5 T.P. TRAITEMENT D'IMAGES

✎ Commencez par enregistrer sous votre répertoire personnel les images suivantes, placées dans le dossier « Image » du répertoire de la classe.

`radiologie1.png`, `radiologie2.jpg`, `granitePolarise.jpg` et `lentillesSc.jpg`

1 Introduction

Lu dans le programme officiel : « Un exemple important de traitement numérique est fourni par les images ponctuelles (ou bitmap) en niveaux de gris ou en couleurs, qui apparaissent fréquemment dans de nombreux contextes expérimentaux ou appliqués (radiologie, échographie, etc.). Les algorithmes de transformation ou d'extraction permettent d'analyser des structures, distributions, comportements et peuvent participer à des démarches de diagnostic. »

On demande donc la connaissance de quelques algorithmes simples opérant sur une image ponctuelle. On connaîtra en particulier des algorithmes du type « à balayage » : éclaircissement, accentuation du contraste, flou, accentuation du contours.

2 Préambule

Il existe de nombreux formats d'image. Les plus fréquent est le format « .jpg » utilisé par exemples par les appareils photo numériques mais il a le défaut de comprimer les images au détriment de la richesse de l'information. On trouve ensuite le format « .bmp » essentiellement dans des contextes scientifiques. Il occupe plus de mémoire mais ne comprime en rien l'image.

Dans tous les cas, si l'image est composée de $m \times n$ pixels, chacun d'entre eux sera codé de façon différente selon que l'image est en niveaux de gris ou pas. Si c'est le cas, l'intensité lumineuse sera codée sur 8 bits et l'image aura deux dimensions. Sinon, dans le cas des images en couleur, c'est l'intensité lumineuse des trois couleurs fondamentales Rouge, Vert et Bleu qui est transcrite dans le domaine de sensibilité de l'oeil humain *via* un triplet de valeurs également codées sur 8 bits. On parle alors de mode « RGB » (pour Red, Green, Blue).
Chaque intensité lumineuse de chaque pixel est donc codée par une valeur comprise entre 0 et $2^8 - 1 = 255$, ce qui permet de distinguer $16,7$ million de couleurs différentes ($256 \times 256 \times 256$).

A titre d'exemple, un pixel noir sera (0,0,0), un pixel blanc sera (255,255,255), un pixel rouge sombre (255,0,0), un pixel vert (0,255,0), un pixel bleu (0,0,255), un pixel magenta (255,0,255), un pixel jaune (255,255,0) etc.

Pour se familiariser avec cet encodage, ouvrir l'image `granitePolarise.jpg` à l'aide d'un logiciel de traitement d'images et pipetez quelques couleurs pour récupérer les coordonnées en mode RGB.

3 Bibliothèques et acquisition des images

Dans l'ensemble de ce TP nous travaillerons avec la bibliothèque `matplotlib.image`.

A priori, `matplotlib` ne permet de travailler qu'avec des images au format « .png ». Ça n'a rien de gênant si on considère qu'il est facile d'utiliser un logiciel de traitement d'image pour exporter au format « .png » des images de format « .bmp » ou « .jpg ». Pour autant on évitera si on le peut ces exportations qui se font au détriment d'une perte d'information et donc de la qualité de l'image.
Aussi utilisera-t-on le module `pillow` qui, une fois installé, permet l'acquisition et le traitement par `matplotlib.image` des formats d'image les plus usités.
✐ Si jamais `pillow` n'est pas installé, il suffira pour l'utiliser de taper sous Pyzo :

```
conda install pillow
```

Les indispensables : : Les images seront lues et affichées grâce aux fonctions `imread` et `imshow`.

A titre de modèle :

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

img = mpimg.imread('radiologie1.png')
(hauteur,largeur,ncouleur) = img.shape # hauteur = nb_lignes ; largeur =
nb_colonnes
plt.imshow(img)
plt.axis(off) # pour effacer l'affichage des axes
plt.show()
```

✐*Remarque :* Pour éviter de définir le chemin où se trouve l'image, on travaillera dans le répertoire où elle se trouve et au premier appel de la fonction on fera un clic droit sur l'onglet de la fonction PYTHON, puis `Exécuter en tant que script`.

Attention ! Observer et comparer le type des valeurs composant `img1 = mpimg.imread('radiologie1.png')` et de `img2 = mpimg.imread('radiologie2.jpg')`.

- Si l'image est au format « .png », par défaut `matplotlib` n'utilise pas `pillow`, et `img` sera une matrice directement exploitable par `numpy`, codée au format : `float32`. On doit noter que les trois niveaux d'image, codés sur 8 bits, ont été transcris en réels flottants, tous compris entre 0 et 1.

- Pour tout autre format, `matplotlib` utilise `pillow` pour les interpréter et `img` sera codé au format `uint8` spécifique des formats d'image.
  Les valeurs, comprises dans l'intervalle $[\![0, 255]\!]$, ne devront pas donner lieu à des opérations algébriques. Pour s'en convaincre, prendre `a = img1[0,0,0]` et calculer `a+a, a+a+a`.

Remarque : Pour une image en trois dimensions, on accédera à chaque niveau de couleur primaire en écrivant : `imgR = img[:,:,0], imgG = img[:,:,1]` et `imgB = img[:,:,2]`

Image en niveau de gris : On calcule, par combinaison linéaire des valeurs RGB, une luminance monochrome. Les coefficients retenus sont ceux qui rendent compte de la sensibilité oculaire (norme standard « NTSC »). Fonctions utiles :

```
Imagengris = .2989 * imgR + .5870 * imgG + .1140 *imgB
plt.imshow(Imagengris,cmap = "Greys_r")
mpimg.imsave (chemin+'nomImage.format',Imagengris) # pour sauvegarde
plt.hist(Imagengris.flatten(),bins =256)
```

Exercice : Compléter la fonction `histPerso()` permettant de retourner à partir d'une image en niveaux de gris une matrice ligne contenant les effectifs de chaque luminance comprise entre 0 et 255.
En faire l'application à `radiologie2.jpg` transformée au préalable en niveaux de gris ainsi qu'à `granite-Polarise.jpg` dont on donnera l'histogramme de chaque niveau de couleur.

✎ **Remarque :** Pour obtenir, les dimensions d'une image (2 ou 3), il suffit d'utiliser la méthode `ndim`

`img.ndim` vaut 3 et `Imagengris.ndim` vaut 2.

4 Création d'une image

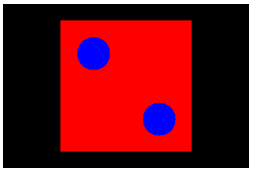
Il est souvent utile de pouvoir créer une image qui illustre, par exemple, les résultats d'une modélisation. Si on souhaite une image en couleur, il suffit de créer un tableau numpy à trois dimension (hauteur,largeur,3) en initialisant chacune de ses valeurs à zéro (tableau noir). On exécutera pour ça :

```
imgCree = numpy.zeros((hauteur,largeur,3))
```

(Si l'image est en niveaux de gris, on se contentera de `imgCree = numpy.zeros((hauteur,largeur))`)

Il suffit alors de modifier les coefficients du tableau pour former l'image de son choix, à condition que les formes en jeu soient relativement simples...

Exemple : Compléter la fonction `creeImage()` pour créer et afficher, comme ci-dessous, la face 2 d'un dé cubique de côtés 160×160, inscrite sur un fond noir de hauteur 200 et largeur 300.



5 Quelques algorithmes de transformation

✎ **Important :** On souhaite faire des transformations sur une image sans pour autant altérer l'image d'origine. On commencera donc par la copier en se souvenant que l'affectation `B = A` n'est pas, sous Python, une copie mais créé une variable `B` qui pointe vers le même objet que `A`.
On écrira donc :

```
from copy import deepcopy
B = deepcopy(A)
```

5.1 seuillage

5.1.1 Le cas des images en couleur

Supposons qu'on dispose d'une image en couleurs sur laquelle on souhaite n'afficher que les zones dont la couleur est proche de celle d'un point A de couleur connue. Ce travaille est utile, par exemple, dans le cadre de recherche de certaines surfaces...
On pourra utiliser la distance euclidienne et dire que la « distance » entre les couleurs d'un point M et du point G est faible si, ε étant fixé par l'utilisateur :

$$d(M,A) = \sqrt{(R_A - R_M)^2 + (G_A - G_M)^2 + (B_A - B_M)^2} < \varepsilon$$

Il suffit alors d'initialiser un tableau noir `imageSeuillee` de même taille que celle de l'image à traiter. Pour tout point M dont la distance en couleur à A est inférieure à ε, on affecte la valeur 1 dans `imageSeuillee`. Dès lors, les zones dont la teinte est proche de celle de A s'affichent en blanc.

Exercice : Prendre l'image `granitePolarise.jpg` et, après avoir complété la fonction `seuillage1(img,coule` afficher une image seuillée des zones proches du bleu codé en RGB par $(78, 66, 168)$.

✐ ***Remarque méthodologique :*** Il peut être utile de récupérer, *via* un simple clic, une couleur à partir d'une image affichée à l'écran.

– On utilisera en premier lieu :

```
coord = plt.ginput(n=1)
```

Une image étant affichée dans la fenêtre active, cette méthode attend un clic gauche de la souris sur l'image pour retourner une liste de la forme $[(a, b)]$ dans laquelle (a, b) désigne les coordonnées du point prélevé.
✐ *Noter que :* `coord[0]=(a,b)` et donc `coord[0][0]=a` et `coord[0][1]=b`

Attention : Par défaut, les coordonnées sont données dans un repère dont l'origine est dans le coin supérieur gauche de l'image, l'axe des abscisses étant l'axe horizontal et l'axe des ordonnées l'axe verticale.
En conséquence, si M a pour coordonnées (a, b), cela signifie qu'il est sur la ligne b et la colonne a de la matrice *img* ! !.. soit :

`ligne = coord[0][1]` et `colonne = coord[0][0]`

– On récupère ensuite la couleur du pixel sélectionné en écrivant :

```
couleur = (img[ligne,colonne,0],img[ligne,colonne,1],img[ligne,colonne,2])
```

5.1.2 le cas des images en niveaux de gris

On commencera par accentuer le contraste (voir section suivante). On utilisera ensuite la fonction `histo(img)` pour déterminer un seuil permettant de scinder l'image en deux zones distinctes.
Il suffit alors de demander :

```
plt.imshow(imgNbContrastee>seuil,cmap="Greys_r")
```

5.2 Accentuation du contraste

On travaillera ici avec l'image `lentillesSc.jpg`.

① Commencer par transformer cette image en niveaux de gris.

② Montrer en quoi l'histogramme de cette image permet d'en augmenter le contraste.
Écrire une fonction `ameliore_contraste1()` permettant, une image du type `Grayscale` et son histogramme étant données, de retourner une nouvelle image dont les pixels sont codés de 0 à 255.

✐ On veillera à taper `plt.imshow (img,cmap='Greys_r',vmin=0,vmax=255)` pour vérifier l'accentuation du contraste (sans quoi `matplotlib`, qui est très performant, trace par défaut les images avec un contraste maximum...).

③ Étudier la fonction $f \longmapsto 1/2 + 1/2 \times sin(\pi(x/255 - 1/2))$.
Expliquer en quoi elle permet d'améliorer le contraste obtenu précédemment.

5.3 Détection des bords

5.3.1 Opérateur gradient

les coefficients d'une image de type `Grayscale` interprétée matriciellement peuvent-être vue comme les valeurs d'une fonction `intensité lumineuse` en chaque point de coordonnées (i, j). Lorsqu'au voisinage d'un point on assiste à une brusque variation de niveaux de gris, on peut considérer que ce point se situe sur un « bord ».
Un outils analytique est particulièrement bien adapté pour détecter les directions de plus grande pente et l'importance de cette pente. Il s'agit du **gradient** dont la norme sera un bon indicateur des dénivelés de la luminance d'une image.

Dans le cas discret, les dérivées partielles de la fonction `img` peuvent être approximées par de simples différences :

$$\begin{aligned}
&\frac{\partial img}{\partial x}(i,j) \approx \Delta_x img(i,j) = img(i+1,j) - img(i,j)\\
&\frac{\partial img}{\partial y}(i,j) \approx \Delta_y img(i,j) = img(i,j+1) - img(i,j)\\
&\quad \text{et donc } \left\|\overrightarrow{grad(img)}(i,j)\right\| = \sqrt{(\Delta_x)^2 + (\Delta_y)^2}
\end{aligned}$$

Exercice 1 : Écrire une fonction `detectBords1()` qui exploite les propriétés du gradient et retourne une image qui, à partir de `pause.jpg` permet à ma fille Jeanne, sept ans, de faire un coloriage de son héro préféré.

5.3.2 Opérateur de Prewit et Sobel

Dans la pratique, les contours des objets sont rarement suffisamment nets pour observer des saut de luminance importants sur un pixel.
Certains informaticiens ont donc eu l'idée de travailler sur le voisinage de chaque pixel en considérant la matrice 3×3 centrée sur chacun d'eux, à savoir :

$$\begin{pmatrix} (i-1,j-1) & (i-1,j) & (i-1,j+1) \\ (i,j-1) & (i,j) & (i,j+1) \\ (i+1,j-1) & (i+1,j) & (i+1,j+1) \end{pmatrix}$$

Ils ont dans ce contexte, remplacés les dérivées partielles au pixel de coordonnée (i, j) par des dérivées dite « directionnelles » selon l'axe des x et l'axe des y...
On considère alors les « masques » suivants :

$$\begin{pmatrix} 1 & 0 & -1 \\ c & 0 & -c \\ 1 & 0 & -1 \end{pmatrix} \text{ et } \begin{pmatrix} -1 & -c & -1 \\ 0 & 0 & 0 \\ 1 & c & 1 \end{pmatrix}$$

✐ Note : Pour $c = 1$ on parle de masques de Prewitt et pour $c = 2$ de masques de Sobel.

En multipliant terme à terme chaque matrice 3×3 prises sur l'image à étudier par H_x et H_y on obtient en chaque pixel à l'exclusion de ceux de ceux du bord deux dérivées directionnelles qui permettent le calcul du gradient.

Exercice 2 : Écrire une fonction `detectBords2()` qui mette en pratique l'un ou l'autre de ces deux opérateurs.