

TP1: Détection et Appariement de Points Caractéristiques

Analyse et Indexation d'images

ROB317

Yu WANG

October 22, 2021

1 Introduction

L'objectif de ce TP est de (1) se familiariser avec la bibliothèque de traitement d'images *OpenCV* sous Python, et (2) expérimenter, critiquer et mettre en œuvre différentes techniques de représentation pour l'appariement de structures locales dans les images, en suivant les différentes étapes de la problématique :

- **Détecteur:** Comment réduire le support de la présentation.
- **Descripteur:** Quelle information attacher à chaque point du support.
- **Métrique:** Quelle mesure utiliser pour appairer des points.
- **Recherche:** Comment coder et parcourir l'espace des points d'un modèle.

Remarque : Tous les résultats (code, images, etc.) présentés dans ce rapport peuvent être trouvés sur ce GitHub : <https://github.com/Rescon/ROB317>

2 Format d'images et Convolution

2.1 Q1 : Comparaison entre les deux méthodes dans *Convolution.py*

Les résultats de l'exécution du *Convolution.py* sont présentés dans la Figure 1. La première différence que on constante en exécutant le code *Convolution.py* est le temps d'exécution des deux méthodes : pour la Méthode directe, il faut environ 0.5s, alors que pour la Méthode *filter2D*, il faut seulement environ 0.7ms.

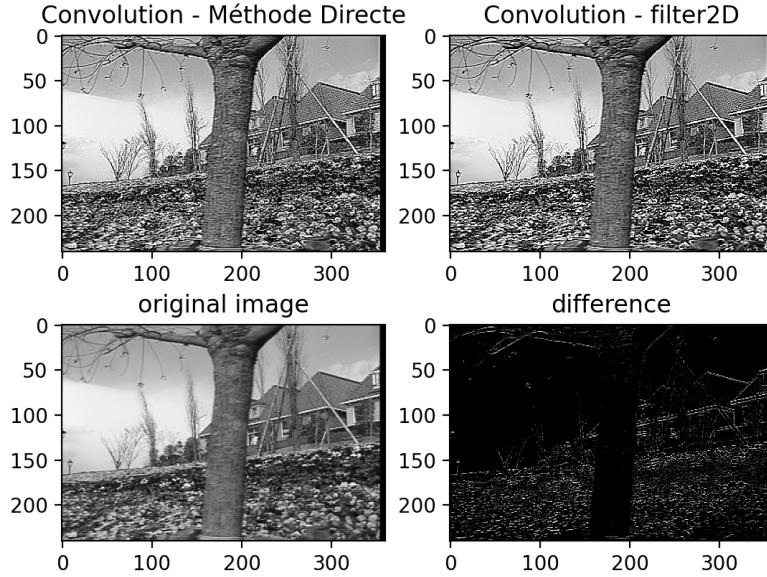


Figure 1: Résultat

Il est clair que la Méthode *filter2D*, une fonction de la bibliothèque *OpenCV*, optimise l'opération de convolution en conséquence, ce qui la rend beaucoup plus efficace que la méthode qui effectue un calcul pixel par pixel pour réaliser la convolution (Méthode directe).

En fait, la convolution est distributive par rapport à l'addition (comme Eq. (2.1))

$$h * (g + k) = h * g + h * k \quad (2.1)$$

La fonction *filter2D* tire parti de cette propriété pour optimiser l'opération de convolution. Par exemple, avec une image de taille (H, W) et un noyau de taille (K, K) (typiquement le noyau de convolution est un carré), en utilisant la méthode itérative sur la matrice, on a environ HWK^2 ordres de grandeur à compléter. Mais avec *filter2D* on a $2HWK$ car le noyau serait divisé en deux sous-noyaux et l'opération serait divisée en deux étapes : la première nécessiterait des opérations de l'ordre de HWK et la seconde nécessiterait également HWK . Cette amélioration introduite par la fonction *filter2D* est donc de l'ordre de $\frac{K}{2}$. La complexité de calcul a été réduite de $O(k^2)$ à $O(k)$, ce qui représente une accélération significative.

Si on regarde les résultats, on constate que les deux images obtenues avec

les deux méthodes sont très similaires, mais elles ne sont pas identiques : cela est dû au fait que on a créé des artefacts différents dans les deux images.

De plus, les fonctions de la bibliothèque *OpenCV* et *Matplotlib* utilisées dans cette section sont les suivantes:

- *cv2.imread(img)*: lire l'image à partir du data-base
- *cv2.getTickCount()*: Obtenir des informations sur le moment présent
- *cv2.copyMakeBorder(img, 0, 0, 0, 0, cv2.BORDER_REPLICATE)*: Ajout d'une autre couche de contour pour remplir les bords de l'image. En utilisant *BORDER_REPLICATE*, on peut dupliquer le contour ajouté aux lignes ou aux colonnes de la bordure.
- *cv2.getTickCount()*: Obtenir des informations sur le moment présent
- *cv2.getTickCount()*: Obtenir des informations sur le moment présent
- *cv2.getTickFrequency()*: Fréquence de comptage de l'horloge. On peut obtenir le nombre réel de secondes à partir de celle-ci et des informations obtenues par *cv2.getTickCount()*.
- *cv2.filter2D(img, -1, kernel)*: Effectuer la convolution entre l'image et le noyau d'une manière optimisée. -1 signifie que l'image de sortie aura les mêmes dimensions que l'image original.
- *plt.subplot()*: crée une figure et une grille de sous-parcelles en un seul appel, tout en offrant un contrôle raisonnable sur la façon dont les parcelles individuelles sont créées.
- *plt.imshow(img3, cmap='gray', vmin=0.0, vmax=255.0)*: Tracer l'échelle de gris avec une valeur comprise entre 0 et 255.
- *plt.title('title')*: ajouter le titre à l'image.
- *plt.show()*: Afficher les plots préparé sur l'écran.

2.2 Q2 : Expliquer le rehaussement de contraste introduit par le noyau de convolution

Tout d'abord, on peut effectuer une décomposition du noyau de convolution comme [2.2](#).

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} = [1] - \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad (2.2)$$

La première partie est un noyau unitaire et la deuxième partie concerne le noyau du *Laplacien* en 4-connexité comme (2.3).

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad (2.3)$$

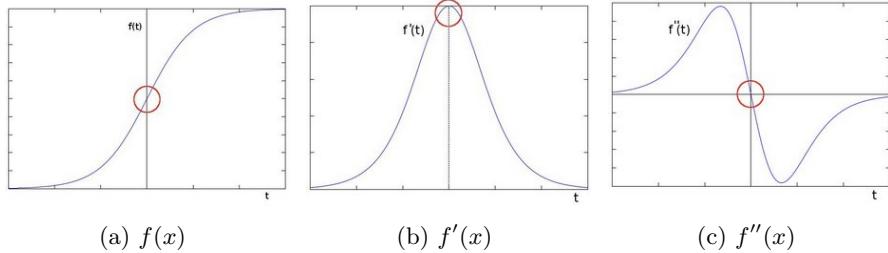


Figure 2: L'échelle de gris d'image et dérivées de premier et de second ordre

Comme le Figure 2, pour l'opérateur de Laplace, lorsque le niveau de gris du pixel central d'un voisinage est inférieur au niveau de gris moyen des autres pixels de son domaine, le niveau de gris de ce pixel central doit être réduit davantage, et lorsque le niveau de gris du pixel central d'un voisinage est supérieur au niveau de gris moyen des autres pixels de son voisinage, le niveau de gris de ce pixel central doit être augmenté davantage, ce qui permet d'obtenir une netteté de l'image. Comme les bords d'une image sont les zones où le niveau de gris saute, l'opérateur de Laplace est utile pour la détection des bords. Les techniques d'amélioration normales ont des difficultés à localiser les lignes de bord pour les bords abrupts et les bords qui changent lentement. Cet opérateur, cependant, peut être utilisé pour déterminer le point de passage à zéro entre les pics positif et négatif de la différentielle quadratique, et il est plus sensible aux points isolés ou aux extrémités, ce qui le rend particulièrement adapté aux applications où l'objectif est de mettre en évidence des points isolés, des lignes isolées ou des extrémités de lignes dans une image. Bien sûr, comme pour l'opérateur de gradient, l'opérateur de Laplace renforce également le bruit dans l'image et parfois, lorsqu'on utilise l'opérateur de Laplace pour la détection des bords, l'image peut être lissée au préalable.

Lorsque on applique le noyau à l'image originale, le résultat sera la convolution de l'image elle-même avec ce noyau, ce qui, en termes de pertinence de la convolution, peut être considéré comme la différence entre l'image elle-même (car elle sera le résultat de la convolution avec le noyau unitaire) et le Laplacien de l'image (le résultat de la convolution de l'image avec le Laplacien en 4 connexions). Étant donné que le Laplacien met en évidence les zones de l'image où de forts gradients sont présents, comme les bords, on peut noter que la différenciation entre l'image et son Laplacien donne un renforcement du contraste sur l'image originale.

2.3 Q3 : les convolutions qui approximent les dérivées seconde

La détection des bords est une partie importante du traitement des images. Pour cette raison, on utilise souvent les dérivées de premier et de second ordre d'une image pour effectuer la détection des bords. En général, la dérivée de second ordre donne une vue plus détaillée de la limite de l'objet que la dérivée de premier ordre. Cependant, il est évident que la dérivée de second ordre est également plus sensible aux points de bruit et peut amplifier l'effet du bruit. Toutefois, il est important de bien prétraiter l'image à l'aide d'un lissage gaussien avant d'appliquer ces types de filtres, car ils peuvent également mettre en évidence le bruit dans l'image. Dans une région à échelle de gris uniforme, il existe un point de bruit qui devient plus isolé et plus visible après le traitement par dérivation de second ordre, en particulier dans la région à échelle de gris lissée où le point de bruit est considérablement amplifié.

Dans ce qui suit, on va montrer les résultats de trois filtres qui utilisent des dérivées partielles de second ordre pour la détection des bords sur les images.

Définition de la dérivée partielle de second ordre I_{xx}

Le noyau de convolution $[1, -2, 1]$ est utilisé pour l'approximation de I_{xx} . Le code ci-dessus montre une implémentation de la convolution d'image en utilisant la méthode directe d'approximation et la méthode de filtre 2D disponible dans OpenCV de I_{xx} . Les résultats obtenus sont présentés dans la Figure 3.

```

1 val = -2 * img[y, x] + img[y, x - 1] + img[y, x + 1]
2 img2[y, x] = min(max(val, 0), 255)
3
4 kernel = np.array([[1, -2, 1]])
5 img3 = cv2.filter2D(img, -1, kernel)

```

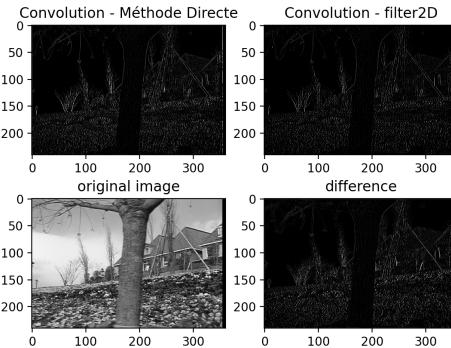


Figure 3: Comparaison de différents méthodes de convolution sur I_{xx}

Dérivées partielles de second ordre I_{xy}

Le noyau de convolution $\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$ est utilisé pour l'approximation de I_{xy} . Le code ci-dessus montre une implémentation de la convolution d'image en utilisant la méthode directe d'approximation et la méthode de filtre 2D disponible dans OpenCV de I_{yy} , Les résultats obtenus sont présentés dans la Figure 4.

```

1 val = 1 * img[y - 1, x] - img[y, x - 1] - img[y + 1, x] + img[y, ...
   x + 1]
2 img2[y, x] = min(max(val, 0), 255)
3
4 kernel = np.array([[0, 1, 0], [-1, 0, 1], [0, -1, 0]])
5 img3 = cv2.filter2D(img, -1, kernel)

```

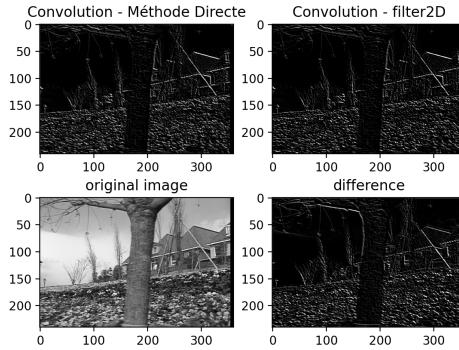


Figure 4: Comparaison de différents méthodes de convolution sur I_{xy}

Dérivées partielles de second ordre I_{yy}

Le noyau de convolution $\begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}$ est utilisé pour l'approximation de I_{yy} . Le code ci-dessus montre une implémentation de la convolution d'image en utilisant la méthode directe d'approximation et la méthode de filtre 2D disponible dans OpenCV de I_{yy} , Les résultats obtenus sont présentés dans la Figure 5.

```

1 val = -2 * img[y, x] + img[y - 1, x] + img[y + 1, x]
2 img2[y, x] = min(max(val, 0), 255)
3
4 kernel = np.array([[1], [-2], [1]])
5 img3 = cv2.filter2D(img, -1, kernel)

```

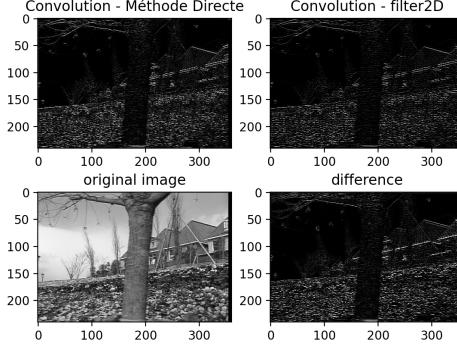


Figure 5: Comparaison de différents méthodes de convolution sur I_{yy}

Enfin, il est important de noter que pour obtenir un affichage correct de l'image, il est nécessaire de tenir compte du fait qu'après l'opération de convolution, les pixels de l'image résultants peuvent être obtenus avec des valeurs de gris supérieures ou inférieures au niveau de gris autorisé. Afin de résoudre ce problème, l'image finale est limitée à l'intervalle [0, 255].

3 DéTECTEURS

3.1 Q4 : La fonction d'intérêt correspondant au déterminant de la matrice hessienne

A l'ordre 2, la grandeur de base est la matrice hessienne :

- Ses vecteurs propres (resp. ses valeurs propres Λ_H et λ_H) correspondent aux directions (resp. intensités) de courbure principale.
- Sa norme de Frobénius, $\|H_I\|_F$ mesure l'intensité de la courbure globale.

$$H_I = \begin{bmatrix} \frac{\partial^2 I}{\partial x^2} & \frac{\partial^2 I}{\partial x \partial y} \\ \frac{\partial^2 I}{\partial x \partial y} & \frac{\partial^2 I}{\partial y^2} \end{bmatrix} \quad (3.1)$$

La matrice hessienne (comme Eq. (3.1)) est une matrice qui décrit la courbure de la fonction. Le code pour calculer du déterminant de la matrice hessienne est le suivant.

```

1 kernel_xx = np.array([[1, -2, 1]])
2 kernel_yy = np.array([[1], [-2], [1]])
3 kernel_x = np.array([-1, 0, 1])
4 kernel_y = np.array([1, 0, -1])
5
6 Ixx = cv2.filter2D(Theta, -1, kernel_xx)
7 Iyy = cv2.filter2D(Theta, -1, kernel_yy)
8 Ix = cv2.filter2D(Theta, -1, kernel_x)
9 Ixy = cv2.filter2D(Ix, -1, kernel_y)
10 Theta = Ixx*Iyy - Ixy**2

```

Les points d'intérêt représentent l'intensité de la courbure globale des bords. Le changement apparent des pixels au bord indique une augmentation du gradient directionnel. La force de la courbure globale est plus grande aux bords par rapport à la matrice hessienne. Le code pour calculer des maxima locaux et seuillage est le suivant.

```

1 Theta_maxloc = cv2.copyMakeBorder(Theta, 0, 0, 0, 0, ...
                                     cv2.BORDER.REPLICATE)
2 d_maxloc = 3
3 seuil_relatif = 0.01
4 se = np.ones((d_maxloc, d_maxloc), np.uint8)
5 Theta_dil = cv2.dilate(Theta, se)

```

La dilatation est l'opération qui consiste à trouver la valeur locale maximale. Le noyau de convolution est convolué avec le graphique, c'est-à-dire qu'il calcule la valeur maximale des points de pixels dans la zone couverte par la convolution (réflétant la localité) et attribue cette valeur maximale aux pixels spécifiés par le point de référence. L'opération d'expansion peut être exprimée comme (3.2). Le résultat est illustré à la Figure 6.

$$A \bigoplus B = \bigcup_{b \in B} A_b \quad (3.2)$$

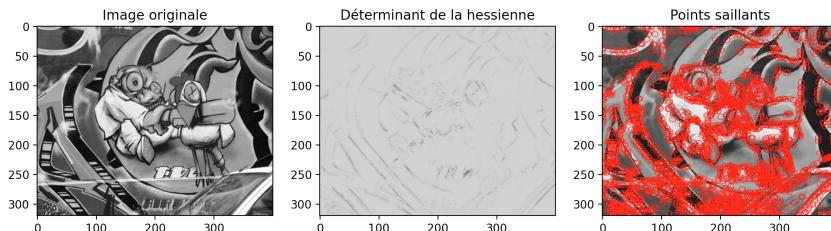


Figure 6: Les points d'intérêt de la matrice hessienne

3.2 Q5 : Les résultats du détecteur DoH

Dans le problème précédent, on a obtenu des résultats pour le cas $d_{maxloc} = 3$ (comme Figure 6). d_{maxloc} est un paramètre qui définit la taille du noyau de dilatation. Nous avons refait l'expérience pour $d_{maxloc} = 10$ et 100, et les résultats sont présentés dans la Figure 7 et la Figure 8.

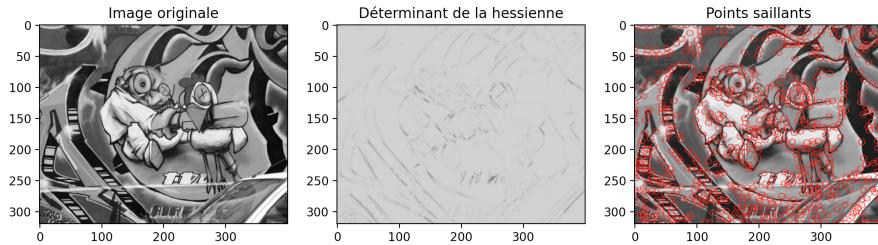


Figure 7: Les points d'intérêt de la matrice hessienne avec $d_{maxloc} = 10$

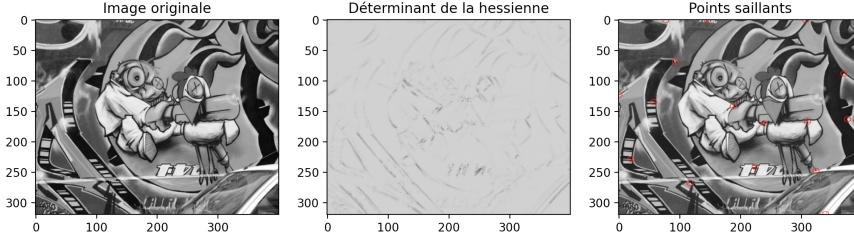


Figure 8: Les points d'intérêt de la matrice hessienne avec $d_{maxloc} = 100$

Avec les résultats, on constate que la valeur de d_{maxloc} est très importante ; si d_{maxloc} est trop petit, il en résulte trop de points d'intérêt et donc des informations redondantes, mais si d_{maxloc} est trop grand, alors on peut difficilement obtenir les points d'intérêt, ce qui nous prive également de beaucoup d'informations importantes.

$$G_\sigma(x, y) = \frac{1}{\sqrt{2\pi}\sigma_2} \exp -\frac{x^2 + y^2}{2\sigma_2} \quad (3.3)$$

Afin d'effectuer de tels calculs sur des échelles multiples, deux méthodes peuvent être utilisées : les dérivées gaussiennes et l'analyse en composantes principales. Les dérivées gaussiennes consistent à appliquer une échelle d'estimation en définissant un noyau de convolution g (par exemple, Eq. (3.3)). En faisant varier la valeur de σ , nous pouvons effectuer des calculs multi-échelles. L'idée

de l'ACP (analyse en composantes principales) est de calculer les vecteurs caractéristiques d'une image. En considérant différents nombres de vecteurs de caractéristiques, nous pouvons également réaliser une analyse multi-échelle.

3.3 Q6 : Les détecteurs ORB et KAZE

Les détecteurs sont des outils qui aident à identifier les parties importantes d'une image, qui peuvent être des coins ou des blobs. Les coins indiquent les variations des caractéristiques autour d'un pixel. Les blobs indiquent des zones de pixels contigus ayant des caractéristiques communes.

Le détecteur ORB

Le détecteur ORB (Oriented FAST and rotated Brief) est une fusion entre le détecteur FAST et le(descripteur BRIEF. Dans cette section, on va se concentrer sur ce que le détecteur ORB ajoute au détecteur FAST : la possibilité de calculer l'orientation des caractéristiques ponctuelles.

FAST est un détecteur qui sélectionne des points dont le voisinage circulaire présente une plage continue assez longue de points significativement lumineux ou sombres. En particulier, pour un point donné, il vérifiera les pixels ayant la même distance au point et évaluera leurs valeurs : si plusieurs pixels proches ont des valeurs similaires, le pixel sera détecté. Il s'agit d'un détecteur multi-échelle efficace qui permet un calcul rapide des points d'intérêt.

Dans *Features_Detect.py* pour le détecteur ORB, le code défini est le suivant.

```
1 kp1 = cv2.ORB_create(nfeatures=250, # Par défaut : 500
2                         scaleFactor=2, # Par défaut : 1.2
3                         nlevels=3)      # Par défaut : 8
```

où:

- **nfeatures** : le nombre de caractéristiques à gérer.
- **scaleFactor** : permet de travailler à plusieurs échelles et indique la quantité utilisée pour modifier la largeur du cercle (schéma à pyramide). Pour *scaleFactor* = 2, chaque niveau aura 4 fois moins de pixels que le précédent.
- **nlevels** : le nombre total de fois que on va modifier la largeur du cercle pour passer de la largeur minimale à la largeur maximale.

En outre, la répétabilité d'un détecteur est une propriété qui permet d'évaluer sa capacité à ne pas modifier les points détectés lorsque l'image est déformée par le traitement. On peut dire qu'un détecteur a une bonne répétabilité si les points mis en évidence sur l'image ne changent pas de position lorsque l'image résultante est déformée.

Pour le détecteur ORB, les résultat est illustré dans la Figure 9. Il est difficile de distinguer un point d'un autre car l'application à différentes échelles avec différents rayons de courbure on donne une image complexe, ce qui nous empêche de comparer facilement les deux images. Pour analyser la répétabilité du détecteur, on s'est concentrés sur les fenêtres et les portes de la maison de gauche : les points détectés dans la première image ne sont plus sélectionnés dans les images voisines, probablement parce que dans l'image de gauche ces points sont devenus des points de contour pour lesquels un cercle de taille suffisante ne peut plus être défini. De plus, si on regarde de près, on verra que certains points d'intérêt ne changent pas de position après la rotation.

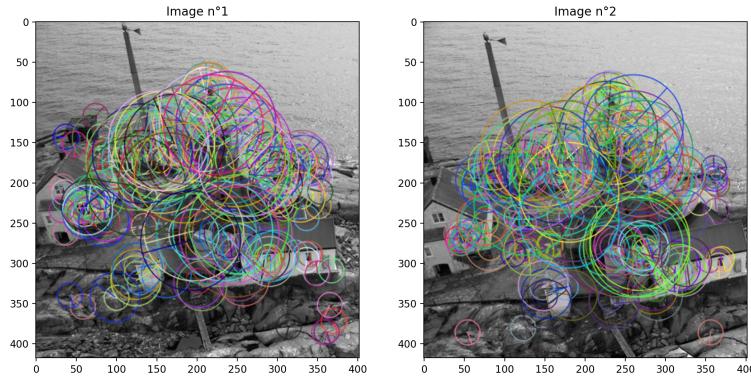


Figure 9: Répétabilité du détecteur ORB

Le détecteur KAZE

Le détecteur KAZE est un détecteur qui combine le principe de la diffusion anisotrope avec les maxima hessiens locaux : il est utilisé pour détecter des caractéristiques bidimensionnelles (horizontales et verticales) dans une image en utilisant une échelle non linéaire. Elle repose essentiellement sur la résolution de l'équation aux dérivées partielles (EDP) du filtre de diffusion anisotrope, ce qui signifie que la fonction de chaleur ne s'étend pas uniformément sur l'image comme le ferait une fonction gaussienne.

Dans *Features-Detect.py* pour le détecteur KAZE, le code défini est le suivant.

```

1  kp1 = cv2.KAZE_create(upright=False,      # Par d faut : false
2                         threshold=0.001,    # Par d faut : 0.001
3                         nOctaves=4,        # Par d faut : 4
4                         nOctaveLayers=4,   # Par d faut : 4

```

5	diffusivity=2) # Par défaut : 2
---	---------------------------------

où:

- **upright** : une valeur booléenne qui, si elle est vraie, utilise des descripteurs qui n'ont pas la propriété d'invariance rotationnelle.
- **threshold** : une valeur permet de choisir un seuil pour l'acceptation ou le rejet d'un point.
- **nOctaves** : le nombre d'ensembles d'images à échelle égale qui seront analysés par le descripteur.
- **nOctaveLayers** : le nombre de sous-niveaux pour chaque niveau de l'échelle.
- **diffusivity** : Choisir le type de diffusion anisotrope : *DIFF_PM_G1* (0), *DIFF_PM_G2* (1), *DIFF_WEICKERT* (2) ou *DIFF_CHARBONNIER* (3). Dans notre exemple, on a une diffusion de type *DIFF_WEICKERT*.

Par rapport à la répétabilité, pour le détecteur KAZE, les résultat est illustré dans la Figure 10. Les points mis en évidence dans l'image de gauche, par exemple le point en haut du poste sur l'image de gauche, correspondent aux points de l'image de droite. En outre, le point au centre de l'image indiquant la fenêtre du bâtiment est le même et ne change pas. Il est plus facile de distinguer un point d'un autre que dans le cas précédent, car le cercle qui lui est associé est plus petit et ne se superpose pas aux autres cercles.

4 Descripteurs et Appariement

4.1 Q7 : le principe des descripteurs attachés aux points ORB et KAZE

Le but des descripteurs est de décrire les points d'intérêt mis en évidence par le détecteur. Leur travail consiste à analyser les voisinages proches de chaque point pour obtenir un vecteur de descripteurs pour chaque point d'intérêt.

Les descripteurs présentent deux caractéristiques principales : l'invariance et la capacité de discrimination. L'invariance permet d'inscrire les mêmes points d'intérêt de la même manière dans deux images distinctes et montre également que le détecteur n'est pas affecté par les transformations géométriques. Le pouvoir discriminant est la capacité de distinguer deux points d'intérêt différents dans deux images distinctes.

Le descripteurs des points ORB

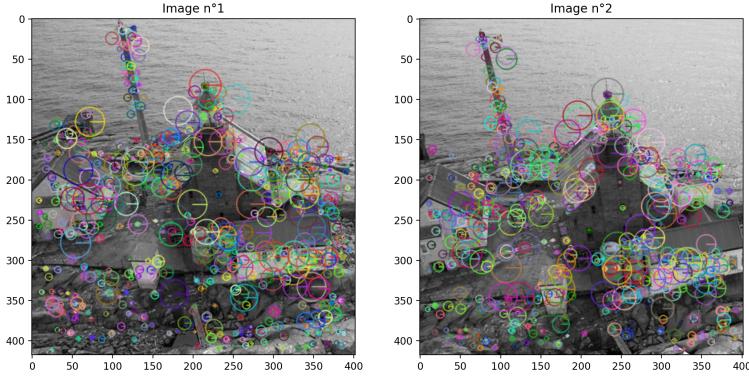


Figure 10: Répétabilité du détecteur KAZE

Le descripteur associé à un point ORB est le descripteur BRIEF (Binary Robust Independent Elementary Feature). Le détecteur Brief consiste à considérer une série de N paires de pixels aléatoires sur la région d'intérêt (c'est-à-dire le voisinage du point d'intérêt) et à utiliser la distance de Hamming ($D_H = \sum_{i=1}^k |x_i - y_i|$). Pour chacune de ces N paires, on va comparer leurs valeurs et si en niveaux de gris, la valeur associée au premier pixel $p(x)$ est inférieure à la valeur du second $p(y)$, on donnera 1, sinon on donnera 0 (comme Eq. (4.1))

$$f(p, x, y) = \begin{cases} 1 & \text{si } p(x) < p(y) \\ 0 & \text{sinon} \end{cases} \quad (4.1)$$

Mais le plus gros problème du descripteur BRIEF est qu'il n'est pas déformé par rotation. On ajoute donc à sa base des informations sur l'orientation et on obtient le descripteur ORB. Pour ce faire, on crée une matrice pour chaque séquence d'échantillonnage N afin de stocker ses informations de position. On crée ensuite une matrice de rotations pour faire pivoter la matrice précédente, qui construit finalement une table de transformation constituée d'un modèle BRIEF pré-calculé. Une fois l'orientation disponible, le descripteur ORB applique une correction aux points sélectionnés qui seront utilisés pour calculer le vecteur du descripteur.

Pour le descripteur ORB, le fait d'être invariant lorsque l'échelle change est dû au fait que la détection est multi-échelle et donc que on utilisera des cercles avec des rayons différents pour trouver les points d'intérêt. De plus, comme

on l'a déjà dit précédemment, la possibilité que chaque point d'intérêt calcule déjà son orientation au niveau du détecteur permet de garantir son invariance rotationnelle. De plus, le fait que deux points soient choisis au hasard fournit un niveau élevé de dé-corrélation, ce qui permet d'éviter la redondance des informations et de fournir des informations différentes pour les descripteurs, permettant finalement une faible corrélation entre les données.

Le descripteurs des points KAZE

Le descripteur associé aux points KAZE est le descripteur SURF (Speed-Up Robust Feature). Pour le calcul des directions, SURF utilise des ondelettes de Haar : elles sont calculées dans un voisinage circulaire de taille 6 fois supérieure à celle de chaque point d'intérêt dans les directions horizontale et verticale. La réponse en ondelettes est représentée en deux dimensions autour du point d'intérêt. Enfin, pour calculer la direction, il faut considérer un cône de 60 degrés : le cône contenant le plus de réponses à l'ondelette donnera la direction associée à ce point.

De plus, pour composer le vecteur du descripteur, la méthode considère un voisinage de points divisé en 64 sous-régions, à l'intérieur desquelles le vecteur est encore calculé à l'aide d'ondelettes de Haar : ses coordonnées horizontales et verticales sont la direction de la somme des réponses horizontales et verticales. On construit ensuite un vecteur v pour chaque région à partir des 4 vecteurs de ses sous-régions : $v = (\sum dx, \sum |dx|, \sum dy, \sum |dy|)$. Le vecteur descripteur est une concaténation de 16 vecteurs v , sa dimensionnalité est donc de 64. Comme le calcul est effectué dans les deux sens, on a la possibilité d'obtenir une plus grande robustesse au bruit.

Comme (1) la fonction de conductivité c utilisée pour la détection du point d'intérêt est appliquée à chaque image. (2) Le descripteur SURF utilise la réponse en ondelettes de Haar, ce qui permet de reconstruire l'orientation. Le descripteur KAZE possède donc la propriété d'invariance rotationnelle.

4.2 Q8 : Les performances des trois stratégies d'appariement de points d'intérêt

Le but d'appariement est de trouver les points correspondants dans une image et dans la même image après transformation, et on va démontrer la performance des descripteurs de cette manière. Dans notre exemple, on va analyser trois méthodes différentes : Cross Check, Distance Ratio Test et FLANN.

Cross Check

La stratégie de vérification croisée, également connue sous le nom de correspondance inversée, consiste à effectuer des correspondances dans les deux sens et à ne retenir que les correspondances. Le but de cette double vérification est de réduire les erreurs dans les cas où le point médian d'une des deux images n'est plus présent. Pour le descripteur ORB et le descripteur KAZE, les résultats sont

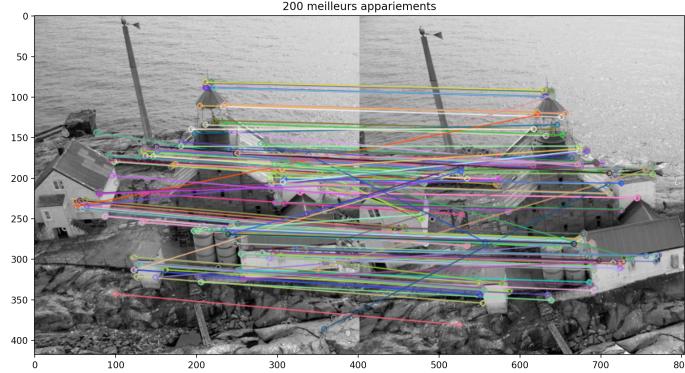


Figure 11: Appariement de Cross Check dans le cas de ORB

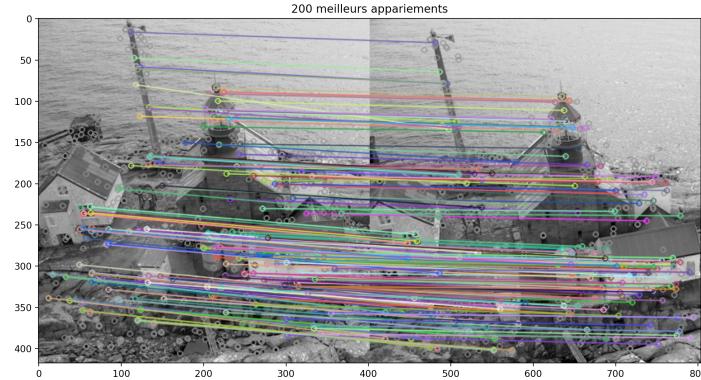


Figure 12: Appariement de Cross Check dans le cas de KAZE

présentés dans la Figure 11 et la Figure 12.

Dans la Figure 11, on constate que la plupart des points trouvent leurs homologues dans l'autre image, seuls certains points associés n'étant pas les bons. Cependant, ces points incorrects sont facilement détectés car la direction des lignes de connexion est différente de la direction de la plupart des points associés. Dans la Figure 12, on constate qu'il n'y a plus d'orientations différentes au sein de l'image : presque toutes les images dans lesquelles on trouve des points équivalents dans une autre image sont caractérisées par des lignes ayant la même orientation.

Distance Ratio Test

Distance Ratio Test utilise le rapport r des distances des deux descripteurs les plus proches. Selon la situation, il est nécessaire de choisir la valeur correcte de r pour préserver l'appariement. Typiquement, la valeur de r devrait être

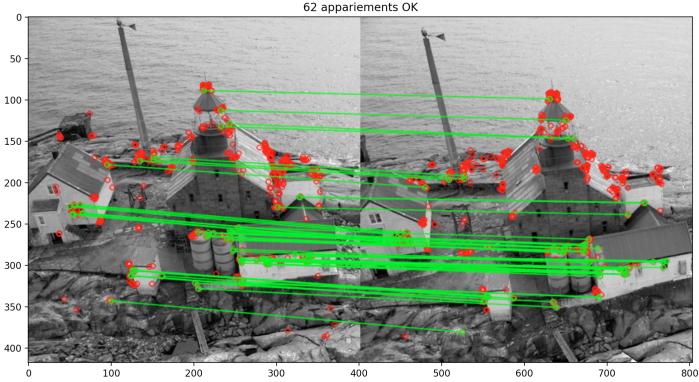


Figure 13: Appariement de Distance Ratio Test dans le cas de ORB

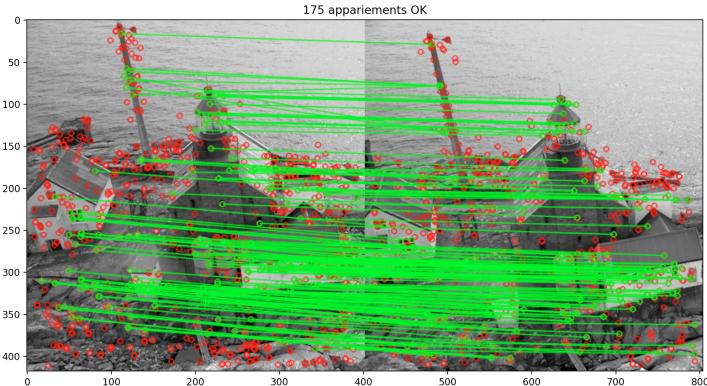


Figure 14: Appariement de Distance Ratio Test dans le cas de KAZE

inférieure à 0.8. pour un rapport $r \geq 0.8$, la probabilité d'une correspondance incorrecte est plus élevée. plus r est proche de 1, plus l'ambiguïté est grande et plus la probabilité de faire une correspondance incorrecte est élevée. Pour le descripteur ORB et le descripteur KAZE, les résultats sont présentés dans la Figure 13 et la Figure 12.

Dans Figure 13, on constate que bien que les correspondances soient toutes correctes, mais le nombre de correspondances trouvées n'a pas changé. Dans la Figure 14, on a plus de correspondances et comme dans la Figure 11, la grande majorité des correspondances semblent correctes et associent un point sur l'image de droite au même point sur l'image transformée.

On peut remarquer que le comportement de mise en correspondance de KAZE est plus efficace dans ce cas, car il parvient à trouver de nombreuses correspondances correctes, environ trois fois plus que dans le cas de ORB. Cela est dû

au fait que l'évaluation ORB est effectuée en utilisant les niveaux de gris de deux points choisis au hasard autour du point d'intérêt et que les résultats de cette opération présentent une plus grande variance dans le point d'intérêt. Le temps est relatif au KAZE de la grille considérée autour du point d'intérêt et l'ondelette de Haar est utilisée pour calculer les caractéristiques de ce point.

On constate que le comportement d'appariement de KAZE est plus efficace dans le cadre de cette stratégie, puisqu'il parvient à trouver de nombreuses paires correctes, environ trois fois plus que dans le cas de ORB. Cela est dû au fait que l'évaluation de ORB est effectuée en utilisant le niveau de gris de deux points choisis au hasard autour du point d'intérêt, une opération qui donne des résultats qui varient davantage dans le temps que KAZE, qui considère la grille autour du point d'intérêt et utilise des ondelettes de Haar pour calculer les caractéristiques des points.

FLANN



Figure 15: Appariement de FLANN dans le cas de ORB

FLANN (Fast Library for Approximate Nearest Neighbor) permet d'effectuer une approche du arbre-KD. Cette méthode permet d'accélérer la recherche des plus proches voisins. La première étape consiste à créer l'arbre en divisant récursivement les données en deux parties sur la base des caractéristiques des points (soit la médiane, la direction de la variance maximale ou une direction aléatoire de la variance maximale). Vient ensuite la partie de la recherche à l'intérieur de l'arbre, qui consiste à chercher dans l'arbre le plus proche voisin de la branche considérée. Pour le descripteur ORB et le descripteur KAZE, les résultats sont présentés dans la Figure 15 et la Figure 16.

Dans ce cas, on constante que les performances obtenues avec le descripteur ORB sont un peu moins bonnes que celles obtenues avec KAZE. Cela s'explique par le fait que le descripteur ORB est un descripteur binaire puisqu'il utilise le descripteur BRIEF et donne le vecteur du descripteur binaire en sortie, en particulier il est calculé en utilisant la distance de Hamming car elle est plus

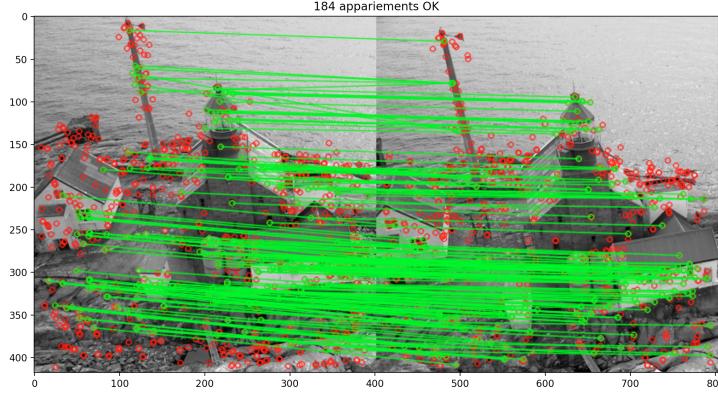


Figure 16: Appariement de FLANN dans le cas de KAZE

efficace pour les vecteurs binaires. Pour le descripteur KAZE, par contre, la dérivée directionnelle est utilisée ainsi que la distance euclidienne (Norme L2).

4.3 Q9 : Une stratégie pour évaluer quantitativement la qualité des appariements

Afin d'évaluer quantitativement la qualité de l'appariement, nous pouvons développer la stratégie suivante.

1. Créer une image standard basée sur une transformation géométrique connue. Pour ce faire, nous pouvons utiliser la fonction *getRotationMatrix2D()* d'OpenCV pour calculer une matrice de transformation représentant la rotation et le zoom de l'image, puis nous utilisons la fonction *warpAffine()* d'OpenCV pour appliquer cette transformation à l'image originale afin d'obtenir une nouvelle image. Dans ce cas, nous pouvons définir notre propre transformation (angle de rotation et facteur d agrandissement).

2. Les points d'intérêt sur la nouvelle image sont calculés et mis en correspondance avec l'image originale. La nouvelle image est ensuite traitée de la même manière que l'image originale : détection des points d'intérêt et calcul des descripteurs. La méthode de mise en correspondance à tester est ensuite exécutée pour faire correspondre les points d'intérêt de la nouvelle image avec ceux de l'image originale. 3.

3. La position théorique des points d'intérêt sur la nouvelle image est calculée. L'idée clé de cette évaluation quantitative est de comparer la correspondance des points d'intérêt sur les deux images calculée pendant l'appariement avec la correspondance théorique. Puisque dans l'étape 1 nous connaissons exactement la forme de la transformation que nous utilisons, nous pouvons calculer la position théorique des points d'intérêt de l'image originale dans la nouvelle

image.

4. Comparaison des deux listes de points et calcul de l'erreur. Une fois que nous avons obtenu les deux correspondances théoriques et appariées sous la forme d'une liste de points situés sur l'image standard, il ne reste plus qu'à les comparer. Dans cette optique, nous calculons pour chaque point apparié la distance entre celui-ci et le point théorique correspondant, ce qui correspond à l'erreur d'appariement. À partir de ces erreurs, nous pouvons calculer l'erreur moyenne de tous les points de correspondance, qui est le premier indicateur pour évaluer la qualité de la correspondance. Ensuite, nous pouvons également déterminer une valeur seuil permettant de juger de la qualité de la correspondance, ce qui constitue la deuxième métrique. Il nous permet d'évaluer davantage la qualité de la correspondance en définissant une erreur de tolérance maximale pour obtenir la proportion de correspondances réussies.