CMP 2015 - TD Python 7

1. Vectorialisation - Révision.
   On travaille avec les tableaux **numpy**. Le nom des tableaux numpy est en principe **ndarray**.
   Exemple : on dispose d'un tableau d'abscisses X et d'un tableau d'ordonnées Y, de même taille. Créer le tableau des normes des vecteurs. Essayer sur un exemple.
   On peut créer des tableaux numpy d'une forme donnée, ce qui comprend les tailles. Par exemple :
   shape=(2,3)
   X=np.ndarray(shape).
   Essayer...

2. Importer une image.
   De la bibliothèque **matplotlib**, il convient d'abord d'importer **pyplot** (alias plt), **image** (alias mim, pour **m**atplotlib**im**age, par exemple), et **cm** (cm est déjà un nom résumé pour colormap).
   Chercher une image (décente...) sur la toile et l'importer dans votre dossier. L'image doit être au format .png. (Jeter un oeil (Wikipedia...) sur ce qu'est le format .png). La syntaxe est :
   monimage = mim.imread('cequej'aitrouvébeauetintéressantàexploiter.png ').
   Vérifier la nature de ce que vous avez créé : **type(monimage), monimage.ndim, monimage.shape**.
   Si votre image est de type $n \times p \times 3$, elle correspond (en pixels) à n lignes (hauteur), p colonnes (largeur), et trois nombres donnant la valeur du Rouge, du Vert, du Bleu, donc codage RGB.
   Si votre image est de type $n \times p \times 4$, elle correspond à n lignes (hauteur), p colonnes (largeur), et quatre nombres donnant la valeur du Rouge, du Vert, du Bleu, et la transparence, donc codage RGBA.
   On va beaucoup utiliser le tranchage (*slicing*). Vous pouvez voir les composantes du pixel (i,j) : **monimage[i,j,3]**. Ou toutes les composante rouges : **monimage[:, :,0]**.
   Nous n'utiliserons pas ici la transparence, qu'on pourra oublier en retenant **monimage[:, :, :3]** (pourquoi ?...).

3. Luminances.
   On crée la luminance en affectant chaque couleur d'un coefficient, soit $L = pR + qG + rB$, avec $p + q + r = 1$. Ceci peut se faire avec un produit scalaire : celui du triplet (R, G, B) avec (p, q, r). Le produit scalaire s'obtient avec **np.dot**.
   On demande le travail suivant (pour voir comment les coefficients des couleurs peuvent être améliorés) :
   plt.subplot(3,1,1) # (prépare 3 lignes, une colonne, premier tracé)
   image1= ... # (luminance avec les coefficients $p = q = r$)
   plt.imshow(image1,cmap= cm.Greys_r) # (donne l'impression en niveaux de gris)
   plt.title("gris = 0.3333R + 0.3333G+0.3333B")
   plt.tight_layout() # (assure une bonne séparation des images)
   plt.subplot(3,1,2) # (second tracé)
   image2= ... # (luminance avec les coefficients $p = .299, q = .587, r = .114$)
   plt.imshow(image2,cmap= cm.Greys_r)
   plt.title("gris = 0.299R + 0.587G+0.114B")
   plt.tight_layout()
   plt.subplot(3,1,3)
   image3= ... # (luminance avec les coefficients $p = .2125, q = .7154, r = .0721$)
   plt.imshow(image1,cmap= cm.Greys_r)
   plt.title("gris = 0.2125R + 0.7154G+0.0721B")
   plt.tight_layout()

4. Pour s'amuser un peu....
   Pour renverser l'image (effet miroir), on prépare d'abord une boîte pour recevoir la copie :
   shape = monimage.shape
   miroir=np.ndarray(shape)
   miroir=np.fliplr(monimage)
   Devinez comment réfléchir dans un lac (le haut et le bas sont échangés) ?
   Ecrire maintenant vos propres fonctions miroir.
   Former une image obtenue par permutation circulaire des trois couleurs.

5. Détection des contours.
   a) On part d'un tableau des luminances, qu'on va ici noter L. On construit le gradient de luminance pour chaque pixel : on approche la dérivée partielle dans la direction horizontale, donc selon les indices i, par L[i+1,j]-L[i,j]. La norme du gradient se calcule avec la composante selon i et selon j.
   Ensuite, on définit un seuil (à fixer expérimentalement). On modifie les valeurs au-dessus du seuil à 1, celles en dessous, à 0.
   Pratique : On construit le tableau des dérivées partielles approchées selon i : c'est la différence des tableaux :
   L[i+1, j], $0 \leqslant i < n-1, 0 \leqslant j < p$ qui égale le tableau L[i,j], $1 \leqslant i < n, 0 \leqslant j < p$
   et du tableau :
   L[i,j], $0 \leqslant i < n-1, 0 \leqslant j < p$.
   On obtient un tableau DPi, qui a une ligne de moins que le tableau initial. De même, le tableau des DPj aura une colonne de moins. Il faut donc rectifier chaque tableau pour que les deux aient la même taille.
   b) On calcule ensuite les normes des gradients. Pour modifier les normes en fonction du seuil, on peut utiliser un **mask** : si T est un tableau, on peut employer la syntaxe **T[(test(T))]=nouvellevaleur**. Le test est une condition sur chaque valeur du tableau : si elle est satisfaite, la valeur correspondante du tableau est remplacé par nouvellevaleur ; sinon, la valeur est laissée.
   c) Une luminance à 1 correspond à blanc, à 0, correspond à noir. On trace ensuite en niveaux de gris.
6. Filtre de Sobel.
   La détection de contour opérée est très grossière. Une façon fine d'apprécier une dérivée partielle discrète est d'utiliser un filtre, il existe plusieurs formules, une des plus précises est celle-ci :
   DPi = L[i,j]+2*L[i+1, j]+ L[i+2,j]- L[i, j+2]-2*L[i+1, j+2] - L[i+2, j+2].
   Formule analogue pour DPj. On procède ensuite comme ci-dessus.