TD 1 traitement des images : Prise en main

Ce premier TD vous permettra de prendre en main la manipulation des images sous Octave.

1) **Dans la fenêtre de commande, tapez « pkg load image » puis « help imdilate », verifier s'il y a des erreurs. S'il y a des erreurs, une mauvaise version est installée ou n'est pas compatible.**
2) **Créez un dossier TD1 avec un chemin de l'arborescence sans accent, ni espace ni caractère spécial (exemple : C:\TP_vision\TD1)**
3) **Créez un script (ex : exo0.m) puis écrire les commandes et répondre aux questions suivantes.**

---

0) Rappels : Quelques rappels élémentaires.

Tapez les lignes suivantes (une par une!) et comprenez ce que Octave renvoie :

```
>>a=1; b=1.5          >>2*a+b^2          >>A = [1 2 3; 4 5 6; 7 8 9]    >>x = [-1.3 sqrt(3) (1+2+3)*4/5 a]
>>x = 3:9             >>x = 0:0.5:2      >> clear x                     >>x = [1 7 5]          >> x(2)
>>x = [x 6 8 10]      >>y = [x x ; 1:12]  >>figure, plot(x)              >> size(x)      >> length(x)

>> B = rand(4,5)      >> C = B'          >>s = size(B)                  >> th = 0.5 ; Bth = B >th
>> D = B + B          >> E = C + B  (pourquoi y a-t-il une erreur?)     >> B(0,0)       >> B(1,1)
>> F = 3*B            >> G = zeros(size(B))   >> H = ones(size(B))      >> figure, imagesc(B)  >> close all
```

Aide : quand vous ne connaissez pas une fonction, tapez «>> help nom_fonction».

A quoi correspondent les fonctions «clear», «clear all», «clc» et «close all» ?

1) Visualiser une image

Les deux commandes suivantes permettent de charger une image puis de la visualiser.

```
>> im = imread("baboon.bmp") ;          >> figure, imshow(im), title('image originale');
```

Lire les aides des fonctions : « imshow », « imagesc » et « imread ». Comprendre le type (double, uint8…).

2) Convertir en niveaux de gris, lire la valeur des pixels et sauver une image

L'image d'entrée est en couleur, convertissez-la en niveaux de gris avec la commande suivante :

```
>> im1=rgb2gray(im);
```

Visualiser « im1 » et vérifier dans l'espace de travail qu'il s'agit d'un seul plan couleur.
Afin de connaître la valeur d'un pixel de coordonnées *(x, y)*, taper « im1(x,y) ».

On peut aussi visualiser les valeurs des pixels dans une partie de « im1 » contenues entre les valeurs de *x1* et *x2* en abscisses et *y1* et *y2* en ordonnées avec la commande (pour visualiser, ne pas écrire les « ; ») :

```
>>im2 = im(x1:x2, y1:y2)
```

Repérer les pixels sombres et brillants, comparez-les aux valeurs affichées.
Afficher im2, vérifier à quel endroit im2 correspond sur im1. Que fait la commande :

```
>>im3 = im(x1:end, y1:end-3)
```

? Enfin, effectuer la même manipulation sur l'image couleur.

Que fait la commande « im3 = im2double(im2) » ?
Repérer les pixels sombres et brillants, comparez-les aux valeurs affichées.

Pour sauvegarder l'image im1 sous différents formats, puis les visualiser en dehors d'Octave ; taper:

```
>>imwrite(im1, 'baboon_ng.jpg');   >>imwrite(im1, 'baboon_ng.tif');   >>imwrite(im1,'baboon_ng.png');
```

3) Normalisation et inverse à passer en fonction

Afin d'effectuer certains traitements sur les images, il est nécessaire de normaliser l'image, c'est à dire de ramener les valeurs des pixels entre 0 et 1. Vous avez constaté que l'image du babouin « im1 » possède des valeurs entre 0 et 255, normalisez cette image dans l'image « im2 » (il est conseillé d'utiliser des valeurs en format *double* avant de normaliser i.e. la commande « double(im1) »). Pour la visualiser, la commande est :

```
>> figure, imagesc(im2), colormap(gray);title('Image normalisee');
```

Comparer les valeurs des pixels :« im1(x,y) » et « im2(x,y) ».

Pour les autres TDs, nous aurons besoin de deux fonctions «*f_normalisation* » et «*f_inverse* » qui prennent en entrée **une image *I*** et qui renvoient une image.

a) Vérifier comment calculer le max et le min d'une image (une seule valeur doit être renvoyée).
b) Créer les deux fonctions suivantes :
f_normalisation : (On appellera la fonction par : >> In = f_normalisation (double(I));)
Ramener les valeurs des pixels dans [0, 1] : *(im – min(im)) / (max(im)-min(im))*.
f_inverse : On inverse les niveaux de gris de l'image, inverser les valeurs des pixels avec la formule : *max(im) – im*. Tester dans un script sur l'image de la lune puis l'adapter pour une image en couleurs (babouin).

Important : Au début de chaque script et fonction, il faut le nom et prénom ainsi que la date de création.

4) Codage de l'image, Normalisation et Courbes gamma

a) Chargez l'image « lune.png » dans la variable L. Vérifiez si L est en niveaux de gris, sinon la convertir en niveaux de gris. Vérifier dans l'espace de travail comment est codée l'image (8 bits ? 16 bits?).
Créer une image « L2 » équivalente à L mais codée sur 16 bits avec la commande : « >>L2 = uint16(L); ».
Vérifier le format dans l'espace de travail ainsi que les maximums de L et L2 (i.e., « >>m1= max(max(L)) »).
b) Créer LL telle que « >> LL = 5*L » et L2 telle que « >> LL2 = 5*L2 », visualiser LL et LL2 avec imagesc, comprendre pourquoi cette différence.
c) Normalisez L dans l'image appelée Ln puis appliquez les courbes gamma telles que Lgamma = Ln^gamma avec gamma = b ou gamma = 1/b (où b>1). **Attention**, on travaille pixel par pixel (quelle est la commande automatique ?).
Quelles différences faites-vous entre gamma = b et gamma = 1/b ?
d) Trouver une valeur de gamma qui permet de mieux mettre en valeur des cratères sur la surface de la lune et une autre valeur permettant de faire apparaître le bord gauche du satellite naturel de la terre.

5) Seuillage

Trouver une méthode vectorielle (i.e., automatique, sans boucle) permettant de garder les pixels d'une image normalisée dont la valeur est supérieure à 0.5. Enfin, trouver un seuil permettant d'extraire les cratères sur l'image Lgamma.

6) Image couleur

Appliquer correctement les courbes gamma sur une image couleur (image du babouin « baboon.bmp ».).

7) Ligne sur une image et signal

Charger l'image « bureau256.png ». Pour afficher une ligne verte, il faut créer une image couleur :
>>Ic = I; Ic(: , : , 2) = I; Ic(: , : , 3) = I;
Pour afficher le signal, utilisez la fonction « plot », et bloquer la taille du signal à la largeur de la ligne (« xlim »).
a) Afficher sur l'image en vert la ligne 85 et sur une autre figure le signal puis en utilisant «subplot».
b) Afficher une ligne sur une image couleur (babouin), visualisez les signaux pour une ligne sur les trois canaux.
c) Faire de même sur l'image en niveau de gris et tout afficher sur la même figure.
d) Servez-vous de cette technique pour la question 4) afin d'afficher les cratères en vert sur l'image de lune.
e) Afficher le signal d'une ligne avant et après avoir appliqué les courbes gamma (cf. plot, option « hold on »).

8) Comprendre un code.

On considère l'image « lune.png » dans la variable *I* et *Igamma* avec gamma=2. Comprendre le code :

```
>> I = double(rgb2gray(imread('lune.png')));
>> I = I(1:end-5, :);  I = f_normalisation(I);  I2 = I.^(2); [L C z]=size(I);
>> Z1 = [I  ones(L, 20)  f_inverse(I)];
>> Z2 = [I2  ones(L, 20)  f_normalisation(I - I2)];
>> [L C z]=size(Z1);  Z = [Z1 ; ones(20, C) ; Z2];
>> figure, imshow(uint8(Z*255)), colormap(gray), title('Comprendre ce code')
```

9) Rehaussement de contraste utilisant un filtre linéaire

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

a) Créer la matrice L1 du masque laplacien suivant :
>> *L1* = [0 -1 0 ; -1 4 -1 ; 0 -1 0] ; (ou >> *L2* = [-1 -1 -1 ; -1 8 -1 ; -1 -1 -1];).
Convoluer une image im1 NG avec L1 :

```
>>imLaplacien = conv2(double(im1), double(L1),'same');
>>figure,  imagesc(f_normalisation(imLaplacien)), colormap(gray),title('image convoluee avec Laplacien');
```

b) Rehausser les contours de im1 en faisant la différence entre les images *im1* et *imLaplacien* multipliée par un petit coefficient $0 < a < 1$: >>*imrehauss = im1 - a * imLaplacien;* Quelles valeurs pouvez-vous prendre pour *a* ?

c) Appliquez le rehaussement de contours sur l'image «lena.bmp» en couleur. Pour cela, extrayez les 3 canaux de couleurs (R,V,B) et appliquez-leurs le même traitement à chacun. Enfin, reconstituez l'image couleur *IC* :

```
>>IC(:,:,1) = imrehaussR; IC(:,:,2) = imrehaussG; IC(:,:,3) = imrehaussB;
>>figure,  imshow(uint8(IC)), title('Rehaussement Couleur');
```

10) Rehaussement de contraste utilisant un filtre non-linéaire

Charger l'image « bureau256.png », appelée *I*.
On considère une fenêtre carrée W de taille mxm centrée sur un pixel *P* de l'image *I* en niveaux de gris (avec m impair), comme illustré à droite (on testera plusieurs tailles de W).
Soit M la moyenne dans la fenêtre W (la valeur de M est différente pour chaque pixel puisque W se déplace). Soit J l'image de sortie, elle est calculée par :
Si M < P alors J(i,j) = max(W).
Sinon J(i,j) = min(W).