Signal et Image

Travaux pratiques - partie Image (préparation du TP)

- Le TP est **individuel**
- Le travail est à faire sous **MATLAB** (sur votre ordinateur personnel ou sur ceux de l'Université). Il existe déjà des fonctions qui font exactement ou au moins en partie les exercices demandés (détection de cercles, de points d'intérêt etc.) ; l'objectif du TP n'est pas de les utiliser, mais de comprendre comment on peut les implémenter (et de voir éventuellement qu'on peut faire mieux assez facilement).

1 Partie 1

L'objectif de la première partie du TP est d'implémenter un détecteur d'objets circulaires par une méthode cumulative de type Hough.

Exercice 1 — La transformée de Hough pour les cercles

On considère une paramétrisation de type (r, c, rad) pour définir un cercle de rayon rad pixels dans l'image, dont le centre est situé sur la ligne r et la colonne c. Comme nous sommes obligés de considérer un ensemble discret de paramètres, on retient toutes les positions $r \in [r_{min}, r_{max}]$ avec un pas de discrétisation de δr, et ainsi de suite pour $c \in [c_{min}, c_{max}]$ avec un pas de discrétisation de δc et pour $rad \in [rad_{min}, rad_{max}]$ avec un pas de discrétisation de δrad.

1. Pour l'image four.png fournie, de taille 100 × 100 pixels, considérons que $r_{min} = 1$, $r_{max} = 100$, $\delta r = 2$. Combien de valeurs discrètes aura-t-on pour la coordonnée r des cercles ? Et si $\delta r = 0.5$?
2. Pour la même images, en supposant que $r_{min} = 1$, $r_{max} = 100$, $\delta r = 1$, $c_{min} = 1$, $c_{max} = 100$, $\delta c = 1$, $rad_{min} = 5, rad_{max} = 100\sqrt{2}, \delta rad = 1$, quel est le nombre total de cercles qu'on peut décrire avec ces trois variables ?
3. Le tableau tridimensionnel acc associe à la case $acc(i, j, k)$ le cercle situé à la i-ème valeur discrète de r, la j-ème valeur discrète de c, et la k-ème valeur discrète de rad. Quel est le cercle associé au $acc(1, 1, 1)$? Au $acc(10, 7, 30)$?
4. Inversement, quelle est la case de l'accumulateur associée au cercle centré dans le pixel (40,40) et de rayon $rad = 13$? Attention : les indices i,j,k doivent être entiers.

Exercice 2 — Implementation du détecteur

1. On rappelle le fonctionnement de l'algorithme de détection de cercles :
   1. (optionnel) Filtrage Gaussien (en cas de bruit ou de détails très fins)
   2. Filtrage de Sobel, calcul de la magnitude de gradient I_{mag} dans chaque pixel
   3. Tous les pixels dont la magnitude est au dessus d'une fraction t de la valeur maximale dans I_{mag} sont considérés comme des pixels du contour. Note : visualisez l'image des contours pour être sûrs que vous avez dedans les contours des objets recherchés.
   4. Initialisez toutes les valeurs de l'accumulateur acc à 0.
   5. Pour chaque pixel de contour, considérez toutes les (r,c) possibles, calculez le rayon rad pour que le cercle situé en (r,c) passe par le pixel respectif, et incrémentez dans l'accumulateur la case qui correspond à (r, c, rad).

FIGURE 1 – Détection de cercles

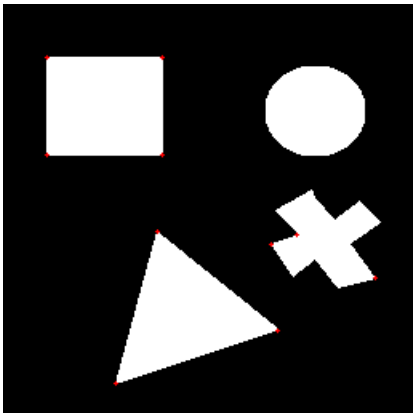

FIGURE 2 – Détection de coins de Harris sur une image synthétique ; les réponses sont indiquées par les petites croix rouges.

6. Identifiez dans l'accumulateur les maximas locaux - les cases avec des valeurs supérieures aux 26 cases voisines (car l'accumulateur est tridimensionnel).
7. Selectionnez les N valeurs les plus grandes, et à partir des indices (i,j,k) récupérez les (r,c,rad) correspondants et visualisez les cercles avec la fonction `showCircle` fournie.

Note 1 : les cercles plus grands reçoivent plus de votes, donc il faudrait normaliser les valeurs de l'accumulateur pour ne pas privilégier les cercles grands.
Note 2 : pour mettre un vote, on peut incrémenter soit par 1, soit par la magnitude du gradient dans le pixel respectif etc.
Essayez de trouver une solution qui fonctionne pour des images variées (voir par exemple Figure 1 pour des images avec ou sans bruit).

2 Partie 2

Vous devez implémenter un détecteur de coins et vous avez le choix entre l'exercice 3 (détecteur de Harris) et l'exercice 4 (détecteur FAST). De manière facultative, vous pouvez implémenter les deux, ce qui vous rapporte des points bonus.

Exercice 3 — Detecteur de Harris

1. Implémentez le détecteur de Harris vu en cours. Justifiez le choix des méthodes et des paramètres utilisés. On rappelle que la détection se fait par

1. l'estimation des dérivées de l'image (par un filtre de Sobel, ou par un filtre dérivée de Gaussienne)
2. le calcul du tenseur de structure à l'aide d'un filtre Gaussien de taille adaptée
3. le calcul, le seuillage et la suppression non-maximale des valeurs de la matrice R

Vous pouvez soit seuiller par rapport à une valeur fixe, soit garder les N premières réponses les plus fortes, soit utiliser une stratégie composée. Dans tous les cas, justifiez bien les choix que vous faites. Faites attention également á implémenter les opérations de filtrage de manière efficace, en exploitant les propriétés de la convolution (séparabilité, associativité etc.).
Testez d'abord votre méthode avec l'image synthétique fournie ; vous devriez obtenir un résultat similaire (mais pas forcement identique) à la Figure 2. Pour dessiner les croix, vous pouvez utiliser la fonction `showCorners` fournie. Par la suite, testez votre détecteur sur les images naturelles.

Exercice 4 — Detecteur FAST

1. Implémentez le détecteur FAST vu en cours. N'utilisez pas la stratégie basée sur la variation de l'entropie (détaillée dans l'article), mais faites un simple parcours séquentiel de la circonférence autour d'un pixel pour décider s'il est un coin. En revanche, utilisez bien la stratégie de l'article pour la suppression non-maximale : parmi plusieurs coins adjacents, on ne garde que celui qui reste coin en dernier si on continue à augmenter le seuil de détection. De la même manière, visualisez le résultat de votre détecteur sur l'image synthétique et sur les images naturelles à l'aide de la fonction `showCorners`.

Exercice 5 — Appariement

1. En faisant tourner un de ces deux détecteurs sur les images `set1-1.png` et `set1-2.png`, vous obtenez deux ensembles de détections qu'on doit associer. Pour faire cela, il faut calculer un score entre une détection de la première image et chaque détection de la deuxième image. La solution suggérée est d'utiliser un score de type sum of squared distances (SSD) entre deux petites régions carrées (patchs) R_1 et R_2 centrées sur les coins c_1 et c_2 qu'on analyse :

$$SSD(c_1, c_2) = \sum_{p_1 \in R_1, p_2 \in R_2, p_1 \leftrightarrow p_2} \Big[I_1(p_1) - I_2(p_2) \Big]^2$$

ou $p_1 \leftrightarrow p_2$ signifie que p_1 et p_2 sont des pixels correspondants dans les deux patchs (situés dans la même location par rapport aux coins c_1 et c_2).
En réalité, pour être robuste aux variations globales d'intensité entre les deux images, on préfère de retirer de chaque coté la moyenne du patch respectif, pour calculer un score ZMSSD (zero-mean sum of squared distances) :

$$ZMSSD(c_1, c_2) = \sum_{p_1 \in R_1, p_2 \in R_2, p_1 \leftrightarrow p_2} \Big[(I_1(p_1) - \mu_1) - (I_2(p_2) - \mu_2) \Big]^2$$

Dans ce cas, plus le score est petit, mieux c'est, et on va apparier un coin avec le correspondant de l'autre image avec le score le plus petit. Néanmoins, il faudra choisir un seuil (la valeur est à justifier) au dessus duquel on refuse même le meilleur appariement. C'est le cas par exemple quand un point de l'image 1 n'apparaît pas dans l'image 2, et on évite ainsi de l'apparier de manière erronée. Si vous avez trop de fausses appariements, vous pouvez utiliser deux autres stratégies plus fines :
— married matching : vous appariez image 1 vers image 2, puis image 2 vers image 1 et vous ne gardez que les points qui se sont choisis réciproquement
— vous regardez le meilleur score et le second meilleur ; pour accepter l'appariement il faut que le meilleur score soit bien meilleur que le deuxième
Une fois que vous avez une liste d'appariements, vous pouvez la visualiser avec la fonction `showMatches` fournie (Figure 3)

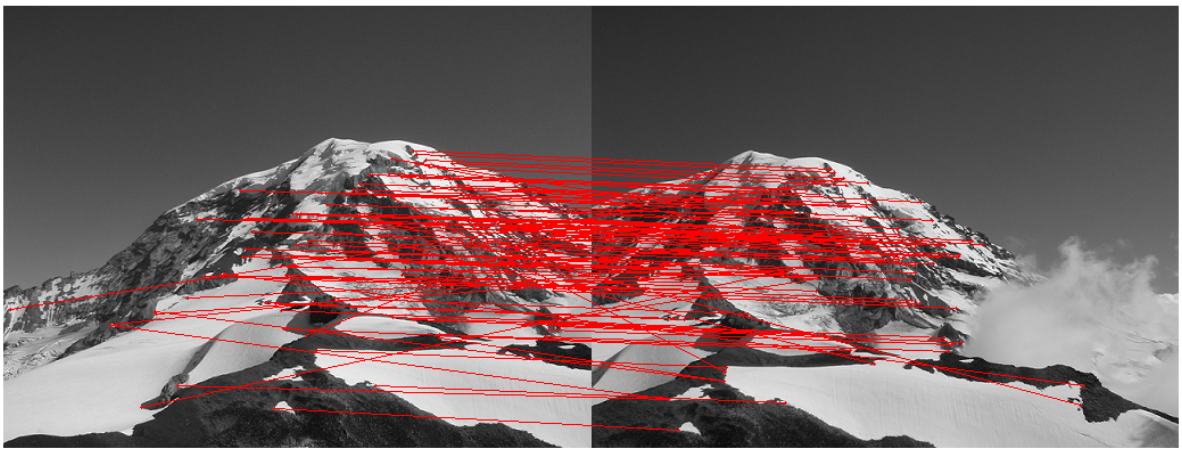

FIGURE 3 – Appariement de coins entre deux images.