IMA201-TP 2 : COMPTE-RENDU

Antoine VERIER

Transformation géométrique

Lena avec rotation ppv **Lena avec rotation bilinéaire**

Visuellement, on observe une netteté plus faible pour la méthode des plus proches voisins : notamment au niveau des bords donc hautes fréquences (chapeau, épaules), les bords sont moins lisses pour la méthode des plus proches voisins. La méthode des plus proches voisins est une méthode plus grossière d'interpolation.

Lena 8 rotations ppv **Lena 8 rotations bilinéaire**

On observe que les 8 rotations avec la méthode du plus proche voisins dégradent fortement la qualité de l'image tandis que les 8 rotations bilinéaires ajoutent légèrement du flou mais la qualité de l'image reste acceptable.
L'apparition des 4 coins noirs est dû au zoom de l'image pendant les rotations de 45° pour ne perdre aucune informations.

Lena avec zoom 1/2

Si on applique la rotation avec un facteur de zoom inférieur à 1, le bruit de l'image augmente : il aurait fallu appliquer un filtre passe-bas avant d'appliquer la rotation de l'image pour gagner en qualité (on aurait eu un peu de flou en fonction du noyau choisi). On utilise le paramètre clip à faux pour atténuer cet effet.

Filtrage linéaire et médian

Get_gau_ker : le paramètre s permet de déterminer le support de la gaussienne, c'est l'écart type de la gaussienne (taille du noyau) : `int(max(3,2*np.round(2.5*s)+1))`

Le noyau est de taille minimale 3.

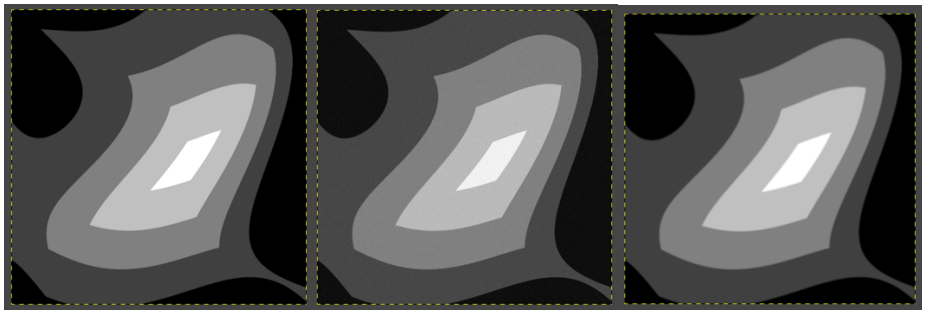


Image originale **Image bruitée (var=3)** **Image filtré (linéaire)**

Sur des zones homogènes (sans discontinuité, variance nulle sur le patch), on peut évaluer le bruit résiduel. En effet, si on prend une zone de l'image précise (où c'est noir par exemple) : l'image bruité possède une variance de 9 sur ce patch tandis que l'image filtré linéairement possède une variance très faible (en appliquant un noyau de taille 3).

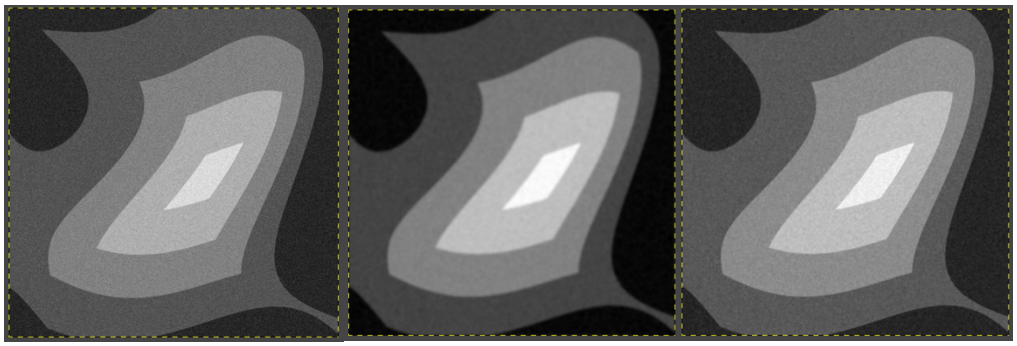
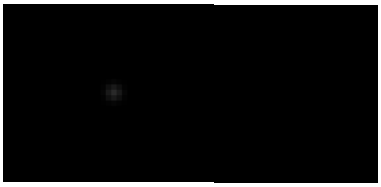


Image avec bruit (var de 10) Image filtre linéaire Image filtre médian

Dans le cas d'un bruit gaussien, le filtrage linéaire semble plus performant que le filtre médiant (sur les zones de basses fréquences et sur les zones de hautes fréquences). L'image du filtre linéaire est plus floue mais possède moins de bruit. Le filtre médian garde plus de bruit mais les bords sont davantage respectés.

Pyra impuls(bruit impulsionnel) Filtre linéaire(filtre gaus) Filtre médian

Le filtre médian est plus efficace pour supprimer le bruit impulsionnel. En effet, la médiane permet de reconstruire plus facilement et efficacement les pixels perdus.



Filtre moyen, Filtre médian
Zooms sur le point lumineux de l'image initiale

Le filtre moyen atténue le pixel en haut à droite. En effet, il y a seulement un pixel non nul donc en réalisant la moyenne, cela atténue le point lumineux sans le faire disparaître.

Le filtre médian supprime le pixel en haut à droite. En effet, + de la moitié des pixels sont noirs donc en appliquant le filtre, cela va prendre la valeur des pixels aux alentours qui sont noirs.

Restauration

Image originale **Image filtre lin (masque gaussien)** **Image filtre inverse**

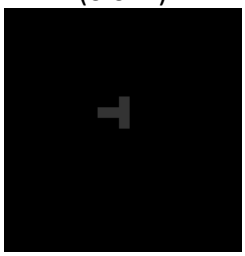
On retrouve l'image originale en appliquant le filtre inverse au filtre linéaire. On a utilisé le même noyau pour le filtre linéaire et le filtre inverse. Si on a Y = A*X avec Y ce qu'on observe (image filtre lin) et on veut retrouver X (image originale), il suffit d'appliquer l'inverse (A-1) : c'est ce qu'on a fait ici. A est le masque gaussien que j'ai choisi.

En ajoutant du bruit avant le filtre inverse :

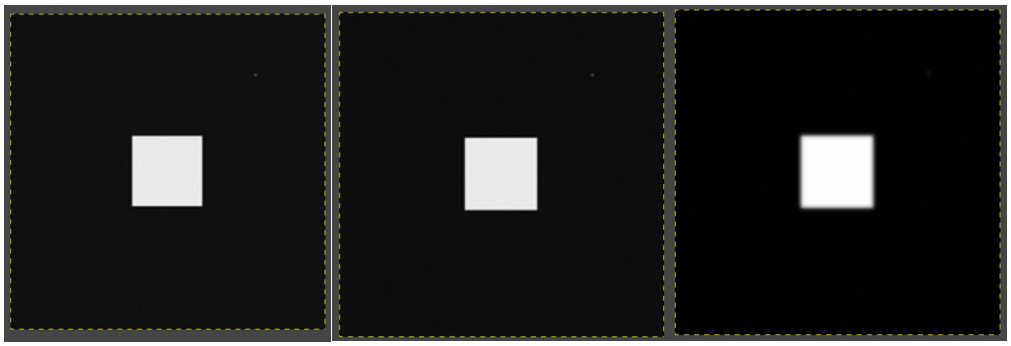
La reconstruction est mauvaise. En effet, dans ce cas on a Y = A*X + B avec B le bruit qu'on ajoute à l'image. Or ensuite on applique A-1 mais on omet l'ajout du bruit donc l'image reconstruite n'est pas fidèle à l'image originale.

Le point lumineux devient un « bloc », donc cela nous donne le noyau de convolution qui est une matrice de taille 3x3 car il y avait seulement un pixel blanc dans l'image initiale :

(0 0 1)
(1 1 1)
(0 0 1)



On retrouve bien l'image originale en appliquant le filtre inverse avec le noyau précédent.



Wiener lambda = 1 Wiener lambda = 2 Wiener lambda = 50

Plus lambda est élevé, plus les hautes fréquences ne sont pas bien conservés par la fonction Wiener. Lambda = 2 semblent convenir

Applications

Comparaison filtre linéaire et médian

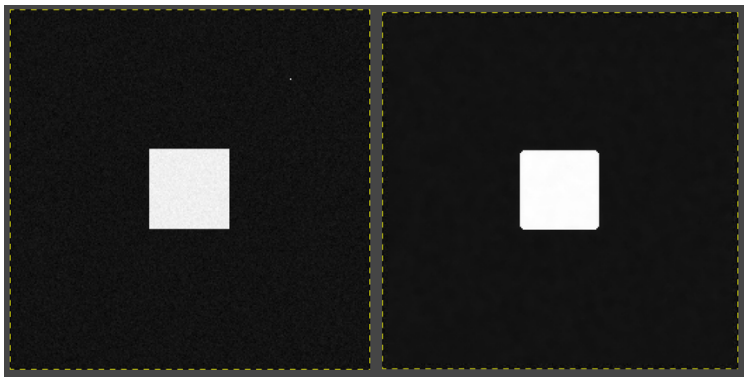


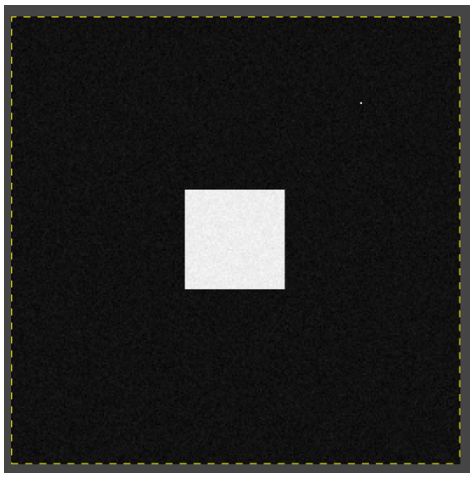
Image bruitée (ecart type de 5) Image filtre médian de rayon 4

```
# %% APPLICATIONS COMPARAISON FILTRAGE LINÉAIRE ET MÉDIAN

im = skio.imread('images/carre_orig.tif')

imnoise = noise(im,5)
viewimage(imnoise)
immedian = median_filter(imnoise, typ =2, r=4)
varmedian = var_image(immedian,0,0,1,1)
viewimage(immedian)
for i in range(100):
    imlin = filtre_lineaire(imnoise, get_cst_ker(i))
    varlin = var_image(imlin,0,0,1,1)
    if np.around(varmedian,3) == np.around(varlin,3):
        print(i)
```

On obtient i = 1, l'image restaurée donne ceci :



Calcul théorique du paramètre de restauration

```
def wiener2(im,K,lamb=0):
    """effectue un filtrage de wiener de l'image im par le filtre K.
       lamb=0 donne le filtre inverse
       on rappelle que le filtre de Wiener est une tentaive d'inversion du noyau K
       avec une regularisation qui permet de ne pas trop augmenter le bruit.
       """
    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

    fX=np.concatenate((np.arange(0,x2+0.99),np.arange(-x2+1,-0.1)))
    fY=np.concatenate((np.arange(0,y2+0.99),np.arange(-y2+1,-0.1)))
    fX=np.ones((ty,1))@fX.reshape((1,-1))
    fY=fY.reshape((-1,1))@np.ones((1,tx))
    fX=fX/tx
    fY=fY/ty

    imt=np.float32(im.copy())
    f_transform = np.fft.fft2(imt)
    f_transform_shifted = np.fft.fftshift(abs(f_transform))
    psd = np.abs(f_transform_shifted) ** 2
    sigmab = im.var()*im.size
    rapport = sigmab/psd


    w2=fX**2+fY**2
    w=w2**0.5

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