Programme et objectifs des TPs de Traitement d'images - M1 Informatique

L'objectif de ces TP est de vous faire implémenter sous Matlab/Octave différents **outils de pré-traitement et d'analyse d'images** sur des images fixes. Le **TP2** sera **à rendre** (par mail ou sur Plubel) **au plus tard le 19 février 2023 (minuit)**. Le **TP5** sera noté en séance.

Programme des TP Matlab :

TP 1 : Histogramme, binarisation et LUT

TP 2 : Étirement, égalisation et spécification d'histogrammes ***(à rendre)***

TP 3 : Filtrage de bruits

TP 4 : Détection de contours

TP 5 : Segmentation d'images ***(noté)***

Pour tester les traitements demandés, plusieurs **images fixes** sont à disposition sur **Plubel** (images de tailles et de formats différents, en couleur ou NG, éventuellement bruitées ou à **faible contraste**).

Pour chaque exercice, il est demandé :
- de créer une **fonction Matlab** permettant de réaliser le traitement demandé
- et de **vérifier** son bon fonctionnement en testant le résultat sur **plusieurs des images** fournies.

I) Indications :

1) Voici un exemple de fonction Matlab pour réaliser l'inversion vidéo d'une image (négatif) :

```
function res = inversion (I)  % fonction qui prend en paramètre une image I
                              % et retourne une image res (négatif)
[m, n, can] = size(I);  % m=nb lignes, n=nb colonnes, can=nb canaux
res = zeros (m, n);     % image résultante (de même taille que I) :
                         % initialisée à 0 partout
if(can > 1)
    I = rgb2gray(I);    % si l'image est en couleur, la transformer en NG
end
res = 255 - I;          % l'inversion
figure

%divise la fenêtre en 1 ligne et 2 colonnes : pour afficher 2 images l'une
%à côté de l'autre
subplot(1, 2, 1)     %sélectionne le premier cadran de la fenêtre
colormap(gray(256))
imagesc(I);          %affichage de l'image originale
subplot(1, 2, 2)     %sélectionne le deuxième cadran de la fenêtre
colormap(gray(256))
imagesc(res);        %affichage du résultat
title(strcat(['Image de taille ', num2str(m), 'x', num2str(n)]));
end
```

2) Utilisation de la **fonction inversion** définie précédemment (en supposant qu'on dispose de l'image « pout.tif », qui se trouve dans le répertoire '**faible_contraste/**') :

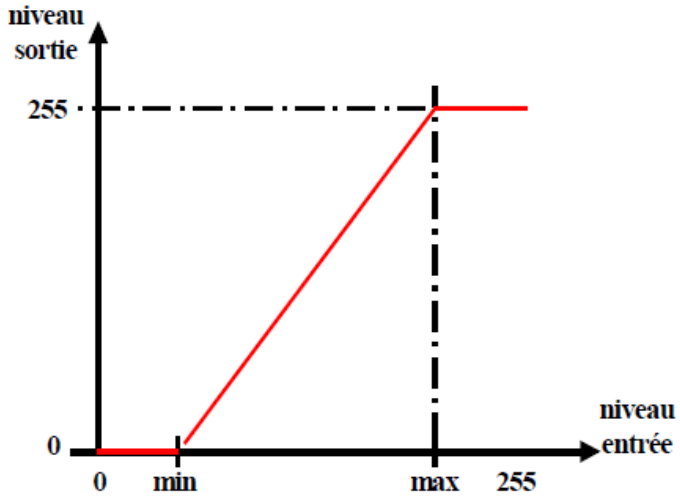
```
> Im = imread('pout.tif');
> inversion(Im);
```

Commentaires : Im est une matrice bidimensionnelle dans le cas d'une image en niveaux de gris. Le type de données de Im est **uint8** (unsigned integer sur 8 bits).

3) Affichage d'une image :

L'affichage d'une image peut se faire avec les fonctions : **image, imagesc et imshow**.

- **imshow** affiche l'image contenue dans le fichier image ou la matrice correspondante. imshow appelle imread pour lire l'image depuis le fichier, mais les données de l'image ne seront pas stockées dans le workspace.
- **image** affiche la matrice en argument comme une image (n'accepte pas de fichier image en paramètre). L'affichage repose sur les valeurs de la matrice qui sont :
  - associées aux index de **la carte des couleurs activée** (colormap)
  - ou directement interprétées comme **valeurs RGB** (true color)
- **imagesc** affiche la matrice en argument comme une image. Contrairement à **image**, elle effectue une remise à l'échelle des valeurs de la matrice avant l'affichage, de manière à utiliser pleinement la carte des couleurs (colormap). Elle utilise donc la LUT suivante (correspondant à un étirement d'histogramme) :



Comparer visuellement les trois fonctions précédentes avec une image, en se servant éventuellement de la commande **figure** avant chacune des commandes **image, imagesc, et imshow**. Ainsi Matlab ouvrira une nouvelle fenêtre d'affichage pour chaque nouvelle image.

Par défaut la **carte de couleur** (colormap) utilisée par Matlab est la carte appelée **'parula'**, qui utilise un dégradé du bleu au jaune. Tapez *'help colormap' ou allez voir directement dans l'aide Matlab,* pour en savoir plus sur les cartes de couleurs disponibles.

TP1 : Histogramme, binarisation et LUT

Nous allons commencer par créer **3 fonctions Matlab : histogramme, binarisation_manuelle** et **binarisation_automatique**.

I) Histogramme

Créer la fonction Matlab **histogramme** qui affiche l'histogramme d'amplitude d'une image en niveau de gris (NG) dans une fenêtre. Si l'image d'entrée est en couleurs, la transformer en une image en NG, grâce à la focntion Matlab « `rgb2gray` », dont l'utilisation est précisée à la page précédente.
On appelle **H** l'histogramme d'amplitude, défini de telle sorte que **H[n]** donne le nombre de pixels de l'image dont le niveau de gris vaut **n**. Comparez votre histogramme avec celui fourni par les fonctions *« imhist » ou « histogram »* de Matlab.

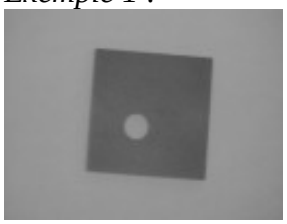
Attention : les indices d'une matrice ou d'un vecteur commencent à 1 !

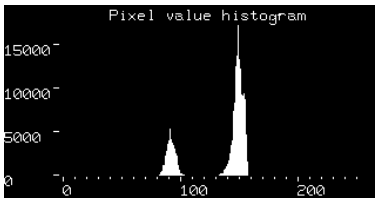
II) Binarisation

Créer les fonctions Matlab **binarisation_man** et **binarisation_auto** pour afficher dans une fenêtre Matlab à gauche l'image d'entrée en NG et à droite le résultat de la binarisation.

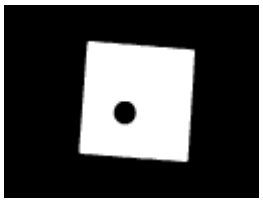
1) Le **seuil** aura été choisi en ayant **visualisé l'histogramme** au préalable (on pourra demander à l'utilisateur de le taper au clavier : fonction « `input` » de Matlab).

Exemple 1 :

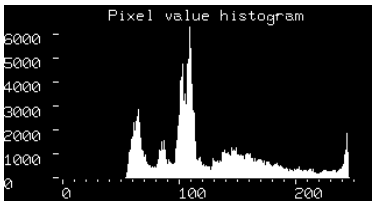


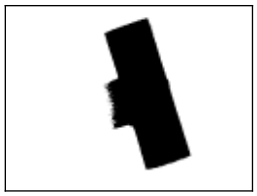


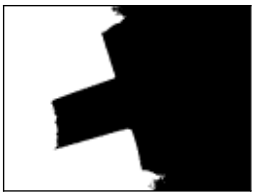
Résultat :



Exemple 2 :







Indication : Matlab est un logiciel de **calcul matriciel**, donc autant que possible, utilisez les calculs matriciels qui sont bien plus efficaces que les boucles « for » imbriquées (gain de temps de calcul).

Exemple :

```
Im = imread('ex.bmp');
s = 128;
n = size(Im,1);              % Nombre de lignes de l'image
m = size(Im,2);              % Nombre de colonnes de l'image
for i=1:n                    % Balayage sur les lignes de l'image
      for j=1:m              % Balayage sur les colonnes de l'image
            if Im(i,j) <= s
                  ImB(i,j) = 0;
            else
                  ImB(i,j) = 1;
            end
      end
end
```

Peut-être remplacé par :

```
Im = imread('ex.bmp');
s = 128;
ImB = Im > s;
```

En entrant la commande « ImB = Im > s », Matlab créé un masque logique « ImB » de même taille que Im. Un pixel (i, j) du masque est à 1 quand la condition « Im(i, j) > s » est vraie et à 0 sinon.

2) Le **seuil** sera déterminé **automatiquement** par la **fonction Matlab** *« graythresh »* (méthode d'**Otsu**) : regarder dans l'aide de Matlab comment elle fonctionne.
Remarque : on pourra afficher dans la fenêtre de traitement la valeur du seuil automatique, avec le code suivant :

```
title(strcat(['Seuil=', num2str(seuil)]));
```

Visualisez maintenant l'histogramme de l'image binarisée, que remarquez-vous ?

III) LUT

Sous Matlab, on peut créer ses **propres LUT** et les appliquer aux images, à l'aide de la fonction **colormap**. Les valeurs de la LUT doivent cependant être normalisées sur l'intervalle [0, 1].

Prenons, par exemple, le cas simple de la matrice **M** définie ci-dessous (de taille 3x3). Pour créer la LUT '**map4C**' affichant les valeurs :

- « 1 » et « 4 » en noir, « 2 » en blanc,
- « 3 » en rouge et « 5 » en vert,

on construit une matrice à 5 lignes et 3 colonnes (chaque ligne précisant un triplet R, G et B).
Pour appliquer cette LUT à l'image affichée, on tape la commande **colormap(map4C)** après la commande **image** :

```
M = [4 5 5 ; 1 2 2 ; 3 3 0]; % ATTENTION, Matlab considère que ce sont des 'double'
% Définition de la LUT = palette de couleurs indexées
% val = 1    2    3    4    5  → correspondances entre les valeurs de M et de la colormap
r = [ 0 ; 1 ; 1 ; 0 ; 0];  % vecteur colonne
g = [ 0 ; 1 ; 0 ; 0 ; 1];  % vecteur colonne
b = [ 0 ; 1 ; 0 ; 0 ; 0];  % vecteur colonne

map4C =[r g b];      %création de la LUT qui convertit les pixels à 1 et 4 en noir (0,0,0),
                     % à 2 en blanc (1,1,1), à 3 en rouge (1,0,0) et à 5 en vert (0,1,0).
image(M);
colormap(map4C)   % indique la palette de couleurs indexées considérée
                  % map4C : intensités entre 0 et 1 pour les canaux R, G et B
```

ATTENTION : **colormap** ne se comporte pas de la même façon avec des valeurs entières ! Observer la différence avec le code suivant :

```
M2 = uint8(M);  % transformation en unsigned int sur 8 bits
figure; image(M2); colormap(map4C)
% val = 0    1    2    3   4 et 5   → correspondances entre les valeurs de M2 et de la colormap
r = [ 0 ; 1 ; 1 ; 0 ; 0];  % vecteur colonne
g = [ 0 ; 1 ; 0 ; 0 ; 1];  % vecteur colonne
b = [ 0 ; 1 ; 0 ; 0 ; 0];  % vecteur colonne
```

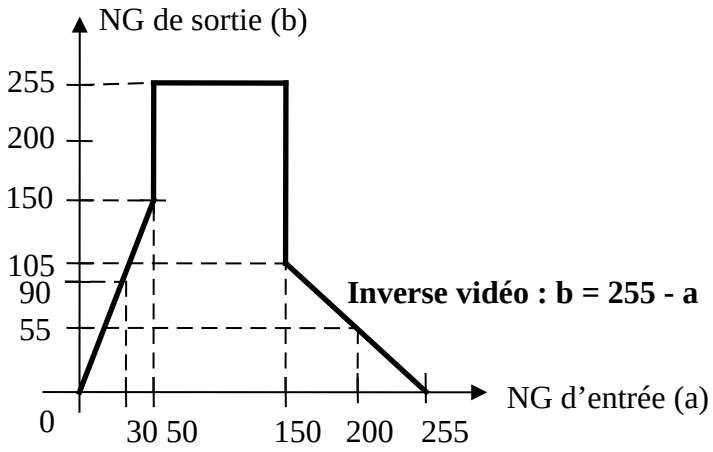
1) Appliquer la transformation (LUT) précisée ci-dessous, pour transformer l'image **« rectangles »** (image en NG) en une image couleur (RGB) :

- pixels de NG = 29 → en Rouge
- pixels de NG = 76 → en Noir
- pixels de NG = 91 → en Jaune (R+G)
- pixels de NG = 150 → en Bleu
- pixels de NG = 226 → en Vert
- pixels du fond (NG = 255) → en Blanc

Utiliser la LUT créée comme paramètre de la fonction **colormap** de Matlab pour **afficher** l'image en couleurs (avec la fonction **image de Matlab).**

Figure 1 : image « rectangles »
Les nb indiquent le NG utilisé pour chaque rectangle de l'image (fond blanc = 255)

2) Créer une fonction qui réalise la transformation suivante à partir d'une image en niveaux de gris (NG) :



Afficher **en NG** l'image résultante avec la fonction **imagesc** de Matlab (en utilisant l'image en NG « lena.png », dans le répertoire **NG/**).