

# **ROB317 - Analyse et Indexation d'Images**

## **TP1 - Détection et appariement de points caractéristiques**

**Bastien Hubert et Michel Bitar**  
3A - Parcours Robotique

# 1 Format d'images et Convolutions

## 1.1 Q2

Le script `Convolutions.py` calcule la convolution de deux façons : par balayage du tableau 2D (calcul direct) et en utilisant la fonction *filter2D* d'OpenCV. Après compilation du code, il a été constaté que la fonction *filter2D* est plus rapide (0.013s) que la méthode directe (0.2s). Ces deux convolutions donnent une image de contraste rehaussé par rapport à celle d'origine (Figure 1).

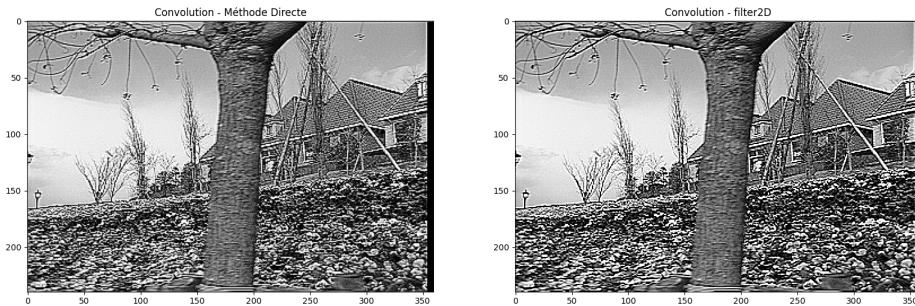


Figure 1: Convolution - méthode direct et filtre2D

Ainsi, le noyau ci-dessous appliqué fonctionne comme un filtre passe haut, permettant le rehaussement des contours et laissant l'image plus bruitée:

$$h = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix} \quad (1)$$

La somme des coefficients de ce noyau est 1, ce qui signifie que la moyenne des niveaux de gris de l'image après traitement est la même que celle de l'image initiale. De plus, ce noyau est la différence entre deux noyaux connus :

$$K = \underbrace{\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}}_A - \underbrace{\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}}_B$$

$A$  est le noyau dont la convolution avec une image est l'identité ( $f * A = f$ ) et  $B$  est un laplacien en 4-connexité, c'est-à-dire un noyau détecteur de contour. La soustraction de l'image par ses contours revient à augmenter le contraste de celle-ci, et par linéarité de la convolution, le noyau  $K$  est un noyau réhausseur de contraste.

## 1.2 Q3

Afin de calculer la convolution qui donne le module du gradient:

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

Il faut tout d'abord calculer les dérivées partielles:

$$\begin{aligned} I_x &= \frac{\partial I}{\partial x} \\ I_y &= \frac{\partial I}{\partial y} \end{aligned} \quad (3)$$

En utilisant les noyaux de convolution  $h_x$  et  $h_y$ :

$$h_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad (4)$$

$$h_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \quad (5)$$

Pour obtenir un affichage correct, il est nécessaire de saturer les valeurs des pixels avec une fonction de la forme :

$$f : \begin{cases} [0, 255] & \rightarrow [0, 255] \\ v & \mapsto \max(0, \min(v, 255)) \end{cases}$$

L'affichage ainsi obtenu est le suivant:

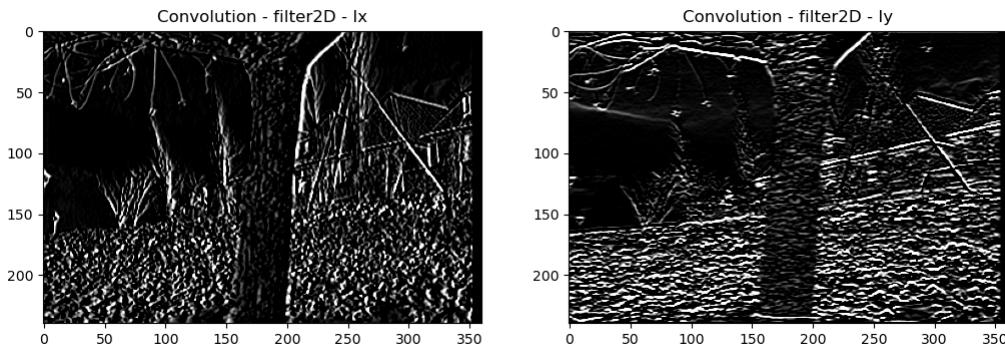


Figure 2: Convolution - filter2D -  $I_x$  et  $I_y$

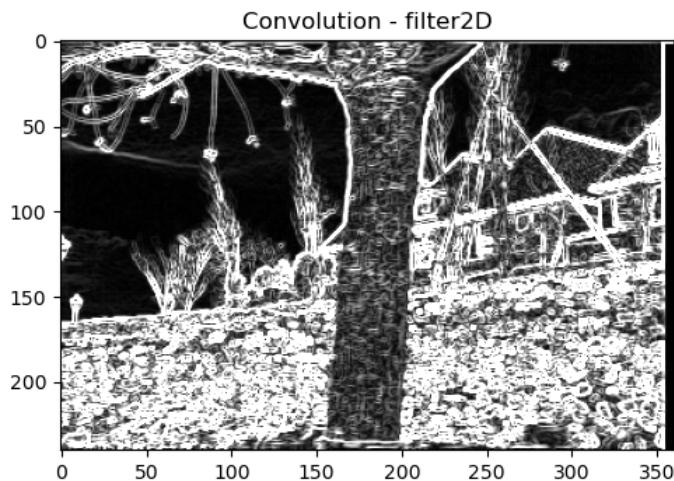


Figure 3: Convolution - filter2D - Norme du gradient

## 2 DéTECTEURS

### 2.1 Q4

Les points d'intérêt de l'image sont calculés par le script *Harris.py* en utilisant la matrice de corrélation :

$$\Xi(x, y) = \begin{pmatrix} \sum I_x^2(x_k, y_k) & \sum I_x(x_k, y_k) \cdot I_k(x_k, y_k) \\ \sum I_x(x_k, y_k) \cdot I_k(x_k, y_k) & \sum I_y^2(x_k, y_k) \end{pmatrix} \quad (6)$$

Après, on utilise la fonction de Harris qui sera calculée en chaque pixel de l'image :

$$\Theta(x, y) = \det(\Xi) - \alpha \cdot \text{trace}^2(\Xi) \quad (7)$$

Avec  $\alpha$  valeur fixe entre 0.04 et 0.06.

Le code complété est le suivant :

```
def Harris( img, W, alpha ):
    ( h, w ) = img.shape;
    # Masques de Sobel
    hx = np.array( [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]] )
    hy = np.array( [[1, 2, 1], [0, 0, 0], [-1, -2, -1]] )
    # Estimations des dérivées premières
    Ix = cv2.filter2D( img, -1, hx )
    Iy = cv2.filter2D( img, -1, hy )
    # Définition de la fonction d'intérêt
    Theta = img * 0
    for i in range( h ):
        for j in range( w ):
            # Calcul de la matrice d'autocorrélation
            # sur la fenêtre de taille W
            H = np.array( [[0, 0], [0, 0]] )
            for i1 in range( i - W, i + W + 1 ):
                for j1 in range( j - W, j + W + 1 ):
                    # Gestion des effets de bord
                    i2, j2 = min( max( 0, i1 ), h - 1 ), \
                        min( max( 0, j1 ), w - 1 )
                    H11 = Ix[ i2, j2 ] * Ix[ i2, j2 ]
                    H12 = Ix[ i2, j2 ] * Iy[ i2, j2 ]
                    H22 = Iy[ i2, j2 ] * Iy[ i2, j2 ]
                    H = H + np.array( [[H11, H12], [H12, H22]] )
            # Calcul de la fonction d'intérêt en ( i, j )
            Theta[ i, j ] = np.linalg.det( H ) - \
```

```

alpha * np.trace( H ) ** 2
return Theta

```

On peut voir sur la figure 4 le résultat obtenu pour une fenêtre  $7 \times 7$ .

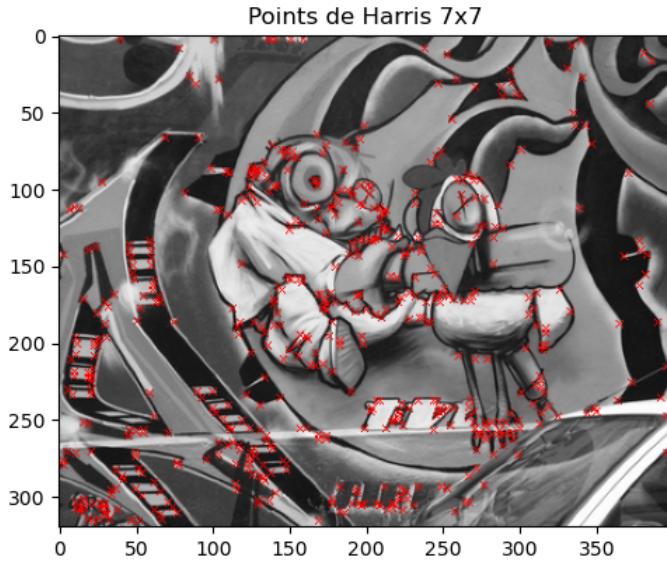


Figure 4: Points de Harris  $7 \times 7$

La dilatation morphologique étend les zones claires de l'image dans un voisinage défini par un élément structurant. Un pixel qui s'est éclairci suite à la dilatation est en fait voisin d'un pixel qui était plus clair que lui dans l'image d'origine. Ce pixel ne correspond donc pas à un maximum local dans l'image originale. C'est de cette manière que la dilatation morphologique participe au calcul des maxima locaux de la fonction d'intérêt *Theta* ; elle permet d'éliminer de la liste des maxima potentiels les voisins des vrais maxima.

## 2.2 Q5

On remarque que les points d'intérêt coïncident avec les points de plus grandes variations de gris dans l'image. Afin de réaliser le calcul sur plusieurs échelles, il faut d'abord appliquer un filtre gaussien pour lisser l'image et changer son échelle à travers un paramètre  $\sigma$ . Ensuite, la fonction Harris sera calculée pour chaque échelle choisie. Le filtre gaussien est:

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (8)$$

La taille de la fenêtre de sommation et la valeur du paramètre  $\alpha$  peuvent grandement impacter cette détection ; Si la fenêtre de sommation est grande, un maximum local de la fonction d'intérêt sera noyé par des informations globales parasites. En conséquence, peu de maxima locaux seront détectés, et la pertinence de ces derniers ne sera pas toujours assurée. Si la fenêtre de sommation est

petite, elle peut mener à un nombre de points détectés très faible. Il est donc nécessaire de choisir la taille de la fenêtre de sommation en adéquation avec l'échelle des points caractéristiques à détecter. Si  $\alpha$  est petite,  $\Theta$  aura plus tendance à prendre de grandes valeurs, ce qui implique une détection des angles là où il n'y a en réalité qu'une ligne. Par contre, si  $\alpha$  est grand,  $\Theta$  risque d'être souvent négatif, ce qui implique un risque de confondre une ligne avec un angle, et donc de ne pas bien détecter les points anguleux de l'image.

La figure 5 montre les variations obtenues lors de la détection des points de Harris en modifiant la taille de la fenêtre de sommation et  $\alpha$  :

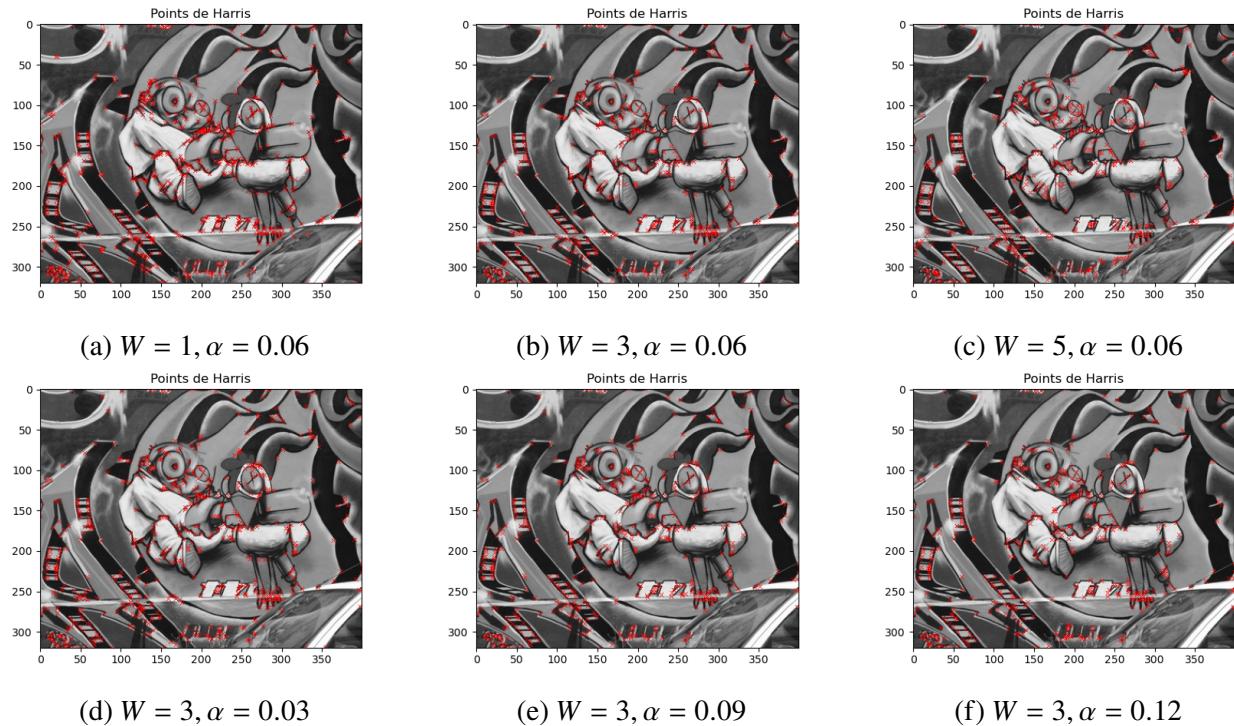


Figure 5: Points de Harris pour différentes valeurs des paramètres  $W$  et  $\alpha$

### 2.3 Q6

*Features\_Detect.py* sert à calculer les zones d'intérêt d'une image en utilisant deux types de détecteurs : *KAZE* et *ORB*. *ORB* est basé sur l'application du détecteur *FAST* à plusieurs échelles caractéristiques, ainsi l'algorithme sélectionne les points qui subissent une grande variation de luminosité relative. Le détecteur *KAZE* utilise le principe de diffusion anisotropique pour travailler sur un espace d'échelles non-linéaire. Ensuite, l'algorithme calcule les extrema locaux du déterminant de la Hessienne afin de déterminer les points d'intérêt. Enfin, une orientation est associée à chaque point sélectionné en calculant l'orientation du gradient dominant dans son voisinage. La compilation du script *Features\_Detect.py* permet de la comparaison de ces deux

méthodes à travers de leur application à deux images pareils. Le détecteur *KAZE* a identifié les points d'intérêt dans l'image 1 et l'image 2 en 0.66s:

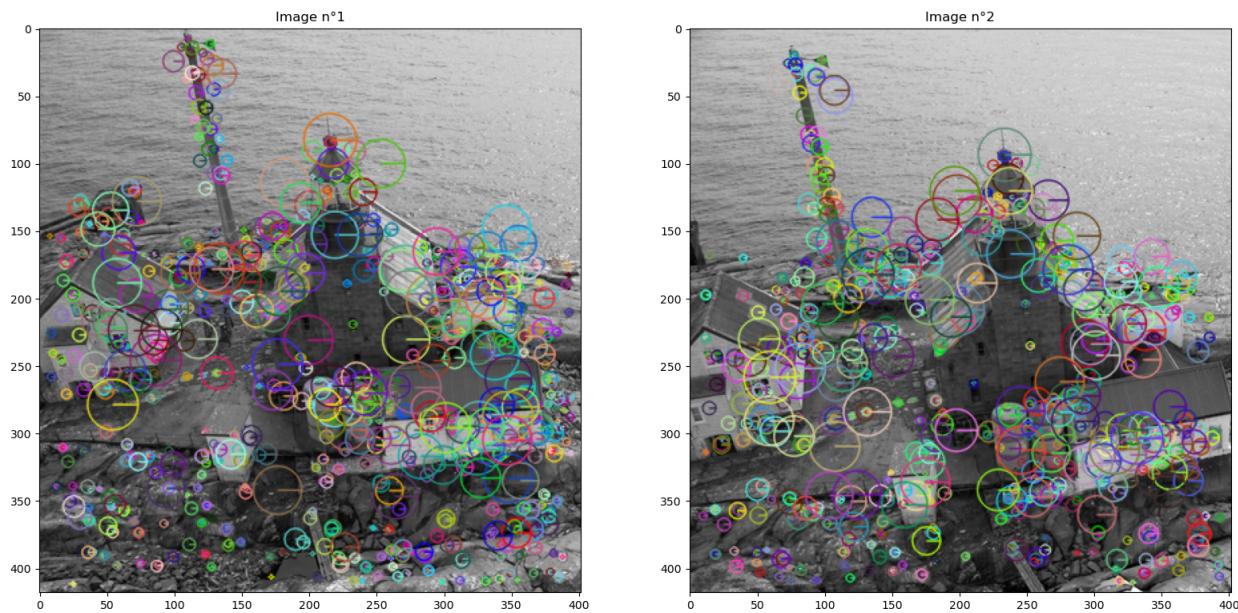


Figure 6: KAZE

Le détecteur *ORB* (pour un nombre de features retenues fixé à 500) a identifié les points d'intérêt dans l'image 1 et l'image 2 en 0.01s:

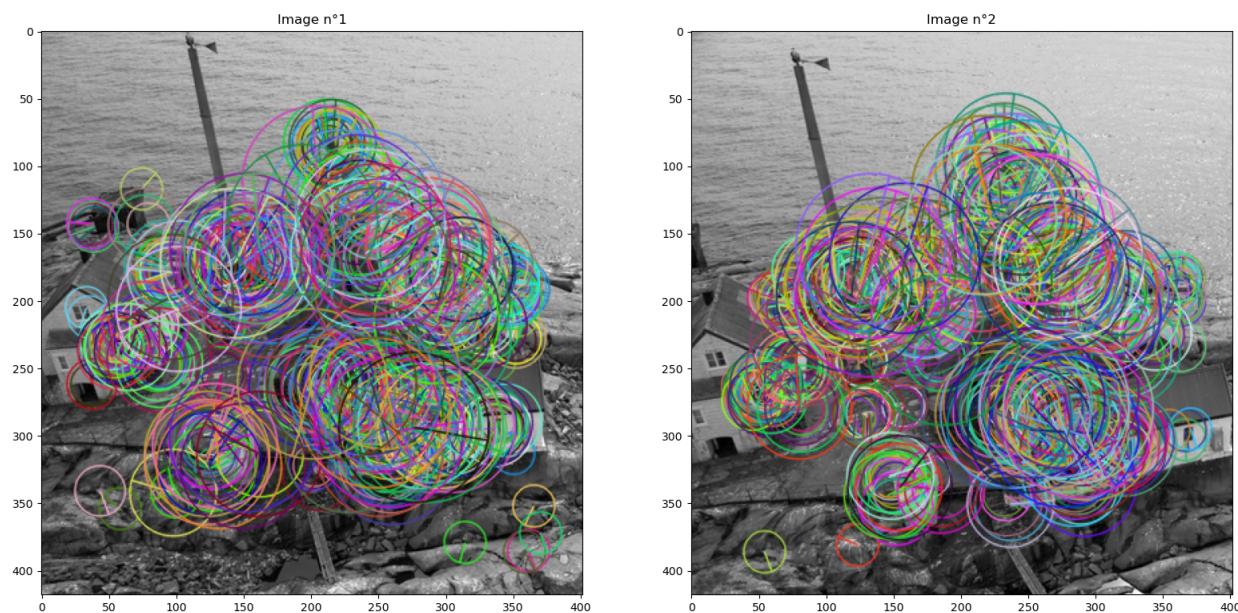


Figure 7: ORB

Les paramètres propres au détecteur *ORB* ayant une influence sur la détection des points d'intérêt sont:

- Le voisinage définissant la taille du cercle de pixels utilisé par la méthode de détection FAST. En pratique, le détecteur ORB utilise un cercle de diamètre de 9 pixels, qui contient donc 16 pixels. Plus le cercle de pixels est grand, plus un point d'intérêt local pourra être perdu au milieu de variations d'intensité globales.
- Le seuil sur la différence entre l'intensité du pixel central et l'intensité des pixels situés sur son voisinage circulaire. Plus le seuil est faible, plus facilement un pixel sera interprété comme un point d'intérêt.
- La taille du patch utilisé pour calculer l'orientation des points d'intérêt. La valeur de l'orientation d'un point d'intérêt est très influencée par la taille de ce patch.
- Le nombre de niveaux de la pyramide multi-échelle de l'image sur laquelle sont détectés les points d'intérêt.
- Le facteur d'échelle de la pyramide multi-échelle, qui définit la manière dont la résolution de l'image est réduite entre chaque niveau de la pyramide.

Les différents paramètres relatifs au détecteur *KAZE* sont:

- Le seuil sur la valeur du déterminant du Hessien de l'image à partir duquel un pixel est accepté comme zone d'intérêt. Plus ce seuil sera faible, plus un pixel sera facilement indiqué en tant que pixel d'intérêt.
- Le nombre d'octaves selon lequel l'espace d'échelle de l'image est discrétisé.
- Le nombre de sous-niveaux selon lequel l'espace d'échelle de l'image est discrétisé.
- Le type de diffusivité utilisé lors de la création de l'espace d'échelle non-linéaire de l'image par diffusion anisotrope. Ce paramètre influe la manière dont les images sont lissées aux différentes échelles de l'espace d'échelle de l'image. Plus la diffusion est importante, plus le lissage de l'image sera marqué entre deux échelles successives.

### 3 Descripteurs et Appariement

#### 3.1 Q7

Le descripteur associé aux points ORB (oriented FAST and rotated BRIEF) est très similaire au descripteur BRIEF, mais dispose en plus d'un mécanisme de compensation des rotations en "guidant" BRIEF en fonction de l'orientation de points-clés, ce qui le rend indépendant des rotations relatives des images. Ce guidage est réalisé par la multiplication matricielle d'une matrice de rotation d'angle donné par le détecteur et de la matrice contenant les coordonnées 2D des pixels détectés par le détecteur. Pour une meilleure invariance par rotation, le descripteur cherche en plus à faire des paires de forte variance avec des échantillons décorrélés, de sorte que chaque nouvelle paire apporte le plus d'information possible. Ceci se fait par évaluation exhaustive gourmande de toutes les paires.

À l'inverse, le descripteur associé aux points KAZE est proche de SURF, en ce sens qu'il utilise la même technique pour déterminer l'orientation locale de l'image par pondération gaussienne des dérivées premières en x et en y des échantillons. Le descripteur effectue ensuite la rotation de l'échantillon selon l'orientation dominante, et calcule le vecteur de description à partir des dérivées premières et de leur norme, qui est ensuite normalisé à 1 pour être invariant par changement de contraste. Puisqu'il fait appel au descripteur SURF, le descripteur associé aux points KAZE s'adapte très bien aux changements d'échelle.

Le détecteur ORB est invariant par changement d'échelle grâce à un FAST multi-échelle pyramidal, et invariant par rotation grâce au calcul d'une orientation caractéristique à partir du centre de gravité de l'imagette étudiée.

Le descripteur ORB est invariant par changement d'échelle car il utilise BRIEF, que l'on sait très résistant aux changements d'échelle grâce à l'utilisation d'une pyramide multi-échelle de l'image, et invariant par rotation grâce à son mécanisme de compensation des rotations et de recherche gloutonne des paires optimales.

Le détecteur KAZE est invariant par changement d'échelle de part son mécanisme de diffusion anisotropique (l'utilisation des maxima locaux du déterminant de la matrice Hessienne pour réduire la conductance de l'image passe très bien aux changements d'échelle), et invariant par rotation car le laplacien et le hessien sont tous les deux invariants par rotation.

Enfin, le descripteur KAZE est invariant par changement d'échelle car il utilise un flou gaussien dans un espace non-linéaire et peut s'adapter à l'échelle du point d'intérêt, et invariant par rotation grâce à la détection de l'orientation dominante des échantillons.

#### 3.2 Q8

Dans les scripts *Features\_Match\_CrossCheck.py* et *Features\_Match\_RatioTest.py*, le calcul des paires se fait par une mise en correspondance exhaustive de toutes les paires puis par sélection des meilleures. La méthode *cross\_check* consiste à calculer les correspondances dans les deux

sens et de ne conserver les appariements que si les résultats de ces deux calculs coïncident. En plus, les erreurs d'appariement sont réduits. Dans cette méthode, pour un détecteur ORB, le calcul de l'appariement a duré 0.013s et la détection des points et le calcul des descripteurs a duré 0.039s. Pour un détecteur KAZE, le calcul de l'appariement a duré 0.013s et la détection des points et le calcul des descripteurs a duré 0.66s.

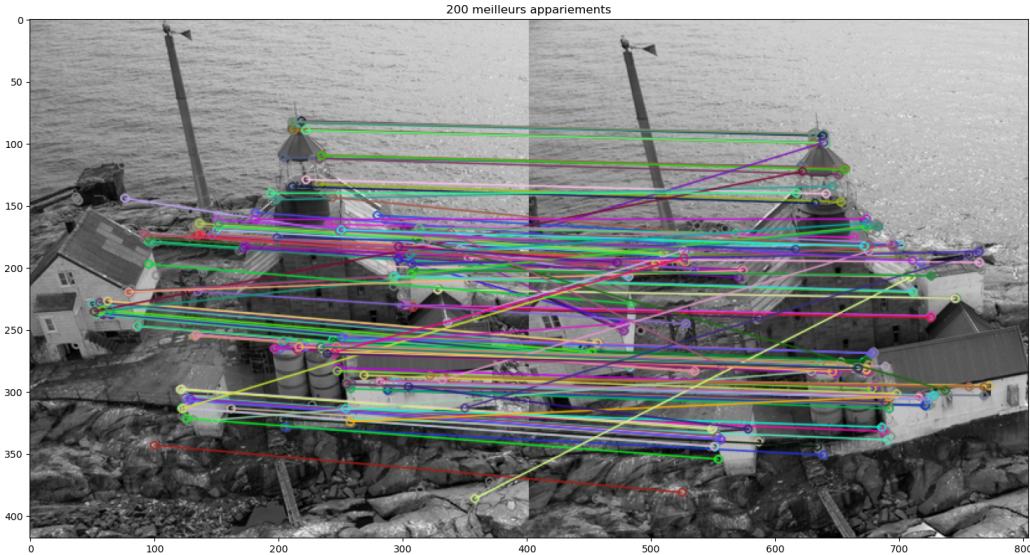


Figure 8: Méthode Cross\_Check ORB

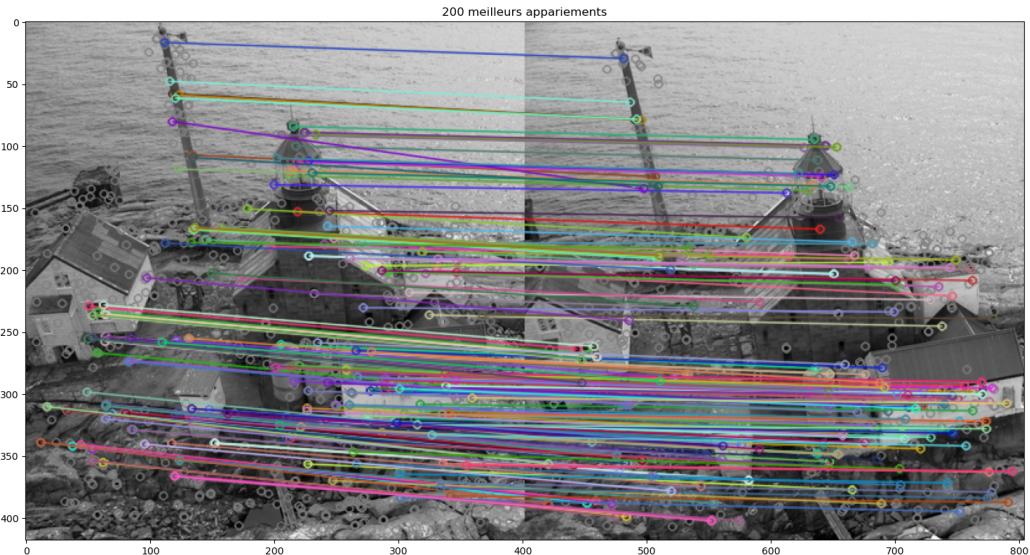


Figure 9: Méthode Cross\_Check KAZE

La méthode *ratio\_test* cherche les pairs de points des deux images qui ont les plus faibles distances entre eux et les conserve. On peut voir dans le code (figure 10) que toutes les paires séparées par

une distance plus grande que 70% de la plus grande distance sont éliminés.

```

70  for m,n in matches:
71      if m.distance < 0.7*n.distance:
72          good.append([m])
73  t2 = cv2.getTickCount()
74  time = (t2 - t1)/ cv2.getTickFrequency()
75  print("Calcul de l'appariement : ",time,"s")
76

```

Figure 10: Code d'élimination des distance > 70%

Dans cette méthode, pour un détecteur ORB, le calcul de l'appariement a duré 0.002s et la détection des points et le calcul des descripteurs a duré 0.0123s. Pour un détecteur KAZE, le calcul de l'appariement a duré 0.003s et la détection des points et le calcul des descripteurs a duré 0.39s.

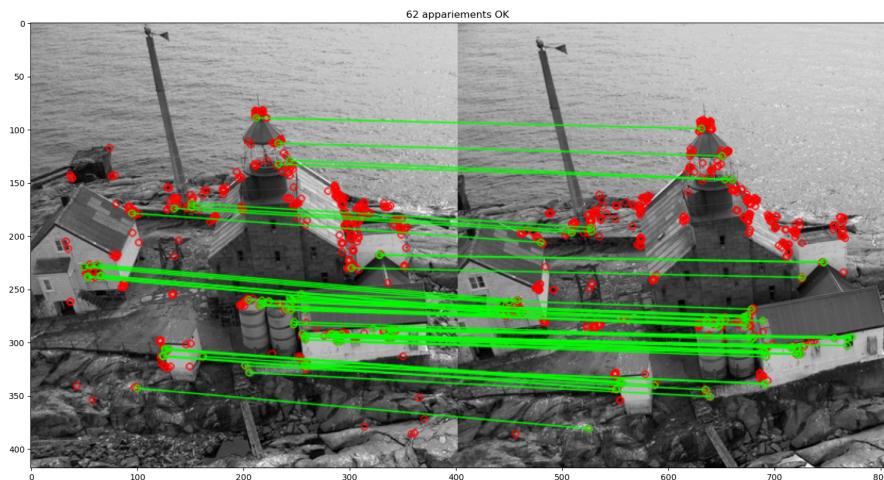


Figure 11: Méthode Ratio Test ORB

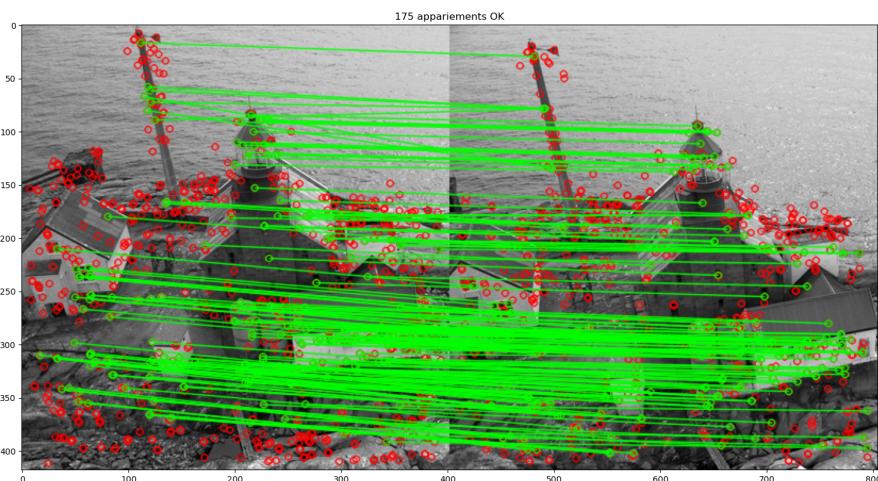


Figure 12: Méthode Ratio Test KAZE

Le script *Features\_Match\_FLANN.py* s'appuie sur la bibliothèque FLANN (Fast Library for Approximate Nearest Neighbors) qui est un algorithme d'appariement d'images pour des recherches rapides de voisins les plus proches approximatifs dans des espaces de grande dimension. Ces méthodes projettent les caractéristiques de grande dimension dans un espace de dimension inférieure, puis génèrent les codes binaires compacts. Ce matcher peut être plus rapide lors de la mise en correspondance d'une grande collection d'entraînement que le matcher force brute. Dans cette méthode, pour un détecteur KAZE, le calcul de l'appariement a duré 0.016s et la détection des points et le calcul des descripteurs a duré 0.32s. Pour un détecteur ORB, le code n'a pas marché et donc on n'a pas eu des résultats. En fait, on utilise ici la méthode des *kd\_trees* qui vient de la bibliothèque FLANN. D'autre part, ORB repose sur des descripteurs qui consistent en des vecteurs de valeurs binaires, ce qui est mal adapté aux kd-trees et c'est pour cela qu'on n'a pas eu de bon résultats avec le détecteur ORB. Finalement, KAZE est un descripteur basé sur une chaîne, donc il faut utiliser ici la distance L1-norm ou L2-norm. ORB est un descripteur binaire et il faut utiliser ici la distance Hamming.

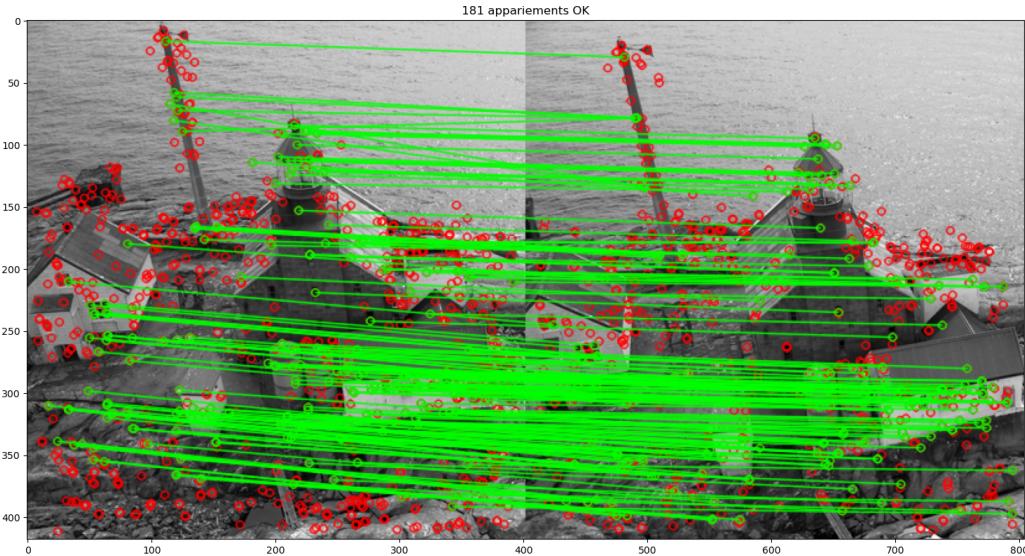


Figure 13: Méthode FLANN KAZE

### 3.3 Q9

Considérons une transformation géométrique connue sous la forme d'une rotation et d'un changement d'échelle. La matrice de transformation correspondant peut être obtenue grâce à la fonction `cv2.getRotationMatrix2D`, tandis que la fonction OpenCV `cv2.warpAffine` permet d'appliquer la transformation géométrique à n'importe quelle image d'origine afin d'obtenir une image déformée.

On effectue un appariement entre l'image d'origine et l'image déformée, et on sauvegarde deux matrices  $M_{\text{origine}}$  et  $M_{\text{assoc}}$  contenant les coordonnées en x et en y respectivement des points détectés sur l'image d'origine et des points associés par appariement sur l'image déformée. On calcule

ensuite le produit matriciel entre  $M_{origine}$  et la matrice de transformation affine pour obtenir  $M_{th}$ , qui représente les coordonnées des points détectés dans le nouvel espace de l'image déformée.

Théoriquement, le point associé à tous points de l'image d'origine par l'appariement devrait être confondu avec celui obtenu par transformation affine à partir du point d'origine correspondant. La mesure des erreurs d'appariement peut se calculer en sauvegardant dans une liste la distance entre chaque paire de points associé/théorique, puis en effectuant une analyse statistique sur cette liste. Le calcul de la moyenne, de la médiane ou encore de l'écart-type permettent de mesurer quantitativement la qualité des appariements obtenus.