TP 9 : TRAITEMENT DES IMAGES, STÉGANOGRAPHIE

Introduction

L'objectif de ce TP est d'aborder quelques algorithmes classiques de traitement d'images, et de les mettre en œuvre avec Python.
Une manière de représenter informatiquement une image est de l'échantillonner en petits éléments (picture element : pixel) supposés de couleur uniforme, suivant un quadrillage. L'image est alors représentée informatiquement par une matrice (carte de points : bitmap), chaque case de la matrice donnant la couleur du pixel correspondant du quadrillage. Les formats les plus connus d'images bitmap sont JPEG, BMP, GIF, TIFF et PNG. La résolution de l'image dépend alors de la finesse du quadrillage.

Selon le format de l'image bitmap, la palette de couleurs est encodée sur un certain nombre de bits, appelée profondeur de l'image (exprimée en bpp = bits par pixel). Les principaux cas sont :

- les images en noir et blanc : 1 bit par pixel
- les images en niveaux de gris : chaque nuance est encodée sur 1 octet, par un nombre compris entre 0 (pour le noir) et 255 (pour le blanc).
- les images couleur au format RGB (Red, Green, Blue). Chaque pixel est décrit par une quantité de rouge, une quantité de vert, et une quantité de bleu. La couleur résultante est obtenue par synthèse additive. On utilise 1 octet pour la quantité de rouge, 1 octet pour la quantité de vert, et 1 octet pour la quantité de bleu, si bien qu'une couleur est représentée sur 3 octets=24 bits. En pratique, on donne un triplet de nombres entre 0 et 255 ((255,0,0) : rouge , (0,255,0) : vert , (0,0,255) : bleu).

En pratique, pour transformer notre image en tableau numpy, nous utiliserons le module PIL (aucune connaissance sur ce module ou un autre n'est exigible).

```
from PIL import Image
import numpy as np
Im = Image.open("image.jpeg")
ImTab = np.array(Im)
```

`ImTab` contient alors un tableau numpy à 3 dimensions (dans la cas d'une image en couleur, 2 pour une image en niveau de gris) de format $(h, l, 3)$, h étant la hauteur de l'image et l sa largeur. On peut alors obtenir cette taille par : `np.shape(Imtab)` . L'exécution de la commande `ImTab[i,j]` renvoie alors une liste de 3 éléments (niveaux de rouge, de vert et de bleu). On aurait pu utiliser également, pour obtenir le même résultat, `ImTab[i,j,:]` (ce qui aurait été plus en phase avec les notations numpy).
Pour afficher une image à partir de son tableau représentatif :

```
Im=Image.fromarray(ImTab)
Im.show()
```

Pour la sauvegarder : `Im.save('image.png',"PNG")` (on peut utiliser d'autres format comme .jpeg ; on changera bien sûr le nom si l'on a créé une nouvelle image).

1 Premières manipulations

Cette première partie est constituée de manipulations élémentaires d'images afin de prendre en main la façon dont elles sont représentées.
On testera les fonctions sur l'image fournie. On pourra enregistrer (ou pas) les images affichées.

1.1 Découpage, symétrie

Question 1.1 – Écrire une fonction `sym` qui prend en entrée un tableau représentant une image et renvoyant le tableau représentant l'image symétrique (par rapport à la droite la coupant verticalement en son milieu). Pour créer un nouveau tableau, on utilisera la commande `np.zeros((h,l,3),dtype=np.uint8)` (h et l bien choisis). Remarquons que par défaut les éléments d'un tableau numpy sont des flottants ; on a imposé ci-dessus le type `int` , codé sur un octet.

Question 1.2 – Écrire une fonction `coin` qui prend en entrée un tableau représentant une image et renvoyant le tableau représentant l'image située dans le coin en bas à droite de l'image (hauteur 200 pixels, largeur 300). On suppose que le tableau initial est assez grand.

1.2 Autour des couleurs

Question 1.3 – Écrire une fonction `neg` qui prend en entrée un tableau `Tab` représentant une image et renvoyant un tableau représentant le négatif de cette image.
Par exemple, si un pixel a pour composantes RGB (232,12,120), alors son pixel correspondant dans le négatif a pour composantes RGB (255-232, 255-12, 255-120).

2 Exemple de masque

Le traitement des images s'appuie souvent sur la transformation de l'image, donnée sous forme d'un tableau I de pixels, par un masque, donné sous forme d'un tableau (matrice carrée) A. Si par exemple A est de taille 3×3, le pixel $I_{i,j}$ est alors transformé en :

$$I'_{i,j} = \sum_{0 \leq k,l \leq 2} A_{k,l}\ I_{i-1+k,j-1+k}\ .$$

Question 2.1 – Écrire une fonction `convol(I,A)` qui transforme `I` par le masque `A` selon le principe expliqué ci-dessus (lorsque `I` est un tableau représentant une image couleur).
La matrice A est supposée de taille $p \times p$; on fera attention aux bords (que l'on ne traitera pas).

On souhaite flouter l'image en remplaçant chaque pixel par la moyenne de ceux contenus dans le carré 3×3 dont il est le centre. On dit alors que l'on applique le masque (ou filtre) :

$$\frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}\ .$$

Question 2.2 – Écrire une suite d'instructions qui floute notre image test selon le principe décrit ci-dessus.
Appliquer plusieurs fois cette fonction à notre image et observer l'effet.

3 Stéganographie

La stéganographie est l'art de dissimuler un contenu. Pour les images, ont peut utiliser les bits de poids faible. Plus précisément : pour chaque pixel, chacun de nos niveaux de R, G, B est codé par un entier entre 0 et 255, qui peut être lui même codé en base 2 sur un octet. On ne modifie pas énormément l'image en modifiant les bits de poids faible de ces nombres. Mieux : à la place de ces quatre bits de poids faible, on peut dissimuler les bits de poids fort d'une autre image, et donc récupérer celle-ci par un traitement approprié.

Question 3.1 – Quelle image se cache dans `image_myst.png` ?
Remarque : on pourra tout d'abord programmer une fonction `baseDeux(n)` qui prend en argument un entier n compris entre 0 et 255 et qui renvoie sa décomposition en base 2 sous forme d'une liste `l` de huit éléments ; ainsi que sa réciproque `baseDix(l)` .
Remarque : attention à la convention adoptée (poids faible à gauche ou à droite de la liste).