Culture Numérique : Atelier 3 (suite)

Prise en main (PIL)

L'objectif de cet atelier est dans une première partie d'utiliser des fonctions utilitaires, ouverture d'un fichier image, affichage d'une image, transformation d'images couleurs en images intensités, etc.

Nous voulons programmer par la suite certaines fonctions de traitement d'images vues dans les ateliers 1 et 2.

La manipulation d'images sera gérée à l'aide de la bibliothèque PIL (Python Imaging Library) qui permet de charger une image sur le disque dur, accéder aux éléments de l'image, afficher ou sauvegarder une image.

/1 Télécharger le fichier « utilitaire2.py »

Ce fichier contient plusieurs fonctions de traitement d'images. Dans ce TP nous allons tester ces différentes fonctions.

Rappel des fonctions vues dans le précédent TP :

```
def ouvrir(nomimage):
    #creer l'objet image  à partir du fichier image, exemple : nomimage='lena.jpg'
    img=Image.open(nomimg)
    # afficher les propriétés de l'image
    print img.format, img.size, img.mode
    # renvoie l'objet image  a partir du fichier image
    return img

def affiche(img):
     # affiche l'image
     img.show()

def imc2img(imc,n): #transforme une image couleur en une image de niveaux de gris
#fonction imc2img qui prend en entrée une image couleur et le numéro d'une bande
#(R,V,B) qui renvoie une image en niveau de gris.
    img=Image.new("L",imc.size)# cree une image de même taille que imc
    pixc=imc.load() # associe le fichier image avec un tableau des pixels de imc
    pixg=img.load() # le tableau des pixels permet la manipulation et les calculs
    width=imc.size[0] # le nombre de colonnes de l'image
    height=imc.size[1] # le nombre de lignes de l'image
# parcours de l'image : lignes/colonnes
```

```
    for j in range(height):
        for i in range(width):
            pixg[i,j]=pixc[i,j][n]
     img.save("img.bmp")# pour sauvegarder
    return img
# exemple d'utilisation
```

Le fichier utilitaire2.py contient les fonctions suivantes :

```
def ouvrir(nomimg):
     …

def affiche(img):
     …

def imc2img(imc, n):
     …

def moyenne(ims):
     …

def moyenneIterative(ims, n):
     …

def mediane(liste):
     …

def MedianeImage(ims):
     …

def MedianeIteratif(ims,n):
     …

def dilatation(ims):
     …

def dilatationIterative(ims, n):
     …

def erosion(ims):
     …

def erosionIterative(ims, n):
     …

def difference(ims0, ims1):
     …

def somme(ims0, ims1):
     …

def binarization(ims, seuil):
     …
```

1-a) La fonction **moyenne(ims)** permet de filtrer une image d'intensité. Tester la fonction **moyenne(ims)** sur une image de votre choix.

b) Même question pour **moyenneIterative(ims, n)** avec n=5 et n=20. Que constatez-vous ?

c) Même question pour **MedianeImage(ims)** et **MedianeIteratif(ims,n)** avec n=5 et n=20.

2-a) Calculer une image **ims1=moyenneIterative(ims, 5)**.

b) Calculer une image **ims2=moyenneIterative(ims, 20)**

c) Calculer la différence entre ims1 et ims2, on pourra utiliser la fonction **imdiff = difference(ims1, ims2).**

d) Binariser l'image imdiff en utilisant la fonction **binarization(imdiff, seuil)** avec seuil=0

e) Ecrire une fonction **polarity(ims)** qui appelle succéssivement les étapes a), b), c) et d)

3-a) Tester la fonction **dilatation(ims)** pour dilater une image.

b) Tester la fonction **erosion(ims)** pour éroder une image.

c) Ecrire une fonction **gradient(ims)** qui calcul la difference entre une image dilatée et une image érodée. Afficher l'image obtenue.

d) Binariser l'image obtenue à l'étape précédente en utilisant la fonction **binarization(ims, seuil)** avec seuil=20