

ROB317 - ANALYSE ET INDEXATION D'IMAGES

TP1 : Détection et Appariement de Points

Caractéristiques

29 octobre 2019

Gabriel Henrique Riqueti

Victor Kenichi Nascimento Kobayashi

ENSTA IP Paris

Format d'images et Convolutions

Q1 :

Tout d'abord, le fichier `Convolutions.py` utilise deux méthodes pour calculer la convolution de l'image 2D `FlowerGarden2.png` : la méthode directe et la méthode `filter2D`.

La méthode directe commence avec la fonction `copyMakeBorder` de la librairie `cv2` (`OpenCV2`). Cela reçoit les paramètres `src=img, top=0, bottom=0, left=0, right=0, borderType=cv2.BORDER_REPLICATE` qui déterminent que l'image de destin est une copie de l'image source sans contour additionnel.

Ensuite, on calcule la valeur de la convolution de chaque élément qui n'est pas contour

de l'image source par le noyau $\begin{vmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{vmatrix}$. Comme les valeurs peuvent être plus négatives ou plus grandes que 255, on change les valeurs négatives pour 0 et les valeurs supérieures à 255 pour 255.

En plus, la méthode `filter2D` est plus compacte. D'abord, on a crée le même noyau utilisé par la méthode directe. Ensuite, la fonction `filter2D` reçoit les paramètres `src=img, ddepth=-1, kernel=`
 $\begin{vmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{vmatrix}$ et l'image de destin est la corrélation de l'image source par le noyau donné.

Notez que le résultat est le même, puisque le noyau est symétrique et par la définition de les opérations de convolution et de corrélation. Notez que la convolution est la corrélation avec le noyau transposé.

La corrélation est définie comme :

$$(F \circ I)(x, y) = \sum_{i=-N}^N \sum_{j=-N}^N F(x+i, y+j)I(i, j)$$

La convolution est définie comme :

$$(F * I)(x, y) = \sum_{i=-N}^N \sum_{j=-N}^N F(x - i, y - j)I(i, j)$$

En ce qui concerne l'efficace de chaque méthode, la différence de temps d'exécution est très grand. La méthode directe prend 0,19 s lorsque la méthode avec la fonction `filter2D` prend 0,00062 s pour calculer la convolution de l'image 2D.

Pour les fonctions de lecture et d'affichage, on utilise `imread` et `imshow` de la librairie `cv2`. La fonction `imread` reçoit les paramètres `filename="..../Image_Pairs/FlowerGarden2.png"` et `flag= cv2.IMREAD_GRAYSCALE` (égal à 0). Ainsi, il retourne une matrice avec le niveau de gris de chaque pixel de l'image `FlowerGarden2.png`. Le type de chaque pixel est `uint8` pourtant on a changé le type pour `float64`. La fonction `imshow` montre l'image correspondant à la matrice donnée comme paramètre en échelle de gris. Comme le type flottant de 64-bits n'est pas bien spécifié, il faut attribuer les valeurs minimum et maximum de la matrice, dans le cas, 0 et 255.

Par ailleurs, de la librairie `matplotlib`, le fichier utilise les fonctions suivantes : `subplot`, `title` et `show`. La fonction `subplot` reçoit comme paramètres les nombre de lignes, les nombre de colonnes et l'index qui est un numéro entre 1 et le nombre de lignes multiplié par le nombre de colonnes. Les graphiques sont placés de gauche à droite et de haut en bas. Puis, la fonction `title` donne un titre au graphique.

Enfin, la fonction `show` montre les figures déjà définies par la fonction `figure` qui ouvre une fenêtre à chaque fois qui est appelée.

Q2 :

Le noyau de convolution fourni est $\begin{vmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{vmatrix}$.

Étant donnés l'image I et le noyau $k = \begin{vmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{vmatrix}$.

Selon la propriété distributive de la convolution par rapport l'addition et la soustraction :

$$I \circ k = I \circ (m - h - v) = I \circ m - I \circ h - I \circ v$$

$$\text{où } m = \begin{vmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{vmatrix}, h = \begin{vmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{vmatrix} \text{ et } v = \begin{vmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{vmatrix}$$

On peut percevoir que $I \circ m = I$, $I \circ h = \frac{\partial^2 I}{\partial x^2}$ et $I \circ v = \frac{\partial^2 I}{\partial y^2}$ par la méthode de différences finies.

$$\text{On peut considérer aussi que } \Delta I = \frac{\partial^2 I}{\partial x^2} \frac{\partial^2 I}{\partial y^2} = \begin{vmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{vmatrix}.$$

Alors la convolution de l'image est l'image soustraite de son laplacien en 4-connexité.

$$I \circ k = I - \Delta I$$

Comme on utilise des voisinages 3x3 pour calculer les laplaciens et son noyau augmente les éléments voisins et réduit l'importance du élément central, le résultat est donc un rehaussement de contraste de l'image original.

En sachant qu'on soustraite de l'image originale cette image avec rehaussement de contraste, l'image finale est l'original avec un rehaussement de contraste. On peut trouver les

images affichées par les méthodes directes et filter2D dans la Figure 1.

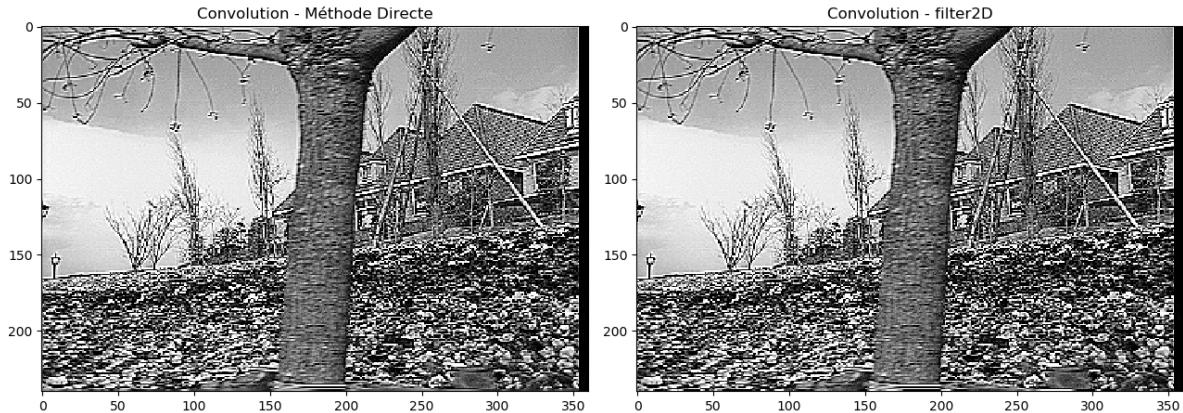


FIGURE 1: Convolution de l'image FlowerGarden2.png équivalent de l'identité moins le laplacien de 4-connexité par les méthodes directe (à gauche) et filter2D (à droite).

Q3 :

Dans le code `Harris.py`, on a ajouté une algorithme pour calculer la dérivé partielle horizontale de l'image par convolution avec le Sobel horizontal par les méthodes directe et indirecte et une algorithme pour calculer la norme euclidienne du gradient de l'image par convolution du filtres de Sobel horizontal et vertical par les méthodes directe et indirecte.

La structure de ces algorithmes se rassemble beaucoup avec la structure du algorithme pour calculer le laplacien de l'image. Donc, il ne vaut mieux que remarquer les nuances.

Les noyaux utilisés sont les masques de Sobel horizontale

$$\begin{vmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix}$$

$$\text{et verticale } \begin{vmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{vmatrix}.$$

Pour utiliser la fonction `filter2D` afin d'implémenter la convolution au lieu de la corrélation, il faut réfléchir horizontalement et verticalement les noyaux à grâce que chaque masque de Sobel n'a que la symétrie horizontale ou verticale. La fonction `flip` de la librairie `numpy` a été

utilisée pour réaliser cette opération.

Concernant l'affichage, on remarque que les valeurs de l'image résultante de la dérivé partielle horizontal de l'image originale peuvent être négatives ou plus grand que 255. Par conséquent, pour bien montrer l'image, nous avons besoin de normaliser l'image résultante. C'est la raison pour laquelle, qu'on a substitué l'image résultante par cela moins son minimum, puis multiplié par 255 et finalement divisé par son maximum moins son minimum. Ainsi, l'image finale a que des valeurs entre 0 et 255. La Figure 2 montre les images finales affichées.

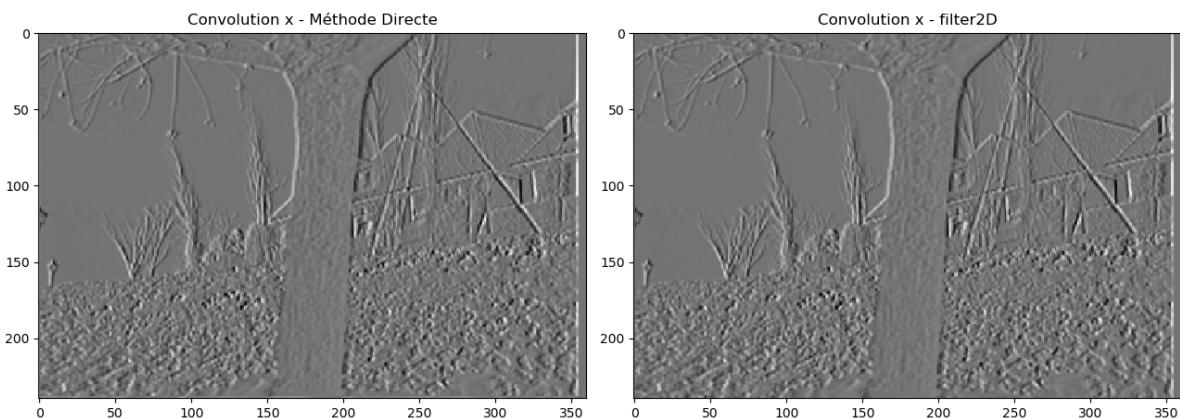


FIGURE 2: Dérivé partielle horizontale de l'image FlowerGarden2.png par convolution avec le filtre de Sobel horizontal par les méthodes directe (à gauche) et filter2D (à droite).

Dans le cas de la norme du gradient, le résultat a toujours des valeurs non négatives, toutefois, ils peuvent être plus grand que 255. Donc, on normalise par la multiplication des valeurs de cette image de la norme du gradient par 255 et la division par le maximum de cette image. L'image finale de cet algorithme est présenté sur la Figure 3.

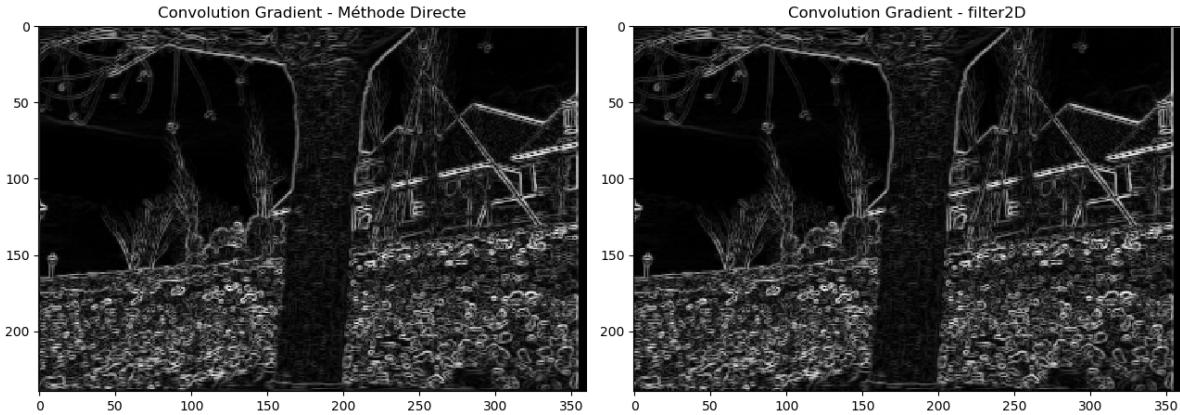


FIGURE 3: Norme euclidienne du gradient de l'image FlowerGarden2.png par convolution avec le filtres de Sobel horizontal et vertical par les méthodes directe (à gauche) et filter2D (à droite).

Détecteurs

Q4 :

Premièrement, il y a la fonction `cv2.dilate` qui calcule le θ de dilatation qui prend deux paramètres prédéfinis : θ et `se`. D'abord, θ s'agit d'une copie de l'image analysée, ensuite, `se` (seuil) est un paramètre qui est typiquement défini 1 % de θ_{max} .

La fonction de dilatation est utilisée pour augmenter les régions de maxima locaux puisque et ensuite on supprime les points de maxima non locaux avec la ligne de code ci-dessous :

— `Theta_maxloc[Theta < Theta_dil] = 0.0`

D'une manière simple, cette fonction va mettre la valeur 0 aux pixels qui sont plus petits que les maxima locaux.

Finalement, une deuxième dilatation qui cette fois utilise le noyau

$$\begin{vmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{vmatrix}$$

Cette fonction est appliquée à la fin de mettre en évidence les points d'intérêt de la fonction de Harris comme montré sur la Figure 4.

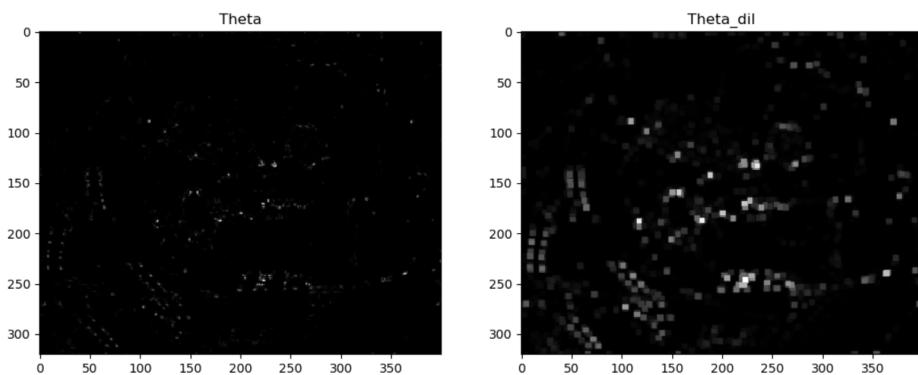


FIGURE 4: Theta avant et après la dilatation

Q5 :

D'abord les résultats qu'on a obtenu avec notre détecteur de Harris a été cet image qui est présentée ci-dessous par la Figure 11 le temps d'exécution de 28.0191 secondes.

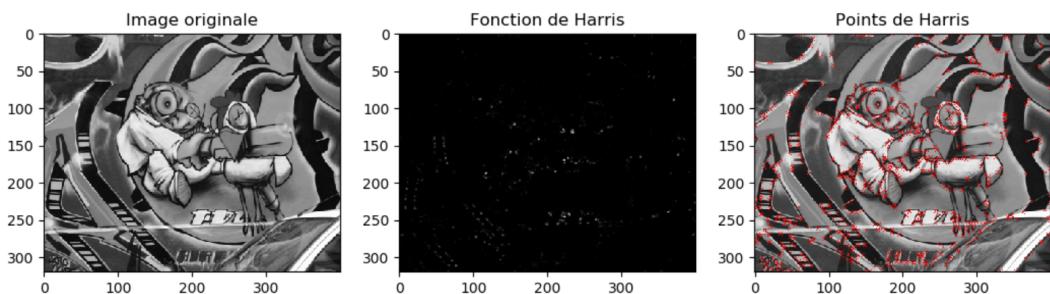


FIGURE 5: Points d'intérêts

Comme prévu, la fonction de Harris sert à détecter les coins. La sous Figure *Points de*

Harris de la Figure 11 montre tous les coins détectés sur cette image par l'algorithme qui a réussi à marquer ces points.

Par ailleurs, les paramètres qui sont possibles de changer pour puisse voir les changements sur l'image sont :

1. `d_maxloc` : la largeur et la hauteur du noyau qui doivent être positives et impaires.
2. `alpha` : un paramètre de pondération déterminé de façon empirique et qui varie, en général, entre 0.04 et 0.06.

En variant la valeur de ces paramètres on a :

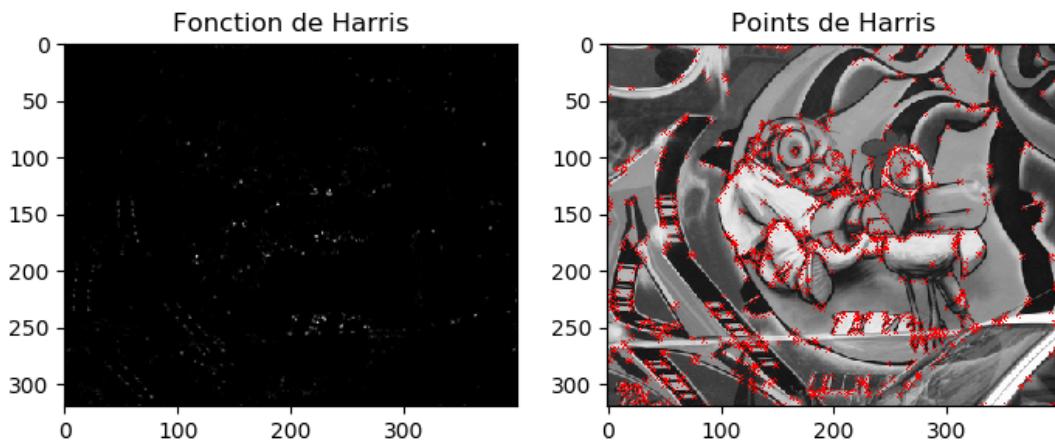


FIGURE 6: $\alpha = 0.04$ et $d_maxloc = 3$

À partir de cet ensemble d'images, il est possible de rendre en compte que la variation du paramètre α ne change pas beaucoup en ce qui concerne la détection de coins par la fonction de Harris. Cependant, la variation du paramètre `d_maxloc` change beaucoup plus.

Par les images, on voit qu'au fur et à mesure qu'on augmente la valeur de `d_maxloc` ont a moins de points d'intérêt détectés et , en fait, il y a du sens car on augmente la taille de la fenêtre, par conséquent il est plus difficile de détecter les petits détails.

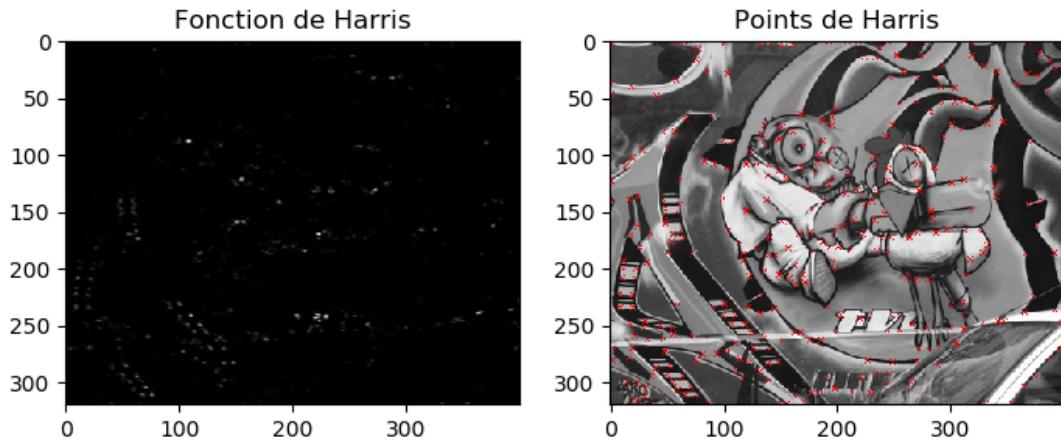


FIGURE 7: $\alpha = 0.04$ et $d_maxloc = 11$

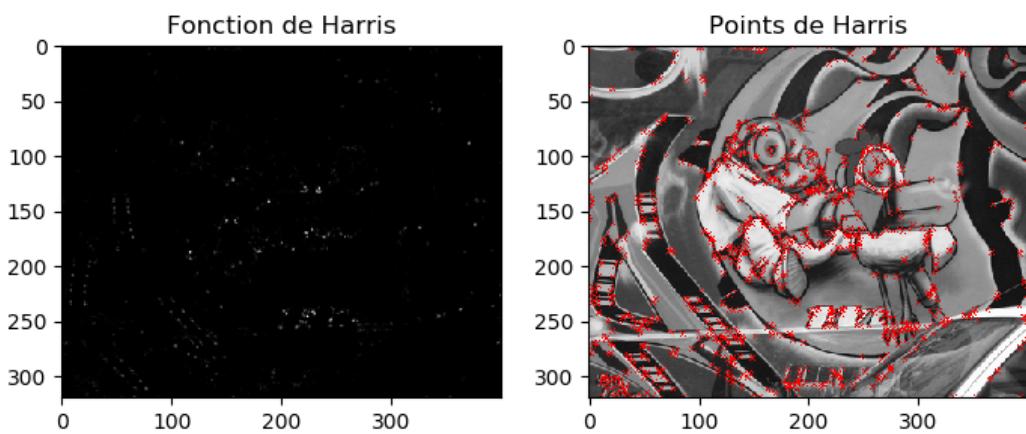


FIGURE 8: $\alpha = 0.06$ et $d_maxloc = 3$

Q6 :

ORB

L'algorithme ORB s'agit de l'ensemble du détecteur oFAST et du descripteur rBRIEF. Le détecteur oFAST ou FAST Keypoint Orientation reçoit le seuil comme paramètre. Ce détecteur choisit les points d'intérêt de l'image en trouvant la différence de niveau de gris entre le centre de la voisinage circulaire et les pixels du contour de cette voisinage, ces différences doivent former deux ensembles contigus : une supérieure au seuil et autre inférieure au opposé du

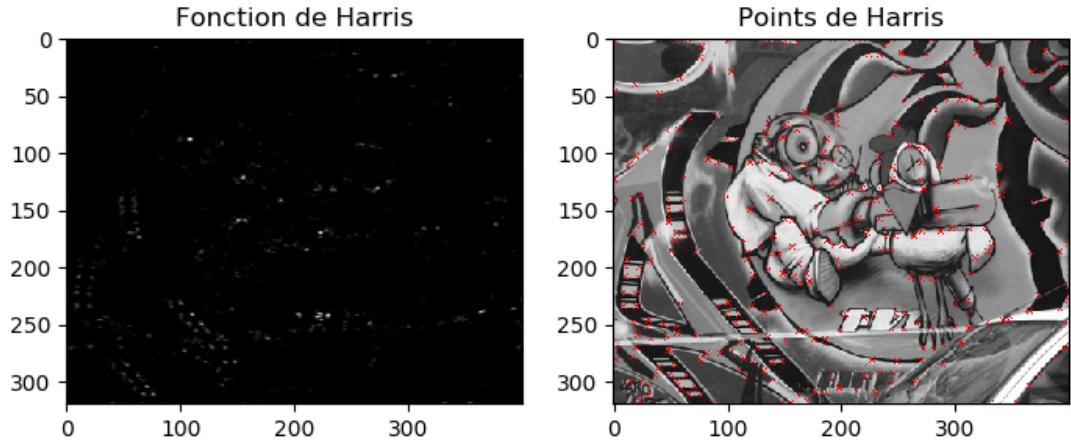


FIGURE 9: $\alpha = 0.06$ et $d_{maxloc} = 11$

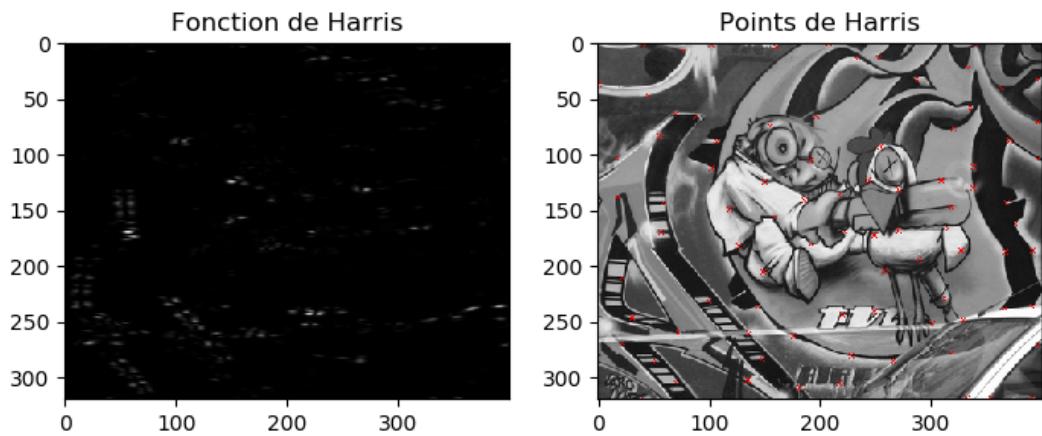


FIGURE 10: $\alpha = 0.04$ et $d_{maxloc} = 31$

seuil.

Ensuite, on utilise le méthode de Harris qui évalue une mesure de coins pour sélectionner la quantité de points spécifiée. Enfin, on ajoute une orientation au points d'intérêt. On calcule la position et la orientation du centroïde à travers de les équations :

$$C = \left(\frac{\sum_{x,y} xI(x,y)}{\sum_{x,y} I(x,y)}, \frac{\sum_{x,y} yI(x,y)}{\sum_{x,y} I(x,y)} \right)$$

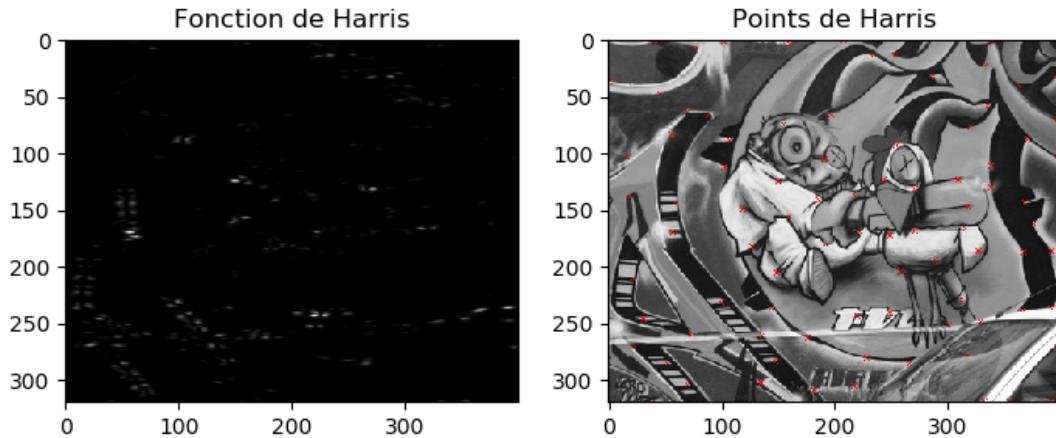


FIGURE 11: $\alpha = 0.06$ et $d_maxloc = 31$

$$\theta = \text{atan}2\left(\frac{\sum_{x,y} yI(x,y)}{\sum_{x,y} xI(x,y)}\right)$$

où x et y sont des positions horizontal et vertical de chaque pixel de la voisinage circulaire détectée et I est le niveau de gris de ces pixels. Le atan2 est la fonction arctan en considérant les quatre quadrants.

Dans la librairie OpenCV, on utilise le constructeur `ORB_create`, qui reçoit les paramètres prédéfinis `nfeatures= 500` , `scaleFactor= 1.2` , `nlevels= 8` , `edgeThreshold= 31` , `firstLevel= 0` , `WTA_K= 2` , `scoreType=ORB_HARRIS_SCORE` , `patchSize= 31` , `fastThreshold= 20` et retourne l'objet de la classe `ORB` pour implémenter le détecteur ORB.

En outre, `nfeatures` est le nombre de points d'intérêt choisis. Comme s'il y a moins points d'intérêt que `nfeatures`, tous les points d'intérêt seront montrés, qui sera différent de la valeur de `nfeatures`.

Pour commencer, on a utilisé le détecteur ORB avec les `nFeatures=250` , `scaleFactor=2` , `nLevels=3` et les autres paramètres comme les valeurs prédéfinies. Le résultat est la Figure 12. Comme on n'utilise que 3 niveaux et une haut taux de décimation de pyramide ou `scaleFactor`, on a une algorithme très vite mais avec bas exactitude. De plus, c'est

difficile d'évaluer les points d'intérêt parce que il y a beaucoup des bouilles grand.

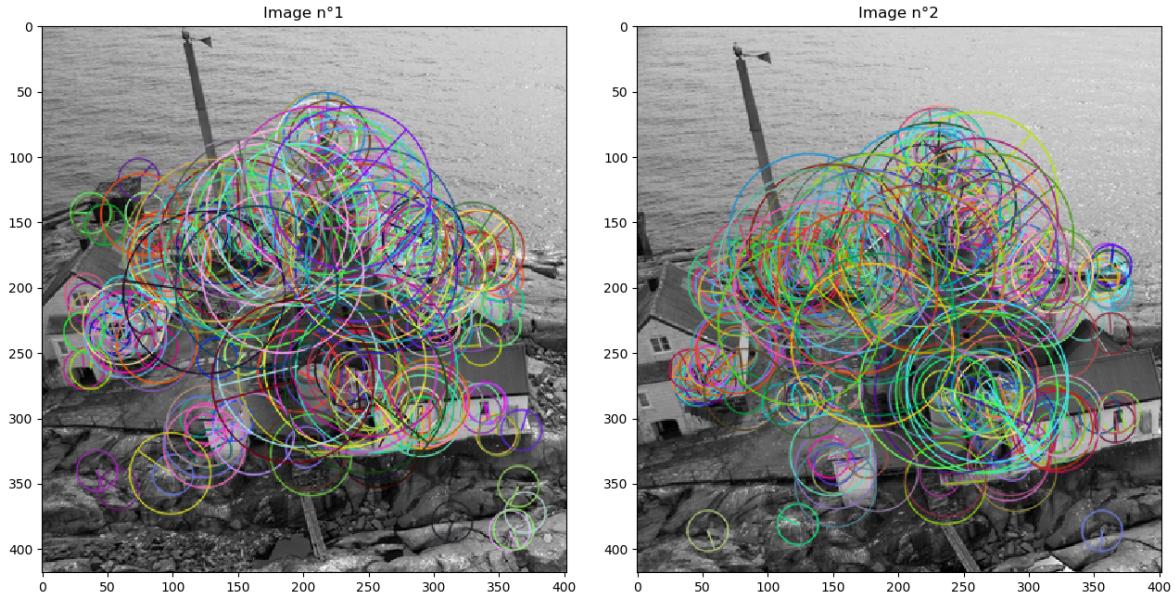


FIGURE 12: Images torsmall1.png (à gauche) et torbsmall2.png (à droite) en utilisant le détecteur ORB avec les paramètres nFeatures=250, scaleFactor=2, nlevels=3, patchFactor=31 et EdgeThreshold=31

Le premier pas pour la amélioration du détecteur était réduire le taille de la voisinage circulaire. Ainsi on a change les valeurs de les paramètres patchSize et edgeThreshold des valeurs prédéfinies pour 9. Le résultat peut être vu dans la Figure 13. patchSize définit le rayon de la voisinage et edgeThreshold définit jusqu'à quelle rayon l'épaisseur de la voisinage s'étend, comme le valeurs sont égals on a un voisinage de un pixel. On peut apercevoir que l'image est plus lisible et que le rayon des points d'intérêt ne sont pas le même en raison de l'usage des représentations de l'image à différent échelle.

Les paramètres scaleFactor et nLevels sont liés au facteur de réduction d'échelle pour niveau de pyramide et le nombre de niveau de pyramide qui sont liés à représentations de l'image à différentes échelles. Le rayon des quelques points d'intérêt sont très grand et peu représentatif à grâce du premier valeur qui est trop grand. En diminuant leur valeur, on obtient la Figure 14 qui est très lisible et montre bien les coins de les deux images.

On peut constater la répétabilité du détecteur visuellement par la vérification que beaucoup des points d'intérêt apparaître dans les deux images sur le même place de les structures

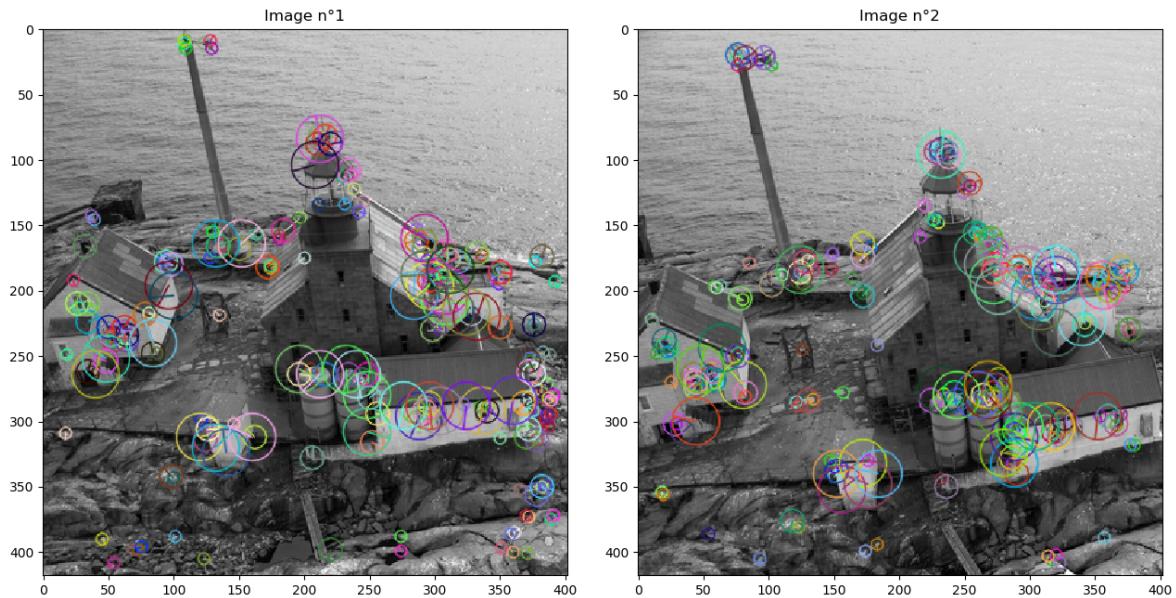


FIGURE 13: Images torsmall1.png (à gauche) et torbsmall2.png (à droite) en utilisant le détecteur ORB avec les paramètres nFeatures=250, scaleFactor=2, nlevels=3, patchFactor=9 et EdgeThreshold=9

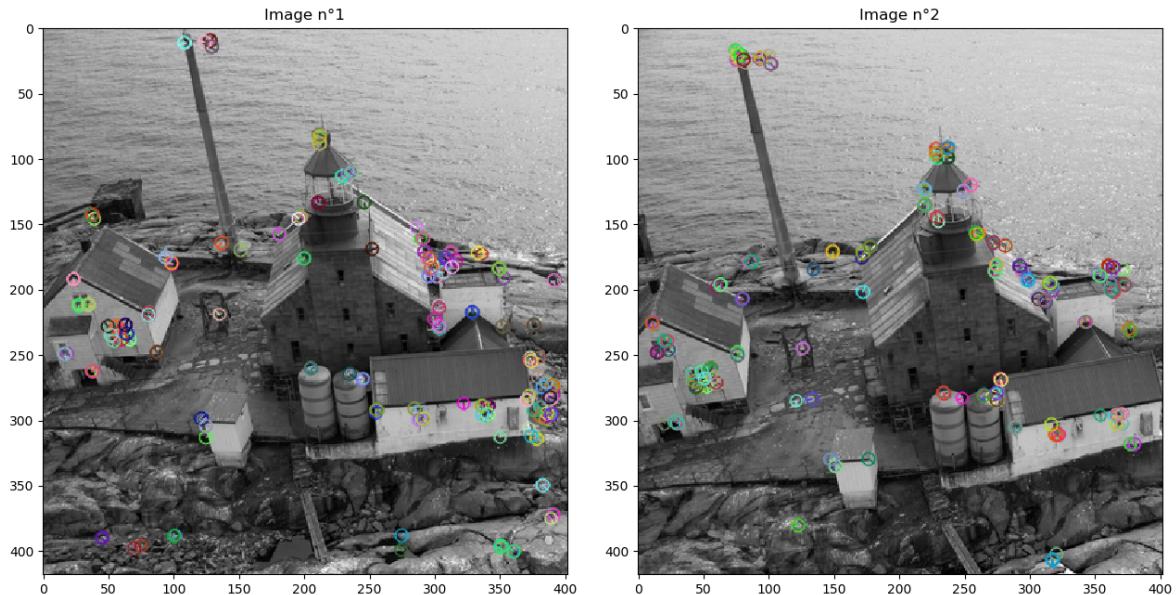


FIGURE 14: Images torsmall1.png (à gauche) et torbsmall2.png (à droite) en utilisant le détecteur ORB avec les paramètres nFeatures=250, scaleFactor=1.05, nlevels=3, patchFactor=9 et EdgeThreshold=9

géométriques de l'images.

KAZE

Le principe du détecteur est : À partir d'une image d'entrée, nous construisons l'espace à échelle non linéaire jusqu'à un temps d'évolution maximal en utilisant des techniques Additive Operator Splitting (AOS) et une diffusion à conductance variable. Ensuite, nous détectons des caractéristiques 2D intéressantes présentant un maximum du déterminant normalisé en fonction de l'échelle du résidant de Hessian par le biais de son espace de balayage linéaire.

Pour détecter les points d'intérêt, nous calculons la réponse du déterminant de Hessian normalisé par l'échelle à plusieurs niveaux d'échelle. Pour la détection de caractéristiques multi-échelles, l'ensemble des opérateurs différentiels doit être normalisé en fonction de l'échelle, car l'amplitude des dérivées spatiales décroît avec l'échelle.

$$L_{Hessian} = \sigma^2(L_{xx}L_{yy} - L_{xy}^2)$$

où (L_{xx}, L_{yy}) sont respectivement les dérivées horizontales et verticales du second ordre et L_{xy} est la dérivée croisée du second ordre. Étant donné l'ensemble des images filtrées de l'espace d'échelle non linéaire Li, nous analysons la réponse du détecteur à différents niveaux d'échelle σ_i . Nous recherchons des maxima d'échelle et de localisation spatiale.

Ensuite, les principaux paramètres de ce détecteur qui ont les valeurs par défaut suivants :

- threshold : Seuil de réponse du détecteur pour accepter le point. Valeur par défaut = 0.001
- nOctaves : Évolution d'octave maximale de l'image. Valeur par défaut = 4
- nOctaveLayers : Nombre de sous-niveaux par niveau par niveau .Valeur par défaut = 4
- diffusivity : Valeur par défaut = 2

Donc, à partir de ces valeurs par défaut nous avons changé chaque paramètre pour un valeur plus petit à fin de regarder les comportement de chaque paramètre sur l'image

résultant.

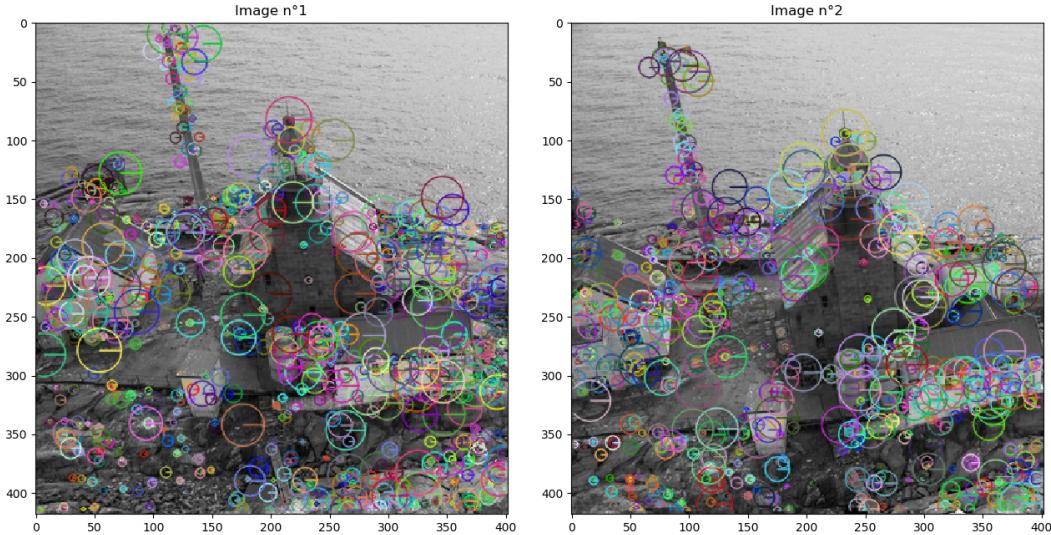


FIGURE 15: DéTECTEUR KAZE AVEC LES PARAMÈTRES PAR DÉFAUT

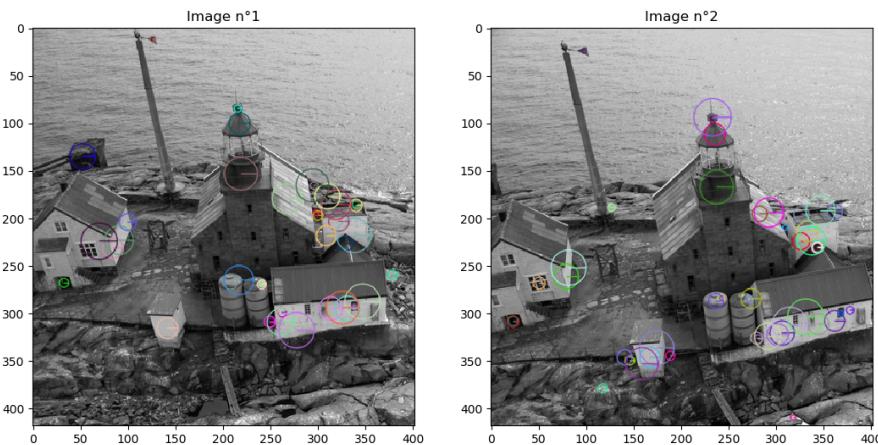


FIGURE 16: Threshold = 0.01

D'abord, le premier paramètre modifié a été le `threshold` qui il s'agit d'un valeur limite que sert comme référence de comparaison, par conséquent, comme nous pouvons vérifier par la Figure 16, lorsqu'on augmente cette valeur on diminue les caractéristiques détectées sur l'image puisque pour un valeur de `threshold` plus haut, moins plus points sont acceptés.

Ensuite, par rapport le paramètre `diffusitynous` avons que ...

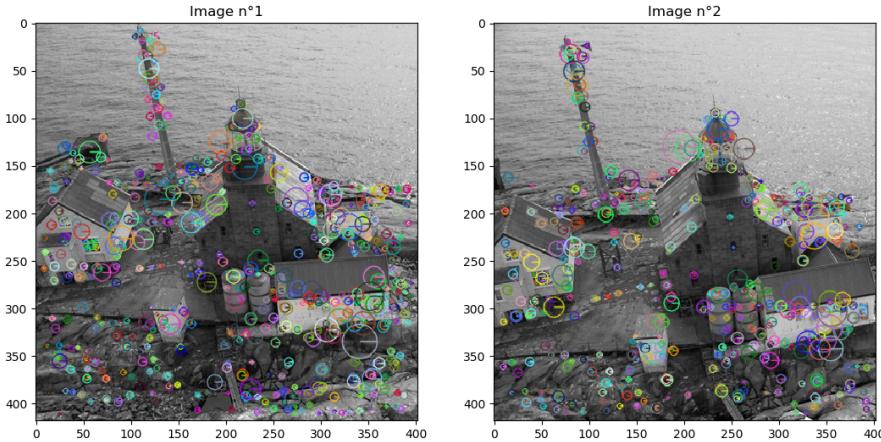


FIGURE 17: Diffusity = 1

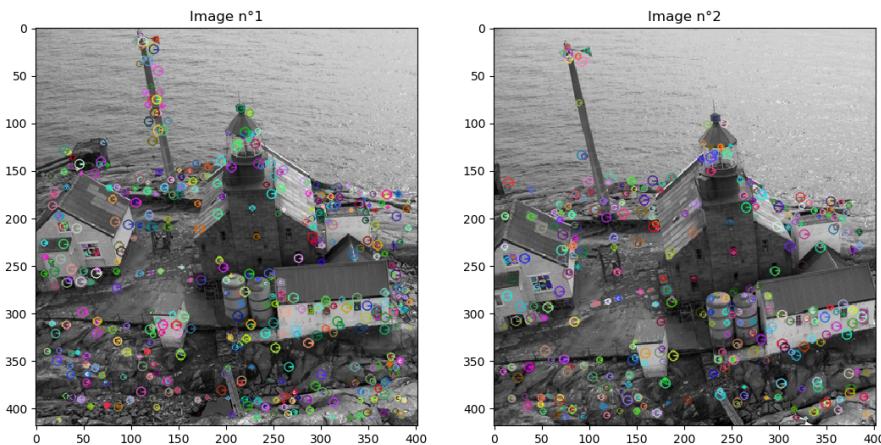


FIGURE 18: nOctaves = 2

Descripteurs et Appariement

Q7 :

ORB

Le descripteur d'ORB s'appelle rBRIEF (Rotated BRIEF) et retourne un vecteur qui décrit chaque patch (p) donné par le détecteur de la façon suivante :

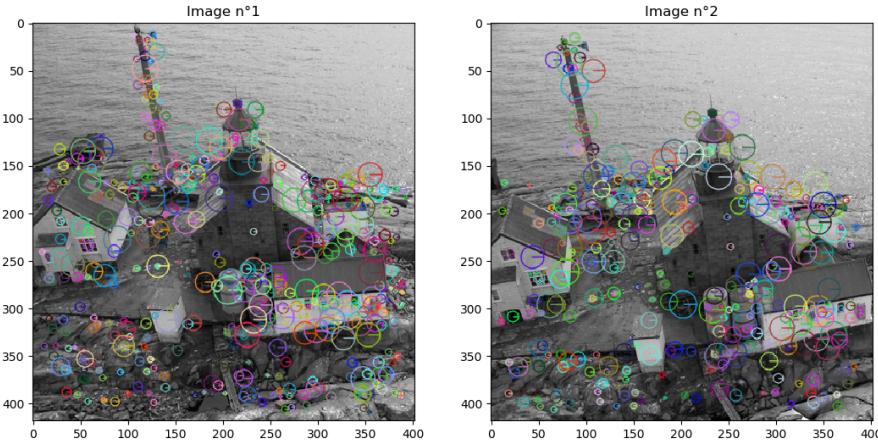


FIGURE 19: nOctavesLayers = 2

$$g_n(p, \theta) = \sum_{i=1}^n 2^{i-1} \tau(p; x_i, y_i) | (x_i, y_i) \in R_\theta S$$

Où n est le nombre de points (pixels) choisi pour le teste, x et y sont les point qui sont comparés, R_θ est la matrice de rotation avec l'angle de rotation θ déjà défini par le détecteur, $S \begin{vmatrix} x_1 & .. & x_n \\ y_1 & .. & y_n \end{vmatrix}$ est l'ensemble des points à être testés et τ est le teste binaire :

$$\tau(p; x, y) = \begin{cases} 1, & p(x) > p(y) \\ 0, & p(x) \leq p(y) \end{cases}$$

Pour augmenter la variance du descripteur, on utilise le méthode suivant pour obtenir un bonne ensemble des testes binaires :

- D'abord, on tourne chaque teste dans chaque patch.
- Ensuite, on range les testes par leur distance de la moyen 0,5.
- Enfin, on exécute une algorithme de recherche gourmand qui retourne un vecteur de taille prédéfini des testes binaires qui ont un corrélation plus petit qu'un seuil.

Pour le descripteur ORB, l'invariabilité par rapport le changement d'échelles vient d'usage

des représentations de l'image à plusieurs échelles qui a une forme de pyramide. Comme les points d'intérêt sont choisis dans chaque étage de la pyramide, le descripteur devient invariant par rapport aux échelles.

Concernant l'invariabilité par rotation, il s'agit de la rotation des tests binaires à travers de la multiplication pour la matrice de rotation R_θ étant θ l'orientation de chaque patch.

KAZE

Pareil à l'algorithme Speeded-Up Robust Features (SURF), nous trouvons l'orientation dominante dans une zone circulaire de rayon $6\sigma_i$ avec un pas d'échantillonnage de taille θ_i . Pour chacun des échantillons dans la zone circulaire, les dérivées du premier ordre L_x et L_y sont pondérées avec une Gaussienne centrée sur le point d'intérêt. Ensuite, les réponses dérivées sont représentées sous forme de points dans l'espace vectoriel et l'orientation dominante est trouvée en faisant la somme des réponses dans un segment de cercle glissant couvrant un angle de $\pi/3$. À partir du vecteur le plus long, l'orientation dominante est obtenue.

Q8 :

Les stratégies d'appariement Ratio Test et Cross Test sont de type d'appariement de force brute. C'est-à-dire qu'ils calculent la distance de la description de chaque point d'intérêt de une image à chaque point d'intérêt de la deuxième image avec une métrique définie et retournent pour chaque point d'intérêt de la première image le point d'intérêt le plus proche. La différence entre les deux approches, c'est qu'il faut calculer aussi le point d'intérêt de la première image plus proche de chaque point d'intérêt de la deuxième image et la méthode en effet retourne les appariements que le point d'intérêt de la première image est le plus proche de le point d'intérêt de la deuxième image et vice versa.

La stratégie d'appariement FLANN ou Fast Library for Approximated Nearest Neighbors

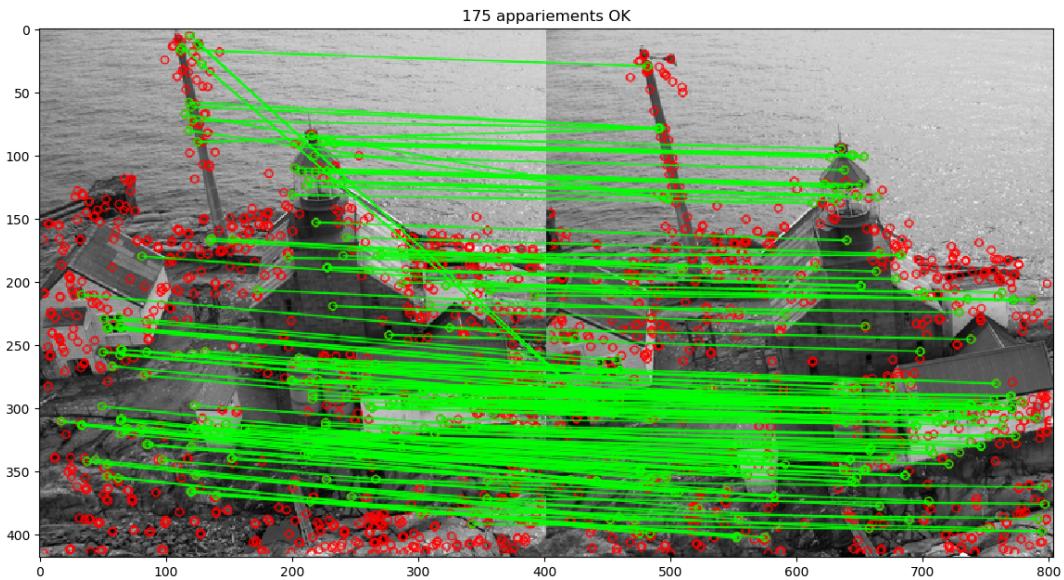


FIGURE 20: Appariement de points d'intérêt en utilisant la stratégie Ratio Test pour le descripteur Kaze.

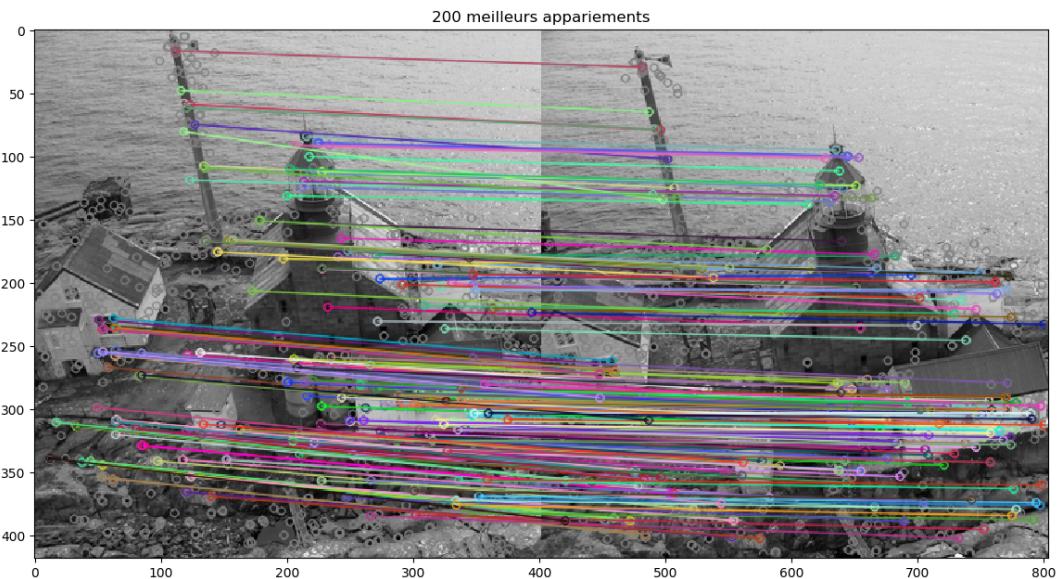


FIGURE 21: Appariement de points d'intérêt en utilisant la stratégie Cross Check pour le descripteur KAZE

est une ensemble des méthodes, comme randomized kd-trees et hierarchical k-means tree, combinées dans une librairie pour maximisé la vitesse d'exécution des appariements en donnant la tolérance nécessaire et peut-être une limite de mémoire.

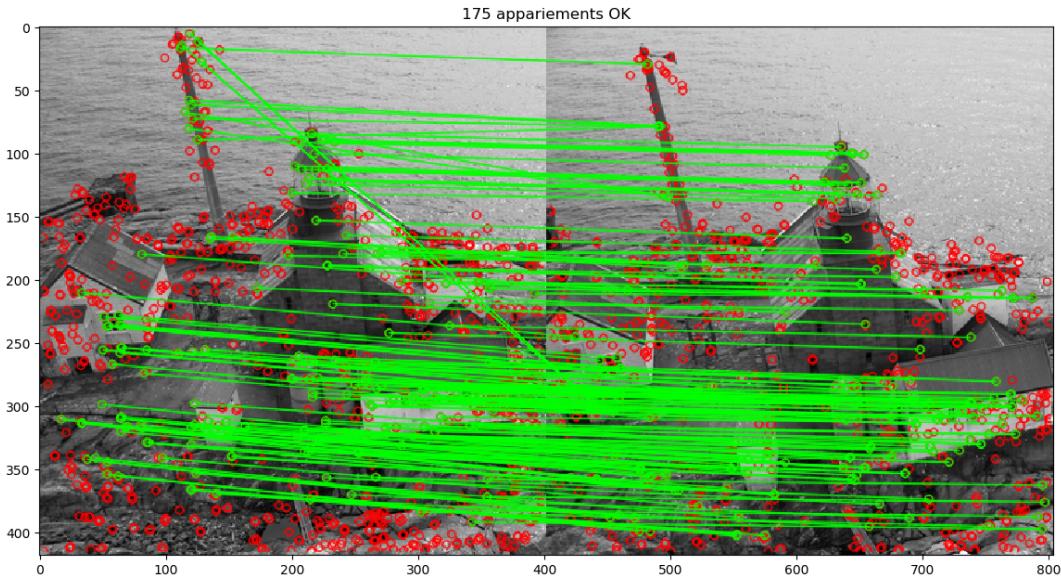


FIGURE 22: Appariement de points d'intérêt en utilisant la stratégie FLANN pour le descripteur KAZE

Pour le descripteur KAZE, on a obtenu trois appariements de points d'intérêt (Figures 20, 21 et 22). La stratégie d'appariement Ratio Test a géré 175 appariements que elle-même dise que sont bons mais en effet 173 appariements sont bons, comme on peut apercevoir dans l'image 20. La stratégie d'appariement Cross Test a géré 199 appariements bons comme on peut apercevoir dans l'image 21. Enfin, la stratégie FLANN a géré 171 appariements bons comme on peut apercevoir dans l'image 22. Alors, on voit que pour cette deux images et le détecteur Kaze, la stratégie d'appariement de force brute Cross Test est la plus performant, après vient la stratégie de force brute Ratio Test, et finalement, la stratégie FLANN.

Pour le descripteur ORB, on a obtenu trois appariements de points d'intérêt (Figures 23, 24 et 25). La stratégie d'appariement Ratio Test a géré 64 appariements bons, comme on peut apercevoir dans l'image 23. La stratégie d'appariement Cross Test a géré 200 appariements que elle-même dise que sont bons, mais en effet on n'a que 185 appariements bons comme on peut apercevoir dans l'image 24. Enfin, la stratégie FLANN a géré 66 appariements bons comme on peut apercevoir dans l'image 25. Alors, on voit que pour cette deux images et le détecteur Kaze, la stratégie d'appariement de force brute Cross Test est la plus performant,

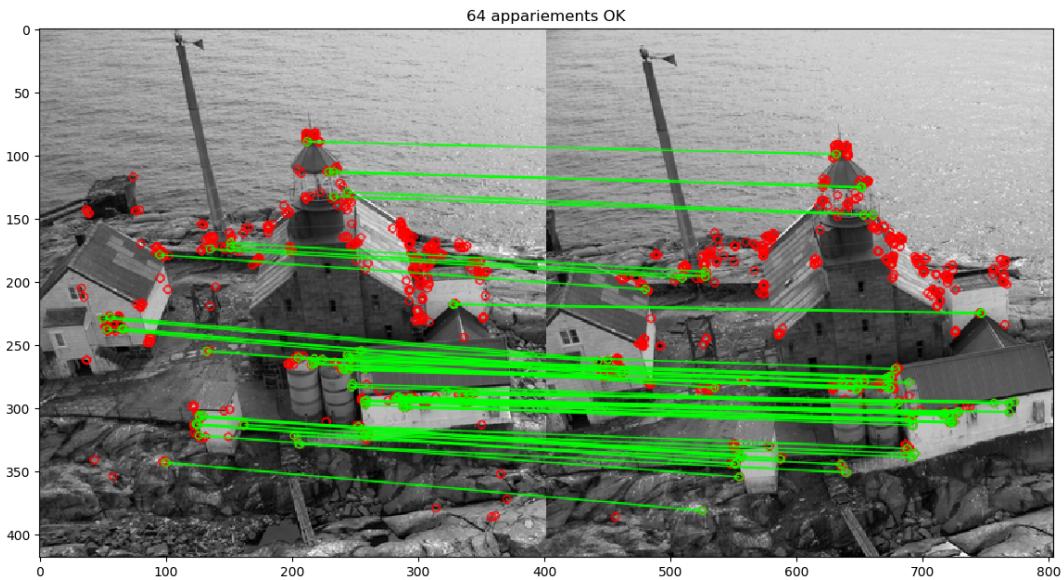


FIGURE 23: Appariement de points d'intérêt en utilisant la stratégie Ratio Test pour le descripteur ORB

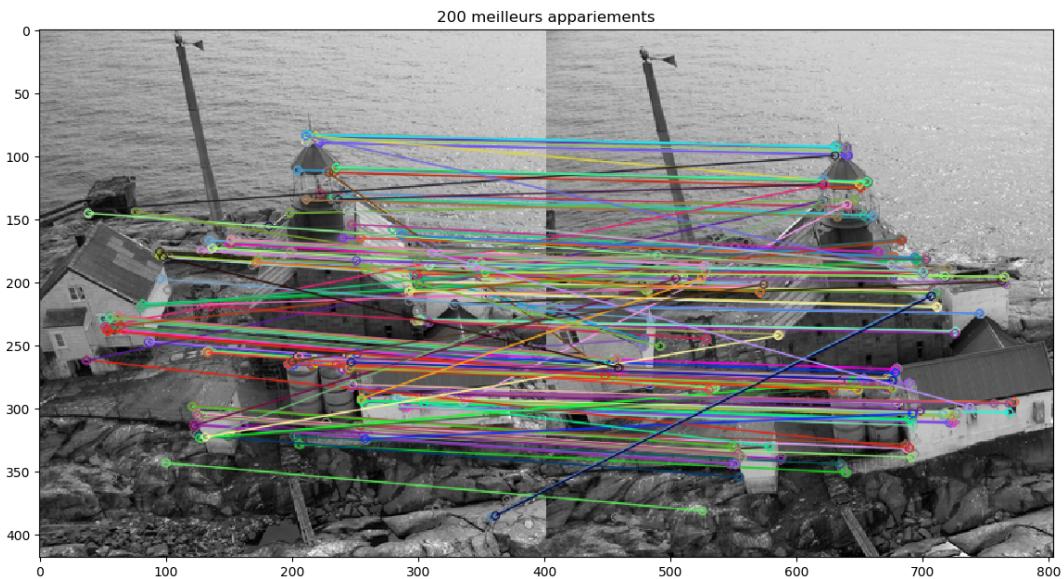


FIGURE 24: Appariement de points d'intérêt en utilisant la stratégie Cross Check pour le descripteur ORB

après vient la stratégie Flann, et finalement, la stratégie de force brute Ratio Test.

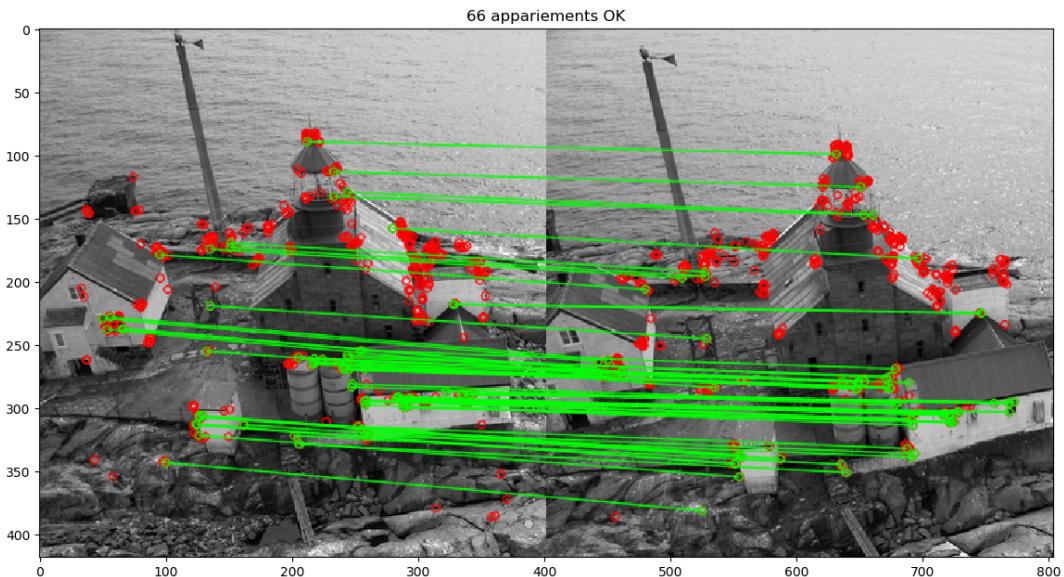


FIGURE 25: Appariement de points d'intérêt en utilisant la stratégie FLANN pour le descripteur ORB

Q9 :

On a développé une stratégie pour évaluer la performance de les trois méthodes d'appariements de points d'intérêt Ratio Test, Cross Check et FLANN entre une image originale et la même image tourné à une angle spécifiée. Le code peut être trouvé dans les codes `Features_Match_RatioTest.py`, `Features_Match_CrossCheck.py` et `Features_Match_FLANN.py`.

L'idée est utiliser les méthodes de détection, description et appariement dans les deux images déjà expliquées. Ensuite, obtenir les points d'intérêt associés dans chaque image et calculer la distance euclidienne entre le point tourné de la première image et le point associé dans la deuxième image. Pour chaque distance inférieure à une valeur de tolérance, on incrémenté un compteur. L'exactitude est calculée comme cette compteur de distances inférieures à la tolérance divisé par la quantité totale de points appariés.

On a exécuté 18 simulations, toujours avec la tolérance de 3 et en changeant la méthode d'appariement, le descripteur et le détecteur, selon Table 1. Cette cadre montre que le détecteur et descripteur ORB est plus invariant par rapport la rotation des images que le détecteur et descripteur KAZE, que la méthode Cross Check est plus fiable que les méthodes Ratio Test

et FLANN, qui ont des résultats pareils et que la rotation toujours augmente la incertitude de l'appariement des points d'intérêt. De plus, on voit que le lien entre l'exactitude et l'angle n'est pas évident.

TABLE 1: Exactitude de méthodes de appariement en changeant l'angle de rotation de l'image, le détecteur et le descripteur.

Angle	Ratio Test		Cross Check		FLANN	
	ORB	KAZE	ORB	KAZE	ORB	KAZE
0	100%	100%	100%	100%	100%	100%
15	76,4%	46,71%	87,2%	73,86%	76,0%	47,71%
60	71,8%	42,76%	85,71%	73,47%	70,8%	42,76%
90	82,4%	55,02%	92,17%	84,36%	80,8%	55,02%
180	78,4%	56,13%	92,29%	85,78%	77,0%	56,13%

Références

OpenCV Image Filtering : <https://docs.opencv.org/2.4/modules/imgproc/doc/filtering.html>

Image file reading and writing : https://docs.opencv.org/3.4/d4/da8/group__imgcodecs.html

OpenCV User Interface : https://docs.opencv.org/2.4/modules/highgui/doc/user_interface.html

matplotlib.pyplot.subplot : https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.subplot.html

matplotlib.pyplot.title : https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.title.html

matplotlib.pyplot.show : https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.show.html

matplotlib.pyplot.figure : https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.figure.html

cv2.dilate : https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

P.F. Alcantarilla, A. Bartoli, & A. J. Davison « KAZE features ». 12th European conference on Computer Vision - Vol. Part VI. Springer-Verlag, Berlin, Heidelberg, 214-227, 2012.

E. Rublee, V. Rabaud, K. Konolige, & G. Bradski « ORB : an efficient alternative to SIFT or SURF », IEEE International Conference on Computer Vision (ICCV), 2011.

Marius Muja, David G. Lowe. Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration, 2009

Multi-Probe LSH : Efficient Indexing for High-Dimensional Similarity Search by Qin Lv, William Josephson, Zhe Wang, Moses Charikar, Kai Li., Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB). Vienna, Austria. September 2007

ORB (Oriented FAST and Rotated BRIEF) : https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_orb/py_orb.html

OpenCV ORB Class Reference : https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_orb/py_orb.html

OpenCV Feature Matching : https://docs.opencv.org/master/dc/dc3/tutorial_py_matcher.html