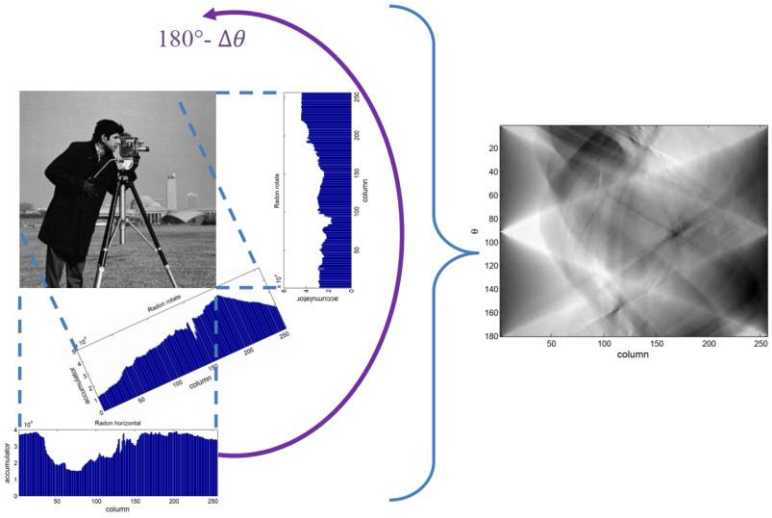
TD 7 UE Vision

La transformée de Radon est abordée pour détecter des droites. Le suivi d'objet dans les vidéos est également abordé sommairement. Une technique de détection de coin par masque et une de débruitage sont à étudier.

1) Transformée de Radon

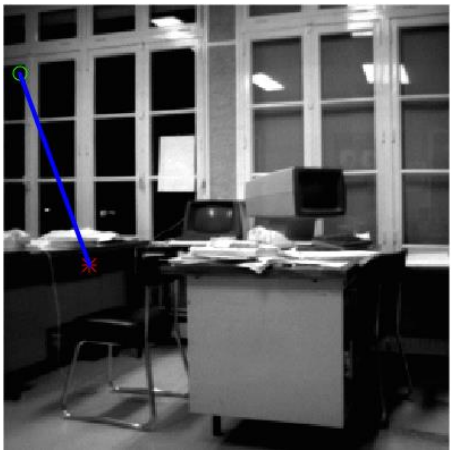


La transformée de Radon correspond à une intégrale le long de colonnes (ou lignes). En effet, l'objectif est de projeter les intensités de l'image le long des rayons orientés de 0 à 180°. Ces projections sont calculées en sommant les intensités le long de chaque colonne suivant un angle donné. L'image à droite illustre trois exemples de projections sous trois angles différents. Toutes les orientations permettent de construire un sinogramme avec en ordonnées l'angle d'orientation pour l'accumulation et en abscisses la largeur de l'image. Voici la formule :

$R_I(\theta, j) = \sum_{l=1}^{L} I_\theta(l, j)$, où I_θ représente une image I tournée d'angle θ, et j le numéro de la colonne.

a) Tout d'abord, regarder l'aide de la fonction « *imrotate* » et tester sur une image.
b) Créer un carré blanc au milieu d'une image noire, regarder ses projections pour l'angle 0°. Regarder la projection pour l'angle 45°. Que remarquez-vous ? Regarder également à 90° et 180°, que constatez-vous ?
c) Créer une fonction « *f_radon.m* » qui calculera la transformée de Radon de 0 à $180^o - \Delta\theta$, où $\Delta\theta$ représente le pas angulaire entre les différentes rotations. La difficulté est de créer un vecteur de taille identique pour chaque orientation. Tester sur le carré et différentes images que vous avez construites.
d) Calculer et essayer de comprendre la transformée de Radon de l'image «Shepp_Logan_phantom_redim.png».
e) Après avoir ajouté des marges noires à l'image « bureau256.png », calculer et essayer de comprendre sa transformée de Radon. Que remarquez-vous ?
f) (Optionnel) La transformée de Radon peut être appliquée pour des images en couleur. On peut également calculer la rétroprojection de la transformée de Radon par la sommation des rétroprojections pour chaque orientation. Etudier les valeurs du SNR/PSNR de la rétroprojection en fonction du nombre d'angles utilisés.

2) Tracer un segment et afficher des points



Charger ou créer une image appelée *I*.
Les commandes suivantes permettent d'afficher les points de coordonnées (x1, y1) et (x2, y2) sur l'image puis d'afficher le segment entre ces points :

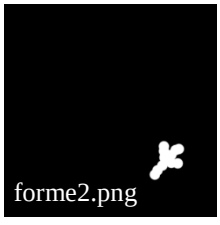
```
>> x1 = 10    % point 1
>> y1 = 40
>> x2 = 50    % point 2
>> y2 = 150
>> figure, imagesc(I), colormap(gray), title('image originale')
>> hold on              % permet de bloquer la figure
>> plot(x1,y1,'go')     % affiche point 1
>> plot(x2,y2,'r*')     % affiche point 2
>> plot([x1 x2], [y1 y2], 'linewidth', 4) % affiche le segment
```

Que se passe-t-il lorsque *y2 = 500* ?

3) Suivi d'objet par différence d'image

Segment entre deux objets : Le but ici est de tracer le segment entre les deux objets extraits des deux images « forme1.png » et « forme2.png ».
1- Créer deux images binaires à partir de ces deux images.
2- Calculer les deux barycentres des deux formes.
3- Faire apparaitre les deux formes sur la même image *I3*.
4- Afficher les barycentres sur l'image *I3* (cf. exercice **2)**).
5- Tracer la droite entre les deux barycentres (figure de droite).


forme1.png


forme2.png

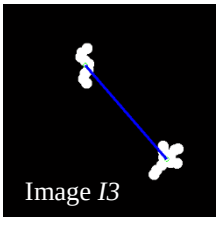
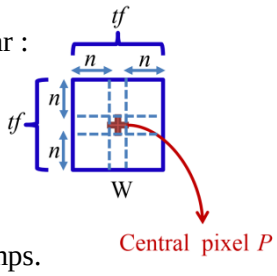


Image *I3*

4) Détection de coins : méthode de Moravek

Soit *tf* la taille de la fenêtre glissante W sur l'image originale *I*. Les coins sont calculés par :



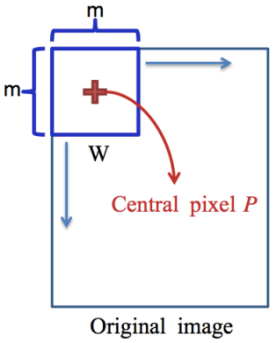
$$J(x,y) = \sum_{i=-n}^{n} \sum_{j=-n}^{n} (I(x+i, y+j) - I(x,y))^2 .$$

Avec $n = \frac{tf-1}{2}$ et (x,y) les coordonnées du pixel *P* au centre du masque (*tf* impair).

On travaillera avec l'image « img74_crop_avec_contours_flous.png » dans un premier temps.

a) Ecrire une fonction « *f_moravek* » qui calcule la formule précédente (attention aux bords de l'image)
b) Normalisez et visualisez *J*.
c) Ecrire une fonction « *f_maxima_locaux* » qui calcule si l'élément central du masque est un maximum local de *J*.
d) Seuillez le résultat et affichez les coins en couleur sur l'image originale (voir questions précédentes).
e) Avec les valeurs *tf* = 7 et un seuil à 0.7, qu'obtenez-vous ?
f) Tester sur l'image « bureau256.png » avec différentes tailles de fenêtre *tf*. Que constatez-vous ?

5) Restauration des images : filtre min-max



Soit *W* une fenêtre glissante de taille *m* sur l'image originale *I* (avec *m* impair).
On ne considère pas la valeur du pixel central pour les calculs suivants.
$i_{max} = \max(I(x,y) \in W \ / \ I(x,y) \neq P)$
$i_{min} = \min(I(x,y) \in W \ / \ I(x,y) \neq P)$

a) Le filtre min-max est defini par :

$$J(x,y) = \begin{cases} P \ \ if \ \ i_{min} \leqslant P \leqslant i_{max} \\ i_{min} \ \ if \ \ P < i_{min} \\ i_{max} \ \ if \ \ P > i_{max} \end{cases}$$

Coder la fonction ci-dessus telle que l'on puisse l'appeler ainsi :

```
>> [ J ] = f_min_max_filter( I, tf );
```

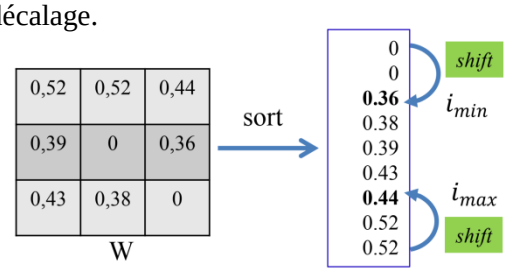
b) Charger l'image du cameraman, puis la bruiter avec les commandes suivantes :

```
>> I = double(imread('cameraman.tif'))/255;
>> I = imnoise(I,'salt & pepper',0.01);
```

Visualiser *I* puis appeler la fonction « *f_min_max_filter* » pour débruiter *I*. Tester différentes valeurs de *tf*.
Vous remarquerez qu'il reste encore des points de bruit. A votre avis, pourquoi ?
Appliquer le filtre sur l'image « angiogra.jpg ». Là aussi, subsiste-t-il encore des résidus.

c) Pour améliorer le filtre, les valeurs choisies ne seront plus ni i_{max}, ni i_{min}.
En effet, les valeurs de W vont être ordonnées et les valeurs choisies seront décalées (avec un shift) par rapport au min et au max comme illustré à droite.
Calculer le filtre min-max sur la matrice ci-contre, puis la version avec décalage.
Ecrire la fonction « *f_min_max_filter_ameliore* » qui fonctionne comme suit (avec *tf = 3*) :

```
>> decal = 2
>> [ J ] = f_min_max_filter_ameliore( I, tf, decal);
```



On se servira de :

```
S = size ( A );
%reshape A.
p = reshape ( A, S(1) * S(2), 1 );    % creation du vecteur
S = sort(p)                           % tri par ordre croissant
```

d) Comparer les fonctions « *f_min_max_filter* » et « *f_min_max_filter_ameliore* ».
e) Tester différents paramètres « *decal* » et « *tf* ».
f) Appliquer deux itération du filtre amélioré sur l'image « angiogra.jpg » avec :

```
>> decal = 2
>> tf = 3
```

g) Tester d'autres paramètres. Que se passe-t-il lorsque $decal > \frac{tf^2+1}{2}$?