TP4

Traitement d'images

I Présentation

En informatique, une image en couleurs (au format RGB) est définie par un ensemble de pixels colorés, constitués d'un mélange de trois couleurs : rouge, vert et bleu. Plus précisément, une image de taille (n, m) est la donnée d'une matrice de $n \times m$ pixels, chaque pixel étant repéré par ses coordonnées (x, y) (ligne et colonne) et contenant un triplet (R, G, B).
Chaque composante du triplet (R, G, B) est un entier compris entre 0 et 255 codant respectivement le niveau de rouge, vert et bleu du pixel, 0 représentant l'absence de la couleur concernée, 255 le maximum. Si les trois valeurs du triplet (R, G, B) valent 0 (resp. 255), on obtient du noir (resp. du blanc). Par exemple, le triplet $(255, 0, 0)$ code du rouge.

Nous allons présenter deux possibilités de travail sur les images sous Python :

- En travaillant directement sur des tableaux de nombres (modules `numpy` et `matplotlib.pyplot`).
- En utilisant un module dédié au traitement d'images (module `Image` de la bibliothèque `PIL`).

II Utilisation des tableaux `numpy`

Pour représenter les pixels de l'image, Python utilise un tableau à trois dimensions. Le premier indice désigne la ligne, le second indice la colonne et le troisième indice désigne le canal de couleur (0 pour rouge, 1 pour vert et 2 pour bleu).
Pour créer un tel tableau à trois dimensions, nous allons utiliser le module `numpy` de Python que l'on importera avec la commande `import numpy as np`.
Ce module contient une fonction `np.zeros(a,b,c)` qui permet de créer un tableau de dimension `a×b×c` dont toutes les composantes sont nulles (image noire).

```
import numpy as np
im1 = np.zeros((10, 10, 3))
```

Pour afficher l'image correspondant à un tableau à trois dimensions, nous allons utiliser le module `matplotlib.pyplot` de Python que l'on importera avec la commande `matplotlib.pyplot as plt`. Si le tableau est stocké dans une variable nommée `im`, on peut l'afficher avec les commandes suivantes.

```
import matplotlib.pyplot as plt
im1 = np.zeros((10, 10, 3))
plt.imshow(im1)
plt.show()
```

Exercice 1

Que fait le programme suivant ?

```
import numpy as np
im1 = np.zeros((10, 10, 3))
im1[2,2] = [0, 255, 0]
im1[5,4,2] = 255
```

Exercice 2

1. Créer une image de fond blanc, de taille 500×500 avec un carré rouge de taille 50×100 dont le coin en haut à gauche est à la position $(200,200)$.
2. Créer une image de taille 255×255 qui crée un dégragé de gris par colonne en partant du noir (à gauche) jusqu'au blanc (à droite).
   On obtient du gris lorsque les trois composantes du triplet (R,G,B) sont égales).

III Utilisation du module `Image` de `PIL`

III.1 - Présentation du module

La bibliothèque PIL permet de créer et de manipuler une image. Voici les fonctions et méthodes utiles pour ce TP.

- La fonction **Image.open** permet d'ouvrir une image : La commande

  ```
  photo = Image.open( 'Adresse_Photo\Nom_Photo' )
  ```

  permet d'ouvrir l'image présente à l'adresse indiquée et stocke les données dans une variable nommée `photo` (en fait, photo contient l'adresse mémoire contenant les données).
- La fonction **Image.new** qui permet de créer une nouvelle image (en mode RGB) La commande

  ```
  photo2 = Image.new("RGB", (xsize, ysize))
  ```

  permet de créer une image de taille $(xsize, ysize)$ au format "RGB" et stocke les données dans une variable nommée `photo2`.

On peut ensuite travailler sur les variables `photo` et `photo2` via les méthodes décrites ci-dessous.

- La méthode **show** permet d'afficher une image. La commande

  ```
  photo.show()
  ```

  permet d'afficher l'image contenue dans la variable `photo`.
- La méthode **size** permet de récupérer la taille d'une image. La commande

  ```
  photo.size
  ```

  renvoie un couple contenant les dimensions de l'image.
- La méthode **getpixel** permet de récupérer un pixel donné dans une image. La commande

  ```
  photo.getpixel((x,y))
  ```

  permet de récupérer le pixel situé en coordonnées (x,y).
  On rappelle qu'un pixel est un triplet (R,G,B) d'entiers compris entre 0 et 255.
- La méthode **putpixel** permet de changer la valeur d'un pixel dans une image. La commande

  ```
  photo.putpixel( (x,y) , newpix )  // newpix est un pixel
  ```

  permet d'attribuer au pixel situé en coordonnées (x,y) le contenu de `newpix`.
- La méthode **load** qui permet d'avoir accès à la totalité des pixels de l'image. La commande

  ```
  pix = photo.load()
  ```

  permet de charger tous les pixels dans la variable `pix`.
  La commande

  ```
  pix[2,3]
  ```

  permet alors de récupérer la valeur du pixel en $(2,3)$.

III.2 - Traitement d'image

Exercice 3 (*Mise en place*)

1. Importer le module `Image` de la bibliothèque `PIL` à l'aide de la commande

   ```
   from PIL import Image
   ```

2. Télécharger sur le site l'image nommée `Puissance_et_finesse.jpg` et noter l'adresse absolue de l'emplacement de ce fichier sur l'ordinateur.
3. Ouvrir et afficher **depuis Pyzo** cette image.
   On utilisera la fonction `Image.open` *et la méthode* `show`.
4. Récupérer la valeur de la composante rouge du pixel de coordonnées $(10,5)$.

Exercice 4 (*Transformations basiques*)

1. Écrire une fonction `" Canal(im,k) "` qui prend en paramètres une image `im` et un entier `k` ($k \in \{0,1,2\}$) et qui renvoie une nouvelle image correpondant au canal k de l'image.
   Le pixel `(R,G,B)` *est remplacé par* `(R,0,0)` *(resp.* `(0,G,0)` *et* `(0,0,B)`*) pour le canal* 0 *(resp.* 1 *et* 2*).*
2. Écrire une fonction `" NB(im) "` qui prend en paramètre une image `im` et qui renvoie une nouvelle image en niveau de gris de l'originale.
   On obtient une image en niveau de gris lorsque chaque pixel (R,G,B) *est remplacé par le pixel* (M,M,M) *où* M *est donné par la formule* $M = 0.2126 \times R + 0.7152 \times G + 0.0722 \times B$ *pour se rapprocher le plus possible de la vision humaine (qui privilégie le vert).*
3. Écrire une fonction `" InversionH(im) "` qui prend en paramètre une image `im` et qui renvoie une nouvelle image, symétrie de l'originale par rapport à l'axe horizontal coupant l'image en deux.

Exercice 5 (*Réduction d'une image*)

1. Écrire une fonction `" Reduction(im,k) "` qui prend en paramètres une image `im` et et un facteur de réduction k (entier) et qui renvoie une nouvelle image, copie de l'originale réduite d'un facteur k^2.
   On ne garde qu'un pixel sur k pixels sur la longueur et un sur k sur la largeur.
2. Réduction avec moyenne : on remplace un pixel par la moyenne des pixels que l'on supprime.
   (a) Calcul de la moyenne des pixels voisins.
   Écrire une fonction `" PixelMoyen(im, x, y, k) "` qui prend en paramètres une image `im`, les coordonnées (x,y) d'un pixel et un entier k et qui renvoie un pixel contenant les valeurs moyennes des k^2 pixels contenus dans le carré de sommet (x,y) (en haut à gauche) de côté k.
   (b) Écrire une fonction `" Reduction2(im,k) "` qui prend en paramètres une image `im` et un facteur de réduction k et qui renvoie une nouvelle image, copie de l'originale réduite d'un facteur k^2
   On ne garde qu'un pixel sur k^2 *pixels et on attribue à ce pixel la valeur moyenne des pixels sue le carré de côté k.*

Exercice 6 (*Flouter une image*)

Écrire une fonction `flou(im,k)` qui prend en paramètres une image `im` et un entier k (facteur de flou) et qui renvoie une nouvelle image, copie floutée de l'originale où chaque pixel est remplacé par la moyenne de ses k^2 pixels voisins.

Exercice 7 (*Détection de contour*)

La détection de contour dans une image se fait par l'application d'un masque de gradient en chaque pixel de l'image dont la mise en application est explicitée ci-dessous.

- On transforme l'image en niveau de gris (noir et blanc).
- On crée une nouvelle image qui contiendra le résultat de la détection de contours.
- On applique le masque du gradient à chaque pixel (X,Y) de l'image en calculant la valeur v suivante : $v = \sum_{i=-1}^{1} \sum_{j=-1}^{1} pix(X+i, Y+j) \times A[i+1][j+1]$ où A représente la matrice du masque : $A = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$.
  Cette opération permet de comparer la luminosité de chaque pixel à celle de ses voisins.
- On fixe un seuil. Si pour un pixel donné la valeur obtenue dans l'étape précédente est inférieure à ce seuil, on donne la valeur $(255,255,255)$ au pixel correspondant dans la nouvelle image (blanc). Dans le cas inverse, on donne la valeur $(0,0,0)$ à ce pixel (noir).

Écrire une fonction `" Contour(im,seuil) "` qui prend en paramètre une image `im` et un entier seuil et qui applique la procédure décrite plus haut pour détecter les contours de l'image.
*On pourra utiliser la méthode **load** qui permet d'avoir accès à la totalité des pixels de l'image. Par ex :*

```
pix = photo.load()  // Charge tous les pixels dans la varialbe pix
pix[2,3]            // Affiche la valeur du pixel en (2,3)
```

Exercice 8 (*Rotation*)

Écrire une fonction `" Rotation(im,alpha) "` qui prend en paramètres une image `im` et un angle alpha et qui renvoie une nouvelle image, résultant de la rotation de l'image originale d'un angle alpha.

On rappelle que l'image $A' = \begin{pmatrix} x' \\ y' \end{pmatrix}$ de $A = \begin{pmatrix} x \\ y \end{pmatrix}$ par la rotation d'angle α est donnée par :

$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$.

On importera les fonctions `cos` et `sin` du module math via la commande : `from math import cos, sin`.