Travaux pratiques de Traitement d'Images sous MATLAB

Année 2018

1 But du TP

Dans ce TP vous apprendrez à :

1. traiter un grand nombre d'images de types différents : TRIMAGO, RGB, Intensité , Indexés , IHS , IH1H2S , JPEG
2. faire des profils lignes des images numériques afin de révéler les valeurs d'intensité sous-jacentes
3. faire des rotations colorimétriques afin de changer une couleur en une autre
4. représenter une image complexe (image radar SLC par exemple) comme une combinaison linéaire d'exponentielles complexes ou une image réelle (image d'intensité par exemple) comme une combinaison linéaire de cosinus phasés en calculant sa transformée de Fourier
5. simuler la célèbre expérience de filtrage optique d"Abbe et Porter
6. analyser le contenu d'une image JPEG pour la décoder
7. améliorer vos compétences en programmation MATLAB.

2 MATLAB

MATLAB (« matrix laboratory ») est un langage de programmation émulé par un environnement de développement du même nom ; il est utilisé à des fins de calcul numérique. Développé par la société The MathWorks, MATLAB permet de manipuler des matrices, d'afficher des courbes et des données, de mettre en œuvre des algorithmes, de créer des interfaces utilisateurs, et peut s'interfacer avec d'autres langages comme le C, C++, Java, et Fortran. Les utilisateurs de MATLAB (environ un million en 2004) sont de milieux très différents comme l'ingénierie, les sciences et l'économie dans un contexte aussi bien industriel que recherche. Matlab peut s'utiliser seul ou bien avec des toolbox (« boîtes à outils »).
L'organisation IEEE a établi un classement de la popularité des langages de programmation en 2014 ; MATLAB se classe dans le TOP 10 [1]. Pour ce classement, IEEE [2] prend en compte le nombre de références sur Twitter [3], le nombre de nouveaux projets sur GitHub [4], le nombre de questions concernant le langage sur StackOverflow [5],

1. 1. Java , 2. C , 3. C++ , 4. Python , 5. C # , 6. PHP , 7. Javascript , 8. Ruby , 9. R , 10. MATLAB
2. L'Institute of Electrical and Electronics Engineers est la plus grande association professionnelle du monde pour le développement de la technologie. Elle compte plus de 400.000 membres répartis dans plus de 160 pays.
3. Twitter est un outil de microblogage géré par l'entreprise Twitter Inc. Il permet à un utilisateur d'envoyer gratuitement de brefs messages, appelés tweets (« gazouillis »), sur internet, par messagerie instantanée ou par SMS. Ces messages sont limités à 140 caractères. Le service est rapidement devenu populaire, jusqu'à réunir plus de 500 millions d'utilisateurs dans le monde fin février 2012, dont environ 200 millions utilisant Twitter au moins une fois par mois
4. GitHub est un service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions Git. En plus d'offrir l'hébergement de projets avec Git, le site offre de nombreuses fonctionnalités habituellement retrouvées sur les réseaux sociaux comme les flux, la possibilité de suivre des personnes ou des projets ainsi que des graphes de réseaux pour les dépôts. Github offre aussi la possibilité de créer un wiki et une page web pour chaque dépôt. Le site offre aussi un logiciel de suivi de problèmes. Alors que le système traditionnel open source amène chaque contributeur à télécharger les sources du projet et à proposer ensuite ses modifications à l'équipe du projet, Github repose sur le principe du fork par défaut : toute personne « forkant » le projet devient publiquement de facto le leader de son projet portant le même nom que l'original. Avec 16,7 millions de dépôts en 2014, GitHub est le plus grand hébergeur de code du monde.
5. Stack Overflow est un site web proposant des questions et réponses sur un large choix de thèmes concernant la programmation informatique.

les références sur Hacker News et Reddit [6] et le nombre d'offre d'emplois sur différents sites demandant une compétence dans ces langages.
MATLAB est un langage orienté vers la programmation "vectorielle" et "matricielle" , donc particulièrement bien adapté au traitement d'images. Il est souvent possible d'écrire du code MATLAB de façon extrêmement concise sans boucle for. C'est ce que vous vous efforcerez de faire durant ce TP.

3 Formats d'images

3.1 Les images traitables par MATLAB

MATLAB est capable de traiter trois types d'images numériques :

1. les images indexées : elles sont composées d'une matrice de données et d'une matrice colormap
2. les images d'intensité : ce sont des matices de données représentant les intensités lumineuses. Les images d'intensité sont rarement sauvegardées avec une colormap mais on utilise toujours une colormap pour les visualiser. Elles sont donc traitées comme les images indexées.
3. les images RGB : elles sont composées de trois matrices de données représentant les intensités dans le rouge (R) , dans le vert (G) et dans le bleu (B)

3.2 Autres formats

Il existe beaucoup d'autres formats d'images numériques :

1. les images matricielles constituées de pixels
   (a) les images JPEG : ce sont des images comprimées avec perte utilisées en photographie
   (b) les images JPEG2000 : utilisées elles aussi en photographies. Le codage JPEG2000 utilise une transformation par ondelettes alors que JPEG utilise la transformation en cosinus
   (c) les images GIF : couramment utilisées sur le web, leur codage est sans perte
   (d) les images PNG : particulièrement appropriées lorsqu'il s'agit d'enregistrer des images synthétiques destinées au web comme des graphiques, des icônes, des images représentant du texte ou des images avec peu de dégradés. PNG a tendance à remplacer GIF.
   (e) les images TIFF : TIFF est un format de conteneur (comme zip) trés flexible, utilisé des scanners industriels aux appareils photo numériques et aux imprimantes.
   (f) les images TRIMAGO : il s'agit d'un format propriétaire sans codage mais contenant un descripteur de l'image
   (g) les images DICOM : utilisées en imagerie médicale
   (h) les images radar SLC (Single Look Complex) : ce sont des images complexes codant une amplitude et une phase
2. les images vectorielles : elles sont composées d'objets géométriques (segments de droite, arcs de cercle, courbes de Bézier, polygones, etc.) définis chacun par différents attributs (forme, position, couleur, remplissage, visibilité) et auxquels on peut appliquer différentes transformations géométriques. Elles sont utilisées pour l'établissement de cartes.

4 Le codage RGB

Il consiste à représenter l'espace des couleurs à partir des trois rayonnements monochromatiques suivants :

1. le Rouge (R) de longueur d'onde 700 nm
2. le Vert (G) (Green en anglais) de longueur d'onde 546,1 nm

6. Reddit est un site web communautaire de partage de signets permettant aux utilisateurs de soumettre leurs liens et de voter pour les liens proposés par les autres utilisateurs. Ainsi, les liens les plus appréciés du moment se trouvent affichés en page d'accueil. Hacker News est centré autour de l'actualité des technologies informatiques, du hacking et des startups, et promeut tout contenu susceptible de « gratifier la curiosité intellectuelle » des lecteurs.

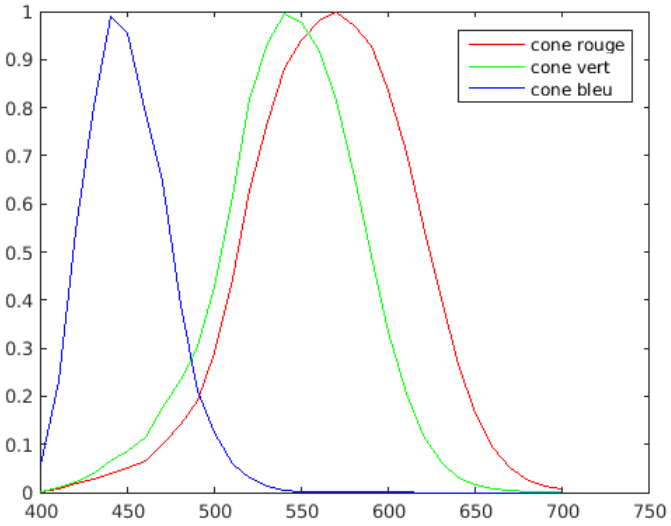


FIGURE 1 – Réponses spectrales des cônes humains

3. le Bleu (B) de longueur d'onde 435,8 nm

Le modèle RGB est le système le plus courant et le plus utilisé car il dérive de la technologie employée dans l'industrie de l'image et du numérique. Moniteurs à tube cathodique ou à cristaux liquides, appareils photos numériques, scanners, utilisent tous un système RGB basé sur le principe additif des trois couleurs primaires, rouge, vert et bleu. En codant chacune des composantes colorées sur un octet, on obtient 256 valeurs pour chaque couleur. Il est donc possible en théorie d'obtenir 256^3, soit 16.777.216 couleurs, c'est-à-dire beaucoup plus que l'œil humain n'est capable d'en discerner (environ 350000). Cette valeur reste cependant théorique, car les écrans ne permettent pas d'afficher un tel nombre de couleurs [7].

5 Matching trichromatique

Par synthèse additive des 3 primaires R,G et B, on obtient la même sensation colorée que celle produite par une lumière monochromatique de longueur d'onde λ_S si l'équation suivante est vérifiée :

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = L_\lambda \begin{pmatrix} M_r(\lambda_r) & M_r(\lambda_g) & M_r(\lambda_b) \\ M_g(\lambda_r) & M_g(\lambda_g) & M_g(\lambda_b) \\ M_b(\lambda_r) & M_b(\lambda_g) & M_b(\lambda_b) \end{pmatrix}^{-1} \begin{pmatrix} M_r(\lambda_s) \\ M_g(\lambda_s) \\ M_b(\lambda_s) \end{pmatrix} = \begin{pmatrix} \bar{r}(\lambda_s) \\ \bar{g}(\lambda_s) \\ \bar{b}(\lambda_s) \end{pmatrix} \quad (1)$$

où L_λ est la luminance de l'objet émettant une lumière monochromatique de longueur d'onde λ_S dans la direction de l'oeil, M_r , M_g et M_b sont les réponses spectrales des cônes rouge, vert et bleu, $\bar{r}$, $\bar{g}$ et $\bar{b}$ sont appelés composantes trichromatiques spectrales ou encore fonctions d'appariement de la couleur (color matching functions)

6 Représentation de la couleur

A côté du codage RGB, on trouve :

1. le codage XYZ [8]
2. le codage R*G*B*W : proche de RGB, il code explicitement le blanc

7. d'après http ://e-cours.univ-paris1.fr/modules/uved/envcal/html/compositions-colorees/representations-couleur/systemes-materiels/rgb.html
8. afin d'éviter l'inconvénient de composantes trichromatiques négatives, les ingénieurs de la CIE ont imaginé le codage XYZ dont les composantes trichromatiques spectrales sont positives ou nulles

3. les codages perceptuels : IHS , IH1H2S , CIELab
4. CMYK ou quadrichromie : utilisé en imprimerie
5. les standards vidéo ou TV : YUV (utilisé dans les standards PAL et SECAM) , YIQ (utilisé dans le standard NTSC).

7 Introduction aux images TRIMAGO et JPEG

Vous venez de télécharger le dossier "Images" contenant deux types d'images numériques

1. les images dont les noms sont suffixés par .tri sont des images au format TRIMAGO (un format "propriétaire"). Ces images sont non comprimées.
2. les images dont les noms sont suffixés par .jpg sont des images au format JPEG. Elles sont donc comprimées.

Pour s'en rendre compte, comparez les volumes des deux images couleur singe.tri et singe.jpg qui représentent la même scène, à savoir le visage d'un mandrill. Leur nombre de pixels est le même : 512 x 512 = 262.144. Mais :

1. singe.tri occupe 786,9 ko soit 3 octets = 24 bits par pixel [9]
2. singe.jpg occupe 78,6 ko soit 2,4 bit par pixel

L'image singe.tri est donc 10 fois plus volumineuse que l'image singe.jpg . Vous vérifierez néanmoins qu'elles paraissent visuellement identiques sur l'écran de votre PC. Cet exemple montre donc la qualité de la compression JPEG.

8 Lecture et affichage des images au format TRIMAGO

Ces images se composent d'un descripteur suivi de l'image proprement dite rangée ligne par ligne et canal par canal dans l'ordre R , V , B s'il s'agit d'une image couleur. Chaque mesure d'éclairement d'un pixel est quantifié sur un octet (entre 0 et 255)
La taille du descripteur est celle d'une ligne de l'image. Sa composition est :
descripteur = (typeimage, etiquette,nlig,ncol,ncan,zonelibre)
où

1. *typeimage* code le type de l'image sur 4 caractères
   (a) IMON s'il s'agit d'une image monochrome (noir et blanc, à niveaux de gris)
   (b) ITRI s'il s'agit d'une image trichrome (couleur) codée en R (rouge) , G (vert) et B (bleu)
2. *etiquette* contient un commentaire sur l'image de 64 caractères
3. *nlig* contient le nombre de lignes de l'image, codé en "integer big endian" [10] sur 32 bits.
4. *ncol* contient le nombre de colonnes de l'image, codé en "integer big endian" sur 32 bits.
5. *ncan* contient le nombre de canaux de l'image, codé en "integer big endian" sur 32 bits.
6. *zonelibre* est une zone libre de ncol - 80 caractères.

Vous allez commencer par lire une image au format TRIMAGO (dans le dossier "Images") puis à l'afficher avec son commentaire sur votre écran de machine. L'image lue sera rangée dans le tableau im. Les '??' indiquent les champs qui doivent être complétés.
Solution : script lecshowtrimago.m

```
%
NOMFIC = uigetfile('Images/*.tri');
%
NOMFIC = ['Images/' NOMFIC];
```

9. la dynamique du capteur étant de 256 (un octet) sur les trois primaires R, V ,B et l'image n'étant pas comprimée, il s'ensuit que son volume est (approximativement) égal à son nombre total de pixels fois 24 bits, soit 24 bits par pixel

10. Le codage "big endian" (en français, mot de poids fort en tête) veut dire que l'octet de poids le plus fort est enregistré à l'adresse mémoire la plus petite, l'octet de poids inférieur à l'adresse mémoire suivante et ainsi de suite. Dans le codage "little endian"(en français, mot de poids faible en tête), c'est l'inverse, l'octet de poids le plus faible est enregistré à l'adresse mémoire la plus petite, etc. Il faut savoir que les processeurs x86 qui se trouvent dans les PC ont une architecture "little endian"

```
imtri = fopen(NOMFIC,'r','??');
%
typeimage = fread(imtri,4,'??')[11];
typeimage = typeimage';
etiquette = fread (imtri,64,'??');
etiquette = etiquette';
nlig = fread(imtri,1,'??');
ncol = fread(imtri,1,'??');
ncan = fread(imtri,1,'??');
fread (imtri,ncol-80);
%
if strcmp(typeimage,'ITRI')
   im = fread(imtri)[12];
   im = reshape(im,'??','??','??');
   im = permute(im,'??');
   him[13] = figure ('Name',etiquette,'Units','??');
   imshow(im/255);
else if strcmp(typeimage,'IMON')
      im = fread (imtri,['??','??']);
      im = im';
      him = figure ('Name',etiquette,'Colormap',gray(256),'Units','??');
      imshow(im,[0,255]);
   else
      disp('ERREUR');
   end
end
fclose(imtri);
```

9 Lecture et affichage des images au format JPEG

Solution : script lecshowjpg.m

```
% Lecture d'une image JPEG
%
NOMFIC = uigetfile('Images/*.jpg');
%
NOMFIC = ['Images/' NOMFIC];
im = imread[14] (NOMFIC);
pas = str2double(input('Pas d"echantillonnage : ,'s')); % pas d'échantillonnage de im
im = double(im(1 :pas :end,1 :pas :end,1 :end)); %sous-échantillonnage de l'image et transformation en double
%
nlig = size(im,1);
ncol = size(im,2);
```

11. Si A = *fread(fileID,sizeA,precision)* , le champ *precision* définit les classes des valeurs d'entrée (celles du fichier *fileID* et de sortie (matrice A). Il y a principalement trois formes d'expression de la chaîne de caractères décrivant ce champ :
    1. *source* . Par exemple 'int16' . Les valeurs d'entrée sont de la classe spécifiée par *source*. A est de type double.
    2. *source => output* . Par exemple 'int8=>char' . Les valeurs d'entrée sont de la classe spécifiée par *source*. La classe de la matrice de sortie A est spécifiée par *output*
    3. **source* . Par exemple '*int16' . Les valeurs d'entrée et la matrice de sortie, A, sont de la classe spécifiée par *source*

    Par défaut, *precision* vaut 'uint8 => double'
12. par défaut, fread lit un fichier octet par octet, l'interprète comme un entier non signé (uint8) et retourne un vecteur de type double (voir la note 11)
13. him est le "handle" , l'identifiant de la figure ; en tapant figure(him), la figure identifiée par him devient visible
14. la fonction *imread* réalise la décompression totale de l'image JPEG alors que la fonction *jpeg_read* (cf.24.2) ne réalise qu'un décodage partiel.

```
%
% Affichage de l'image
him = figure ('Name','Image JPEG','Units','pixels');
if ismatrix(im)
   % L'image JPEG est monochrome
   typeimage = 'IMON';
   ncan = 1;
   colormap(gray(256));
   imshow(im,[0,255]);
else if ndims(im) == 3
      % L'image JPEG est en couleur
      typeimage = 'ITRI';
      ncan = 3;
      imshow(im/255);
   else
      disp('ERREUR');
   end
end
```

10 Conversion des images TRIMAGO en images JPEG

On désire maintenant coder **toutes** les images .tri au format .jpg. Une fois cette conversion effectuée, un calculera pour chaque image .tri le taux de compression obtenu.

Solution : script tri2jpg.m

```
% tri2jpg
%
lecshowtrimago;
NOMFIC = [NOMFIC(1 :find(NOMFIC =='.')),'jpg'];
imwrite(im/255,NOMFIC,'jpg','Comment',etiquette);
disp(imfinfo(NOMFIC));
```

11 Affichage des profils ligne

Le profil ligne d'une image monochrome est le graphe de la fonction qui à tout pixel d'une ligne fait correspondre la valeur quantifiée d'éclairement de ce pixel. Le profil ligne d'une image trichrome RGB est composé des trois graphes des fonctions qui à tout pixel d'une ligne font correspondre les valeurs quantifiées d'éclairement de ce pixel dans les trois canaux primaires R, G et B. [15] L'utilisateur désigne une ligne de l'image avec la souris et le programme affiche son profil.

Solution : script profim.m [16]

```
%% Trace le profil horizontal de la ligne de l'image (monochrome ou couleur) designee par le curseur.
disp('Designer une ligne dont on veut le profil avec le curseur');
[ ,x] = ginput (1);
```

15. Ceci n'est vrai que pour les capteurs Foveon X3 des reflex numériques SIGMA ; ceux-ci enregistrent bien , pour chaque pixel, les trois couleurs primaires R, G et B. Les autres capteurs utilisent un filtre qui ne permet à chaque pixel de ne voir qu'une seule couleur : soit R, soit G, soit B. L'image capturée par les reflex SIGMA est donc une image vue intégralement en couleurs par le capteur, alors que les autres appareils photo numériques restituent une image calculée, reconstituée à partir d'informations partielles correspondant au tiers de l'information totale de l'image réelle. Il en résulte une qualité d'image inconnue jusqu'ici en photo numérique, et un rendu des détails plus riche que celui d'un capteur conventionnel qui comporterait au moins le double de pixels "classiques" obtenus par interpolation. En effet, un capteur conventionnel de 10 millions de pixels n'enregistre que le tiers de l'image finale, avec 2,5 million d'informations en bleu, 5 millions en vert et 2,5 million en rouge et calcule donc les 2/3 de l'image qui lui manquent lors de la capture. L'image issue des reflex numériques SIGMA est plus précise, avec en particulier plus de détails dans les couleurs, plus naturelle, et sans artefacts tels que le moirage.

16. le scipt profim permet de révéler l'image numérique sous-jacente à l'image affichée à l'écran

```
tempy = x*ones(ncol,1)';
tempv = 1 :ncol;
%On efface toute trace de marque anterieure.
eval('delete(hl)','');
%
if all(typeimage == 'IMON')
   hl = line(tempv,tempy,'Color','g');
   %On efface toute trace de marque anterieure.
   eval('delete(hp)','');
   hp = line(tempv,nlig*(1 - im(round(x), :)/256),'Color','r');
else
   if all(typeimage == 'ITRI')
      hl = line(tempv,tempy,'Color','w');
      %On efface toute trace de marque anterieure.
      eval('delete(hpr)','');
      eval('delete(hpg)','');
      eval('delete(hpb)','');
      hpr = line(tempv,nlig*(1 - im(round(x), :,1)/256),'Color','r');
      hpg = line(tempv,nlig*(1 - im(round(x), :,2)/256),'Color','g');
      hpb = line(tempv,nlig*(1 - im(round(x), :,3)/256),'Color','b');
   else
      disp('ERREUR');
   end
end
```

12 Espaces perceptuels

Comme leur nom l'indique, ces espaces se construisent à partir de trois paramètres servant à caractériser l'impression visuelle, à savoir :

1. l'intensité de la couleur (I)
2. sa teinte (en anglais, "hue") (H). La teinte est la forme pure d'une couleur, c'est à dire sans adjonction de blanc [17].
3. sa saturation (S). La saturation mesure le degré de pureté de la couleur. Plus la saturation est grande, plus la couleur est pure.

Pour cela, on définit un axe d'intensité I correspondant à tous les gris et allant du noir au blanc de référence. En se plaçant dans un plan perpendiculaire à cet axe et en tournant autour de cet axe, on parcourt toutes les teintes H. Enfin la distance d'un point à l'axe des intensités mesure la saturation de la couleur.

13 Le codage IHS et le passage de RGB à IHS

Ce codage permet de repérer les couleurs dans un espace perceptuel.
Le passage de RGB à IHS s'effectue en deux étapes, la première est linéaire et la seconde non linéaire.

1. Calcul de : $\begin{pmatrix} I \\ \nu_1 \\ \nu_2 \end{pmatrix} = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ -\frac{1}{2} & -\frac{1}{2} & 1 \\ \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} & 0 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$
2. Calcul de : H = atan2(ν_2, ν_1) [18] et de S = $\sqrt{\nu_1^2 + \nu_2^2}$

17. Cette définition pose néanmoins problème. En effet, elle ne peut s'appliquer au blanc qui est pourtant une couleur. Parler de la forme pure du blanc sans adjonction de blanc est un non-sens car si on retire tout le blanc du blanc il ne reste ... rien !

18. atan2(ν_2, ν_1) donne l'angle θ tel que $\tan(\theta) = \frac{\nu_2}{\nu_1}$, $\sin(\theta) = \frac{\nu_2}{\sqrt{\nu_1^2+\nu_2^2}}$, $\cos(\theta) = \frac{\nu_1}{\sqrt{\nu_1^2+\nu_2^2}}$ et $\theta \in [-\pi, \pi]$. Cette fonction diffère de l'arc tangente atan(y) qui donne l'angle θ tel que $\tan(\theta) = y$ et $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$. Ce serait une erreur que d'écrire H = atan$(\frac{\nu_2}{\nu_1})$.

Solution : fonction rgb2ihs.m

```
function image_ihs = rgb2ihs(image_rgb)
% Fonction effectuant la conversion d'une image RGB en IHS
%
M = [1/3 1/3 1/3; -1/2 -1/2 1; sqrt(3)/2 -sqrt(3)/2 0];
%
nlig = size(image_rgb,1);
ncol = size(image_rgb,2);
image_ihs = zeros(nlig,ncol,3);
%
for nolig=1 :nlig,
  for nocol=1 :ncol,
    image_ihs(nolig,nocol, :) = M * double(squeeze(image_rgb(nolig,nocol, :)));
  end
end
%
v1 = image_ihs( :, :,2);
v2 = image_ihs( :, :,3);
%
H = atan2(v2,v1);
S = sqrt(v1.^2 + v2.^2);
%
image_ihs( :, :,2) = H;
image_ihs( :, :,3) = S;
end
```

14 Programmation du codage IHS des images TRIMAGO et JPEG

Nous désirons coder en IHS les images au format TRIMAGO et au format JPEG. Nous procèderons en deux étapes :

1. Lecture d'une image au format TRIMAGO ou au format JPEG
   Solution : lecimage.m

```
% Lecture d'une image au format TRIMAGO ou au format JPEG
%
NOMFIC = uigetfile('Images/*.*');
suffixe = NOMFIC(find(NOMFIC=='.')+1 :length(NOMFIC));
NOMFIC = ['Images/' NOMFIC];
if strcmp(suffixe,'tri')
lecshowtrimago2;
else
if strcmp(suffixe,'jpg')
lecshowjpg2;
else
disp('ERREUR')
return
end
end
```

2. Codage en IHS
   Solution : fonction rgb2ihs.m

Remarquez aussi qu'atan2(0,0) = 0. Donc si l'on considère un pixel de couleur noire, grise ou blanche (R=G=B), alors H = 0. La "teinte" du noir, du gris ou du blanc (H) est donc aussi celle du bleu qui vaut aussi 0, ce qui semble en contradiction avec la grammaire du mot teinte (on ne dira jamais par exemple que la teinte du blanc est la même que celle du bleu !).

Analysons le résultat du codage en IHS de la mire TV couleur : mireTV.jpg . Il est particulièrement intéressant d'afficher l'image des saturations (S) image_ihs(:, :,3). Que remarquons-nous ?

15 Le passage de IHS à RGB

Programmer le passage du codage IHS au codage RGB
Solution : fonction ihs2rgb.m

16 Calcul et visualisation d'un anneau des couleurs pures

Nous nous intéressons à la composante H d'une couleur RGB. Pour cela, nous allons construire un anneau des couleurs pures ou anneau des teintes. Dans cet anneau toutes les couleurs sont saturées (S=255) et leurs intensités égales à 160.
Solution anneau_teintes.m

```
% Construction de l'anneau des teintes
%
nlig = 256;
ncol = 256;
image_ihs = zeros(nlig,ncol,3);
cx = 128;
cy = 128;
Rmin = 30;
Rmax = 80;
%
for i = 1 :nlig
   for j=1 :ncol
      dist = sqrt((i-cx)^2 + (j-cy)^2);
      if Rmin <= dist && dist <= Rmax
         image_ihs(i,j,1) = 160;
         image_ihs(i,j,3) = 255;
         sintheta = (cx-i)/dist;
         costheta = (j-cy)/dist;
         angle = atan2(sintheta,costheta);
         image_ihs(i,j,2) = angle;
      else
         image_ihs(i,j,1) = 0;
         image_ihs(i,j,3)= 0;
      end
   end
end
%
image_rgb = ihs2rgb(image_ihs);
him = figure ('Name','Anneau des teintes','Units','pixels');
imshow(image_rgb/255);
```

17 Ordre des couleurs

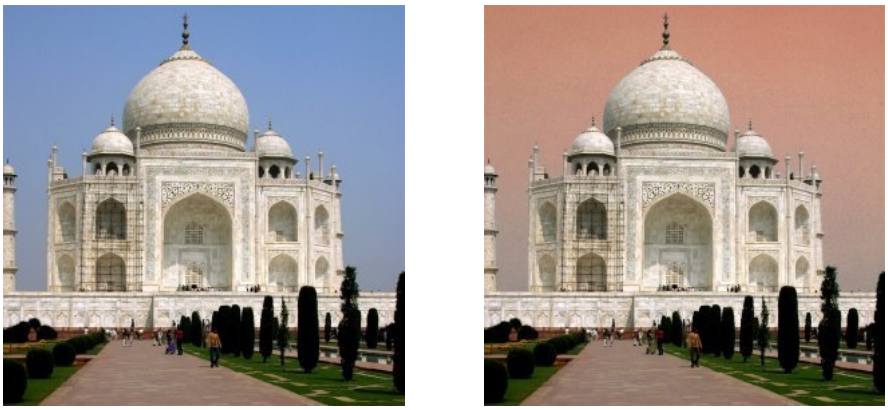
Lorsqu'on tourne sur l'anneau dans le sens trigonométrique, on rencontre successivement les couleurs suivantes : bleu, violet, magenta, rouge, orange, jaune, vert, cyan, bleu, ... [19]

19. On remarquera que les couleurs blanche, noire ou grise sont absentes de cet anneau. Notons aussi que l'on parcourt un continuum de couleurs et non pas un nombre fini de couleurs mais notre vocabulaire ne nous permet que de désigner un nombre

Essayer d'attribuer à chaque nom de couleur un intervalle angulaire délimitant un secteur contenant cette couleur.
Solution teinte.m

18 Rotation des couleurs dans le repère IHS

La rotation des couleurs est une opération qui permet d'échanger un intervalle de couleurs contre un autre. Comme exemple, considérons l'image du Taj Mahal en Inde et supposons que nous voulions échanger son ciel bleu contre un ciel d'aurore.



(a) Image originale (b) Image transformée

FIGURE 2 – Rotation des couleurs

La solution est d'échanger un intervalle contenant la teinte bleue contre un autre contenant la teinte rouge. Les couleurs dont les teintes ne figurent pas dans l'intervalle initial ne seront pas affectées : le Taj Mahal lui-même reste blanc, les pelouses restent vertes, etc. [20]
Dans ce qui suit, nous nous proposons de programmer une rotation des couleurs qui sera cette fois globale : toutes les couleurs présentes dans l'anneau seront échangées contre une autre couleur. Pour cela, l'axe de rotation est l'axe des intensités I du repère IHS. Pour calculer l'angle de cette rotation, on désigne dans une image RGB un pixel dont on va remplacer la couleur C_1 et un autre pixel possédant la couleur de remplacement C_2. On détermine alors les teintes H_1 et H_2 de ces couleurs. L'angle de rotation est alors égal à $H_2 - H_1$

Solution rot_teinte.m

```
% Rotation de teinte sur une image trimago ou jpeg
%
% Lecture d'une image trimago ou jpeg
%
lecimage;
%
% Codage IHS
image_ihs = rgb2ihs(im);
sauv = image_ihs;
%
disp('Désigner une couleur à remplacer');
[y1,x1] = ginput (1);
x1 = round(x1);
```

fini de couleurs.

20. Ce traitement a été fait avec le logiciel GIMP

```
y1 = round(y1);
disp(['x1 = ', num2str(x1),' y1= ',num2str(y1)]);
angle1d = (image_ihs(x1,y1,2)/pi)*180;
disp(['Angle de la teinte à remplacer : ',num2str(angle1d),' degrés ** teinte : ',teinte(angle1d)]);
%
disp(' ');
%
disp('Désigner une couleur de remplacement');
[y2,x2] = ginput (1);
x2 = round(x2);
y2 = round(y2);
disp(['x2 = ', num2str(x2),' y2= ',num2str(y2)]);
angle2d = (image_ihs(x2,y2,2)/pi)*180;
disp(['Angle de la teinte de remplacement : ',num2str(angle2d),' degrés ** teinte : ',teinte(angle2d)]);
%
% Calcul de l'angle de rotation
theta0 = image_ihs(x1,y1,2) - image_ihs(x2,y2,2);
theta0d = num2str((theta0/pi)*180);
%
image_ihs( :, :,2) = image_ihs( :, :,2) - theta0;
image_rgb = ihs2rgb(image_ihs);
%
him = figure ('BackingStore','off','Name',['Rotation de teinte de ',num2str(theta0d), ' degres'],'Units','pixels');
imshow(image_rgb/255);
```

18.1 Cas particuliers du blanc et du noir

Considérons maintenant une image RGB (par exemple MireTV.jpg) possédant un pixel de couleur blanche et un autre pixel possédant une autre couleur. Essayons de remplacer cette autre couleur par du blanc. Qu'observe-t-on ? Expliquer le résultat.
Considérons maintenant une image RGB (par exemple MireTV.jpg) possédant un pixel de couleur noire et un autre pixel possédant une autre couleur. Essayons de remplacer cette autre couleur par du noir. Qu'observe-t-on ? Expliquer le résultat.

19 Le système perceptuel IH1H2S. Extension des teintes au blanc

Bien que le blanc soit une couleur, il n'a pas de teinte H. Le programme rot_teinte.m ne peut donc attribuer une couleur blanche (ou grise) à une autre couleur ou inversement changer du blanc en une autre couleur. Pour qu'il en soit ainsi, une solution va consister à changer le codage RGB en un codage quadridimensionnel R*G*B*W (W représentant le blanc), puis à passer comme précédemment dans un espace perceptuel IH1H2S où l'on trouve cette fois-ci non plus un cercle mais une sphère des couleurs pures ou des teintes (étendues). Dans cette sphère, la teinte blanche est présente. On peut alors par une rotation de cette sphère des couleurs pures changer du rouge en du blanc ou inversement du blanc en du rouge. Pour visualiser le résultat, il suffit de retrouver l'image RGB correspondante, en passant du code IH1H2S au code RGBW puis au code RGB.

19.1 Le passage de RGB à IH1H2S

19.1.1 Le passage de RGB à R*G*B*W

Le passage de RGB à R*G*B*W repose sur la constatation suivante : l'intensité d'une ampoule émettant une lumière blanche est la même que celle produite par trois ampoules rouge, verte et bleu de même intensité.

Donc :

$$\begin{cases} R^* = R - \min(R,G,B) \\ G^* = G - \min(R,G,B) \\ B^* = B - \min(R,G,B) \\ W = \min(R,G,B) \end{cases}$$

19.1.2 Le passage de R*G*B*W à IH1H2S

1. Calcul de : $\begin{pmatrix} I \\ \nu_1 \\ \nu_2 \\ \nu_3 \end{pmatrix} = \begin{pmatrix} \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \\ -\frac{1}{3} & -\frac{1}{3} & -\frac{1}{3} & 1 \\ -\frac{\sqrt{2}}{3} & -\frac{\sqrt{2}}{2} & \frac{2\sqrt{2}}{3} & 0 \\ \frac{\sqrt{2}}{\sqrt{3}} & -\frac{\sqrt{2}}{\sqrt{3}} & 0 & 0 \end{pmatrix} \begin{pmatrix} R^* \\ G^* \\ B^* \\ W \end{pmatrix}$
2. Calcul de : H1 = atan2(ν_3, ν_2) , H2 = asin(ν_1/S) et S = $\sqrt{\nu_1^2 + \nu_2^2 + \nu_3^2}$

Solution rgb2ih1h2s.m

```
function image_ihs = rgb2ih1h2s(image_rgb)
%
% Fonction effectuant la conversion d'une image RGB en IH1H2S
% W = White (Blanc)
%
% Au départ, on a une image rgb : image_rgb
%
% Prétraitement : Passage de RGB à R*G*B*W
% Pour cela, on retire tout le blanc possible de RGB et on le met dans W
R = image_rgb( :, :,1) ;
G = image_rgb( :, :,2) ;
B = image_rgb( :, :,3) ;
W = min(R,G) ;
image_rgb( :, :,4) = min(W,B) ;
%
for k = 1 :3
   image_rgb( :, :,k) = image_rgb( :, :,k) -image_rgb( :, :,4) ;
end
%
M = ? ? ;
% (M (4,4) n'est pas orthogonale)
%
nlig = size(image_rgb,1) ;
ncol = size(image_rgb,2) ;
image_ihs = zeros(nlig,ncol,4) ;
%
for nolig=1 :nlig,
   for nocol=1 :ncol,
      image_ihs(nolig,nocol, :) = M * double(squeeze(image_rgb(nolig,nocol, :))) ;
   end
end
%
v1 = image_ihs( :, :,2) ;
v2 = image_ihs( :, :,3) ;
v3 = image_ihs( :, :,4) ;
%
H1 = atan2(v3,v2) ;
S = sqrt(v1.^2 + v2.^2 + v3.^2) ;
H2 = asin(v1./S) ;
```

```
%
image_ihs( :, :,2) = H1;
image_ihs( :, :,3) = H2;
image_ihs( :, :,4) = S;
end
```

19.2 Le passage de IH1H2S à RGB

Solution ih1h2s2rgb.m

19.3 Rotation de la sphère des couleurs pures étendues (blanc inclus)

Ayant désigné une couleur à remplacer puis une couleur de remplacement, toute couleur dans l'image de départ sera remplacée ssi l'angle de sa teinte avec celle de la couleur à remplacer mesurée sur la sphère des teintes est inférieure à un certain seuil. On obtient une transformation plus générale que celle codée dans le logiciel GIMP puisque le blanc auquel on a maintenant attribué une teinte peut être une couleur à remplacer aussi bien qu'une couleur de remplacement.
A noter que la matrice de rotation autour d'un axe dirigé par un vecteur unitaire $\overrightarrow{u} = (u_x, u_y, u_z)$ et d'angle θ s'écrit : $R = \begin{pmatrix} u_x^2 + (u_y^2 + u_z^2)c & u_x u_y(1-c) - u_z s & u_x u_z(1-c) + u_y s \\ u_x u_y(1-c) + u_z s & u_y^2 + (u_x^2 + u_z^2)c & u_y u_z(1-c) - u_x s \\ u_x u_z(1-c) - u_y s & u_y u_z(1-c) + u_x s & u_z^2 + (u_x^2 + u_y^2)c \end{pmatrix}$
avec $c = \cos(\theta)$ et $s = \sin(\theta)$

Solution rot_teinte2.m

```
%% Rotation de couleur sur une image trimago ou jpeg
% On remplace la couleur si sa teinte est suffisamment proche de la teinte à remplacer
% Elle reste inchangée sinon
% On utilise le codage IH1H2S : le blanc possède donc une teinte
%
M = [1/6 1/6 1/6 1/6; -1/3 -1/3 -1/3 1; -sqrt(2)/3 -sqrt(2)/3 2*sqrt(2)/3 0; sqrt(2)/sqrt(3) -sqrt(2)/sqrt(3) 0
0];
%
%% Lecture de l'image trimago ou jpeg qui va subir une rotation colorimétrique
lecimage;
im_depart = im; %on sauvegarde l'image im dans im_depart

%% Codage IH1H2S de im_depart
image_ihs = rgbw2ih1h2s(im);
%
%% Calcul des matrices V1 ,V2 et V3
V1 = sin(image_ihs( :, :,3));
V2 = cos(image_ihs( :, :,3)) .* cos(image_ihs( :, :,2));
V3 = cos(image_ihs( :, :,3)) .* sin(image_ihs( :, :,2));
%
%% Couleur à remplacer
disp('Désigner une couleur à remplacer');
[y1,x1] = ginput (1);
x1 = round(x1);
y1 = round(y1);
disp(['x1 = ', num2str(x1),' y1= ',num2str(y1)]);
%
psi1d = (image_ihs(x1,y1,2)/pi)*180; % psi1 en degrés
theta1d = (image_ihs(x1,y1,3)/pi)*180; % theta1 en degrés
disp(['Angle azimutal de la teinte à remplacer : ',num2str(psi1d),' degrés']);
disp(['Angle de site de la teinte à remplacer : ',num2str(theta1d),' degrés']);
```

```
%
% Calcul de v11 , v12 , v13 , v1
v11 = V1(x1,y1);
v12 = V2(x1,y1);
v13 = V3(x1,y1);
v1 = [v11;v12;v13];
%
disp(' ');
%
close(him);
%% Couleur de remplacement
% on lit l'image qui contient la couleur de remplacement
%
disp('Lire l'image contenant la couleur de remplacement');
lecimage;
% L'image de remplacement est im
%
disp('Désigner une couleur de remplacement');
[y2,x2] = ginput (1);
x2 = round(x2);
y2 = round(y2);
disp(['x2 = ', num2str(x2),' y2= ',num2str(y2)]);
%
% Codage IH1H2S de im(x2,y2).
% Attention im est l'image de remplacement pas l'image de depart im_depart!
%
% Prétraitement : Passage de RGB à R*G*B*W
% Pour cela, on retire tout le blanc possible de RGB et on le met dans W
%
r = im(x2,y2,1);
g = im(x2,y2,2);
b = im(x2,y2,3);
minrgb = min(r,g);
minrgb = min(minrgb,b);
%
rstar = r - minrgb;
gstar = g - minrgb;
bstar = b - minrgb;
w = minrgb;
%
result = M*[rstar;gstar;bstar;w];
%
% Calcul de v21 , v22 , v23 , v2
v21 = result(2);
v22 = result(3);
v23 = result(4);
v2 = [v21;v22;v23];
v2 = v2/norm(v2);
%
h1 = atan2(v23,v22);
s = sqrt(v21^2 + v22^2 + v23^2);
h2 = asin(v21/s);
%
psi2d = (h1/pi)*180; % psi2 en degres
theta2d = (h2/pi)*180; % theta2 en degres
```

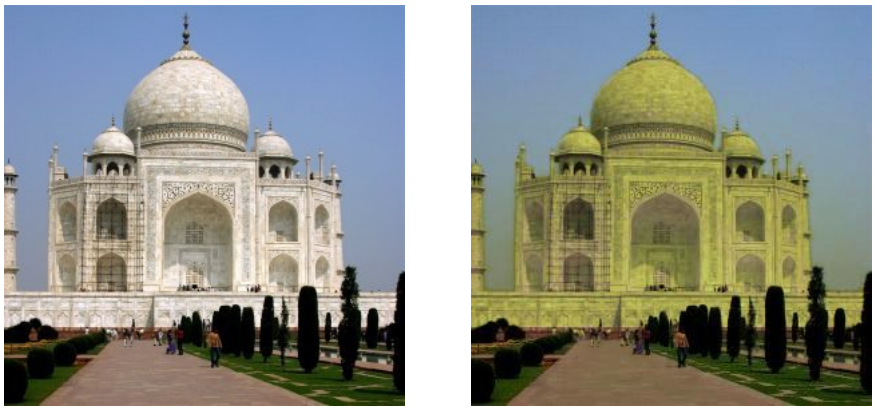
```
disp(['Angle azimutal de la teinte de remplacement : ',num2str(psi2d),' degrés']);
disp(['Angle de site de la teinte de remplacement : ',num2str(theta2d),' degrés']);
disp(' ');
%
%% Lecture du seuil de remplacement en degrés
seuild = input('Seuil de remplacement en degrés (0<= seuil <= 180) ? ','s');
seuil = (str2double(seuild)/180)*pi; % seuil en radians
%
%% Calcul de la matrice de rotation R
%
% Calcul de l'axe de rotation
%
% Produit vectoriel pour trouver l'axe
u = cross(v1,v2); % v1 et v2 sont unitaires
%
% Calcul de l'angle de rotation theta
%
sintheta0 = norm(u); % sintheta > 0 !
%
% Produit scalaire pour trouver costheta
costheta0 = sum(v1.*v2);
%
theta0 = atan2(sintheta0,costheta0); % theta0 est l'angle de la rotation
theta0d = (theta0/pi)*180; %theta0 en degrés
%
% Calcul de R
R = rotationmat3D(theta0,u);
%
%% Rotation des couleurs dans im_depart
nlig = size(im_depart,1);
ncol = size(im_depart,2);
%
I = image_ihs( :, :,1);
S = image_ihs( :, :,4);
%
V = zeros(size(im_depart));
V( :, :,1) = S .* V1;
V( :, :,2)= S .* V2;
V( :, :,3) = S .* V3;
%
% Rotation
for nolig=1 :nlig
   for nocol=1 :ncol
      V(nolig,nocol, :) = R * squeeze(V(nolig,nocol, :));
   end
end
%
%
image_ihs = cat(3,I,V( :, :,1),V( :, :,2),V( :, :,3));
%
%% Décodage IH1H2S vers RGBW de l'image transformée
image_rgb = zeros(nlig,ncol,4); % en fait c'est image_rgbw !
%
for nolig=1 :nlig
   for nocol=1 :ncol
```

```
        image_rgb(nolig,nocol, :) = M \ squeeze(image_ihs(nolig,nocol, :)) ;
    end
end
%% Décodage RGBW vers RGB de l'image transformée
% Posttraitement : on remet le blanc dans R G et B
%
for k = 1 :3
    image_rgb( :, :,k) = image_rgb( :, :,k) + image_rgb( :, :,4) ;
end
%
image_rgb = image_rgb( :, :,1 :3) ; %image_rgb est l'image ayant subi une rotation colorimétrique GLOBALE d'angle theta0
%
%% Détermination des angles de rotation des teintes
THETA = zeros(nlig,ncol) ;
for nolig=1 :nlig
    for nocol = 1 :ncol
        v2 = [V1(nolig,nocol) ;V2(nolig,nocol) ;V3(nolig,nocol)] ;
        u = cross(v1,v2) ;
        sintheta = norm(u) ;
        costheta = sum(v1.*v2) ;
        THETA(nolig,nocol) = atan2(sintheta,costheta) ;% THETA(nolig,nocol) est l'angle entre la teinte de couleur à remplacer et celle de la
        % couleur du pixel (nolig,nocol)
    end
end
%% Filtrage de l'image par un filtre sigmoïde sigmf
% les couleurs dont les teintes sont proches de la teinte a
% remplacer seront remplacées par une combinaison lineaire de la teinte à remplacer et de la teinte de remplacement.
image_rgb_filtre = zeros(size(im_depart)) ;
a = 200*seuil ;
FILTRE = sigmf(THETA,[a seuil]) ;
for k=1 :3
    image_rgb_filtre( :, :,k) = FILTRE.*im_depart( :, :,k) + (1-FILTRE).*image_rgb( :, :,k) ; %image_rgb est maintenant l'image filtree
end
% Projection sur l'orthant positif ou nul
image_rgb_filtre = (image_rgb_filtre >= 0).*image_rgb_filtre ;
%
%% Affichage
him = figure ('BackingStore','off','Name',['Rotation de teinte de ( ',num2str(theta0d),' ) degres'],'Units','pixels') ;
imshow(image_rgb_filtre/255)
```

19.4 Quand la France devient espagnole

Notre objectif est de trouver une suite de transformations élémentaires permettant de passer de l'image du drapeau français à celle du drapeau espagnol.

Solution fr2es.m



(a) Image originale (b) Image transformée

FIGURE 3 – Rotation des couleurs

(a) Image originale (b) Image transformée

FIGURE 4 – Rotations géométrique et colorimétrique

20 Etude de la transformée de Fourier

20.1 Normalisation de l'image

Normaliser l'image im (notée **imn**) de telle sorte que la moyenne des intensités soit nulle et que son énergie soit égale à l'unité.
Solution : normalise.m

20.2 Calcul de la transformée de Fourier normalisée de im

Si
imf = fft2(im) ;
alors

$$imf(u+1,v+1) = \sum_{k=0}^{nlig-1} \sum_{l=0}^{ncol-1} im(k+1,l+1) W_l^{-uk} W_c^{-vl} \quad u = 0 \dots nlig-1 \quad v = 0 \dots ncol-1$$

$W_l = e^{\frac{2i\pi}{nlig}}$ $W_c = e^{\frac{2i\pi}{ncol}}$ [21]

21. on remarquera que la fonction de MATLAB fft2 n'est pas normalisée : il n'y a pas conservation de l'énergie

Calculer la transformée de Fourier normalisée de l'image *normalisée* imn (notée imfn) [22] ; imfn est donc une **représentation orthogonale** de imn
Solution : fft2n.m

Que doit valoir imfn(1,1) ?

Vérifier la conservation de l'énergie par fft2n :

Solution : conservation.m

20.3 Vérification de la relation de conjugaison

La relation à vérifier est donc :

$$IM(1-f_x, 1-f_y) = IM^*(f_x, f_y) \quad \forall f_x, f_y$$

où IM est la matrice de Fourier indicée par les fréquences spatiales en x (lignes) et en y (colonnes).
Pour vérifier la relation de conjugaison, vous établirez les relations existant entre f_x et u d'une part, f_y et v d'autre part, puis vous exprimerez la relation de conjugaison sur imf (ou sur imfn).

Solution : vconjg.m

```
% Verification de la relation de conjugaison
%
if all(abs(imfn(1,2 :1+ncol/2)-conj(imfn(1,ncol :-1 :1+ncol/2))) == zeros(1,ncol/2))
   disp('Relation de conjugaison verifiee sur (1,2 :1+ncol/2)') [23]
else
   disp('ATTENTION Relation de conjugaison NON verifiee sur (1,2 :1+ncol/2)')
end
%
if '??'
   disp('Relation de conjugaison verifiee sur (2 :1+nlig/2,1)')
else
   disp('ATTENTION Relation de conjugaison NON verifiee sur (2 :1+nlig/2,1)')
end
%
if '??'
   disp('Relation de conjugaison verifiee sur (2 :1+nlig/2,2 :1+ncol/2)')
else
   disp('ATTENTION Relation de conjugaison NON verifiee sur (2 :1+nlig/2,2 :1+ncol/2)')
end
```

20.4 Affichage du module de la transformée de Fourier de l'image normalisée imn

```
imfa = abs(imfn) ;
```

Solution : afficher_imsc0.m

20.5 Déplacer l'origine de la transformée de Fourier au centre de l'image

Revenons à l'image initiale im et à sa transformée de Fourier non normalisée imf.
Pour centrer cette transformée de Fourier (le résultat est **imfc**) , on utilise la fonction fftshift. Afficher le module

22. la transformée de Fourier normalisée de im décompose l'image im sur les images de base des "exponentielles complexes" lesquelles forment une base **orthonormée** de $C^{nlig*ncol}$
23. Pour afficher du texte en couleur dans la fenêtre de commande (Command Window) on utilisera la fonction **cprintf** fournie (et non pas disp)

de imfc.
Calculer ensuite la transformée de Fourier inverse de imfc et vérifier que l'on obtient une nouvelle image $\tilde{im}$.

$$\tilde{im}(x,y) = (-1)^{x+y} im(x,y)$$

Démontrer cette dernière relation.

Solution : centrefourier.m

21 Compression de l'image par transformée de Fourier

On testera deux méthodes :

1. Annulation des coefficients de hautes fréquences
2. Annulation des coefficients de plus faibles poids

21.1 Annulation des coefficients de hautes fréquences

Pour faire cela, on reprend l'image centrée **imfc** et on la multiplie par un masque *mask* centré de taille (*hauteur,largeur*). Les coefficients du masque valent 1 à l'intérieur du masque et 0 à l'extérieur.

1. Construire le masque[24] *mask*
2. Comprimer l'image et l'afficher (début de solution : imfc0 = mask.*imfc ;)
3. Faire varier la taille du masque

Solution : comp0.m

21.2 Annulation des coefficients de plus faibles poids

Etant donnée une erreur quadratique eps, on cherche le plus petit nombre de coefficients de Fourier non nuls qui fournissent une erreur inférieure à eps. L'idée est de mettre à zéro les coefficients de Fourier qui sont les plus faibles. Il est donc nécessaire de les trier.

Solution : comp_opt.m

21.3 Comparaison des deux méthodes de compression

Il s'agit de savoir quelle est la meilleure méthode de compression. Pour cela on doit comparer les deux images comprimées imc0 et imc1 lorsqu'on met à 0 le même nombre de coefficients de fourier.

Solution : comparaisoncompressions.m

```
% Comparaisons compressions 0 et opt
%
%% Compression par annulation des coefficients de Fourier de plus faibles poids (compression opt)
%
imfc1 = imf;
energie = norm(imf,'fro')^2; % somme des modules au carre des coefficients de Fourier non normalises
pourcentage = str2double(input('pourcentage de conservation de l'énergie de l'image :,'s'));
seuilerreur = (1 - pourcentage/100)*energie;
%
[imftri,index] = sort(abs(imf( :))); % on trie les modules des coefficients par ordre croissant
%
% attention sort(imf( :)) est erronné car on trie alors selon la partie
% réelle (cf. doc)
```

24. On remarquera que la matrice *mask* est de rang 1 ; donc $mask = c' * l$ où c est un vecteur ligne et l un vecteur colonne ; que valent c et l ?

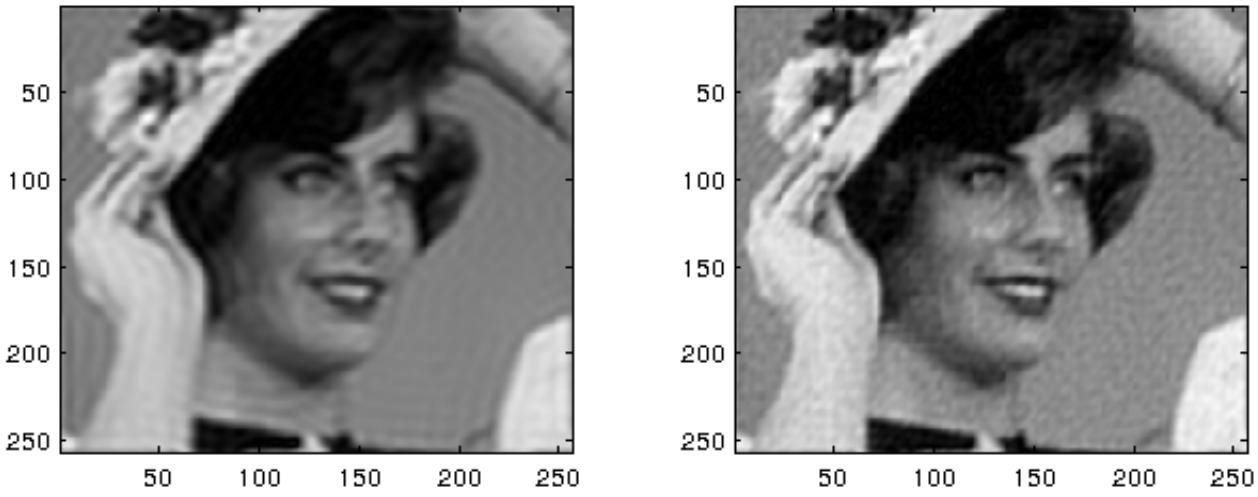


FIGURE 5 – Comparaison des deux méthodes de compression

```
%
erreur2 = 0;
i = 1;
erreur2 = erreur2 + abs(imftri(i))^2;
while erreur2 <= seuilerreur
    k=rem(index(i)-1,nlig)+1;
    l=floor((index(i)-1)/ncol)+1;
    imfc1(k,l)=0;
    i=i+1;
    erreur2=erreur2 + abs(imftri(i))^2;
end
nombrea0=i-1; % nombre de coefficients mis a 0
cprintf('blue',['Nombre de coefficients mis à 0 : ',num2str(nombrea0)]);
disp(' ');
pourcent0 = 100*nombrea0/(nlig*ncol);
cprintf('blue',['Pourcentage de coefficients mis à 0 : ',num2str(pourcent0)]);
disp(' ');
%
imc1 = ifft2(imfc1);
imc1 = abs(imc1); % imc1 est l'image comprimée par annulation des coefficients de plus faibles poids
%
%% Compression par annulation des coefficients de hautes fréquences spatiales (compression 0)
% Construction du masque
hauteur = round(sqrt(nlig*ncol - nombrea0));
largeur = hauteur;
c = '??'
l = '??'
mask = c'*l;
% Visualisation du masque
xori = (9*ncol)/4;
yori = 0;
hmask = figure ('BackingStore'[25], 'on' ,'Color','k','Colormap',gray(256),...
```

25. Le mot "BackingStore" désigne un buffer utilisé par MATLAB pour stocker une copie du contenu de la fenêtre d'une figure. Par défaut, Backingstore est on ; lorsque la mémoire de votre système est limitée, il faut établir BackingStore à off pour récupérer

```
'Name','Masque','Units','pixels',...
'Position', [xori , yori , ncol , nlig ]);
imagesc(mask);
%
% Filtrage par le masque mask : les coefficients en dehors du masque sont
% mis a 0.
%
imfc0=imfc.*mask; % imfc est la transformee de Fourier centree
%
imc0 = ifft2(imfc0);
imc0 = abs(imc0); % imc0 est l'image comprimee par annulation des coefficients de hautes frequences spatiales

%
%% Affichage des deux images comprimees au meme taux (meme nombre de coéfficients de Fourier mis a 0)
xori = 0;
yori=0;
h2 = figure ('BackingStore','on' ,'Color','k','Colormap',gray(256),...
'Name',['imc0 et imc1 nombre de 0 : ',num2str(nombrea0)],'Units','pixels',...
'Position', [xori , yori , round(2.8*ncol) , nlig ]);
subplot(1,2,1);
imagesc(imc0);
subplot(1,2,2);
imagesc(imc1);
```

22 Calculer et afficher quelques images de base réelles pour différents couples fréquentiels

On supposera pour simplifier que le nombre de lignes de l'image est égal au nombre de colonnes : nlig = ncol = n. Les images de base de la représentation de Fourier sont réelles lorsque l'image est réelle. Elles sont formées des images des fonctions sinus et cosinus [26] paramétrées par les fréquences spatiales :

$f_x = 0, \frac{1}{n}, \frac{2}{n}, \ldots, \frac{1}{2}$

$f_y = 0, \frac{1}{n}, \frac{2}{n}, \ldots, 1 - \frac{1}{n}$

ce qui fait bien 2 (sinus + cosinus) $* \frac{n}{2}(f_x) * n(f_y) = n^2$ images de base pour une image (n,n).
Pour calculer ces images de base, le plus simple est de commencer par calculer leurs transformées de Fourier.

Solution : basereellefourier.m

```
% Calcul et affichage d'une image de base reelle de la transformee de
% Fourier d'une image reelle
% on supose que nlig == ncol!!
%
rep='t';
n = nlig;
while rep  = 's' & rep  = 'c'
```

la mémoire utilisée par ce buffer. De même, lorsqu'on cherche à rafraîchir l'écran rapidement, on doit établir BackingStore à off pour éviter de réécrire à la fois dans le buffer et dans la fenêtre de la figure

26. on peut donc dire que la transformée de Fourier décompose une image réelle en ses composantes en sinus et en cosinus ou mieux encore en une somme de composantes en cosinus déphasées les unes par rapport aux autres. On a en effet :

$$\frac{n}{2} im(x,y) = \frac{1}{2} IM(0,0) + \sum_{\substack{f_x, f_y = 0 \\ (f_x,f_y) \neq (0,0)}}^{\frac{1}{2}, 1-\frac{1}{n}} |IM(f_x,f_y)| \cos(2\pi(f_x x + f_y y) + \phi(f_x,fy)) \qquad \text{où} \quad IM(f_x,f_y) = |IM(f_x,f_y)| e^{i\phi(f_x,fy)}$$

Par leurs phases $\phi(f_x, fy)$ ces images de base dépendent de l'image im elle-même (ce qui ne sera pas le cas pour la représentation en cosinus)

```
    rep = input ('Desirez-vous afficher un sinus ou un cosinus (s/c) ?', 's');
end
u = str2double(input ('n * frequence en x ? ','s'));
v = str2double(input ('n * frequence en y ? ','s'));
%
% mise a zero de imf
basef = zeros(n,n);
%
if rep == 'c'
    if u  = 0 && v  = 0
        basef(u+1,v+1)= '? ?';
        basef(n+1-u,n+1-v)= '? ?';
    elseif u == 0
        basef(1,v+1)= '? ?';
        basef(1, n+1-v)= '? ?';
    else
        basef(u+1,1)= '? ?';
        basef(n+1-u,1)= '? ?';
    end
else
    %rep == 's'
    if u  = 0 && v  = 0
        basef(u+1,v+1)= '? ?';
        basef(n+1-u,n+1-v)= '? ?';
    elseif u == 0
        basef(1,v+1)= '? ?';
        basef(1, n+1-v)= '? ?';
    else
        basef(u+1,1)= '? ?';
        basef(n+1-u,1)= '? ?';
    end
end
%
base=ifft2(basef)*n;
base = 10000 * real(base);
base = base -min(base( :));
%
map = exp(1.*gray(63))-1;
map = map/max(max(map));
hbase = figure('BackingStore','on','Colormap', map, 'Units', 'pixels');
imagesc(base);
axis('square');
% Tracer un profil ligne de base
profimfunct(base);
```

23 Affichage de résultats

Solution : affichage1.m

```
%% Affichage des principaux résultats de l'étude de la transformée de Fourier
%
xori = 0;
yori= 0;
h1 = figure ('BackingStore','on' ,'Color','k', 'Colormap',gray(256),...
```

```
'Name','Resultats sur l'etude de la Transformee de Fourier','Units','pixels',...
'Position', [xori , yori , 2*ncol , 2*nlig ]);
%
a(1) = subplot(3,3,1);
image(ind2rgb(im,jet(256)))27;
%
a(2) = subplot(3,3,2);
imfa = abs(imf);
ecart_type=std(std(imfa));
imfa = round((imfa/ecart_type)*256);
image(imfa);
%
a(3) = subplot(3,3,3);
imfa = abs(imfc);
ecart_type=std(std(imfa));
imfa = round((imfa/ecart_type)*256);
image(imfa);
%
a(4) = subplot(3,3,4);
imagesc(mask);
%
a(5) = subplot(3,3,5);
image(ind2rgb(round(imc0),jet(256))) [27];
%
a(6)= subplot(3,3,6);
image(ind2rgb(round(imc1),jet(256))) 27;
%
sincos3; % calcule sin30 , cos30 et cosphi30 [28]
%
a(7) = subplot(3,3,7);
imagesc(sin30);
%
a(8) = subplot(3,3,8);
imagesc(cos30);
%
a(9) = subplot(3,3,9);
imagesc(cosphi30);
%
ht1 = title(a(1),'image initiale');
set(ht1,'Color','white');
ht2 = title(a(2),'TF');
set(ht2,'Color','white');
ht3 = title(a(3),'TF centrée');
set(ht3,'Color','white');
ht4 = title(a(4),'masque');
set(ht4,'Color','white');
ht5 = title(a(5),'compression par masque');
set(ht5,'Color','white');
ht6 = title(a(6),'compression optimale');
```

27. Afin de mieux comparer visuellement les résultats des deux compressions, on affiche les images monochromes en pseudo-couleurs plutôt qu'en niveaux de gris ; pour cela, on utilise la table de couleur jet(256) et on convertit l'image indexée im en une image non-indexée RGB au moyen de la fonction ind2rgb. L'affichage se fait alors en RGB et tout se passe comme si on affichait alors une image indexée par la table jet(256) alors qu'elle était préalablement indexée par la table de la figure où elle s'inscrivait à savoir gray(256)

28. sin30 ,cos30 et cosphi30 sont respectivement les images de base en sinus, cosinus et cosinus phasés (cf note 26) calculées aux fréquences $f_x = \frac{3}{n}$ et $f_y = 0$

```
set(ht6,'Color','white');
ht7 = title(a(7),'sin(3,0)');
set(ht7,'Color','white');
ht8 = title(a(8),'cos(3,0)');
set(ht8,'Color','white');
ht9 = title(a(9),'cosinus phase (3,0)');
set(ht9,'Color','white');
%
```

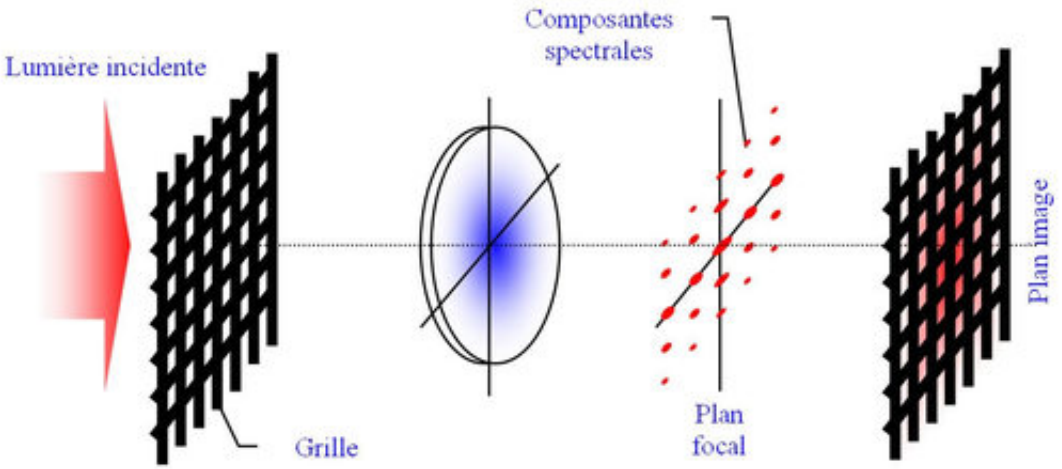


FIGURE 6 – L'expérience d'Abbe et Porter (issu du site *www.optique-ingenieur.org*)

24 Simulation numérique des expériences d'Abbe et Porter

La transformée de Fourier (continue) joue un grand rôle en optique [29].Ceci a été prouvé de façon éclatante par les expériences réalisées par Abbe en 1893 et plus tard par Porter en 1906. Nous nous proposons à présent de réaliser une simulation numérique de ces expériences. Celles-ci consistent à éclairer à l'aide d'une source de lumière cohérente une grille métallique. Dans le plan focal d'une lentille apparaît alors la transformée de Fourier de cette grille et finalement dans le plan image, les différentes composantes de Fourier transmises par la lentille se recombinent pour former une réplique de cette grille. En plaçant divers obstacles (par exemple un diaphragme à iris ou une une fente) dans le plan focal, on modifie la transformée de Fourier (et donc l'image) de la grille de différentes façons.

La similation numérique de cette expérience va consister à programmer les étapes suivantes :

1. Lecture du fichier grillec.tri
2. Calcul de la transformée de Fourier centrée
3. Construction d'une fente horizontale de hauteur 10 pixels et de largeur 250 pixels (par exemple)
4. Filtrage par la fente horizontale : les coefficients de Fourier en dehors de la fente sont mis à 0
5. Calcul de l'image filtrée
6. Rotation de 90° de la fente pour la rendre verticale
7. Filtrage par la fente verticale
8. Calcul de l'image filtrée
9. Affichage des résultats de l'expérience dans une même figure avec de haut en bas et de gauche à droite :
   (a) Affichage de l'image de la grille
   (b) Affichage des intensités (amplitudes au carré) des coefficients de Fourier
   (c) Affichage de l'image filtrée par la fente horizontale
   (d) Affichage du spectre de la grille filtrée par la fente horizontale
   (e) Affichage de l'image filtrée par la fente verticale
   (f) Affichage du spectre de la grille filtrée par la fente verticale

Solution : AbbePorter.m

29. La prise de conscience de l'utilité de la transformée de Fourier dans l'analyse des systèmes optiques est due en grande partie à P.M. Duffieux dont les travaux de recherche aboutirent en 1946 à la publication de l'ouvrage : "L'intégrale de Fourier et ses applications à l'Optique", Faculté des Sciences, Besançon.

sectionEtude de la compression JPEG

24.1 Diagrammes de compression et de décompression

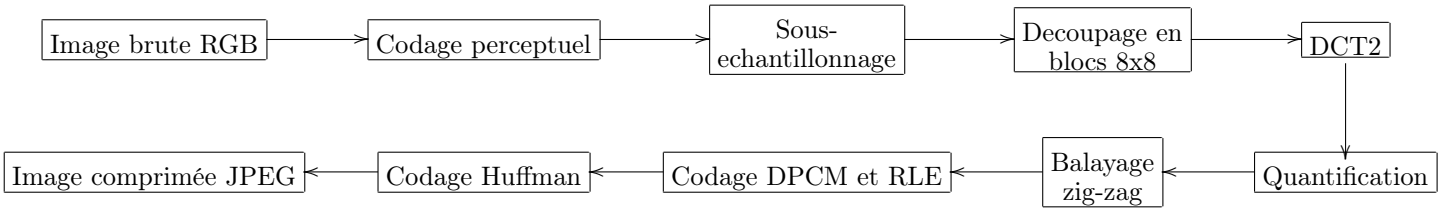


FIGURE 7 – Compression JPEG

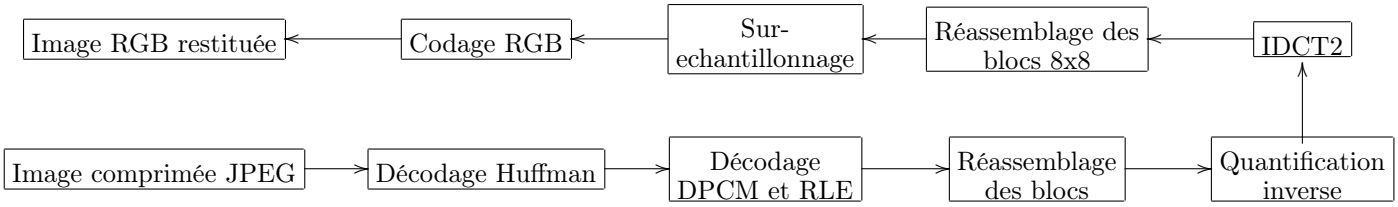


FIGURE 8 – Décompression JPEG

24.2 La fonction jpeg_read

Nous commençons par décoder partiellement le contenu d'une *image monochrome comprimée JPEG* (telle que Lena.jpg[30]) Pour ce faire nous allons utiliser le fichier MEX **jpeg_read.mexa64**[31]. Le fichier jpeg_read.mexa64 contient la fonction **jpeg_read** écrite en C, laquelle réalise les décodages Huffman[32] et Run-Length du train de bits et le réarrangement des coefficients DCT après quantification en une matrice de blocs 8x8.

```
jobj = jpeg_read('Images/Lena.jpg') ;
```

Le résultat se trouve ici dans la structure jobj qui contient 14 champs dont les deux plus importants sont :

1. coef_arrays : cellule contenant la matrice composée des coefficients DCT des blocs 8x8
2. quant_table : matrice 8x8 des coefficients utilisés pour la quantification

24.3 Décompression grossière de l'image JPEG

Notre premier objectif est de décomprimer l'image JPEG le plus simplement possible sans utiliser la transformée en cosinus inverse, ni la table de quantification. Pour ce faire nous allons seulement utiliser la composante continue de chaque bloc qui n'est autre que le coefficient DCT pour un couple fréquentiel nul (situé en haut et à gauche de chaque bloc)
Solution : decode0

24.4 Décompression de l'image JPEG sans utiliser la table de quantification

La décompression se fera en calculant les transformées en cosinus inverse de chaque bloc 8*8
Solution : decode1

30. Lena est un morceau de photo d'une playmate prise dans le numéro de novembre (miss novembre) 1972 du magazine Playboy. Elle sert d'image de test pour les algorithmes de traitement d'image et est devenue de facto un standard industriel et scientifique.
31. Les fichiers .mex sont des fichiers écrits en C, C++ ou en Fortran, qui une fois compilés, peuvent être utilisés dans MATLAB de la même manière que les fichiers .m A l'inverse, ils permettent aussi d'appeler directement des fonctions MATLAB dans du code C, C++ ou Fortran.
32. le code d'Huffman est un code entropique qui permet d'obtenir une longueur moyenne de codage de chaque symbole qui soit proche de l'entropie laquelle représente la longueur minimale de codage possible

24.5 Décompression de l'image JPEG en utilisant la table de quantification

Cette fois, on multiplie chaque coefficient DCT par son coefficient de quantification correspondant.
Solution : decode2

24.6 Comparaison des résultats

Comparer visuellement les trois images décomprimées et les classer par ordre de qualité visuelle décroissante. Pour cela, on affichera les trois images côte à côte dans une même figure.
Solution : affichage2.m

25 Récapitulatif des principaux identificateurs des variables du Workspace

— im : image initiale
— imn : image normalisée (moyenne des intensités nulle et énergie égale à 1)
— imf : transformée de Fourier, non normalisée, de im
— imfn : transformée de Fourier, normalisée de imn
— imfc : transformée de Fourier centrée et non normalisée de im
— imfa : module de imfn ou de imfc
— imtilde : transformée de Fourier inverse (non normalisée) de imfc
— imc0 : image Fourier-comprimée par annulation des coefficients de hautes fréquences spatiales
— imc1 : image Fourier-comprimée par annulation des coefficients de plus faibles poids
— base : image de base réelle (sinus ou cosinus) de la représentation de Fourier d'une image réelle
— jobj : structure d'une image JPEG
— im0 : image JPEG-décodée par decode0
— im1 : image JPEG-décodée par decode1
— im2 : image JPEG-décodée par decode2