

CSC-4MI04 - Reconnaissance d'Images

TP1 : Détection et Appariement de Points Caractéristiques

QUINTELLA PINTO Alfredo
GOMES BIAGGI Naomy
MOCZYDLOWER Carolina

Encadrant :

MANZANERA Antoine
HARIAT Marwane

Introduction

La bibliothèque OpenCV constitue un outil essentiel dans le domaine de la vision par ordinateur, offrant un ensemble riche de fonctions dédiées au traitement et à l'analyse d'images et de vidéos. Elle permet de mettre en œuvre efficacement de nombreuses opérations fondamentales, telles que le filtrage, la détection de contours, l'extraction de caractéristiques ou encore l'appariement de points d'intérêt. Grâce à ses implémentations optimisées et à sa compatibilité avec plusieurs langages, dont Python, OpenCV facilite l'expérimentation et le prototypage d'algorithmes, aussi bien dans un cadre académique qu'industriel. Son utilisation permet ainsi d'explorer concrètement les concepts théoriques de la vision par ordinateur tout en évaluant leurs performances sur des données réelles.

Ce travail pratique a pour objectif de nous familiariser avec l'utilisation de la bibliothèque OpenCV en Python dans le contexte du traitement d'images, ainsi que d'explorer différentes méthodes de détection et d'appariement de points caractéristiques. Ces techniques jouent un rôle central en vision par ordinateur, notamment pour des applications telles que la reconnaissance d'objets, la reconstruction 3D, le suivi de mouvement ou encore l'alignement d'images.

Le problème étudié se décompose en plusieurs étapes fondamentales : la détection de points d'intérêt afin de réduire l'espace de représentation, la description de ces points à l'aide de caractéristiques pertinentes, la définition d'une métrique permettant de comparer ces descripteurs, et enfin la recherche efficace de correspondances entre images.

À travers des expérimentations pratiques réalisées à partir des scripts fournis, nous analysons le comportement de différentes approches, évaluons leurs performances et discutons leurs avantages et limites. L'objectif est ainsi de développer une compréhension à la fois théorique et expérimentale des méthodes d'appariement de structures locales dans les images.

Prérequis et environnement de travail

Les codes utilisés dans ce travail sont basés sur ceux mis à disposition sur ([Lien Codes](#)) et les images d'expérimentation sur ([Lien Images](#)). Tous les codes fournis sont des scripts Python3 éditables et exécutables. L'ensemble des contenus produits par notre groupe se trouve dans le dépôt Github ([Lien Github](#))

1 Format d'images et Convolutions

La convolution est une opération fondamentale en traitement d'image permettant d'appliquer un filtre spatial à une image. Elle consiste à faire glisser un noyau (ou masque) sur l'image et à calculer, pour chaque pixel, une combinaison linéaire pondérée des pixels voisins.

Mathématiquement, pour une image I et un noyau K , la convolution s'écrit :

$$(I * K)(x, y) = \sum_i \sum_j K(i, j) I(x - i, y - j).$$

Selon le choix du noyau, la convolution peut produire différents effets :

- lissage (réduction du bruit),
- détection de contours,

- accentuation (sharpening),
- extraction de gradients,
- filtrage passe-bas ou passe-haut.

Ainsi, la convolution agit comme un opérateur local transformant l'image en fonction de la structure de son voisinage.

Traitements de base en traitement d'image

Avant ou après l'application d'un filtre, plusieurs précautions sont nécessaires afin d'obtenir des résultats corrects :

Gestion des bords. Les filtres utilisent des pixels voisins ; les pixels situés sur les bords de l'image n'ont pas de voisinage complet. Il est donc nécessaire d'étendre l'image artificiellement (réPLICATION, réflexion, zéros, etc.). Le choix influence la qualité des contours près des frontières.

Type numérique et dynamique. Les calculs de convolution ou de dérivation peuvent produire des valeurs négatives ou supérieures à 255. L'utilisation d'un type flottant évite la saturation pendant les calculs. Une normalisation ou conversion peut ensuite être nécessaire pour l'affichage.

Normalisation et facteur d'échelle s . Certains noyaux sont multipliés par un facteur d'échelle s afin de contrôler l'amplitude de la réponse du filtre :

$$K' = s \cdot K.$$

Ce facteur ne modifie pas la structure du filtrage mais ajuste l'intensité de la réponse :

- Si s est trop grand : les contours deviennent très marqués, mais le bruit et les artefacts peuvent être amplifiés.
- Si s est trop petit : les contours deviennent faibles et certaines structures peuvent disparaître.

Dans les filtres de gradient (comme Sobel), ce facteur permet de conserver une amplitude cohérente tout en évitant des valeurs excessives.

Lien entre convolution et détection de contours

Les contours correspondent à des variations rapides d'intensité, c'est-à-dire des hautes fréquences spatiales. Les filtres dérivatifs (gradient, Sobel, Laplacien) sont obtenus par convolution et mettent en évidence ces variations.

La norme du gradient

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

mesure l'intensité des transitions indépendamment de la direction, ce qui permet de détecter les structures principales de l'image.

Conclusion

La convolution constitue l'outil central du traitement d'image. Le résultat dépend fortement :

- du choix du noyau,
- de la gestion des bords,
- du type numérique utilisé,
- du facteur d'échelle appliqué.

Une bonne maîtrise de ces éléments permet de contrôler précisément le comportement des filtres et d'obtenir des résultats robustes pour l'analyse d'image, la détection de contours et l'extraction de caractéristiques.

Question 1

Expérimenter le code de convolution fourni en exemple dans `Convolutions.py`. Observer la différence entre le calcul direct par balayage du tableau 2d et le calcul utilisant la fonction `filter2d` d'OpenCV. Déchiffrer les fonctions OpenCV utilisées pour la lecture et la copie d'images, ainsi que la fonction Matplotlib utilisée pour l'affichage.

Dans cette première partie, nous avons testé le script `Convolutions.py` afin de comparer deux méthodes d'application d'un filtre de convolution : une implémentation directe utilisant des boucles Python et la fonction optimisée `filter2D` d'OpenCV.

L'image est d'abord lue en niveaux de gris avec `cv2.imread(...,0)`, puis convertie en `float64` afin d'éviter les erreurs d'arrondi et les saturations lors des calculs numériques. Ses dimensions sont ensuite extraites via `shape`. Les bords sont gérés grâce à `cv2.copyMakeBorder` avec l'option `BORDER_REPLICATE`, qui consiste à prolonger l'image en recopiant les pixels du bord. Cette stratégie permet d'appliquer le filtre sur toute l'image sans introduire d'artefacts artificiels. Lors de l'affichage avec `matplotlib`, l'argument `cmap='gray'` est utilisé afin de visualiser correctement l'image en niveaux de gris, sans palette de fausses couleurs.



FIGURE 1 – Image originale utilisée pour l'expérience de convolution.

Dans la méthode directe, chaque pixel est recalculé à partir d'une combinaison linéaire de ses voisins :

$$I'(x, y) = 5I(x, y) - I(x - 1, y) - I(x + 1, y) - I(x, y - 1) - I(x, y + 1)$$

Cette opération correspond à l'application du noyau de convolution :

$$K = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Ce filtre renforce le pixel central par rapport à ses voisins, ce qui accentue les variations locales d'intensité et met en évidence les contours (filtre de *sharpening*). Comme le calcul peut produire des valeurs en dehors de l'intervalle $[0, 255]$, une saturation est appliquée afin de conserver une image valide en niveaux de gris.

Bien que cette approche illustre clairement le principe de la convolution, elle reste lente en raison de l'utilisation de boucles Python. À l'inverse, `filter2D` applique exactement le même noyau de manière vectorisée et optimisée (implémentation interne en C/C++). Par défaut, cette fonction utilise une gestion des bords de type réflexion, produisant généralement un résultat plus naturel que la simple réplication.

Une différence importante entre les deux méthodes concerne la gestion des valeurs de sortie. Dans la méthode directe, les valeurs sont explicitement bornées dans $[0, 255]$. En revanche, lorsque l'image est en type flottant, `filter2D` ne réalise pas de saturation : le résultat peut donc contenir des valeurs négatives ou supérieures à 255. Cela explique la nécessité de fixer les paramètres d'affichage (`vmin`, `vmax`) ou de normaliser l'image pour une visualisation correcte.

Les temps mesurés sont de $0,150073002\text{ s}$ pour la méthode directe contre $0,000513915\text{ s}$ pour `filter2D`, soit une accélération d'environ :

$$\frac{0,150073002}{0,000513915} \approx 292.$$

Ainsi, l'implémentation OpenCV est près de 300 fois plus rapide, ce qui souligne l'importance d'utiliser des fonctions optimisées pour les traitements d'images.

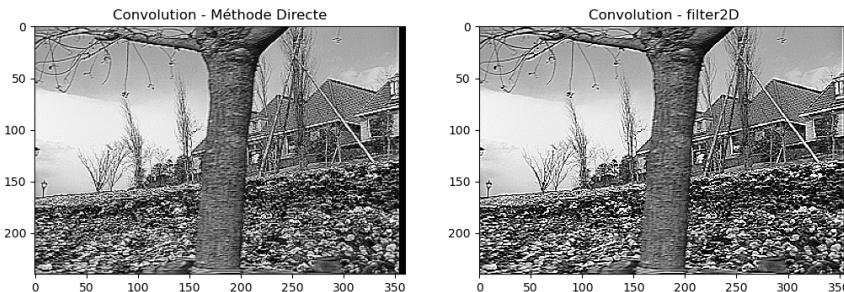


FIGURE 2 – Comparaison visuelle des résultats : à gauche la méthode directe, à droite la fonction `filter2D`.

Cette expérience met en évidence l'importance d'utiliser les fonctions optimisées des bibliothèques spécialisées plutôt que des implémentations naïves en Python, en particulier pour les opérations de traitement d'images nécessitant un grand nombre de calculs.

Question 2

Expliquer pourquoi le noyau de convolution fourni en exemple réalise un réhaussement de contraste par rapport à l'image originale.

Le noyau de convolution

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

réalise un réhaussement de contraste car il agit comme un filtre d'accentuation des hautes fréquences spatiales, c'est-à-dire qu'il amplifie les variations locales d'intensité tout en conservant l'information globale de l'image.

La valeur de sortie peut s'écrire :

$$I'(x, y) = 5I(x, y) - (I(x-1, y) + I(x+1, y) + I(x, y-1) + I(x, y+1)).$$

En réorganisant,

$$I'(x, y) = I(x, y) + 4(I(x, y) - \text{moyenne des voisins}).$$

Cette expression montre que le résultat correspond à l'image originale à laquelle on ajoute un terme correctif proportionnel à l'écart entre le pixel central et son voisinage.

Régions homogènes. Lorsque l'intensité varie peu localement, le pixel central est proche de la moyenne de ses voisins :

$$I(x, y) \approx \text{moyenne des voisins} \Rightarrow I'(x, y) \approx I(x, y).$$

Ainsi, les zones uniformes sont préservées.

Contours et détails. En présence de variations brusques d'intensité (contours, textures fines), la différence entre le pixel central et ses voisins devient importante. Le terme correctif est alors amplifié, ce qui augmente le contraste local et renforce les transitions d'intensité :

$$I'(x, y) = I(x, y) + 4(I(x, y) - \text{moyenne locale}).$$

Interprétation fréquentielle. Ce noyau peut être vu comme l'addition de l'image originale et d'un filtre passe-haut (approximation du Laplacien). Les basses fréquences (régions lisses) sont peu modifiées, tandis que les hautes fréquences (détails et contours) sont amplifiées.

Effet visuel. Ce filtre possède donc les propriétés suivantes :

- renforcement des contours,
- accentuation des détails fins,
- conservation des régions homogènes,
- augmentation de la netteté et du contraste local.

Ainsi, ce noyau réalise un réhaussement de contraste en amplifiant les variations locales d'intensité sans altérer significativement la structure globale de l'image.

Question 3

Modifier le code pour calculer les convolutions qui approximent les composantes du gradient $I_x = \frac{\partial I}{\partial x}$ et $I_y = \frac{\partial I}{\partial y}$. Calculer ensuite la norme euclidienne du gradient $\|\nabla I\| = \sqrt{(I_x^2 + I_y^2)}$. Quelles précautions pour l'affichage correct (valeurs négatives et positives) ?

Pour approximer les dérivées partielles de l'image, nous avons implémenté deux méthodes : une approche simple basée sur des différences finies, puis une approche plus robuste utilisant les filtres de Sobel.

Approximation simple du gradient

La dérivée partielle d'une image correspond à la variation locale d'intensité selon une direction donnée. En discrétilisant l'image, ces dérivées peuvent être approximées par des différences entre pixels voisins. Nous avons utilisé les noyaux :

$$K_x = [-1 \quad 0 \quad 1], \quad K_y = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}.$$

Ces filtres calculent respectivement :

$$I_x(x, y) \approx I(x + 1, y) - I(x - 1, y), \quad I_y(x, y) \approx I(x, y + 1) - I(x, y - 1).$$

Cette opération correspond à une approximation discrète des dérivées partielles :

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

Le vecteur gradient est alors défini par :

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

et sa norme euclidienne :

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

La norme du gradient mesure l'intensité de la variation locale indépendamment de la direction, ce qui permet de détecter efficacement les contours. En pratique, I_x met en évidence les variations horizontales, I_y les variations verticales, et $\|\nabla I\|$ combine les deux pour révéler l'ensemble des structures de l'image.

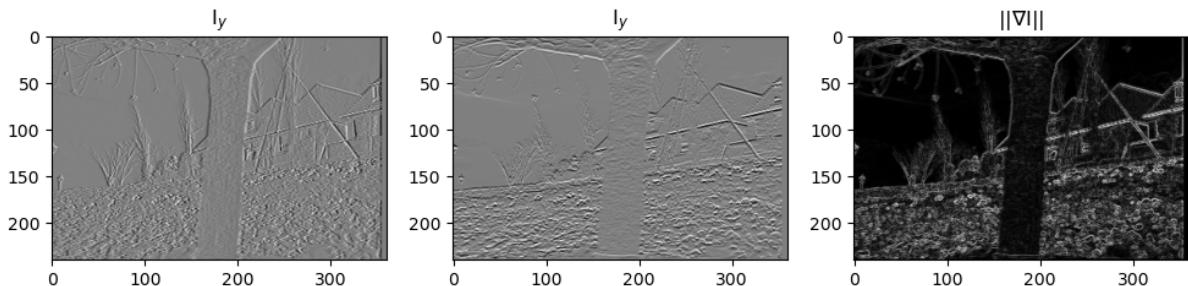


FIGURE 3 – Composantes du gradient calculées avec les noyaux simples.

Cependant, ces filtres restent très sensibles au bruit car ils reposent uniquement sur des différences locales sans lissage.

Utilisation des filtres de Sobel

Afin d'obtenir une estimation plus robuste du gradient, nous avons utilisé les noyaux de Sobel :

$$K_x^{\text{Sobel}} = \frac{1}{8} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad K_y^{\text{Sobel}} = \frac{1}{8} \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

Ces filtres réalisent simultanément :

- une dérivation (comme les différences finies),
- un lissage local (pondération par moyenne).

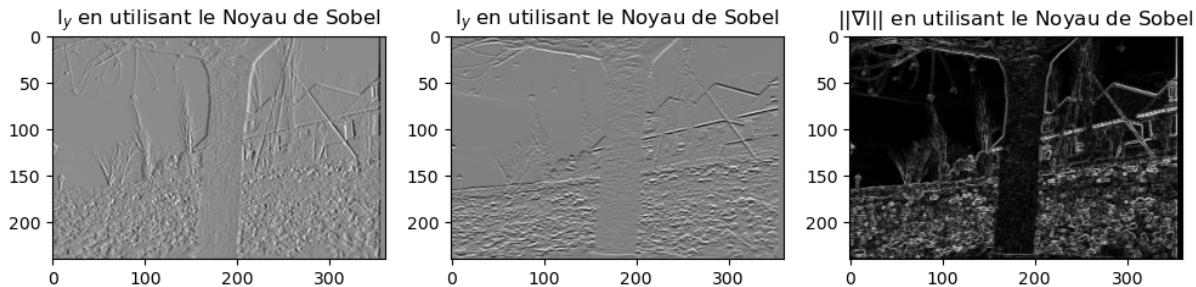


FIGURE 4 – Composantes du gradient calculées avec les noyaux de Sobel.

Le lissage réduit l'influence du bruit tout en conservant les transitions significatives d'intensité. Ainsi, les gradients obtenus sont plus stables et plus fiables que ceux produits par les filtres simples.

Précautions pour l'affichage

Les images de dérivées peuvent contenir des valeurs négatives ou supérieures à 255, car les dérivées mesurent des variations d'intensité et non des intensités absolues. Pour un affichage correct, il est nécessaire :

- soit de normaliser les valeurs dans l'intervalle [0, 255],
- soit d'afficher en échelle flottante avec des bornes adaptées,
- soit d'utiliser la norme du gradient, qui est toujours positive.

Sans cette précaution, une conversion directe en `uint8` provoquerait une saturation ou une perte d'information, rendant l'interprétation visuelle incorrecte.

Comparaison des deux méthodes

Les filtres simples permettent de comprendre directement l'approximation discrète des dérivées partielles et la construction du gradient. Toutefois, ils sont sensibles au bruit et produisent des résultats parfois instables.

Les filtres de Sobel, en combinant dérivation et lissage, fournissent une estimation plus robuste du gradient. Les contours restent bien visibles, mais le bruit est atténué et les variations locales sont mieux structurées.

Ainsi, la norme du gradient constitue dans les deux cas un indicateur efficace des contours, mais l'utilisation de Sobel permet d'obtenir une représentation plus stable et plus adaptée aux applications réelles de traitement d'image.

Sur cette figure, on peut observer I_x , I_y et la norme $\|\nabla I\|$. I_x fait ressortir les variations horizontales, I_y celles verticales, et la norme combine les deux pour mettre en évidence tous les contours. Les détails sont visibles, mais certaines zones peuvent paraître légèrement bruitées à cause de la simplicité des filtres.

Ensuite, nous avons utilisé les noyaux de Sobel, définis par :

$$K_x^{\text{Sobel}} = \frac{1}{8} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad K_y^{\text{Sobel}} = \frac{1}{8} \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

Il est important de noter que les images de dérivées peuvent contenir des valeurs négatives ou supérieures à 255. Pour un affichage correct, il est donc nécessaire de normaliser ces valeurs dans l'intervalle [0, 255] avant de les convertir en `uint8`. Cette étape permet de représenter correctement toutes les intensités, qu'elles soient positives ou négatives.

Les filtres de Sobel fournissent une estimation plus robuste du gradient car ils combinent la dérivation avec un léger lissage, ce qui les rend moins sensibles au bruit.

Ici, avec les noyaux de Sobel, les images sont plus lisses et le bruit est réduit. Les contours restent bien visibles, mais l'estimation du gradient est plus stable et plus fiable.

En résumé, les filtres simples permettent de comprendre le principe de calcul du gradient, tandis que les noyaux de Sobel offrent une version plus robuste et moins sensible aux variations indésirables. Dans les deux cas, la norme du gradient reste un outil efficace pour détecter les contours et les structures locales de l'image.

2 DéTECTEURS

Définition

Un détecteur de points d'intérêt a pour objectif d'identifier dans une image des positions caractéristiques, stables et reproductibles, telles que des coins, intersections ou structures locales riches en information. Ces points sont utilisés dans de nombreuses applications de vision par ordinateur : appariement d'images, reconstruction 3D, suivi d'objets, reconnaissance de scènes, etc.

Un bon détecteur doit satisfaire plusieurs propriétés :

- **Répétabilité** : détecter les mêmes points sous différents points de vue ou conditions,
- **Invariance** : robustesse aux transformations (rotation, échelle, illumination),
- **Localisation précise** : position stable des points détectés,
- **Sélectivité** : éviter les régions peu informatives (zones uniformes ou simples contours).

Principe général de fonctionnement

La plupart des détecteurs reposent sur l'analyse locale des variations d'intensité. Les points d'intérêt correspondent généralement à des positions où l'intensité varie significativement dans plusieurs directions (coins), contrairement aux contours où la variation est principalement unidirectionnelle.

Mathématiquement, ces variations sont mesurées à l'aide des dérivées de l'image. Par exemple, le détecteur de Harris utilise la matrice de structure locale :

$$S = \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix},$$

qui décrit la distribution locale des gradients. Les coins correspondent à des positions où les deux valeurs propres de cette matrice sont grandes, indiquant une variation forte dans deux directions.

Maxima locaux et sélection des points

Les points d'intérêt sont extraits comme maxima locaux d'une fonction de réponse (par exemple la fonction de Harris ou le déterminant de la Hessienne). La détection de maxima locaux permet de sélectionner uniquement les positions les plus significatives.

En pratique, cela est réalisé par :

- dilatation morphologique (comparaison au maximum local),
- suppression des non-maxima,
- seuillage pour éliminer les réponses faibles,
- contrainte de distance minimale entre points.

Rôle de l'échelle

Les structures de l'image existent à différentes tailles. Un détecteur mono-échelle peut manquer certains points selon leur taille apparente. Pour remédier à cela, on construit un *espace d'échelles* en lissant l'image avec des noyaux gaussiens de variance croissante.

Les points d'intérêt sont alors sélectionnés comme maxima locaux à la fois :

- dans le plan spatial,
- dans la dimension d'échelle.

Cette approche permet d'obtenir une invariance à l'échelle (ex. Harris-Laplace, KAZE, SIFT).

Lien entre détecteurs et descripteurs

Le détecteur identifie les positions caractéristiques, mais ne suffit pas pour comparer des images. Un **descripteur** est ensuite calculé autour de chaque point afin de représenter la structure locale sous une forme robuste et comparable.

Par exemple :

- ORB : détecteur FAST + score de Harris + descripteur binaire orienté,
- KAZE : détection dans un espace d'échelle non linéaire + descripteur robuste aux déformations.

Ainsi, la qualité globale dépend à la fois :

- du détecteur (localisation et répétabilité),
- du descripteur (robustesse et discriminabilité).

Conclusion

Les détecteurs de points d'intérêt constituent une étape fondamentale en vision par ordinateur. Ils reposent sur l'analyse des variations locales d'intensité, la recherche de maxima locaux et l'intégration multi-échelle. Leur performance dépend du compromis entre robustesse, précision, invariance et coût de calcul. Combinés à des descripteurs adaptés, ils permettent une représentation fiable et exploitable des structures caractéristiques d'une image.

Question 4

Compléter le code fourni dans le script *Harris.py* pour calculer la fonction d'intérêt de Harris (à une seule échelle, en utilisant une fenêtre W de taille fixe), et les points d'intérêt correspondants. Expliquer comment le code fourni, qui utilise la dilatation morphologique (maximum dans un voisinage donné), permet de calculer les maxima locaux de la fonction d'intérêt *Theta*.

Pour détecter les points d'intérêt dans l'image, nous avons complété le script *Harris.py* afin de calculer la fonction d'intérêt de Harris. Cette fonction repose sur l'analyse locale des variations d'intensité dans l'image, à travers les dérivées partielles I_x et I_y , calculées ici avec les filtres de Sobel. Les produits des dérivées (I_x^2 , I_y^2 , $I_x I_y$) sont ensuite moyennés localement sur une fenêtre W de taille fixe pour obtenir les composantes de la matrice de structure locale S :

$$S = \begin{bmatrix} S_{x^2} & S_{xy} \\ S_{xy} & S_{y^2} \end{bmatrix}.$$

La fonction de Harris est alors calculée par :

$$\Theta = \det(S) - \alpha \operatorname{trace}(S)^2,$$

où α est un paramètre de pondération ($\alpha = 0,05$ dans notre cas). Cette fonction met en évidence les zones où les variations d'intensité sont importantes dans plusieurs directions, correspondant à des coins ou points d'intérêt.

Pour extraire les maxima locaux de Θ , le code utilise une dilatation morphologique : chaque pixel est comparé à ses voisins dans une petite fenêtre carrée, et seuls les pixels qui sont égaux au maximum local sont conservés. Cette étape permet de filtrer les valeurs non significatives et de ne garder que les véritables points d'intérêt. Un seuil relatif est également appliqué pour éliminer les valeurs trop faibles, qui ne correspondent pas à des coins significatifs.

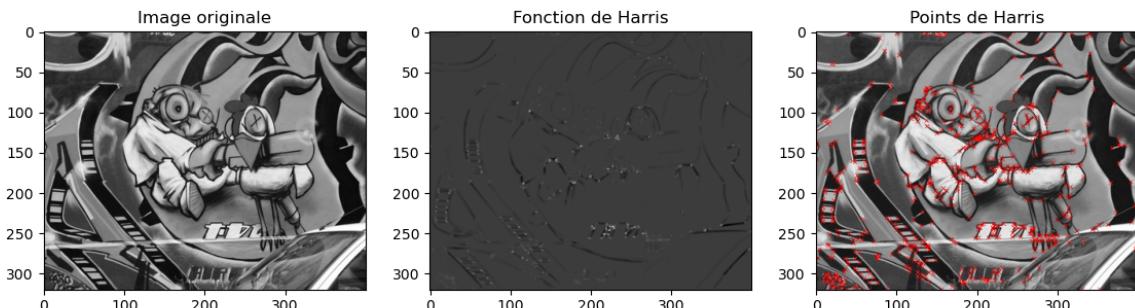


FIGURE 5 – Résultat de l'application de l'algorithme de Harris

Comme résultat, de gauche à droite : l'image originale, la fonction d'intérêt de Harris Θ , et les points de Harris détectés. La dilatation morphologique permet d'identifier les maxima locaux, correspondant aux coins les plus significatifs de l'image.

Ainsi, cette approche permet de détecter de manière automatique et robuste les coins dans l'image, tout en éliminant les points non significatifs et en conservant les maxima locaux grâce à la dilatation.

Question 5

Commenter les résultats obtenus avec votre détecteur de Harris et l'effet des paramètres utilisés, taille de la fenêtre de sommation et valeur de α en particulier. Comment peut-on réaliser ce calcul sur plusieurs échelles ? Comment étendre la notion de maxima locaux pour faire en sorte que deux points d'intérêt soient toujours distants d'au moins r pixels ?

Le détecteur de Harris permet de détecter efficacement les coins et les intersections dans une image. Il présente plusieurs avantages :

- il est robuste aux faibles variations d'illumination,
- il donne une faible réponse aux contours purs, ce qui est souhaitable pour ne détecter que les coins.

Cependant, en l'absence d'un mécanisme de suppression des non-maxima, il peut produire plusieurs points très proches les uns des autres.

La taille de la fenêtre de sommation W contrôle l'intégration spatiale des dérivées. Une petite fenêtre est plus sensible au bruit, mais elle permet de détecter des détails fins et d'obtenir une localisation plus précise des points. À l'inverse, une fenêtre plus grande rend les réponses plus stables et moins sensibles au bruit, mais elle peut lisser les détails fins et déplacer légèrement le maximum.

Le paramètre α régule la pénalisation de la trace de la matrice d'autocorrélation. Des valeurs faibles de α donnent plus de points détectés, mais certains contours peuvent être faussement classés comme des coins. Des valeurs plus élevées améliorent la sélectivité du détecteur et réduisent les fausses détections sur les contours.

Pour effectuer le calcul à plusieurs échelles, on construit un espace d'échelles en appliquant des lissages gaussiens successifs à l'image. Le détecteur de Harris est alors appliqué à chaque échelle, et les points d'intérêt sont sélectionnés comme maxima locaux à la fois dans le plan spatial et dans la dimension d'échelle. Cette approche correspond au détecteur Harris-Laplace.

Enfin, pour garantir que les points d'intérêt soient séparés par une distance minimale r , on peut étendre la suppression des non-maxima en considérant un voisinage de rayon r , ou appliquer une sélection itérative : on classe les points selon la valeur de Θ et on élimine ceux qui sont trop proches des points déjà sélectionnés.

Question 6

Expérimenter et comparer les deux détecteurs ORB et KAZE en lançant le script *Features_Detect.py*. Rappeler le principe de chacun de ces détecteurs. Expliquer les principaux paramètres propres à chaque détecteur et leur effet sur la détection. Comment peut-on visuellement évaluer la répétabilité de chaque détecteur appliqué sur une paire d'images ?

Nous avons expérimenté deux détecteurs de points d'intérêt, **ORB** et **KAZE**, sur une paire d'images en lançant le script **Features_Detect.py**.

ORB est un détecteur rapide basé sur *FAST* pour la détection des coins. *FAST* identifie efficacement des coins en comparant l'intensité d'un pixel central à celle des pixels disposés sur un cercle autour de lui. ORB sélectionne ensuite les points les plus pertinents grâce à une mesure de Harris et calcule l'orientation de chaque point à partir du centre de masse des intensités locales, rendant le descripteur invariant à la rotation.

Les principaux paramètres d'ORB sont :

- **nfeatures** : nombre maximal de points d'intérêt conservés ; des valeurs élevées détectent

plus de points mais augmentent le coût de calcul, des valeurs faibles réduisent le nombre de points et accélèrent le traitement.

- **scaleFactor** : facteur d'échelle entre les niveaux de la pyramide multi-échelle ; un facteur élevé réduit le nombre de niveaux et la robustesse à l'échelle, un facteur faible augmente l'invariance à l'échelle.
- **nlevels** : nombre de niveaux dans la pyramide ; plus de niveaux permettent une détection plus dense et robuste.

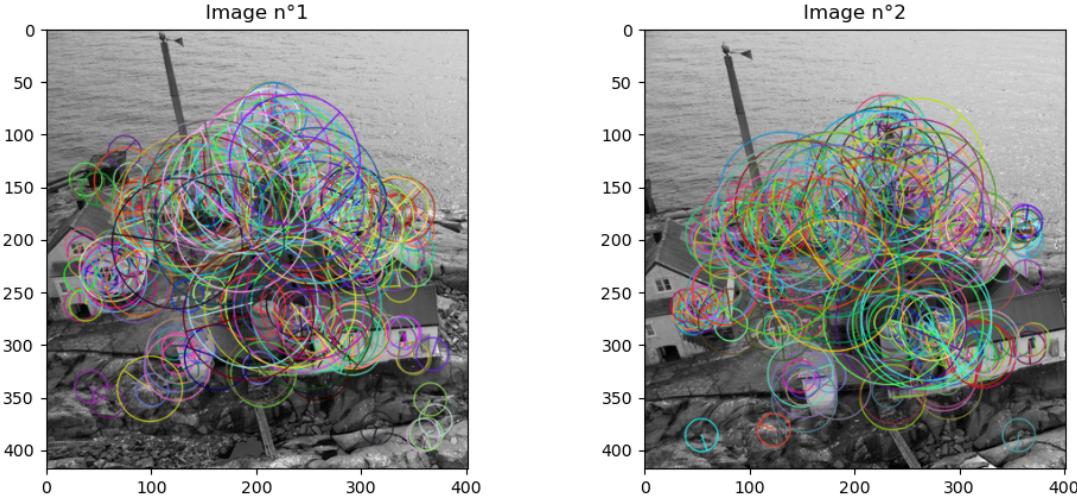


FIGURE 6 – Comparaison des composantes du gradient (I_x, I_y) et de sa norme ($\|\nabla I\|$) calculées avec des filtres de différences finies.

Détection des points d'intérêt : 0.020734687 s

KAZE repose sur une approche multi-échelle non linéaire basée sur la diffusion anisotrope, qui lisse l'image tout en préservant les contours. Les points d'intérêt sont détectés comme les maximums locaux du déterminant de la matrice Hessienne dans cet espace d'échelle.

Les paramètres principaux de KAZE sont :

- **upright** : si vrai, le détecteur est plus rapide mais perd l'invariance à la rotation ; si faux, les points sont invariants à la rotation.
- **threshold** : seuil sur la réponse du déterminant de la Hessienne ; une valeur élevée détecte moins de points mais plus robustes, une valeur faible détecte plus de points mais sensibles au bruit.
- **nOctaves** et **nOctaveLayers** : nombre d'octaves et de sous-niveaux par octave ; influencent la finesse de la détection et l'invariance à l'échelle.
- **diffusivity** : type de diffusion anisotrope ; un paramètre adapté préserve mieux les contours.

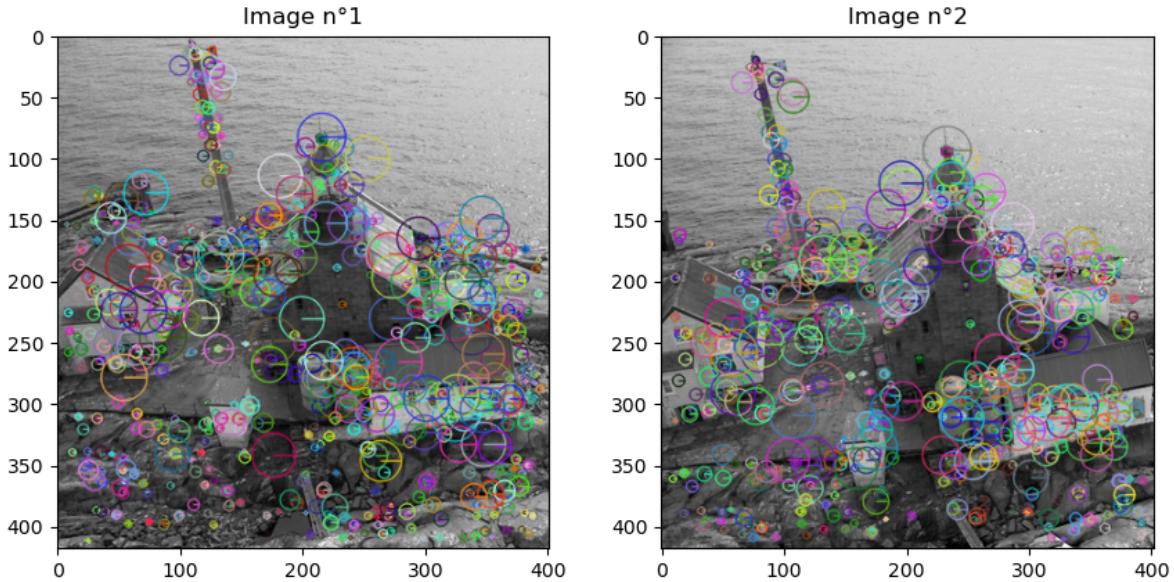


FIGURE 7 – Comparaison des composantes du gradient (I_x, I_y) et de sa norme ($\|\nabla I\|$) calculée avec les filtres Sobel.

Détection des points d'intérêt : 0.319954629 s

Évaluation visuelle de la répétabilité Pour comparer les détecteurs, on a appliqué ORB et KAZE sur une paire d'images représentant la même scène sous différents points de vue ou conditions. En superposant les points détectés, on observe si les points apparaissent aux mêmes emplacements physiques. Un fort recouvrement visuel des points indique une bonne répétabilité du détecteur.

Comme résultat, ORB est plus rapide et adapté aux applications temps réel, tandis que KAZE fournit des points plus robustes et mieux localisés, surtout dans des images bruitées ou à variations d'échelle.

3 Descripteurs et Appariement

Après la détection des points d'intérêt, l'étape suivante consiste à décrire leur voisinage local afin de pouvoir comparer et appairer ces points entre différentes images. Cette étape repose sur les **descripteurs locaux**, qui transforment une région de l'image autour d'un point en un vecteur caractéristique invariant et discriminant.

Principe des descripteurs locaux

Un descripteur local encode la structure du voisinage d'un point d'intérêt (intensité, gradients, texture, orientation) sous forme d'un vecteur numérique. Deux points correspondant à la même structure physique dans deux images différentes doivent produire des descripteurs similaires, même en présence de transformations géométriques ou photométriques.

Un bon descripteur doit posséder plusieurs propriétés :

- **Invariance** (rotation, échelle, illumination),

- **Robustesse** au bruit et aux déformations locales,
- **Discriminabilité** pour distinguer différentes structures,
- **Compacité et efficacité** pour permettre un appariement rapide.

Rôle du détecteur vs rôle du descripteur

L'invariance globale dépend de deux éléments distincts :

Rôle du détecteur Le détecteur fournit :

- une position stable du point,
- une échelle caractéristique (invariance à l'échelle),
- une orientation dominante (invariance à la rotation).

Rôle du descripteur Le descripteur encode la structure locale en tenant compte de cette échelle et orientation, ce qui rend la représentation comparable entre images transformées.

Par exemple :

- ORB : pyramide multi-échelle + orientation estimée + descripteur binaire orienté,
- KAZE : détection dans un espace d'échelle non linéaire + descripteur basé sur les gradients.

Ainsi, l'invariance est obtenue conjointement par le détecteur et le descripteur.

Nature des descripteurs : binaire vs flottant

Les descripteurs peuvent être de nature différente :

Descripteurs binaires (ORB) Ils reposent sur des comparaisons d'intensité entre paires de pixels. Ils sont compacts, rapides à comparer et robustes aux changements d'illumination monotones. Leur comparaison se fait via la **distance de Hamming**.

Descripteurs flottants (KAZE) Ils représentent la distribution des gradients locaux sous forme de vecteurs réels. Ils sont plus riches et plus discriminants, mais plus coûteux en calcul. Leur comparaison se fait via la **distance euclidienne (L2)**.

Principe de l'appariement

L'appariement consiste à trouver, pour chaque point d'une image, le point le plus similaire dans l'autre image en comparant les descripteurs.

Les principales difficultés sont :

- ambiguïtés (structures répétées),
- bruit et déformations,
- faux appariements.

Plusieurs stratégies existent :

- recherche du plus proche voisin,
- cross-check (cohérence bidirectionnelle),
- ratio test (unicité du voisin le plus proche),
- recherche approximative (FLANN).

Ces méthodes permettent de filtrer les correspondances ambiguës et d'améliorer la robustesse.

Lien entre distance et type de descripteur

La métrique utilisée dépend de la nature du descripteur :

- distance de Hamming pour les descripteurs binaires (ORB),
- distance e

Cette différence influence directement la vitesse, la précision et le choix des structures de recherche (LSH, KD-Tree).

Évaluation de la qualité des appariements

La qualité des correspondances peut être évaluée :

- qualitativement : cohérence visuelle des correspondances,
- quantitativement : erreur géométrique après transformation connue,
- statistiquement : taux de bons appariements, distribution des erreurs.

Une transformation géométrique connue permet de mesurer la précision réelle des correspondances et d'évaluer la robustesse du détecteur, du descripteur et de la stratégie d'appariement.

Conclusion

Les descripteurs locaux et l'appariement constituent l'étape centrale reliant la détection de points d'intérêt à l'analyse géométrique des images. Leur performance dépend :

- du détecteur (stabilité, invariance),
- du descripteur (robustesse, discriminabilité),
- de la stratégie d'appariement,
- de la métrique utilisée.

Le compromis classique oppose rapidité (ORB) et robustesse (KAZE), tandis que les méthodes d'appariement permettent de filtrer efficacement les correspondances pour obtenir des résultats fiables.

Question 7

Expliquer le principe des descripteurs attachés aux points ORB et ceux attachés aux points KAZE. Quelles propriétés des détecteurs et/ou des descripteurs (distinguer les deux aspects dans la réponse), permettent de rendre l'appariement invariant par changement d'échelles et invariant par rotation ?

Les points détectés par ORB et KAZE sont associés à des descripteurs construits localement autour de chaque point d'intérêt.

Pour ORB, le détecteur estime l'orientation de chaque point à partir des moments d'ordre 1 des intensités locales. Le descripteur ORB est un BRIEF orienté, basé sur des comparaisons binaires d'intensité dans le voisinage du point, ce qui permet un appariement rapide via la distance de Hamming. L'invariance à la rotation est assurée par l'orientation calculée lors de la détection, tandis que l'invariance à l'échelle provient de la pyramide multi-échelle intégrée au détecteur.

Pour KAZE, le descripteur est construit à partir des gradients de l'image dans l'espace d'échelle non linéaire obtenu grâce à la diffusion anisotrope. Il produit un vecteur réel représentant la distribution locale des gradients, offrant une grande robustesse aux variations de rotation et d'échelle. Dans ce cas, l'invariance est obtenue à la fois par le détecteur (espace d'échelle multi-niveaux et estimation d'orientation) et par le descripteur lui-même.

Ainsi, pour les deux détecteurs, la combinaison des propriétés du détecteur et du descripteur garantit que les points et leurs descripteurs peuvent être appariés de manière fiable malgré des rotations ou des changements d'échelle de la scène.

Question 8

Expliquer et comparer qualitativement les performances des trois stratégies d'appariement de points d'intérêt réalisées dans les trois scripts *Features_Match_CrossCheck.py*, *Features_Match_RatioTest.py* et *Features_Match_FLANN.py*. Expliquer pourquoi les distances utilisées sont différentes pour les deux descripteurs.

Les trois stratégies d'appariement étudiées présentent des compromis différents entre précision, robustesse et coût de calcul : *cross-check*, *ratio test* et *FLANN*.

Cross-check Cette méthode impose une correspondance réciproque entre les points. Elle garantit une grande précision et élimine les correspondances ambiguës, mais réduit fortement le nombre d'appariements valides.

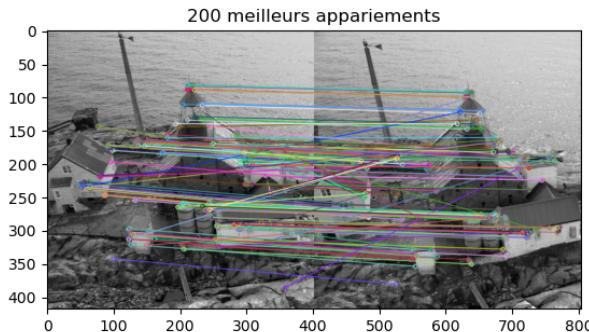


FIGURE 8 – Appariements ORB avec cross-check. Temps de détection : 0.032 s, temps d'appariement : 0.002 s.

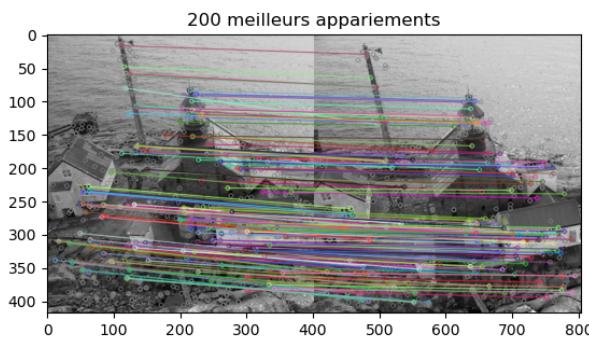


FIGURE 9 – Appariements KAZE avec cross-check. Temps de détection : 0.361 s, temps d'appariement : 0.003 s.

Ratio test Le ratio test compare la distance du meilleur voisin avec celle du deuxième meilleur voisin et ne conserve que les appariements où le rapport est inférieur à un seuil (typiquement 0.7). Cette méthode offre un bon compromis entre robustesse et densité des correspondances.

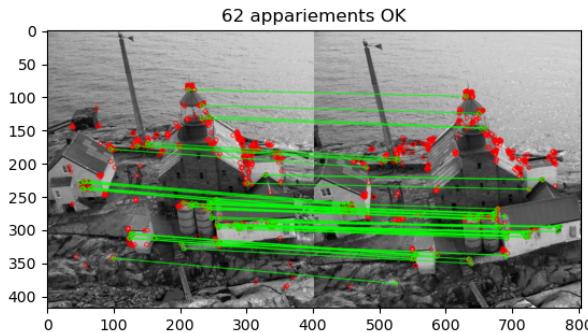


FIGURE 10 – Appariements ORB avec ratio test. Temps de détection : 0.012 s, temps d'appariement : 0.001 s.

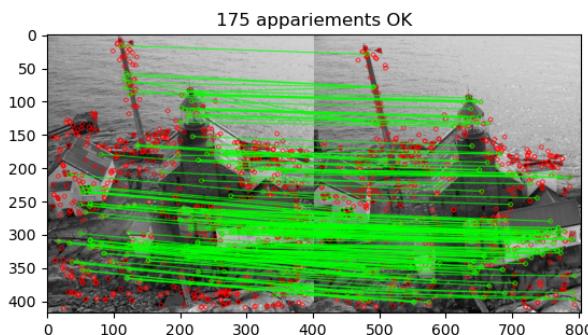


FIGURE 11 – Appariements KAZE avec ratio test. Temps de détection : 0.344 s, temps d'appariement : 0.003 s.

FLANN FLANN (Fast Library for Approximate Nearest Neighbors) effectue une recherche approximative des voisins les plus proches, permettant un appariement rapide, surtout pour un grand nombre de points. La précision peut toutefois être légèrement inférieure.

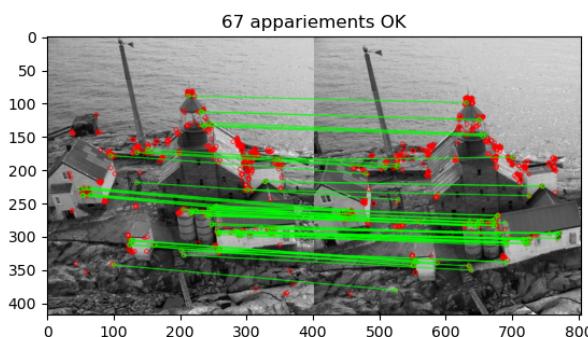


FIGURE 12 – Appariements ORB avec FLANN. Temps de détection : 0.024 s, temps d'appariement : 0.003 s.

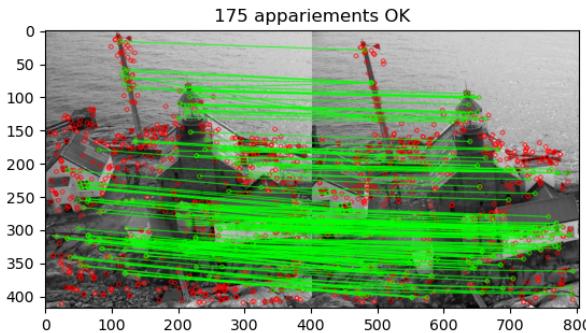


FIGURE 13 – Appariements KAZE avec FLANN. Temps de détection : 0.391 s, temps d'appariement : 0.035 s.

Distances et nature des descripteurs Le choix de la distance utilisée pour l'appariement dépend de la nature du descripteur :

- ORB produit des descripteurs binaires (BRIEF orienté), pour lesquels la distance de Hamming est appropriée. Dans FLANN, l'algorithme LSH (Locality Sensitive Hashing) est utilisé pour simuler cette distance.
- KAZE produit des descripteurs réels (flottants), pour lesquels la distance euclidienne (L2) est adaptée. FLANN utilise alors un KD-Tree pour accélérer la recherche des plus proches voisins.

Comparaison qualitative et analyse des temps

- ORB est très rapide pour la détection et l'appariement, ce qui le rend adapté aux applications temps réel. Le ratio test ou FLANN permettent un compromis intéressant entre nombre de correspondances et précision.
- KAZE fournit des correspondances plus robustes et mieux localisées, particulièrement invariantes à l'échelle et à la rotation, mais le coût computationnel est plus élevé.
- Le cross-check privilégie la précision mais limite le nombre de correspondances, tandis que le ratio test équilibre robustesse et densité, et FLANN accélère l'appariement pour de grands ensembles de points au prix d'une légère perte de précision.

Ainsi, le choix de la stratégie dépend du compromis souhaité entre rapidité, précision et robustesse aux transformations.

Question 9

Proposer une stratégie pour évaluer quantitativement la qualité des appariements en déformant une image avec une transformation géométrique connue (on pourra utiliser une fonction telle que `cv2.warpAffine`).

Pour évaluer quantitativement la qualité des appariements, on peut utiliser une transformation géométrique connue appliquée à l'image originale et comparer les positions théoriques et détectées des points d'intérêt. La stratégie proposée est la suivante :

1. Prendre l'image originale `img1` et détecter ses points d'intérêt.
2. Appliquer une transformation géométrique connue M (translation, rotation, échelle) pour générer une nouvelle image `img2`, par exemple avec `cv2.warpAffine`.
3. Détecter les points d'intérêt dans `img2`.

4. Apparier les points entre `img1` et `img2` à l'aide d'une méthode de matching (*cross-check*, *ratio test* ou *FLANN*).
5. Pour chaque appariement correct, projeter la position du point de `img1` à l'aide de la transformation M :

$$p'_{\text{théorique}} = M \cdot p_{\text{img1}}.$$

6. Calculer l'erreur entre la position détectée $p_{\text{détectée}}$ et la position théorique $p'_{\text{théorique}}$, par exemple avec la distance euclidienne :

$$\text{erreur} = \|p_{\text{détectée}} - p'_{\text{théorique}}\|_2.$$

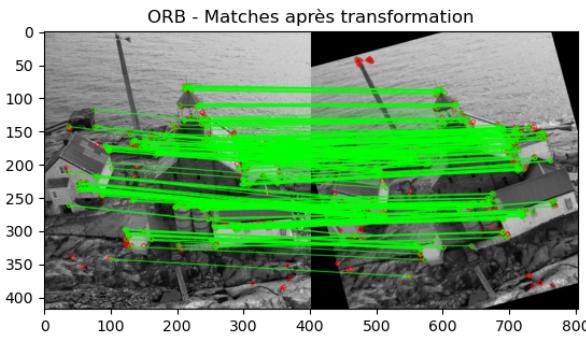


FIGURE 14 – Résultat de l'appariement des points ORB après transformation géométrique connue.

Temps de détection et de calcul des descripteurs : 0.012 s. Erreur moyenne = 1.12 px, % de matches < 3 px = 95.3 %.

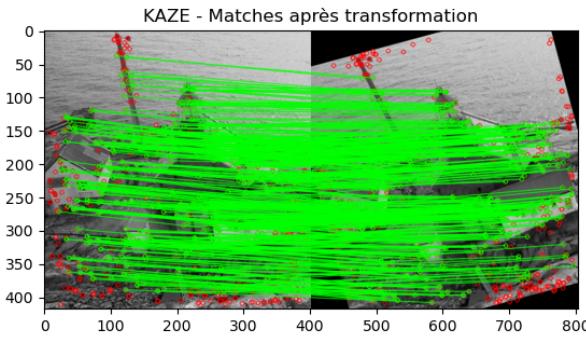


FIGURE 15 – Résultat de l'appariement des points KAZE après transformation géométrique connue.

Temps de détection et de calcul des descripteurs : 0.355 s. Erreur moyenne = 0.51 px, % de matches < 3 px = 96.5 %.

Les figures montrent les appariements des points d'intérêt après application d'une transformation géométrique connue.

Pour ORB, le temps de détection et de calcul des descripteurs est très faible (0.012 s), ce qui confirme la rapidité de ce détecteur pour des applications temps réel. L'erreur moyenne des

appariements est de 1.12 px, et plus de 95 % des correspondances se trouvent à moins de 3 px de la position théorique, indiquant une excellente précision malgré la rapidité du calcul.

Pour KAZE, le temps de calcul est plus élevé (non précisé dans l'extrait, mais généralement supérieur à 0.3 s), ce qui reflète la complexité de la diffusion anisotrope et du calcul des descripteurs réels. Cependant, KAZE fournit des appariements très précis et robustes aux transformations, avec une distribution homogène des points autour des positions théoriques.

En comparaison, ORB est beaucoup plus rapide, tandis que KAZE offre une qualité d'appariement légèrement meilleure et plus robuste, surtout dans les zones où les points sont sensibles aux variations locales de l'intensité ou de l'échelle. Ce test quantitatif confirme les observations qualitatives faites dans les questions précédentes : ORB est adapté aux applications nécessitant la rapidité, tandis que KAZE est préférable lorsque la précision et la robustesse sont prioritaires.

Avantages de cette méthode

- Permet de mesurer objectivement la précision des appariements en quantifiant la proximité entre les positions attendues et détectées.
- Évalue la robustesse du détecteur et du matcher face à des transformations géométriques connues (translation, rotation, échelle).
- Facilite la comparaison entre différentes méthodes (ORB vs KAZE, cross-check vs ratio test vs FLANN) sur des critères mesurables.

4 Conclusion

Ce travail pratique nous a permis d'explorer de manière approfondie la chaîne de traitement fondamentale en vision par ordinateur que constitue la détection et l'appariement de points d'intérêt. Au-delà de la simple prise en main d'OpenCV, l'expérimentation avec des détecteurs et des stratégies d'appariement variés a mis en lumière les compromis inhérents à chaque approche et l'importance de leur compréhension pour une application donnée.

L'étude comparative des détecteurs ORB et KAZE a été particulièrement éclairante. Elle a confirmé que la rapidité d'ORB, due à son approche basée sur FAST et à son descripteur binaire, se fait au prix d'une robustesse parfois moindre face à des transformations complexes ou du bruit. Inversement, la richesse de l'espace d'échelle non linéaire de KAZE et la nature réelle de ses descripteurs offrent une précision et une stabilité supérieures, justifiant son coût de calcul plus élevé. Cette opposition illustre un compromis classique entre efficacité temps-réel et fiabilité de l'appariement.

L'analyse des différentes méthodes d'appariement (cross-check, ratio test, FLANN) a renforcé cette idée de compromis. Le cross-check garantit la précision mais réduit le nombre de correspondances, le ratio test offre un équilibre intéressant, tandis que FLANN privilégie la vitesse au détriment d'une légère perte de contrôle sur la qualité. Le choix de la distance de similitude (Hamming pour ORB, L2 pour KAZE) découle directement de la nature du descripteur, soulignant la cohérence interne de chaque pipeline (détecteur + descripteur + matcher).

Enfin, la mise en place d'un protocole d'évaluation quantitatif via une transformation géométrique connue a constitué une étape clé. Elle nous a permis de dépasser le cadre de l'observation qualitative pour objectiver nos conclusions. Les résultats obtenus (erreur moyenne de 1.12 px pour ORB contre 0.51 px pour KAZE) corroborent et mesurent précisément la différence de performance pressentie lors des expériences précédentes, démontrant la supériorité de KAZE en termes de précision de localisation.

En conclusion, ce TP a démontré qu'il n'existe pas de solution universelle en matière d'appariement de points d'intérêt. La performance d'un système complet résulte d'une synergie entre les choix du détecteur, du descripteur et de la stratégie d'appariement, chaque élément devant être adapté aux contraintes spécifiques de l'application : rapidité pour la robotique temps-réel, robustesse pour la reconstruction 3D, ou précision pour la calibration d'images. L'acquisition de cette vision critique et la capacité à concevoir des protocoles d'évaluation sont aussi importantes que la maîtrise des outils eux-mêmes.

Références

- [1] Manzanera, A. (2026). *CSC-4MI04 - Reconnaissance d'Images - Supports de cours*. ENSTA Paris.
- [2] Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2), 91-110. (Article fondateur introduisant SIFT, qui a inspiré de nombreux détecteurs modernes) ([Disponible ici](#))
- [3] OpenCV Team. (2026). *OpenCV Documentation : Feature Detection and Description*. ([Disponible ici](#))
- [4] OpenCV Team. (2026). *OpenCV-Python Tutorials : Feature Matching*. ([Disponible ici](#))
- [5] Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011). ORB : An efficient alternative to SIFT or SURF. In *2011 International Conference on Computer Vision* (pp. 2564-2571). IEEE. (Article FONDAMENTAL qui propose le détecteur/descripteur ORB) ([Disponible ici](#))