TD 3 UE Vision

Le but de ce TD est de pouvoir détecter certaines couleurs, les échanger, les inverser et de changer d'espace.

I - Manipulation des couleurs

Pour commencer, nous allons nous familiariser avec les images en couleur. Charger l'image « baboon.bmp » que l'on nomera « *im* ». On écrira le résultat dans l'image « *imgf* » de la même taille que « *im* » :
>> s=size(im); imgf = uint8(zeros(s)); % image résultat

a) Afin d'isoler le nez rouge du babouin, construire l'image filtrée où ne sont retenus que les pixels dont l'intensité en rouge est supérieure à un seuil donné. Donner une valeur de la variable seuil qui permet d'obtenir le nez complet tout en éliminant un maximum des autres pixels de l'image (cette valeur doit être déterminée "à la main" par essais successifs). **Attention : le calcul prendra un peu de temps.**
Aide : les valeurs des pixels de couleurs bleue et verte doivent également être inférieures à un second seuil. Il faut aussi vérifier que les valeurs du rouge sont supérieures au bleu ainsi qu'au vert.

b) De la même manière, isoler les pixels bleus de l'image du babouin. Attention, les valeurs des pixels de couleurs bleus sont moins importantes que celles des pixels rouges, il faut donc choisir un seuil inférieur à celui de la question précédente. Afin de s'en rendre compte des valeurs, taper la commande suivante :
>> figure, image(im);
A l'aide de la souris, repérer les coordonnées d'un pixel rouge puis d'un bleu, comparer les valeurs du canal bleu et rouge en affichant les valeurs des pixels repérés.

c) Créer une image où, pour chaque pixel dont les valeurs du bleu sont supérieures à un seuil, et **aussi supérieures aux valeurs du vert et du rouge,** on échangera le canal rouge avec celui du bleu (toujours sur l'image du babouin).

d) A la place du bleu et du rouge, faire apparaître du vert dans l'image du babouin. Pour cela, si le bleu est supérieur à un seuil et aux deux autres canaux, échanger les canaux bleu et vert. Si le rouge est supérieur à un seuil et aux deux autres canaux, intervertir les canaux rouge et vert.

II – Création de fonctions pour des images en niveaux de gris

Dans la dernière partie de ce TD, nous aurons besoin de deux fonctions qui prennent en entrée **une image *I* en niveaux de gris** et qui renvoient une image.

1) Vérifier comment calculer le **max et le min** d'une image (**attention** une seule valeur doit être renvoyée).
2) Si elles n'ont pas encore été écrite au TD1, 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*. (Attention au calcul du *max !*). Tester sur l'image lune puis les trois canaux de l'image babouin.

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

III – Espace YUV

L'espace YUV représente les couleurs à l'aide d'une composante Y et deux composantes de chrominance (*U, V*) correspondantes aux composantes bleue et rouge dans des coordonnées réduites : $Cg = Y – U – V$.
a) Charger l'image « peppers.png ». Appelez le canal rouge R, le vert G et le bleu B.
b) En utilisant la fonction max, chercher le maximum de l'ensemble des 3 canaux, appelé M.
c) Normaliser chacun de trois canaux par rapport à M. (Vérifier le résultat.)

Y représente la composante de luminance, soit une moyenne pondérée par la sensibilité humaine :

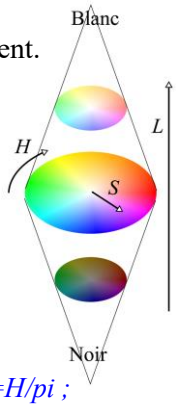
$$Y = 0.3 * R + 0.59 * G + 0.11 * B$$

Les composantes de chrominance *U* et *V* représentent respectivement le contraste bleu/jaune et rouge/cyan :

$$U = 0.49 * (B – Y) \quad \text{et} \quad V = 0.88 * (R – Y)$$

d) Visualiser les images *Y, U, V* et *Cg* puis construire l'image en couleurs YUV suivante :
>> YUV(:, :, 1) = Y ; YUV(:, :, 2) = U ; YUV(:, :, 3) = V ; imshow(YUV), title('YUV')
e) Comparer le résultat avec ceux obtenus à l'aide de la fonction « rgb2gray ».
f) Regarder aussi le résultat sur l'image « baboon.png », « peppers.png » et « peppers_compress.jpg ».

IV – Espace HSV



a) Charger l'image « peppers.png ». On garde les mêmes notations pour les couleurs que précédemment.
L'espace HSV (Hue = teinte, Saturation, Valeur/luminance) caractérise les couleurs plus intuitive :
b) La luminance *L* caractérise l'intensité totale d'un point lumineux coloré : *L = (R + G + B) / 3*.
c) La saturation *S* varie entre 0 et 1. Plus la couleur est pure, plus la saturation est élevée et inversement, plus la saturation est faible, plus la couleur est délavée (0 = gris, 1 = couleur pure).

$$S = 1 - 3 * min(R, G, B) / L.$$

d) L'ensemble des couleurs représentables dans l'espace HSV est délimité par des cônes face à face de même base. Ainsi, on calcule *H* de la façon suivante :$H = acos\frac{(R-G)+(R-B)}{2\cdot\sqrt{(R-G)^2+(R-B)(G-B)}}$

Attention, si $B > G$, alors $H = \pi - H$.

e) Visualiser L, S, H puis construire l'image couleur : *>>HSV(:,:, 1) = L; HSV(:,:, 2) = S; HSV(:,:, 3) =H/pi ;*
f) Visualiser *HSV* et faire de même avec « baboon.png », puis « peppers_compress.jpg », quel constat ?

V – Détection de peau dans les images en couleur

Dans cette partie, nous allons, comparer 3 méthodes de détection de peau dans les images en couleurs. Les deux premières méthodes seront des comparaisons entre la valeur d'un pixel de peau avec les valeurs de tous les autres pixels de l'image. La troisième méthode proposée est issue d'un apprentissage où des coefficients sont estimés à partir d'images comportant des visages, des parties de corps ou des corps entiers.

Charger dans Octave/Matlab **une image couleur avec un visage**, une image assez simple (une image de vous ?) nommée « im ». La visualiser avec la commande suivante : *>> figure, image(im);*

1) Distance RVB.

A l'aide de la souris, repérer les coordonnées d'un pixel appartenant à de la peau, de coordonnées L (ligne) et C (colonne). Vérifier qu'il s'agit du bon pixel en affichant du bon pixel :
>> figure, imshow(I), hold on, plot(L, C, 'g+') ;
Dans un script, pour le pixel considéré, extraire les valeurs du canal rouge, vert et bleu notée :(R_0, V_0, B_0).
Déclarer l'image de peau *imskin* (image de double) de la taille de *im* :
>> s=size(im); imskin = double(zeros(s(1),s(2))); % déclaration pour chaque section suivante
Maintenant, calculer la distance RVB :$imskin = \sqrt{(R-R_0)^2 + (V-V_0)^2 + (B-B_0)^2}$,où *(R, V , B)* représentent les valeurs en rouge, vert et bleu pour chaque pixel de *im*.
Enfin, normaliser *imskin*, inverser (fonctions que vous avez créées précédemment) et seuiller le résultat afin d'extraire au mieux un visage (visualiser avec « imagesc »). Pour un résultat de meilleur qualité, lisser I avec une gaussienne en utilisant la fonction « *fspecial* ("gaussian") » (regarder l'aide).

2) Distance normalisée

Choisir le même pixel que la partie précédente, avec donc les mêmes valeurs(R_0, V_0, B_0).
Travaillez dans un nouvel espace (Lab) à l'aide de $a_0 = \frac{R_0}{(R_0+V_0+B_0)}$ et $b_0 = \frac{V_0}{(R_0+V_0+B_0)}$.
Pour chaque pixel, calculer $a = \frac{R}{(R+V+B)}$ et $b = \frac{V}{(R+V+B)}$puis la distance dans ce nouvel espace :
$imskin = \sqrt{(a-a_0)^2 + (b-b_0)^2}$.
Enfin, normaliser *imskin*, inverser et seuiller le résultat afin d'extraire au mieux un visage. Vérifier si« *fspecial* ("gaussian") » améliore les résultats.

3) Estimation de la peau par apprentissage [1]

Dans cette section, nous travaillons sur toute l'image. Pour chaque pixel de *im*, calculer les valeurs *(a, b)* de la section 2) précédente. Pour chaque pixel, on a :

$$imskin = \begin{pmatrix} a-0.46 & b-0.31 \end{pmatrix} \begin{pmatrix} 0.003517 & -0.00085 \\ -0.00085 & 0.000613 \end{pmatrix} \begin{pmatrix} a-0.46 \\ b-0.31 \end{pmatrix}$$

Enfin, normaliser *imskin*, inverser et seuiller le résultat afin d'extraire au mieux un visage.
Vérifier si« *fspecial* ("gaussian") » améliore les résultats.

4) Insérer le visage qui a été détecté avec la couleur originale dans une autre image couleur.

On vérifiera la taille des deux images avant insertion (commande « *size* ») et on peut boucher les trous (*>> help imfill*).

Références : [1] Sidibe, D., Montesinos, P., & Janaqi, S. (2006). **A simple and efficient eye detection method in color images.** In *International Conference Image and Vision Computing New Zealand 2006* (pp. 385-390).