Université Sidi Mohamed Ben Abdellah — Année universitaire : 2018/2019
Faculté des sciences Dhar El Mahraz — **Master WISD**

TP 6 : Images couleur

I. Représentation RGB

1) Images synthétiques

	R	G	B
Couleur 1	255	0	0
Couleur 2	0	255	0
Couleur 3	0	0	255
Couleur 4	255	255	0
Couleur 5	0	255	255
Couleur 6	255	0	255
Couleur 7	255	255	255
Couleur 8	0	0	0
Couleur 9	30	30	30
Couleur 10	200	200	200
Couleur 11	255	165	0
Couleur 12	139	69	19

a) Créer 12 images couleur 8-bits X1, X2, ..., X12 de taille 200 × 300, unies, et dont la couleur est définie par le code RGB donné dans le tableau ci-dessus. Les afficher, et noter pour chacune d'entre elle à quelle couleur elle correspond.

b) A partir de ces observations, créer une image couleur 8-bits Y1 de taille 200 × 300 représentant un carré jaune sur fond rouge (le carré sera situé vers le milieu de l'image). L'afficher.

2) Images naturelles

a) Ouvrir l'image *autumn.tif* et la stocker dans une matrice X. Quelle est sa taille ? Sur combien de bits est-elle codée ? Afficher l'image.

b) Afficher d'abord la composante rouge de l'image, puis la composante verte, et enfin la composante bleue. Commenter.

c) Comment faire pour afficher la composante rouge non pas en nuances de gris, mais en rouge ? Afficher la composante rouge en rouge.

d) Quel est le code RGB de la couleur du ciel ? Pour le savoir il suffit de regarder la valeur stockée par exemple dans le pixel $(m, n) = (1, 1)$. Mettre ce code dans un vecteur (de dimension 3) qu'on appellera ciel.

e) Nous allons modifier ce ciel, qui est de couleur grise, pour lui donner une belle couleur bleue ! Pour cela, nous allons sélectionner tous les pixels de l'image ayant des couleurs proches de celles stockées dans le vecteur ciel, et modifier leur couleur.

   i. Ecrire un script qui permettra de sélectionner tous les pixels ayant une couleur située à une distance euclidienne avec la couleur ciel inférieure à un seuil T. Cela revient à calculer des distances euclidiennes entre des vecteurs de dimension 3. Nous allons créer une image binaire booléenne *Y_bin* (noir et blanc) : les zones blanches correspondent aux pixels sélectionnés, et les zones noires aux zones hors sélection. Afficher *Y_bin* pour différentes valeurs de T jusqu'à ce que tous les pixels correspondant au ciel soient sélectionnés.

   Nb : la distance euclidienne est définie dans l'espace 3D comme suit :

   $$d = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2 + (Z_1 - Z_2)^2}$$

   ii. Stocker les composantes rouge, verte et bleue de X respectivement dans des matrices R, G et B.

   iii. Mettre les pixels définis par *Y_bin* dans l'image R à la valeur r = 173. Faire la même chose pour G et B avec respectivement les valeurs g = 216 et b = 230.

f) Reconstruire une image Y formée des composantes RGB stockées dans R, G et B. Afficher Y : le ciel est maintenant bleu !

II. *Chrominance, luminance et espace couleur*

1) *Rappels de cours*

On s'intéresse ici à l'espace de représentation *YCbCr*, qui est utilisé pour la représentation JPEG des images couleurs. Au lieu de représenter une image 8-bits avec une composante rouge, une composante verte et une composante bleue, les trois composantes utilisées sont les suivantes :

- La composante Y, correspond à la luminance, il s'agit d'une conversion de l'image couleur en une image en nuances de gris. On peut donc traiter cette composante exactement comme on le faisait avec les images en nuances de gris (filtrage, représentation de Fourier, etc...). Si l'on avait dans l'image originelle trois composantes R, G et B, on a :

$$Y \approx 0.3R + 0.6tt + 0.1B$$

- La composante Cb correspond à la chrominance bleue qui est calculée comme étant la composante bleue moins Y (multiplié par une constante) :

$$Cb = \lambda b(B - Y)$$

où $\lambda b \approx 0.6$ est une constante

- La composante Cr correspond à la chrominance rouge qui est calculée comme étant la composante rouge moins Y (multiplié par une constante) :

$$Cr = \lambda r(R - Y)$$

où $\lambda r \approx 0.7$ est une constante

Comme on veut que Cb et Cr aient des valeurs entre 0 et 255 (pour pouvoir les coder de la même façon que la composante Y), on rajoute en général un terme constant égal à 128 pour s'assurer que ce soit le cas. On obtient finalement le triplet (Y, Cb, Cr) à partir du triplet (R, G, B) par le système d'équations suivant :

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & 0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \times \begin{bmatrix} R \\ R \\ B \end{bmatrix}$$

2) *Etude sous MATLAB*

a) Ouvrir l'image *mandrill.tif*, la stocker dans une matrice *X_rgb* et la renormaliser. Afficher l'image.

b) Convertir l'image (en RGB) en *YCbCr* grâce à la fonction *my_rgb2ycbcr.m* fournie et stocker le résultat dans une matrice *X_ycbcr*. Afficher sur la même figure la composante Y, la composante Cb et Cr. Commenter.

c) Stocker la composante Y dans une matrice Y, la composante Cb dans une matrice Cb, et la composante Cr dans une matrice Cr. Nous allons tenter l'expérience suivante : nous allons dégrader de façon volontaire les composantes de chrominance, et garder telle quelle la composante de luminance, et voir quelles sont les conséquences sur l'image reconstituée.

i. Pour cela, commencer par remettre les valeurs de Cb et Cr entre 0 et 255.

ii. Puis, quantifier les matrices Cb et Cr sur 2 bits.

Etant donné une image quantifiée sur `b1` bits, on peut utiliser la commande suivante pour la re-quantifier sur `b2` bits

```
X = floor(X / 2^(b1-b2));   % Quantification de b1 bits vers b2 bits
% floor permet de calculer la partie entiere d'un nombre
```

iii. Enfin, remettre les valeurs de Cb et Cr entre 0 et 1.

iv. Construire ensuite une matrice 3D *Z_ycbcr* ayant Y comme composante Y et Cb et Cr comme composantes Cb et Cr.

v. Reconstruire grâce à la fonction *my_ycbcr2rgb.m* fournie une image RGB *Z_rgb*. Afficher sur la même figure *X_rgb* et *Z_rgb*. Commenter.

vi. Essayer de quantifier Cb et Cr sur 4 bits et observer le résultat. Commenter.