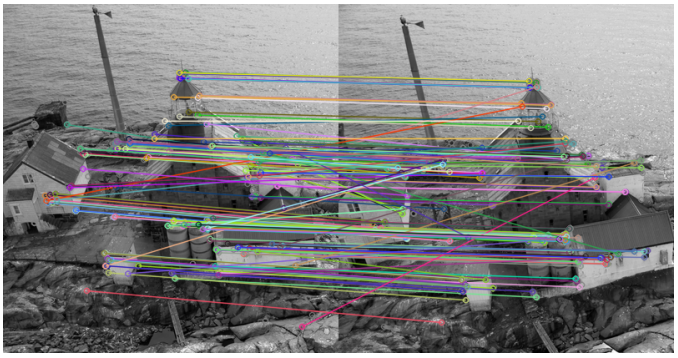


Fig. 1. Cross-check matching results using ORB.

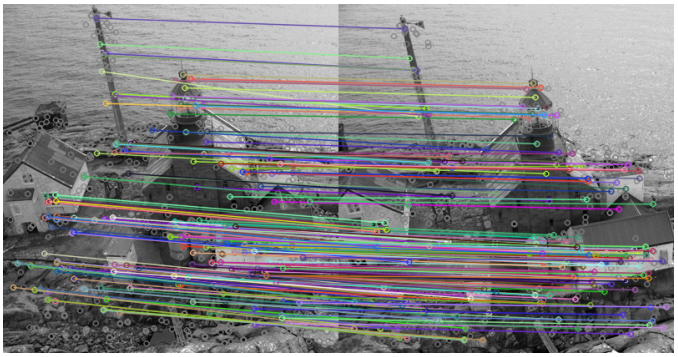


Fig. 2. Cross-check matching results using KAZE.

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.

TABLE I
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

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. 3 shows ORB with ratio test, and Fig. 4 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 I.

We also include the FLANN matching results for visual comparison. Fig. 5 shows ORB with FLANN, and Fig. 6 shows KAZE with FLANN.

When analyzing the results, mainly the matching between the two images using ORB and cross-check matching, it is

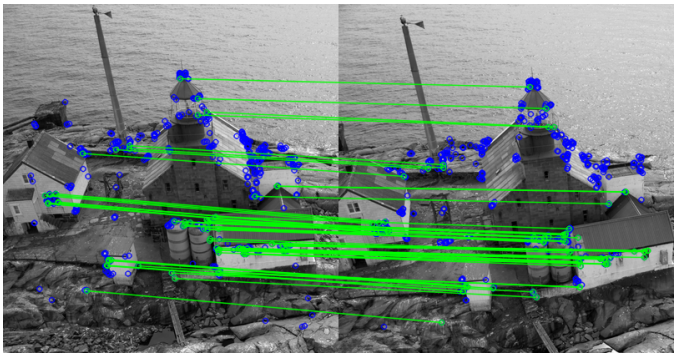


Fig. 3. Ratio-test matching results using ORB.

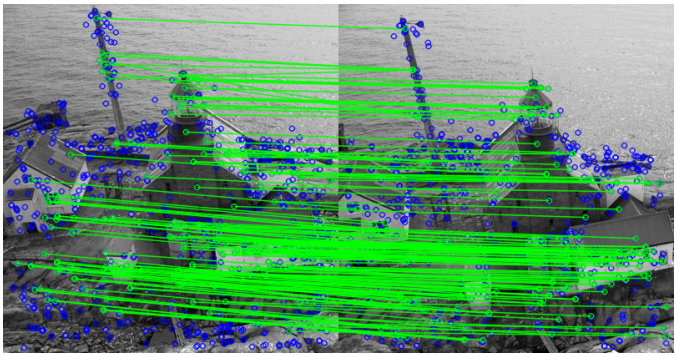


Fig. 4. Ratio-test matching results using KAZE.

TABLE II
MEASURED RUNTIME FOR RATIO-TEST MATCHING.

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

TABLE III
MEASURED RUNTIME FOR FLANN MATCHING.

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

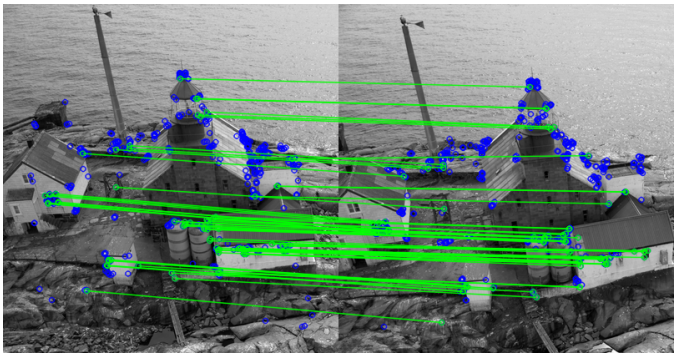


Fig. 5. FLANN matching results using ORB.

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.

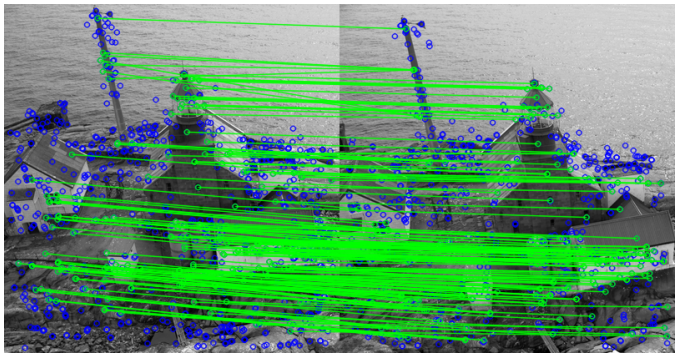


Fig. 6. FLANN matching results using KAZE.

As shown in Figs. 3 and 4, 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. (22):

$$b_i = \mathbf{1}[I(x_i) < I(y_i)] \in \{0, 1\}. \tag{22}$$

Then the complete descriptor is given by Eq. (23):

$$\mathbf{b} = (b_1, \ldots, b_n) \in \{0, 1\}^n. \tag{23}$$

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. (24), which is then normalized:

$$\mathbf{d} \in \mathbb{R}^{64}. \tag{24}$$

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. (25):

$$d_2(\mathbf{d}, \mathbf{d}') = \|\mathbf{d} - \mathbf{d}'\|_2 = \sqrt{\sum_{i=1}^{n} (d_i - d'_i)^2}. \tag{25}$$

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.

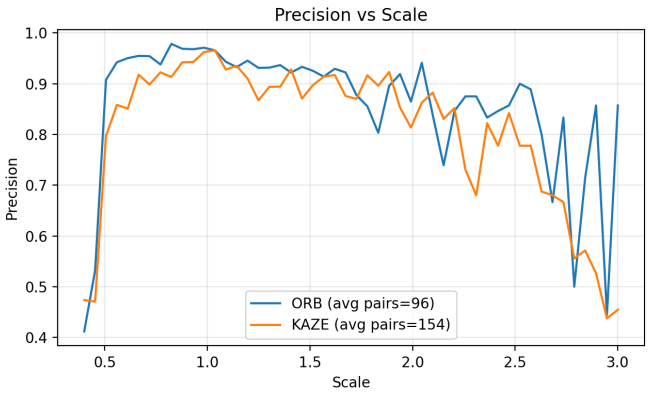


Fig. 7. Precision as a function of scale for ORB and KAZE.

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. 7.

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. 8. This result was expected, mainly due to the inclusion of a rotation-

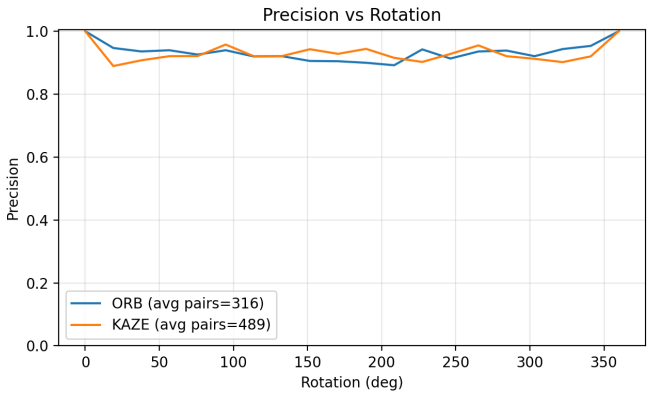


Fig. 8. Precision as a function of rotation for ORB and KAZE.

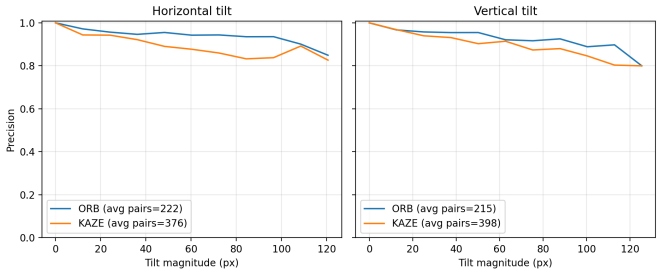


Fig. 9. Precision under viewpoint changes for ORB and KAZE (horizontal and vertical tilt).

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. 9, 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.

II. Détecteurs

Dans le contexte de l'analyse d'images et de vidéos, il est souvent nécessaire de détecter des points d'intérêt, par exemple des bords, des coins, etc. C'est important pour des tâches d'appariement : comparer les descripteurs de deux images et obtenir des correspondances, estimer des transformations d'images (alignement), faire du suivi dans des vidéos à partir de ces points d'intérêt, et de l'odométrie visuelle pour estimer le mouvement de la caméra.

A. *Fonction d'intérêt d'Harris*

C'est ici qu'intervient la fonction (ou le détecteur) de Harris, qui est idéal quand la rapidité et la stabilité sont nécessaires. Cependant, lors de forts changements d'échelle, par exemple quand un objet apparaît beaucoup plus grand ou plus petit entre différents *frames*, le détecteur de Harris est moins efficace.

La fonction d'intérêt de Harris se base sur le fait d'assigner une valeur à chaque pixel. Cette valeur mesure à quel point ce pixel est une *intersection*, c'est-à-dire un *coin*.

Dire qu'un pixel est un coin signifie qu'on cherche à quel point il ressemble à l'intersection de deux bords. Autrement dit, à quel point ce pixel correspond à un endroit où l'image change fortement dans deux directions perpendiculaires. C'est utile, parce qu'avec une petite fenêtre de pixels (la fenêtre W), on peut estimer à quel point ce qu'on observe change au niveau de la structure.

Intuitivement, si on déplace légèrement la fenêtre W autour d'un pixel, on observe :

- **Zone plane (sans texture)** : la fenêtre change très peu $\rightarrow$ ce n'est pas un coin.
- **Bord** : la fenêtre change peu en se déplaçant *le long* du bord, mais change beaucoup en le *traversant*.
- **Coin** : la fenêtre change beaucoup dans presque n'importe quelle direction.

Ce que fait Harris, c'est utiliser les gradients I_x et I_y pour savoir à quel point l'intensité change selon x et selon y. Si autour du pixel il y a un changement fort dans une seule direction, on est plutôt sur un bord ; mais si les changements forts sont dans deux directions, on parle d'un coin. Ainsi, Θ,

la réponse de Harris, est définie pixel par pixel : plus Θ est grande, plus le pixel est un *coin*.

Dans le détecteur de Harris, la fonction d'intérêt se définit comme :

$$\Theta = R = \det(M) - k\,(\mathrm{trace}(M))^2$$

Où k est une valeur empirique de pénalisation : plus elle est grande, plus le critère est strict, donc moins de coins sont détectés. M est la matrice de structure obtenue à partir des gradients I_x et I_y, calculés après un lissage gaussien de l'image, avec le paramètre σ qui contrôle le niveau de lissage (réduction du bruit et des détails fins). La matrice M se définit par :

$$M(x,y) =$$

$$= \sum_{(u,v)\in W} w(u,v) \begin{pmatrix} I_x(u,v)^2 & I_x(u,v)I_y(u,v) \\ I_x(u,v)I_y(u,v) & I_y(u,v)^2 \end{pmatrix}$$

Où $w(u,v)$ est une fonction de poids (pondération) qui donne plus d'importance aux pixels proches du centre de la fenêtre W. Ainsi, la somme se fait sur une fenêtre autour du pixel analysé.

- I_x^2 mesure à quel point l'image change dans la direction horizontale.
- I_y^2 mesure à quel point l'image change dans la direction verticale.
- $I_x I_y$ mesure la corrélation entre ces deux variations.

Et concernant les valeurs propres de M : si les deux sont grandes, alors R est grand et le pixel est un *coin*. Si l'une est grande et l'autre petite, alors R devient négatif et le pixel correspond à un *bord*. Si les deux sont petites, alors R est petit et le pixel est une zone plane.

Enfin, le déterminant de M correspond au produit des valeurs propres, tandis que la trace correspond à leur somme. Mais la trace ne distingue pas bien les bords des coins, c'est pour cela qu'on la pénalise (avec le terme en k). La trace mesure surtout le changement global, alors que le déterminant met mieux en évidence un vrai coin.

Par ailleurs, cette fonction d'intérêt est calculée à une seule échelle : on utilise un seul niveau de lissage et une seule taille de fenêtre pour calculer M.

B. Dilatation Morphologique

Dans le traitement d'images, on cherche souvent à obtenir une valeur représentative d'un voisinage. Dans le cas de la dilatation morphologique, cette valeur correspond, pour chaque pixel, au maximum dans ce voisinage. Ce voisinage est défini par un élément structurant, par exemple une fenêtre 3×3.

Dans le cas de Harris, cette dilatation morphologique est utilisée pour détecter les maxima locaux puis faire la suppression des non-maxima. Autrement dit, l'objectif est de garder les pics (les coins) les plus forts.

Dans le code de `Harris.py`, la variable `se` définit la fenêtre/le voisinage comme une matrice de 1 de taille `d_maxloc`, avec `d_maxloc = 3` :

```
d_maxloc = 3
se = np.ones(
    (d_maxloc, d_maxloc), np.uint8
)
```

Ensuite, l'instruction suivante fait que chaque pixel prend la valeur la plus grande de sa fenêtre :

```
Theta_dil = cv2.dilate(Theta, se)
```

Après, avec l'instruction suivante, on réalise une comparaison pixel par pixel :

```
Theta_maxloc[Theta < Theta_dil] = 0.0
```

Si la valeur originale est différente de la valeur dilatée, alors ce pixel n'était pas un maximum local, donc sa valeur est supprimée (mise à zéro). Dans le cas contraire, la valeur est conservée car c'est bien un maximum local.

Enfin, on applique un seuil relatif pour éliminer les maxima trop faibles :

```
Theta_maxloc[
    Theta < seuil_relatif * Theta.max()
] = 0.0
```

C. Résultats

Le script `Harris.py` a été exécuté 50 fois avec `-stats` sur l'image `Graffiti0.png`. Les mesures suivantes ont été obtenues. Les résultats ci-dessous ont été mesurés avec les paramètres suivants :

```
SUM_WINDOW_SIZE = 5
HARRIS_K = 0.04
MAXLOC_NEIGHBORHOOD_SIZE = 3
RELATIVE_THRESHOLD = 0.01
```

- Dimension de l'image (niveau de gris) : 320×400.
- Type de l'image (niveau de gris) : `float64`.
- Dimension de l'image (couleur) : $320 \times 400 \times 3$.
- Type de l'image (couleur) : `uint8`.

TABLE IV
STATISTIQUES SUR 50 EXÉCUTIONS DU DÉTECTEUR DE HARRIS

Métrique	Valeur
Temps moyen [s]	0.003224673
Variance du temps [s^2]	0.000002632168
Écart-type du temps [s]	0.001622396
Cycles/pixel moyen [cpp]	25.192756
Variance cycles/pixel [cpp^2]	160.654768461
Écart-type cycles/pixel [cpp]	12.674966

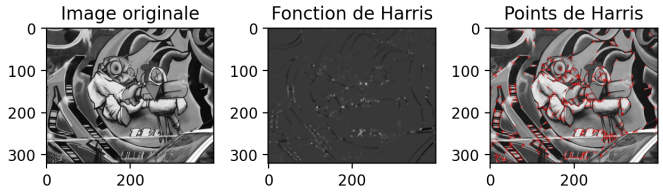
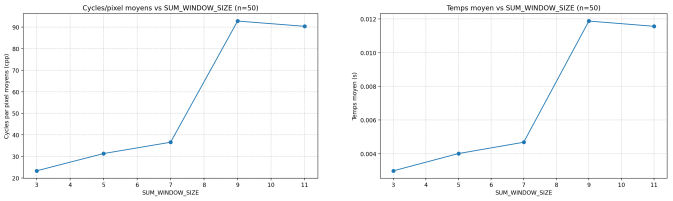


Fig. 10. Visualisation des points d'intérêt détectés par Harris sur `Graffiti0.png`.

D. Analyse Paramétrique du Détecteur de Harris

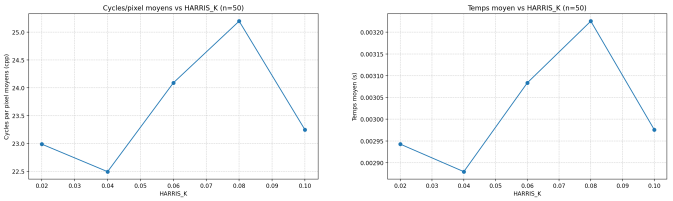
Cette étude paramétrique a été réalisée avec la commande `python Harris.py -stats 50 -plots`. L'échantillon correspond à 50 exécutions du calcul de Harris pour chaque valeur testée. Pendant chaque balayage, les autres paramètres restent fixés à : `SUM_WINDOW_SIZE = 5`, `HARRIS_K = 0.04`, `MAXLOC_NEIGHBORHOOD_SIZE = 3`, `RELATIVE_THRESHOLD = 0.01`.



(a) Cycles/pixel vs `SUM_WINDOW_SIZE` **(b)** Temps moyen vs `SUM_WINDOW_SIZE`

Fig. 11. Impact de `SUM_WINDOW_SIZE` sur le coût de calcul.

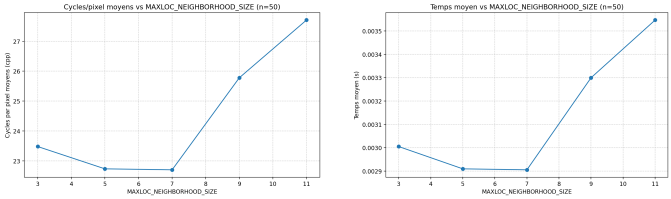
Quand `SUM_WINDOW_SIZE` augmente, le temps moyen et les cycles/pixel augmentent nettement, car la convolution locale est plus coûteuse. Du point de vue détection, une grande fenêtre stabilise la mesure de structure (moins sensible au bruit), mais tend à lisser les détails fins et à réduire le nombre de coins faibles détectés. À l'inverse, une petite fenêtre capte plus de micro-variations (plus de coins potentiels), au prix d'une sensibilité plus forte au bruit et donc d'un risque accru de faux positifs.



(a) Cycles/pixel vs `HARRIS_K` **(b)** Temps moyen vs `HARRIS_K`

Fig. 12. Impact de `HARRIS_K` sur le coût de calcul.

La variation de `HARRIS_K` modifie peu le coût (même pipeline de calcul, mêmes opérateurs principaux), ce qui est cohérent avec les courbes de temps/CPP. En revanche, l'effet sur la sélection des coins est important : un `HARRIS_K` faible est plus permissif (plus de points retenus, y compris des points ambigus proches des bords), tandis qu'un `HARRIS_K` élevé renforce la pénalisation de la trace, donc la sélection devient plus stricte et le nombre de coins tend à diminuer, avec une meilleure stabilité géométrique.



(a) Cycles/pixel vs taille du voisinage NMS **(b)** Temps moyen vs taille du voisinage NMS

Fig. 13. Impact de la taille du voisinage NMS sur le coût de calcul.

Quand `MAXLOC_NEIGHBORHOOD_SIZE` augmente, le coût augmente de manière modérée, car la dilatation/non-max suppression parcourt un voisinage plus large. Ce paramètre agit surtout sur la densité finale des points : un voisinage grand rend la suppression non-maxima plus agressive (moins de coins, mieux espacés, moins redondants), tandis qu'un voisinage petit laisse passer plus de maxima locaux proches les uns des autres (plus de coins, mais plus de redondance spatiale).

En synthèse, il existe un compromis classique entre sensibilité et robustesse : augmenter la sensibilité aux détails produit souvent plus de coins, mais aussi plus de bruit et de faux coins ; renforcer la robustesse réduit les faux positifs, mais peut éliminer des coins faibles utiles pour certaines scènes. Pour ce jeu d'images, la configuration de base (`SUM_WINDOW_SIZE` = 5, `HARRIS_K` = 0.04, voisinage NMS = 3) reste un compromis raisonnable.

E. Calcul de Harris sur Plusieurs Échelles

Dans l'implémentation actuelle de `Harris.py`, le détecteur est mono-échelle : on utilise un lissage gaussien fixe (`sigma = 1.0`) ainsi qu'une fenêtre de sommation fixe pour construire la matrice de structure. Cette approche fonctionne bien pour une échelle donnée, mais elle perd en robustesse lorsque la taille apparente des objets varie entre les images.

Remarque. Le paramètre σ représente le niveau de lissage gaussien : plus σ est grand, plus le lissage est fort, et plus les détails fins de l'image sont atténués (voire effacés).

Pour réaliser le calcul de Harris sur plusieurs échelles, on travaille dans l'espace d'échelle (x, y, σ) :

1) Définir un ensemble d'échelles $\sigma_1, \sigma_2, \ldots, \sigma_n$ (ou construire une pyramide gaussienne).
2) Pour chaque σ_i, calculer I_x, I_y, la matrice M_{σ_i}, puis la réponse de Harris R_{σ_i}.
3) Normaliser les réponses entre échelles pour les rendre comparables.
4) Appliquer une suppression de non-maxima en 2D à chaque échelle, puis l'étendre en 3D : un point est conservé s'il est maximal dans son voisinage spatial et aussi par rapport à σ_{i-1}, σ_i et σ_{i+1}.

5) Appliquer un seuil, puis reprojeter les points dans l'image originale en conservant leur échelle caractéristique.

Les principaux compromis sont les suivants :

- Plus d'échelles : meilleure robustesse aux changements de taille, mais coût de calcul plus élevé.
- Petites échelles : plus de détails et potentiellement plus de coins, mais aussi davantage de sensibilité au bruit.
- Grandes échelles : détection plus stable, mais risque de perdre des coins fins ou faibles.

Finalement, le lien avec la distance minimale r entre points d'intérêt est le suivant. Pour garantir que deux points détectés restent séparés d'au moins r pixels, on peut utiliser une suppression de non-maxima avec un rayon r, ou une ANMS (*Adaptive Non-Maximum Suppression*). Cela réduit la redondance spatiale et améliore la répartition des coins dans l'image.

Au lieu d'utiliser un rayon fixe identique partout, l'ANMS attribue à chaque coin candidat un rayon adaptatif r_i, défini comme la distance au coin plus fort le plus proche. C'est-à-dire que

- un coin très fort, isolé, obtient un grand rayon r_i ;
- un coin faible situé près d'un coin plus fort obtient un petit rayon.

Ensuite, on trie les candidats selon r_i et on conserve les points avec les plus grands rayons. On garde ainsi des coins à la fois fiables (bonne réponse Harris) et bien répartis spatialement, ce qui est souvent préférable à une sélection locale très dense dans une petite zone de l'image.

REFERENCES

[1] 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

[2] 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

[3] 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

[4] 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

[5] 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

[6] 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

[7] 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

[8] 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

[9] 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

[10] 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