Traitement des images numériques

TP 5 : Images couleur

Université Paris 13, Institut Galilée
Master Ingénierie et Innovations en Images et Réseaux - 1ère année

2017-2018

Consignes

- Récupérer le fichier `TP5.zip` sur le site

  `http://www.laurentoudre.fr/tin.html`

- Ouvrir MATLAB et créer un répertoire de travail. Dézipper le fichier `TP5.zip` dans ce répertoire.
- A la fin de la séance, récupérer les scripts que vous avez écrits et les envoyer par e-mail au chargé de TP ainsi qu'à vous même afin de les conserver pour la prochaine séance.

Rendu

- Trois fichiers : `TP5Partie1.m`, `TP5Partie2.m` et `TP5Partie3.m` Chaque fichier doit contenir votre nom, votre prénom et la date.
- Compte-rendu succinct à rendre à la fin de la séance, contenant les observations, commentaires et réponses aux questions. Le compte rendu doit contenir votre nom et votre prénom.

Plan de l'étude

1 Représentation RGB

1.1 Rappels de cours

La représentation la plus courante pour les images couleur est la représentation RGB (R : red, G : green, B : blue). Une image couleur est représentée non pas par une matrice, mais par trois matrices, représentant respectivement l'intensité en rouge, vert et bleu. Les écrans d'ordinateurs reconstituent les différentes couleurs par synthèse additive à partir de ces trois couleurs primaires. Si l'image est quantifiée sur 8 bits, alors chacun des pixels des trois matrices peut prendre des valeurs comprises entre 0 et 255.

Sous MATLAB, une image couleur est représentée par une matrice en 3D, n'ayant plus uniquement deux indices m et n, mais trois indices m, n et c (pour couleur). Une image couleur g de dimension $M \times N$ s'écrira donc :

$$g(:,:,1) = \begin{bmatrix} g(1,1,1) & \cdots & g(1,N,1) \\ \vdots & \ddots & \vdots \\ g(M,1,1) & \cdots & g(M,N,1) \end{bmatrix} \qquad g(:,:,2) = \begin{bmatrix} g(1,1,2) & \cdots & g(1,N,2) \\ \vdots & \ddots & \vdots \\ g(M,1,2) & \cdots & g(M,N,2) \end{bmatrix}$$

$$g(:,:,3) = \begin{bmatrix} g(1,1,3) & \cdots & g(1,N,3) \\ \vdots & \ddots & \vdots \\ g(M,1,3) & \cdots & g(M,N,3) \end{bmatrix}$$

La composante $g(:,:,1)$ correspond à la couleur rouge, $g(:,:,2)$ à la couleur verte et $g(:,:,3)$ à la couleur bleue. Les commandes MATLAB déjà utilisées pourront être adaptées facilement de la façon suivante :

- Pour créer une matrice 3D sous MATLAB, on pourra utiliser les commandes suivantes :

```
X = zeros(M,N,3);   % Cree une matrice de taille M x N x 3 ne contenant que des 0
% M : nombre de lignes
% N : nombre de colonnes
% 3 : nombre de composantes couleur
X = ones(M,N,3);    % Cree une matrice de taille M x N x 3 ne contenant que des 1
```

- Pour afficher une image couleur sous MATLAB, on utilisera la commande suivante. Attention, on fera attention de bien renormaliser l'image avant de l'afficher (comme les images en niveau de gris).

```
figure      % Cree une nouvelle figure
imshow(X/255)   % Affiche l'image X 8-bits (renormalisee)
```

- Pour modifier les valeurs d'une matrice 3D sous MATLAB, on peut utiliser selon les cas :

```
X(3,10,:)=0;     % Le pixel appartenant a la ligne 3 et a la colonne 10
                 % est mis a 0 sur toutes les composantes couleur
X(4,:,2)=0;      % Tous les pixels appartenant a la ligne 4
                 % sont mis a 0 uniquement sur la composante verte
X(:,3:8,:)=0;    % Tous les pixels appartenant aux colonnes de 3 a 8
                 % sont mis a 0 sur toutes les composantes couleur
X(5:9,6:10,1)=0;% Tous les pixels appartenant a la fois aux lignes de 5 a 9 et
                 % aux colonnes de 6 a 10 sont mis a 0 sur la composante rouge
```

- Pour convertir une image couleur en image en niveaux de gris, on peut utiliser la commande `rgb2gray` :

```
% X : image couleur
Y = rgb2gray(X);
% Y : image en nuances de gris
```

1.2 Etude sous MATLAB

1.2.1 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

1. Créer sous MATLAB un script vide nommé `TP5Partie1.m`
2. 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.
3. A partir de ces observations, et de ce qui a été fait au TP1, 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.
4. Créer une image couleur 8-bits `Y1` de taille 200×300 représentant une croix orange sur fond bleu (la croix sera située vers le milieu de l'image). L'afficher.

1.2.2 Images naturelles

1. Créer sous MATLAB un script vide nommé `TP5Partie2.m`
2. 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.
3. Afficher d'abord la composante rouge de l'image, puis la composante verte, et enfin la composante bleue. Commenter.
4. Comment faire pour afficher la composante rouge non pas en nuances de gris, mais en rouge ? Afficher la composante rouge en rouge.
5. 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`.
6. 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.
   (a) Nous allons 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.

Pour sélectionner tous les pixels dont la couleur (vecteur de dimension 3) est située à une distance euclidienne inférieure à T, par rapport à une couleur de référence $[r, g, b]$, on peut utiliser la commande :

```
% X : Image couleur sur 8 bits
% [r g b] : couleur de reference
% T : seuil
Y_bin = (sqrt(abs(X(:,:,1)- r).^2 +abs(X(:,:,2)-g).^2 + abs(X(:,:,3)-b).^2) < T);
```

(b) Stocker les composantes rouge, verte et bleue de `X` respectivement dans des matrices `R`, `G` et `B`.

(c) 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 $b = 230$.

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

2 Chrominance, luminance et espace couleur

2.1 Rappels de cours

Il arrive que la représentation RGB ne soit pas la plus pertinente pour représenter une image couleur. C'est notamment le cas du filtrage, où le filtrage individuel de chaque composante couleur ne donne pas nécessairement un bon rendu visuel sur l'image finale. On définit donc d'autres espaces couleur qui reviennent à projeter les composantes RGB sur trois autres composantes (ne correspondant plus nécessairement à des couleurs primaires). Il existe différents espaces couleur dans lesquels cette transformation peut se faire selon les applications (image, vidéo, vidéo HD, etc...), mais dans tous les espaces, il y deux composantes que l'on appelle *chrominance*, qui comportent une information sur la teinte et la saturation (proportion de blanc), et une composante que l'on appelle *luminance*, qui peut être vue comme une conversion en nuances de gris de l'image couleur, et qui rend compte des zones sombres ou claires.

On s'intéresse ici à l'espace de représentation YCbCr full-range, 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 :
$$Y \approx 0.3R + 0.6G + 0.1B$$
- 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 étantla 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 :

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

Ce système garantit que si l'image couleur dans la représentation RGB prend des valeurs entre 0 et 255, alors sa représentation dans la représentation YCbCr prend également des valeurs entre 0 et 255. Notons qu'il est parfaitement possible d'inverser ce système d'équation pour retrouver R, G et B, à partir de Y, Cb et Cr. Il existe d'autres représentations YCbCr pour la télévision standard (SDTV) et pour la télévision HD (HDTV).

Ce système de transmission et de codage de la couleur été nécessaire pour plusieurs raisons :

- Assurer une restitution de la couleur plus fiable. Le signal Y, bien que noir et blanc, contient en effet toutes les infos pour les trois couleurs, et Cb et Cr étant des différences par rapport à Y, une valeur de soustraction erronée entre Y et Cr ou Cb (en cas de problème de transmission) permet de rectifier plus aisément le tir.
- Dissocier l'information en noir et blanc Y de l'information en couleurs Cb et Cr afin de diminuer la définition de la couleur seule, tout en restant en deçà de la perception visuelle humaine. En effet, l'oeil humain est plus sensible à la luminance qu'à la chrominance. Il est par conséquent possible de dégrader la chrominance d'une image tout en gardant une bonne qualité.

2.2 Etude sous MATLAB

1. Créer sous MATLAB un script vide nommé `TP5Partie3.m`
2. Ouvrir l'image `mandrill.tif`, la stocker dans une matrice `X_rgb` et la renormaliser. Afficher l'image.
3. 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.

   La fonction `my_rgb2ycbcr.m` prend en entrée une image couleur renormalisée et renvoie une image dans l'espace YCbCr renormalisée (tous les coefficients sont compris entre 0 et 1).

   ```
   X_ycbcr = my_rgb2ycbcr(X_rgb);
   ```

4. 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.
   (a) Pour cela, commencer par remettre les valeurs de `Cb` et `Cr` entre 0 et 255.
   (b) 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
   ```

   (c) Enfin, remettre les valeurs de `Cb` et `Cr` entre 0 et 1.
   (d) Construire ensuite une matrice 3D `Z_ycbcr` ayant `Y` comme composante Y et `Cb` et `Cr` comme composantes Cb et Cr.
   (e) 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.
   (f) Essayer de quantifier `Cb` et `Cr` sur 4 bits et observer le résultat. Commenter.