MI 204

Détection et Appariement de Points Caractéristiques

ZHANG Yuheng, JIN Pin

16 Janvier 2024

Format d'images et Convolutions

Q1

— Méthode directe : 0,065230542 secondes
— Méthode filter2D : 0.000280709 secondes

La méthode directe de convolution applique une opération de convolution en itérant sur les pixels de l'image (à l'exclusion des bords) et en appliquant un noyau spécifique. Le noyau est un simple filtre d'amélioration des contours qui rend l'image plus nette en mettant en évidence les contours. L'utilisation par la méthode directe de boucles imbriquées en Python pour itérer sur les pixels de l'image introduit un surcoût considérable. Python, qui est un langage interprété, est beaucoup plus lent à exécuter des boucles, en particulier des boucles imbriquées profondes, par rapport à des langages de niveau inférieur comme C ou C++.

La fonction filter2D est nettement plus efficace que la méthode directe. Cette efficacité provient du fait que filter2D est implémenté en C++ et optimisé pour la performance, y compris l'utilisation du traitement parallèle et d'autres optimisations qui ne sont pas disponibles dans le code Python de haut niveau utilisé dans la méthode directe.

— OpenCV Fonctions
  - cv2.imread(filename, flags) : lit une image à partir du fichier spécifié par nom de fichier.
  - cv2.copyMakeBorder(src, top, bottom, left, right, borderType, value) : copie l'image source src dans une nouvelle matrice avec des bordures ajoutées autour d'elle. Les paramètres top, bottom, left et right spécifient l'épaisseur des bordures dans les directions correspondantes. Le paramètre borderType spécifie le type de bordure à ajouter
— MatPlotLib
  - plt.imshow(X, cmap=None, norm=None, aspect=None, ...) : affiche une image (ou un tableau 2D régulier) X sur une trame 2D régulière. Le paramètre cmap spécifie la table des couleurs. Les paramètres vmin et vmax permettent de contrôler la normalisation des couleurs
  - plt.subplot(nrows, ncols, index) : organise les parcelles dans une structure en grille.
  - plt.title(label, fontdict=None, loc=None, ...) : définit le titre des axes actuels.
  - plt.show() : affiche toutes les figures ouvertes et les blocs jusqu'à ce que les figures soient fermées.

Q2

Center Pixel Weighting : The center of the kernel has a weight of 5. This means that the current pixel's value is multiplied by 5 in the output image. Surrounding Pixels Weighting : Each of the four direct neighbors (up, down, left, and right) of the central pixel is given a weight of -1.

This setup creates a subtractive effect from the central pixel's value based on the intensity of its immediate neighbors. The subtraction caused by the negative weights around the central positive weight emphasizes the high-frequency components of the image, which correspond to edges and fine details. In essence, areas of the image with rapid intensity changes (i.e., edges) are boosted.

The process exaggerates the difference between light and dark areas, making the dark parts darker and the light parts lighter, relative to each other.

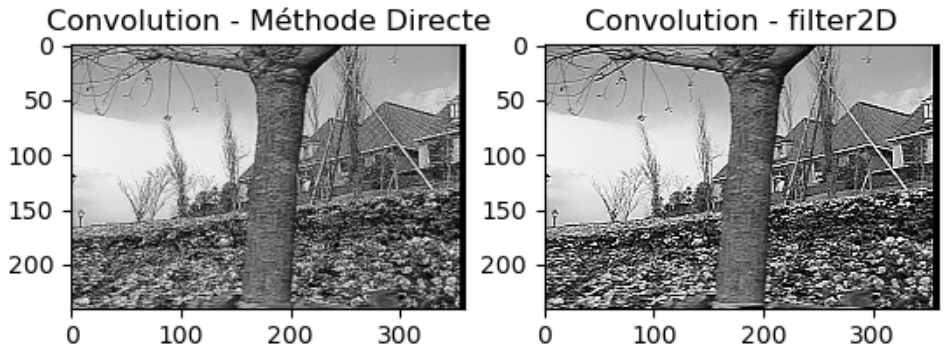

FIGURE 1 – Convolutions

Q3

Le code modfié est ajouté dans le fichier *convolution-modified.py*. Le résultat modifié est la fig 2.

Voici les précautions à prendre pour l'affichage :

— Normalisation : Normaliser les valeurs du gradient dans l'intervalle [0, 255].
— Conversion de Type : Convertisser les valeurs normalisées en un type de données approprié, tel que uint8 pour l'affichage avec OpenCV ou Matplotlib.
— Suppression des Négatifs : Étant donné que les valeurs négatives n'ont pas de sens dans le contexte de l'intensité des pixels, on les ramène généralement à zéro. Alternativement, on peut prendre la valeur absolue des gradients avant la normalisation.

Détecteurs

Q4

Le code Harris est complet dans le fichier *Harris.py*.

Pour fournir le code, on fait :

— Calcul du gradient de l'image à l'aide des opérateurs de Sobel pour I_x et I_y.
— Calcul des produits des dérivées $I_{xx} = I_x \cdot I_x$, $I_{xy} = I_x \cdot I_y$, et $I_{yy} = I_y \cdot I_y$.
— Application d'un filtre gaussien pour lisser les produits des dérivées.
— Calcul de la fonction de réponse de Harris : $R = \det(M) - k \cdot (\text{trace}(M))^2$, où M est la matrice du second moment $\begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}$ et k est un facteur de sensibilité.

Pour calculer les maxima locaux de la fonction d'intérêt Theta :

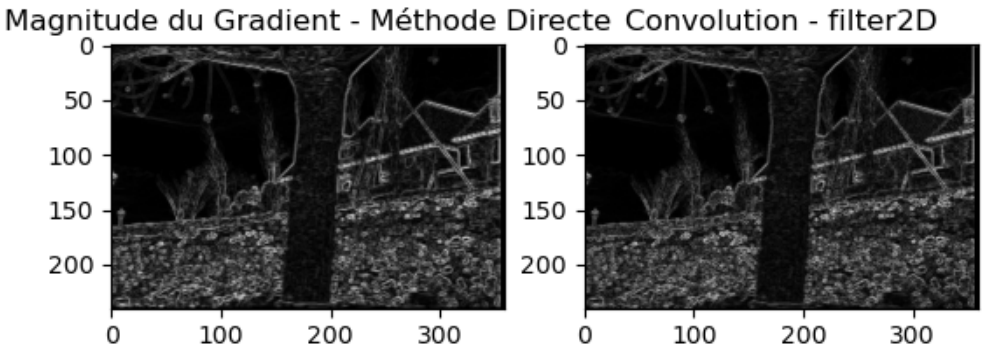

FIGURE 2 – Convolution modifié

— cv2.dilate(Theta, se) : remplace chaque pixel de Theta par la valeur maximale trouvée dans son voisinage. La taille de l'élément structurant détermine la taille de ce voisinage.
— Theta_maxloc[Theta < Theta_dil] = 0.0 : met à zéro chaque pixel de la matrice Theta originale qui n'est pas égale à la version dilatée Theta_dil n'est pas un maximum local.
— max_val = seuil_relatif * Theta.max() : garantit que seuls les maxima les plus significatifs sont pris en compte. Le seuil relatif ne conserve que les valeurs dans Theta_maxloc qui sont à un certain pourcentage de la valeur maximale dans la matrice Theta.

Q5

Le résultat du code est la fig 3. Des régions très lumineuses représentent des scores élevés de la fonction de coin de Harris. Les points semblent être bien distribués autour des coins et des bords de l'image, ce qui suggère que la fonction de Harris a été efficace pour détecter les points caractéristiques.

Taille de la Fenêtre de Sommation influence la zone sur laquelle les gradients sont calculés. Une fenêtre plus grande peut fusionner les coins proches et les rendre indétectables, tandis qu'une fenêtre trop petite peut être trop sensible au bruit et détecter des faux coins.

Valeur de α équilibre la réponse de Harris entre les coins, les bords et les surfaces plates. Une valeur trop élevée peut ignorer des coins valides, tandis qu'une valeur trop faible peut détecter de faux positifs.

Pour calculer sur plusieurs échelles, on peut utiliser une pyramide d'images où l'image est progressivement réduite, et le détecteur de Harris est appliqué à chaque niveau de cette pyramide. Cela permet de détecter des points d'intérêt à diverses résolutions, capturant ainsi des caractéristiques qui ne sont visibles qu'à certaines échelles.

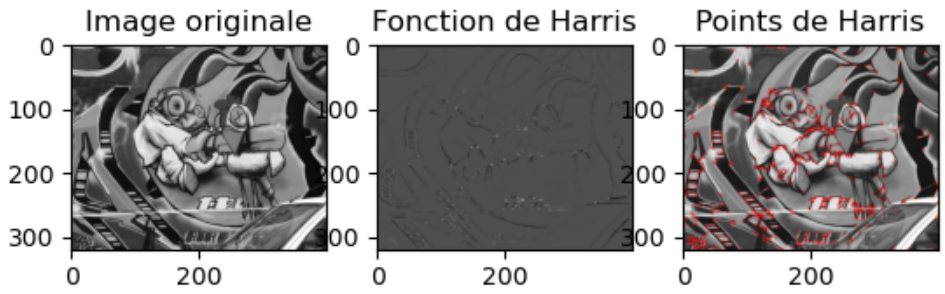

FIGURE 3 – Harris

Pour garantir qu'aucun point d'intérêt ne soit trop proche d'un autre (distants d'au moins r pixels), une suppression non maximale stricte peut être utilisée après la détection des coins. On peut appliquer une suppression non maximale avec une fenêtre de taille r et ne conserver que le point d'intérêt le plus fort dans cette fenêtre.

Q6

Le résultat d'expérience ORB est dans la fig 4, celui KAZE est dans la fig 5.

— ORB est une combinaison de FAST keypoint detector et BRIEF descriptor avec quelques modifications pour améliorer la performance. ORB est rapide et efficace, idéal pour les applications en temps réel.
  - nfeatures : contrôle le nombre de points d'intérêt détectés. Moins de features peuvent accélérer le processus de détection mais risquent de manquer des caractéristiques importantes.
  - scaleFactor : influence la robustesse des points d'intérêt aux changements d'échelle. Un facteur plus grand peut accélérer la détection, mais risque de perdre des détails fins.
  - nlevels : affecte la granularité de la pyramide d'échelle. Moins de niveaux peuvent réduire la sensibilité aux changements d'échelle.

— KAZE offre une détection de caractéristiques basée sur des méthodes non linéaires de filtrage d'échelle, ce qui le rend bon pour capturer des détails fins et complexes.
  - upright : est utilisé lorsque l'orientation des caractéristiques n'est pas importante ou pour améliorer les performances de calcul.
  - hreshold : influence le nombre de caractéristiques détectées. Un seuil plus bas détecte

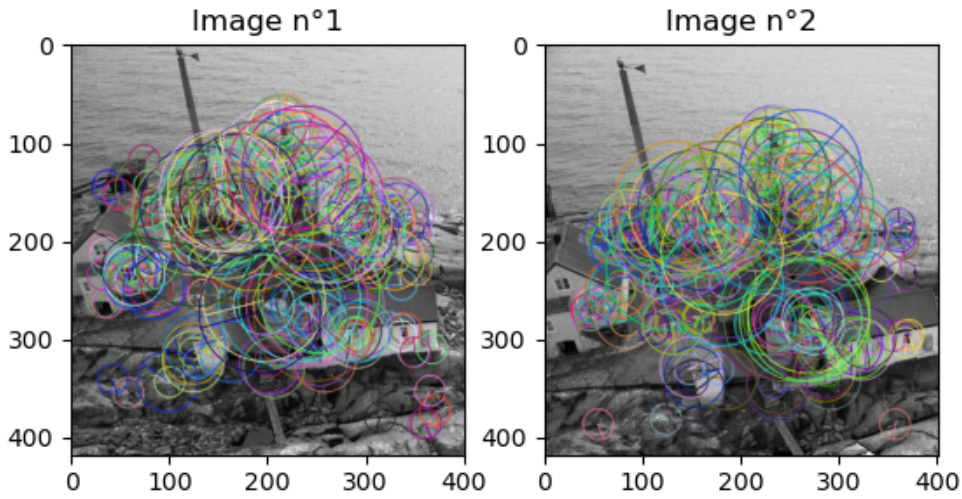

FIGURE 4 – Detect - ORB

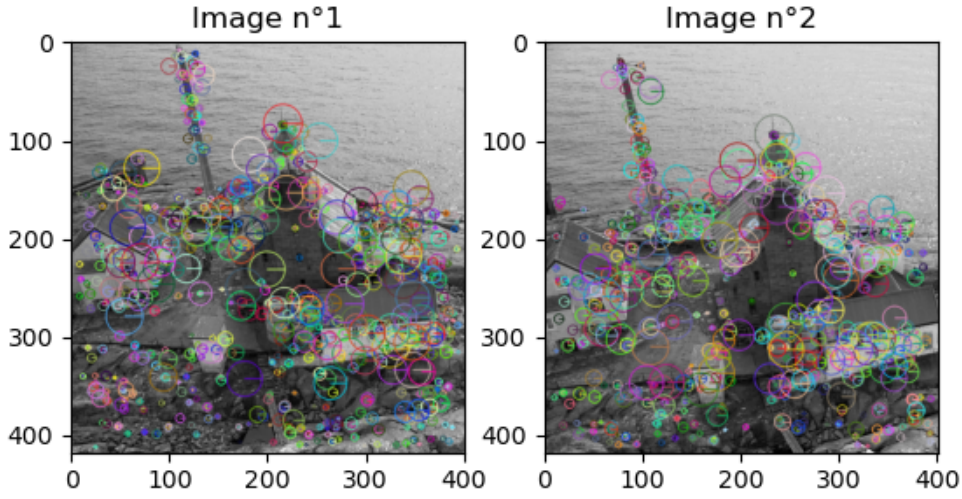

FIGURE 5 – Detect - KAZE

plus de points, mais peut inclure des caractéristiques bruitées.
- nOctaves et nOctaveLayers : contrôlent la robustesse aux changements d'échelle. Plus il y a d'octaves et de couches, plus la détection est susceptible d'être robuste, mais cela peut augmenter le temps de calcul.
- diffusivity : sélectionne le type de filtre de diffusion spatiale, ce qui affecte la manière dont les caractéristiques à différentes échelles sont détectées.

La répétabilité d'un détecteur peut être évaluée visuellement en examinant la cohérence de la localisation des points d'intérêt entre plusieurs images d'une même scène. Idéalement, pour être considérés comme répétables, les points d'intérêt doivent apparaître dans les mêmes emplacements et avec des échelles similaires, malgré les variations d'éclairage, d'échelle, ou de perspective.

Descripteurs et Appariement

Q7

— ORB utilise les descripteurs BRIEF (Binary Robust Independent Elementary Features) mais les améliore pour accroître la robustesse à la rotation. ORB applique une rotation sur les descripteurs BRIEF basée sur l'orientation du point d'intérêt.
- Invariance à la Rotation : Chaque point d'intérêt est associé à un angle d'orientation calculé en utilisant l'intensité des pixels autour du point. Les patchs autour des points d'intérêt sont ensuite tournés en fonction de cet angle, ce qui rend le descripteur invariant à la rotation.
- Invariance à l'Échelle : ORB utilise une pyramide d'images pour détecter les points d'intérêt à différents niveaux d'échelle. Cette invariance est obtenue grâce à la détection de points à multiples échelles et en tenant compte de l'échelle lors de l'appariement des descripteurs.

— KAZE utilise des descripteurs basés sur des gradients locaux qui sont capturés dans une pyramide d'images filtrées non linéairement. Le filtrage non linéaire permet de mieux capturer des caractéristiques à des échelles fines.
- Invariance à la Rotation : KAZE aussi attribue une orientation aux points d'intérêt qui est utilisée pour orienter le patch autour du point avant le calcul du descripteur.
- Invariance à l'Échelle : KAZE est conçu pour être invariant à l'échelle en utilisant une approche de filtrage d'espace d'échelle non linéaire. Les descripteurs sont calculés à différentes échelles en utilisant une pyramide d'échelle, où chaque octave correspond à une version de l'image filtrée à une certaine échelle.

Q8

TABLE 1 – Détection points et calcul descripteurs

	Cross Check	FLANN	Ratio Test
ORB	0.135515875 s	0.101825208 s	: 0.116421875 s
KAZE	0.153695708 s	0.150157167 s	0.151714542 s

— Cross Check : est une méthode où un match est considéré comme valide seulement si le point A dans l'image 1 est le meilleur match pour le point B dans l'image 2 et vice-versa.

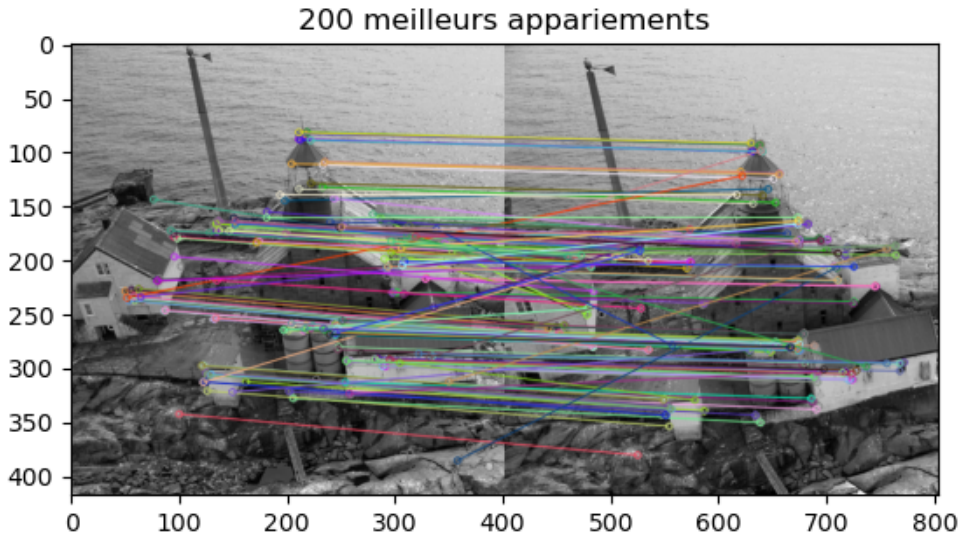

FIGURE 6 – Crosscheck - ORB

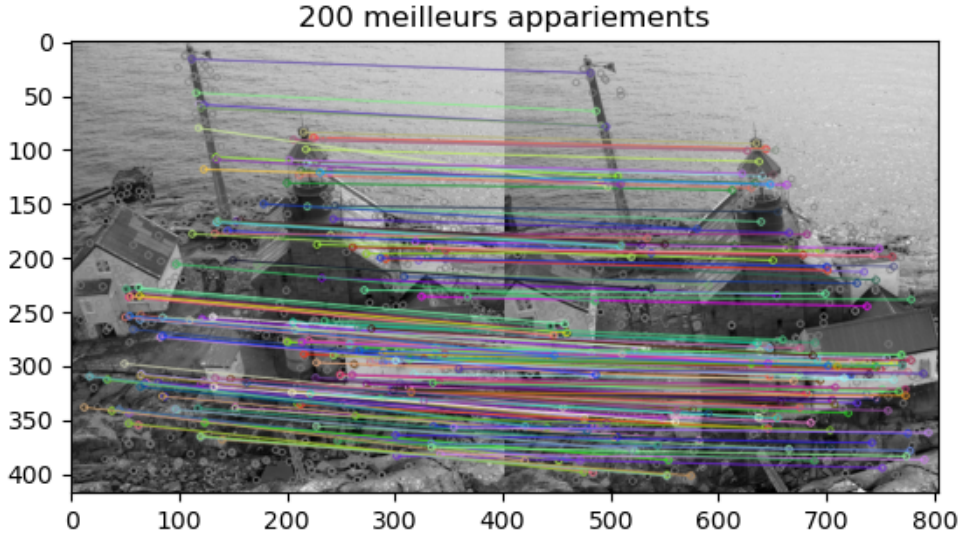

FIGURE 7 – Crosscheck - KAZE

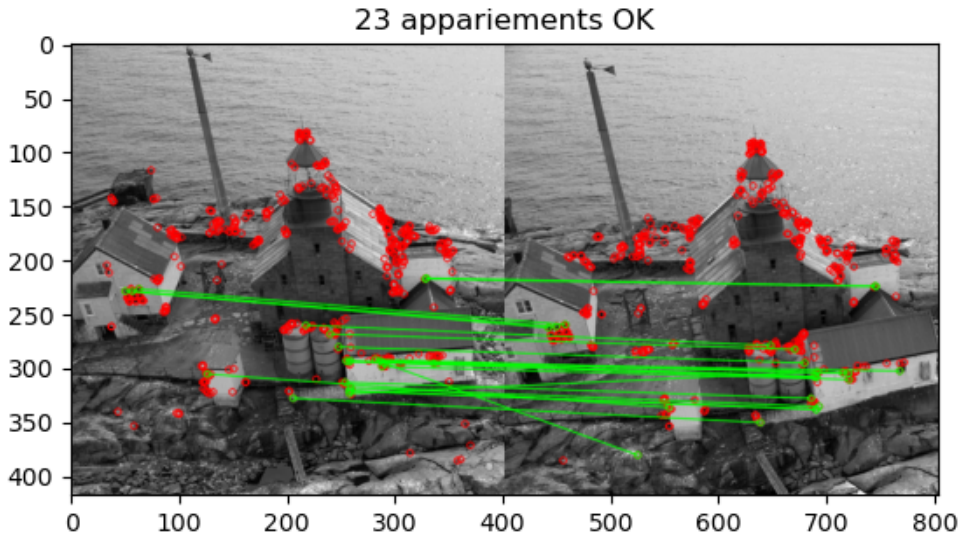

FIGURE 8 – FLANN - ORB

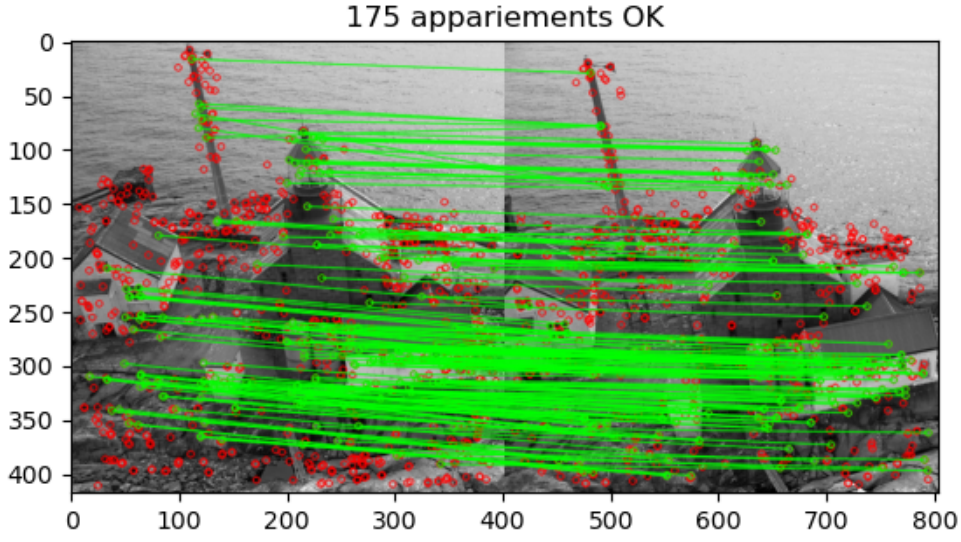

FIGURE 9 – FLANN - KAZE

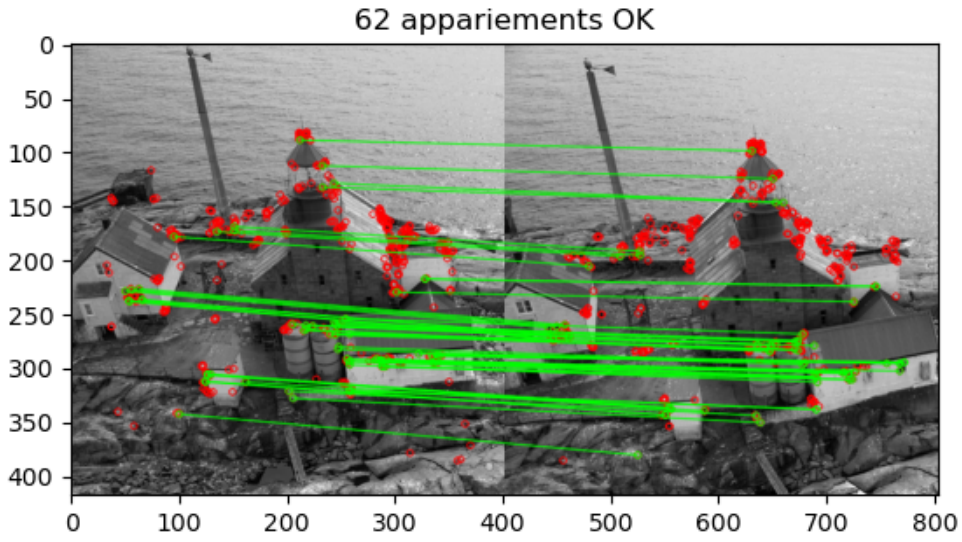

FIGURE 10 – Ratio - ORB

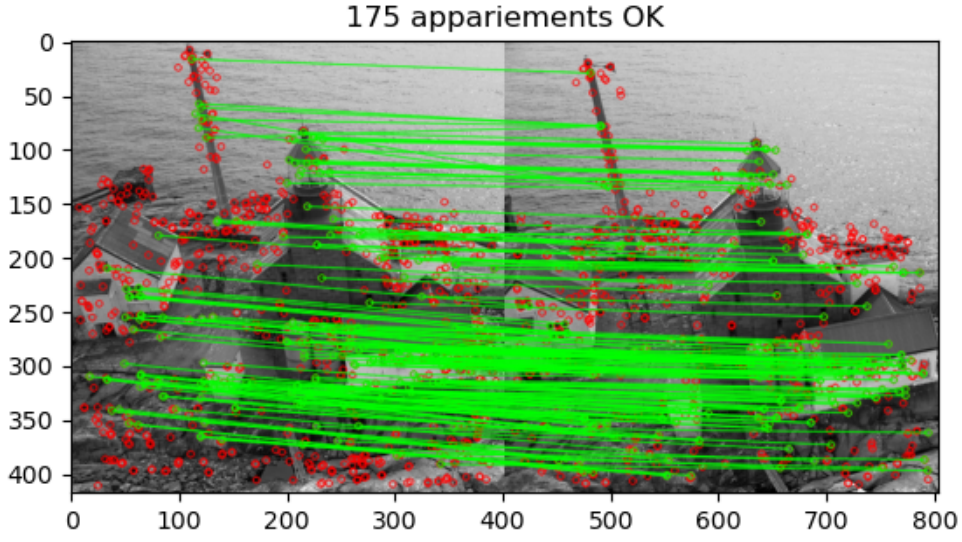

FIGURE 11 – Ratio - KAZE

TABLE 2 – Calcul de l'appariement

	Cross Check	FLANN	Ratio Test
ORB	0.002022625 s	0.00436425 s	: 0.002372791 s
KAZE	: 0.003676166 s	0.021711542 s	0.003567375 s

C'est une approche stricte qui tend à réduire les fausses correspondances mais peut aussi éliminer des correspondances correctes, surtout en présence de bruit ou de changements importants entre les images.
  - ORB : Les matches d'ORB montrent moins de correspondances. Il est généralement beaucoup plus rapide en sacrifiant une partie de la précision.
  - KAZE : Les descripteurs KAZE sont plus détaillés et moins discriminants que ceux d'ORB, en raison de la plus grande sensibilité aux changements dans les descripteurs.

— FLANN : est utilisé pour les descripteurs à haute dimension avec un grand nombre de points d'intérêt.
  - ORB : Il y ades incohérences et une performance suboptimale. FLANN n'est pas le choix le plus adapté car les descripteurs ORB sont binaires et FLANN est optimisé pour des descripteurs en virgule flottante.
  - KAZE : FLANN est plus adapté pour KAZE en raison de la haute dimensionnalité des descripteurs. Il peut fournir une recherche rapide et efficace des correspondances.

— Ratio Test : est basé sur l'idée qu'une bonne correspondance doit avoir une distance beaucoup plus courte à la meilleure correspondance qu'à la deuxième meilleure correspondance. Cette technique est souvent utilisée pour éliminer les correspondances ambiguës.
  - ORB : montre un grand nombre de correspondances. Car les descripteurs ORB, étant binaire, ont souvent des distances qui sont soit très petites pour les correspondances correctes, soit assez grandes pour les autres.
  - KAZE : peut produire des distances plus subtiles entre les correspondances, enayant des descripteurs basés sur des gradients. Le ratio test peut aider à distinguer les meilleures correspondances en se basant sur ces nuances.

En comparaison qualitative, il semble que KAZE fournisse des matches plus précis mais moins nombreux en raison de la nature plus riche des descripteurs, tandis qu'ORB, avec ses descripteurs plus simples et plus rapides, peut produire plus de correspondances mais avec un risque potentiellement plus élevé de faux positifs. Les stratégies de vérification croisée et de test de ratio aident à affiner ces correspondances en éliminant les associations ambiguës ou moins probables.

Q9

Pour évaluer quantitativement la qualité des appariements d'une image déformée par une transformation géométrique connue, ici dans la fig 12 on utilise une transformation affine ou homographie au lieu d'une image déjà transformée mais inconnue. Par exemple, une rotation, une mise à l'échelle, ou une translation...

Puisque on connait la transformation appliquée, on peut prédire où chaque point d'intérêt devrait se trouver dans l'image transformée. On compare la position prédite avec la position obtenue par appariement. La différence entre ces positions donne une mesure de l'erreur pour chaque appariement. Un exemple est dans la fig 13.

Le code complet d'évaluations quantitative se trouve dans le fichier Features-Match-FLANN-

```python
# Définition de la transformation
rows, cols = img1.shape[:2]
M = cv2.getRotationMatrix2D((cols/2, rows/2), 45, 1)
img2 = cv2.warpAffine(img1, M, (cols, rows))
```

FIGURE 12 – Exemple de transformation connue

```python
errors = []
for sublist in good:
    m = sublist[0]
    # Récupérer les points de l'original et de l'image transformée
    original_pt = np.array([pts1[m.queryIdx].pt[0], pts1[m.queryIdx].pt[1], 1])
    transformed_pt = np.array([pts2[m.trainIdx].pt[0], pts2[m.trainIdx].pt[1]])

    # Appliquer la transformation connue au point d'intérêt de l'original
    predicted_pt = M.dot(original_pt)

    # Calculer l'erreur
    error = np.linalg.norm(transformed_pt - predicted_pt[:2])
    errors.append(error)

# Calcul des métriques
mean_error = np.mean(errors)
median_error = np.median(errors)
print("Erreur moyenne :", mean_error)
print("Erreur médiane :", median_error)
```

FIGURE 13 – Évaluation qualitative

quantitative.py.