Éleve: Rafael Senna Benatti
Discipline: IMA201(a)
TP Filtrage et restauration

2. Transformation géométrique

- Quelle différence y-a-t-il entre la méthode à plus proche voisin et la méthode bilinéaire?

L' interpolation à plus proche voisin s'agit d'une dont le valeur d'un point est estimé en utilisant le valeur de le plus proche voisin dans le grille, alors il faut seulement calculer qui est le voisin plus proche.
L'interpolation à la méthode bilinéaire utilise plus de points pour faire l'interpolation. On utilise quatres points voisins et calcule une ligne entre eux, après, on utilise ces liaisons pour faire le calcul pour le point manquant. L' interpolation à la méthode bilinéaire demande plus de calculs, mais elle est plus précise.

- Que constatez-vous sur une image qui aurait subi huit rotations de 45 degrés (en bilinéaire et en plus proche voisin)?

Plus proche voisin:

Méthode bilinéaire:

Je peux constater que l'image qui a subi les rotations en méthode à plus proche voisin est plus nette, mais il y a des défauts d'interpolation plus visibles que dans l'image qui a subi les rotations en utilisant l'interpolation bilinéaire.

- Que constatez-vous si vous appliquez la rotation avec un facteur de zoom inférieur à 1 (par exemple 1/2) ? Qu'aurait-il fallu faire pour atténuer l'effet constaté ?

Je peux constater que l'image ci-dessus est encore plus pixelisée qu'avant. Pour résoudre ce problème il faut faire l'interpolation avec un filtrage passe-bas avant de zoomer l'image.

L'effet de pixellisation est maintenant réduit.

3. Filtrage linéaire et médian

- Expliquer le rapport entre la taille du noyau (size) renvoyé par get_gau_ker et le paramètre cette commande.

Le paramètre s envoyé à get_gau_ker s'agit du écart type du kernel gaussien. L'écart type influence l'ouverture de la courbe gaussienne, la dimension de notre kernel doit être plus grande pour comporter une écart type plus grande. En outre, la dimension de notre kernel doit être impair pour avoir le plus grand valeur au centre du kernel. Le kernel ne peut pas avoir de dimension 1 sinon nous n'aurons pas de filtrage/blur, alors, le taille minimum est 3x3.
Le calcul 2*np.round(2.5, s) est presque la même chose de multiplier s par 5, cette multiplication suffit pour rencontrer un valeur assez grande pour comporter les valeurs importantes de la gaussienne dans le kernel.

- Après avoir ajouté du bruit à une image simple telle que pyramide.tif ou carré_orig.tif et avoir filtré le résultat avec des filtres linéaires, expliquez comment on peut évaluer (sur des images aussi simples) la quantité de bruit résiduel (la commande var_image donne la variance d'une partie d'une image).

On peut mésurer la variance dans un kernel appliqué à une zone homogène. Puis, avec ce resultat ce possible calculer l' écart-type qui caractérise le bruit.

- Appliquer un filtrage médian `a une image bruitée et comparer le résultat avec un filtrage linéaire.

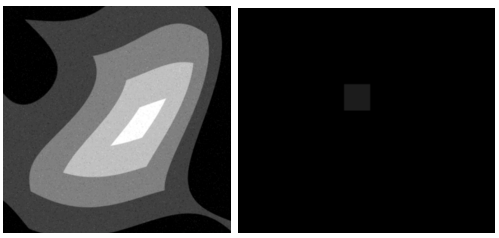
Filtrage linéaire:

Filtre médian:

Le filtre médian a provoqué un effet de flou plus fort sur l'image, dans ce cas le filtre moyen a eu un meilleur résultat.

- Faites une comparaison linéaire/médiane sur l'image pyra-impulse.tif. Que constatez-vous ? Expliquer la différence de comportement entre filtrage linéaire et médian sur le point lumineux situé en haut à droite de l'image carre_orig.tif.

filtre linéaire:



filtre médian:

Pour l'image pyra-impulsionnel le filtre médian a eu un mieux résultat parce que le filtre médian s'applique bien pour le bruit impulsionnel dont les valeurs du bruit sont extrêmes.
Pour le point lumineux dans carre_orig.tif le filtre médian l'a filtré complètement et le point est disparu, parce que le point semble comme le bruit impulsionnel, le filtre linéaire a calculé une moyenne, alors, le point lumineux n'est pas disparu, mais il est devenu plus gris.

4. Restauration

- Appliquez un filtre linéaire à une image puis utilisez la fonction filtre inverse 2 . Que constatez vous ? Que se passe-t-il si vous ajoutez très peu de bruit à l'image floutée avant de la restaurer par la commande précédente ?

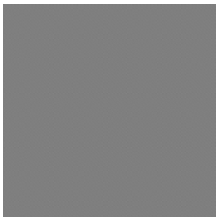
Après appliquer le filtre linéaire, j'ai eu ce résultat:

En appliquant la fonction filtre_inverse sur l'image au dessur:

Nous pouvons voir que le filtre inverse a fait l'image retourner à son état initial.

Si on ajoute de bruit à l'image, le filtre inverse va déranger l'image, parce que si on a de bruit, que sont de haute fréquence, le filtre inverse ne vas pas atténuer, parce que c'est un filtre pas haut, mais va atténuer le reste de l'image que n'a pas toujours de haut fréquence.

=====>

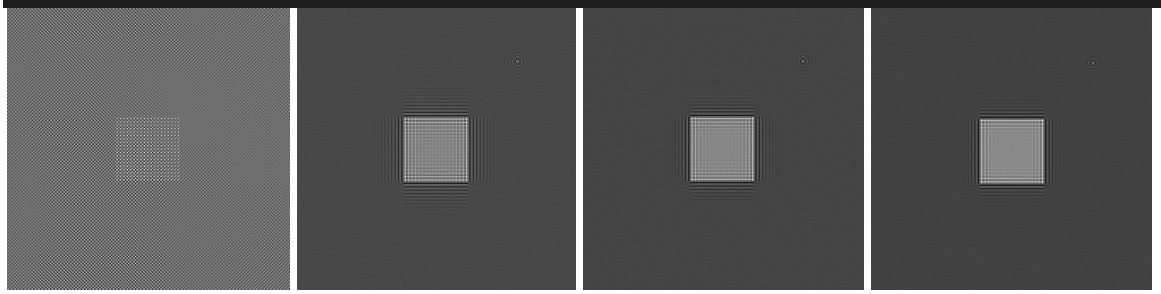


- Comment pouvez-vous déterminer le noyau de convolution qu'a subi l'image carre flou.tif?

L'image carre_orig.tif a un pixel blanc qui prend la forme de la lettre T dans l'image carre_flou.tif, alors, cette forme remet a le noyau convolutionel.
[0, 0, 255]
[255, 255, 255]
[0, 0, 255]

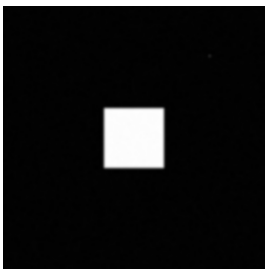
- Après avoir ajouté du bruit à cette image, utilisez la fonction wiener pour restaurer cette image.

```
i = 0
im=skio.imread('images/carre_orig.tif')
im = noise(im, 2)
while (i < 10):
    im_orig = wiener(im, np.ones((3, 3)), i)
    i = i + 3
    viewimage(im_orig)
```

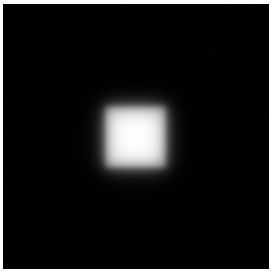


On peut observer que l'image se rapproche de l'initiale quand on engrand le valeur de lambda, bien que on peut atteindre une valeur optimale, parce que si on engrand beaucoup le valeur de lambda l'image va avoir de bleur.

lambda = 5000



lambda = 100000



5. Applications

- Pour une image simple telle que carre orig.tif et un bruit d'´ecart-type 5, trouver la taille du noyau constant qui r´eduit le bruit dans les mˆemes proportions qu'un filtre m´edian circulaire de rayon 4. (explicitez l'algorithme utilis´e)

```
# On ouvre l'image et choise l'écart-type du bruit
im = skio.imread('images/carre_orig.tif')
inputDp = 5
im = noise(im, inputDp)

# Puis on calcule le rapport entre les écart-types avant et après
filtrage
outputDpMedian = var_image(median_filter(im,typ=2,r=4), 0, 0, 50,
50)**0.5
dpRatioMedian = inputDp/outputDpMedian
print(outputDpMedian)

# Puis, on va parcourir les tailles du kernel et à chaque
itération
# on va calculer le rapport entre les écart-types pour le
filtrage par noyau
# si on rencontre un valeur supérior a l'écart-type calculé avant
# on arrête le loop et prendre ce taille du kernel
i = 1
kernel_gaus = get_cst_ker(i)
im1 = filtre_lineaire(im.copy(), kernel_gaus)
outputDpNoyau = im1[0:12,0:12].var()**0.5
```

```
dpRatioNoyau = inputDp/outputDpNoyau
while(((outputDpNoyau > outputDpMedian)) and i < 7):
    i = i + 2
    kernel_gaus = get_cst_ker(i)
    im1 = filtre_lineaire(im.copy(), kernel_gaus)
    outputDpNoyau = im1[0:12,0:12].var()**0.5
    dpRatioNoyau = inputDp/outputDpNoyau


    # On montre les valeurs
    print(f'La taille du kernel est: {i}')
    print(f'L\'écart-type trouvé: {outputDpNoyau}')
```

5.2 Calcul théorique du paramètre de restauration

```
def wienerSpectre(im,K):
    fft2=np.fft.fft2
    ifft2=np.fft.ifft2
    (ty,tx)=im.shape
    (yK,xK)=K.shape
    KK=np.zeros((ty,tx))
    KK[:yK,:xK]=K
    x2=tx/2
    y2=ty/2

    #tranformee de Fourier de l'image degradeee
    g=fft2(im)
    sigmaS2 = abs(g)**2
    var_bruit = var_image(im, 0, 0, 50, 50)
    #transformee de Fourier du noyau
    k=fft2(KK)
    #fonction de mutilplication
    mul=np.conj(k)/(abs(np.conj(k))**2+(var_bruit/sigmaS2))
    #filtrage de wiener
    fout=mul

    # on effectue une translation pour une raison technique
    mm=np.zeros((ty,tx))
    y2=int(np.round(yK/2-0.5))
    x2=int(np.round(xK/2-0.5))
    mm[y2,x2]=1
    out=np.real(ifft2(fout*(fft2(mm))))
    return out
```