Cours de traitement d'images - Corrigé Labo 1:

Familiarisation avec Matlab
Manipulations de base sur les niveaux de gris
Images calculées
Introduction a la Transformée de Fourier

7 Novembre 2005

1. ***Quantification d'une image***
   L'image "Lena" en niveaux de gris peut être obtenue

   (a) soit en la lisant directement à partir du disque

   ```
   >> X = imread('lena.tif');
   ```

   (b) soit à partir de l'image `lena_color.tif`.

   ```
   >> Y = imread('lena_color.tif');
   >> X = rgb2gray(Y);
   ```

   Pour visualiser l'image avec n niveaux de gris, il existe de nombreuses variantes.

   (a) Utiliser les fonctionalités de `imshow` :

   ```
   >> figure(1)                    % Sélectionner la figure 1
   >> imshow(X);                   % Afficher lena avec 256 niveaux de gris
   >> figure(2)                    % Sélectionner la figure 2
   >> n=128;                       % Choisir le nombre de niveaux de gris
   >> imshow(X,n)                  % Afficher l'image à n niveaux
   ```

   (b) Quantification "manuelle" :

   ```
   >> figure(1)                      % Sélectionner la figure 1
   >> imshow(X);                     % Afficher lena avec 256 niveaux de gris
   >> figure(2)                      % Sélectionner la figure 2
   >> n=128;                         % Choisir le nombre de niveaux de gris
   >> d=256/n;                       % Déterminer le pas de quantification
   >> W=uint8(floor(double(X)/d)*d); % Quantifier. On notera que on ne peut
                                       multiplier ou diviser un uint8, et
                                       qu'il faut donc passer par une conversion
                                     au format double
   >> imshow(W)              % Afficher l'image à n niveaux
   ```

(c) Quantification "manuelle" au format `double` :

```
>> I=double(X)/255;          % Convertir l'image en double
>> figure(1)                     % Sélectionner la figure 1
>> imshow(I);                    % Afficher lena avec 256 niveaux de gris
>> figure(2)                     % Sélectionner la figure 2
>> n=128;                        % Choisir le nombre de niveaux de gris
>> d=256/n;                      % Déterminer le pas de quantification
>> imshow(floor(I*255/d)/(n-1)) % Afficher l'image à n niveaux
```

La dernière ligne applique la quantification (`floor(I*255/d)`), ce qui resulte dans des valeurs entières dans l'intervalle $[0, n-1]$ qu'on normalise ensuite à l'intervalle $[0, 1]$.

(d) Quantification en utilisant la fonction `gray2ind` :

```
>> I=double(X)/255;          % Convertir l'image en double
>> figure(1)                     % Sélectionner la figure 1
>> imshow(I);                    % Afficher lena avec 256 niveaux de gris
>> figure(2)                     % Sélectionner la figure 2
>> n=128;                        % Choisir le nombre de niveaux de gris
>> [X,map]=gray2ind(I,n);        % Convertir en indexée avec n niveaux
>> imshow(X,map)                 % Afficher l'image à n niveaux
```

L'apparition de faux contours est visible à partir de 16 niveaux de gris environ. Bien entendu, ce seuil dépend de la qualité du système d'affichage, de la lumière ambiante, etc.

2. ***Loi de Weber***

(a) *Créer une fonction `weber`.*

```
function img=weber(L1, L2, Lb)
%
% WEBER cr\'eation d'une image pour tester la loi de Weber
%   WEBER cr\'ee une image de taille 600x600 dont le fond a une
%   intensit\'e Lb, avec au centre un carr\'e de taille 160x160 dont
%   la moiti\'e gauche a une intensit\'e L1 et la moiti\'e droite une
%   intensit\'e L2.
%

img = ones(600,600)*Lb;
img(221:380,221:300)=L1;
img(221:380,301:380)=L2;
img=uint8(img);
```

(b) *Déterminer votre constante de Weber.*

Sur la figure 1, on a représenté différentes valeurs de α en fonction de l'intensité L_1. Ces valeurs reflètent l'acuité d'un individu à différencier des niveaux de gris, dans des circonstances particulières. Les différents facteurs qui influencent ce résultat sont : l'acuité visuelle du testeur, la période de la journée durant laquelle les mesures ont été prises, la configuration du moniteur (contraste, luminosité,...), la lumière ambiante, etc..

Comme on le voit, le rapport $\Delta L/L_1$ n'est constant que sur une certaine gamme d'intensités. Pour les faibles intensités, la relation entre α et L_1 n'est plus linéaire : le système visuel humain a plus de peine à distinguer des niveaux de gris de faible intensité. Un phénomène analogue peut être observé dans les tons clairs, mais n'apparaît pas dans la figure 1, car la luminosité de l'écran était relativement faible lors de la mesure.

Dans cet exemple, la constante de Weber vaut approximativement 1.5% lorsque $L_b = 10$, et 1.2% lorsque $L_b = 200$.

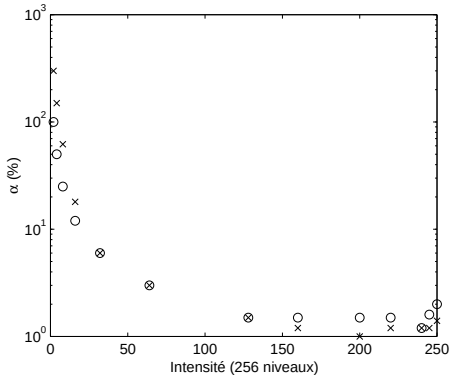

FIG. 1 – Différences à peine perceptibles d'intensité ΔL, en fonction de L_1. 'o' : $L_b = 10$, 'x' : $L_b = 200$.

3. ***Re-échantillonnage d'une image***

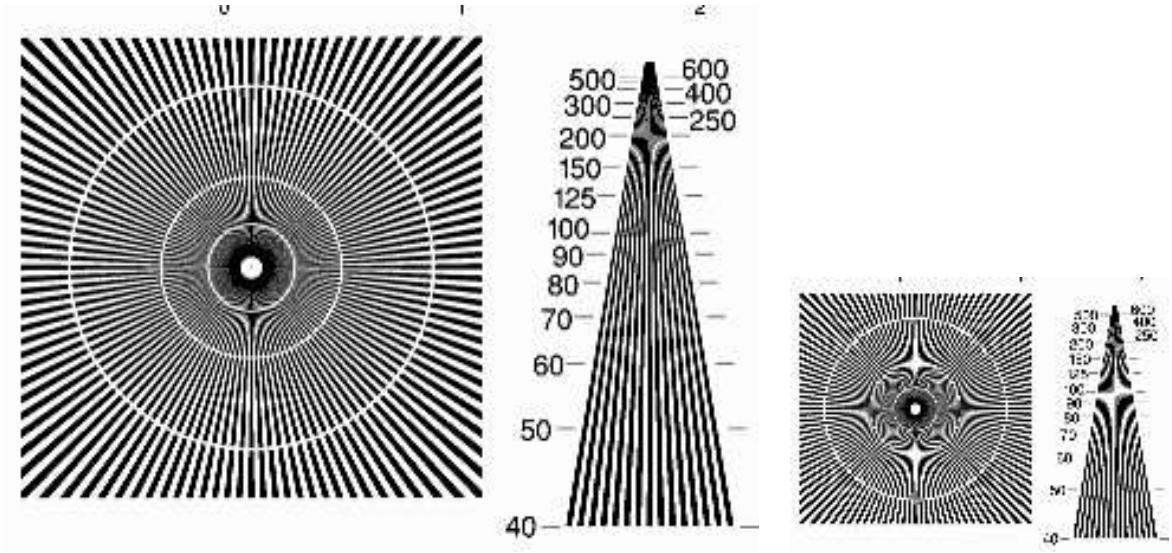

FIG. 2 – Images sous-échantillonées par un facteur 2 (à gauche) et 4 (à droite)

Deux solutions sont possibles :

(a) *Charger l'image*

```
sub4=imread('sub4.tif');
```

(b) *Ecrire une fonction dans* `subsample.m` *:*

```
function Y=subsample(X)
% SUBSAMPLE sous-échantillonnage d'une image par 2
%   SUBSAMPLE sous-échantillonne une image par un facteur 2 dans
```

```
%   chaque direction.
%
%   I2=SUBSAMPLE(I) retourne dans I2 la version sous-échantillonnée
%   de I. Les colonnes et les lignes sont sous-échantillonnées par 2.
%
%   Cette fonction utilise une double boucle 'for'

  s=size(X);
  height=s(1);  % nombre de lignes de l'image
  width=s(2);   % nombre de colonnes de l'image
  % Creer une matrice de zeros de taille moitié dans chaque direction
  Y=zeros(round(height/2),round(width/2));
  % Remplir Y avec les données de X en prenant un pixel sur deux
  for i=1:round(height/2)
   for j=1:round(width/2)
     Y(i,j)=X(2*i-1,2*j-1);
   end
  end
```

Puis l'utiliser de la façon suivante :

```
>> Y=imread('sub4.tif');
>> Y2=subsample(Y);
>> image(Y2)
>> colormap(gray)
>> truesize
```

On peut également utiliser les fonctionnalités de Matlab (beaucoup plus rapide à écrire et surtout à exécuter) :

```
function Y=subsample2(X)
% SUBSAMPLE2 sous-échantillonnage d'une image par 2 (optimisé)
%   SUBSAMPLE2 sous-échantillonne une image par un facteur 2 dans
%   chaque direction.
%
%   I2=SUBSAMPLE2(I) retourne dans I2 la version sous-échantillonnée
%   de I. Les colonnes et les lignes sont sous-échantillonnées par 2.
%
%   Cette fonction utilise le sous-indexage Matlab
  s=size(X);
  Y=X(1:2:s(1),1:2:s(2));
```

Cette fonction s'utilise exactement de la même façon que la précédente.

Pour obtenir l'image sous-échantillonnée par 4 :

```
>> Y4=subsample(Y2);
>> image(Y4)
>> colormap(gray)
>> truesize
```

(c) *Afficher l'image.*

Voir dessus.

(d) *Effets du sous-échantillonnage :*

L'image originale est dégradée par le sous-échantillonnage (cf. figure 2). On note l'apparition de nouvelles figures géométriques, appelées "Figures de Moiré", notamment dans les zones contenant des fréquences spatiales élevées. En effet, aprés l'opération de sous-échantillonnage, le critère de Nyquist n'est plus satisfait. Il y a alors "repliement" du spectre de l'image, ce qui provoque l'apparition de figures de Moiré.

4. ***Transformée de Fourier discrète bidimensionelle (FFT)***

Le chargement et le calcul de la FFT et IFFT de `res_chart.tif` se font de la manière suivante :

```
>> x = imread('res_chart.tif');
>> figure(1); imshow(x);
>> y = fft2(x);
>> x2 = ifft2(y)
>> figure(2); imshow(real(x2),[0,255]);
```

Théoriquement, on s'attend a retrouver exactement l'image originale. En pratique, on observe deux différences : premièrement, alors que `x` était originalement codé en `uint8` (entier non signé sur 8 bits), elle est evidemment transformée en nombre réel (`double`) avant la transformée de Fourier, et de même pour le résultat `x2`. Une conséquence de ce changement de format est que l'on ne peut utiliser les paramètres par défaut de `imshow` mais que l'on doit lui spécifier la game de valeurs que peut prendre `x2`, soit `[0,255]`. Deuxièmement, alors que le résultat de la transformée inverse devrait théoriquement être réel, on observe en pratique une partie imaginaire non nulle suite à des erreurs d'arrondis. C'est pourquoi on affiche uniquement sa partie réelle `real(x2)`.

Pour les images `mit.tif` et `camman.tif`, on charge, affiche, transforme et affiche la FFT de la manière suivante :

```
>> x = imread('mit.tif');
>> figure(1); imshow(x);
>> y=fft2(x);
>> ymod = abs(fftshift(y));
>> figure(2); imshow(log(ymod),[]);
```

Pour afficher la transformée de Fourier (complexe !) de l'image. Il est intéressant de représenter la norme de la transformée de Fourier, qui représente l'amplitude des fréquences spatiales de l'image. La fonction `fftshift` permet de recentrer y pour visualiser l'amplitude du contenu fréquentiel avec les basses fréquences au centre de l'image (domaine principal). Etant donné la grande variabilité des amplitudes possible, il est plus informatif de les afficher en échelle logarithmique, i.e. `log(ymod)`. Enfin, comme on ne connaît pas a priori la gamme des valeur que peut prendre `log(ymod)`, on donne l'intervalle vide `[]` comme argument à `imshow`, qui l'ajuste automatiquement aux valeurs minimales et maximales de l'image àfficher. Notons que de manière alternative, on aurait pu utiliser de la fonction `imagesc` qui elle aussi utilise toute la gamme dynamique de l'image pour la visualisation, et permet en outre de choisir une carte de couleur quelconque. Par exemple, pour afficher l'image en niveau de gris, on peut écrire

```
>> figure(2); imagesc(log(ymod)); colormap(gray); truesize
```

Une propriété intéressante de la transformée de Fourier discrète est sa valeur à l'origine pour laquelle on peut vérifier que `sum(sum(x)) = y(1,1) = max(max(y))`.

La phase du contenu fréquentiel de l'image est plus difficile à appréhender. Afin de se rendre compte de l'importance de l'information contenue dans la phase, on peut visualiser l'image de phase comme suit :

```
>> z = ifft2(y./abs(y));
>> imshow(real(z),[-0.01,0.01]);
```

Ces résultats sont donnés dans la figure 3. On notera que l'on utilise un intervalle `[-0.01,0.01]` pour visualiser `real(z)`. On aurait pu se contenter de l'intervalle par défaut `[]`, mais l'alternative que l'on a choisi permet une meilleure visualisation des résultats. L'image de la figure 4 représente l'image de la

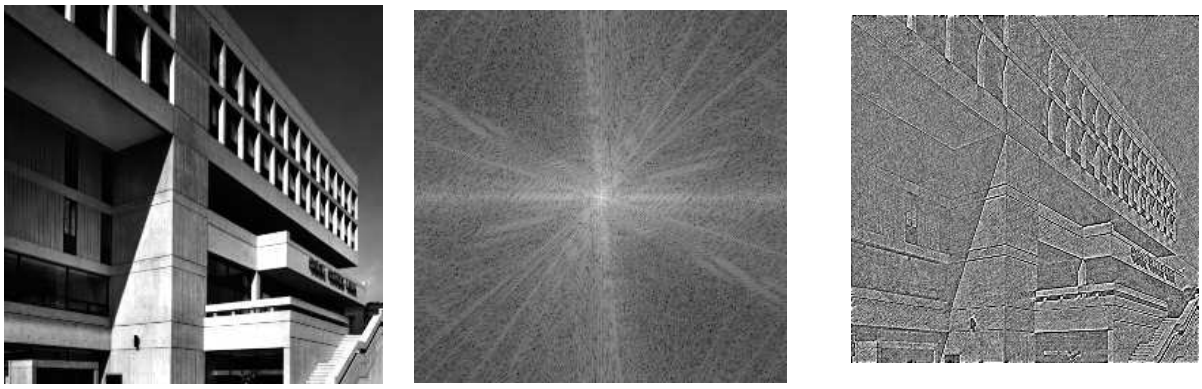

FIG. 3 – Image originale (à gauche), logarithme du module de la transformée de Fourier recentrée (au centre) et image de phase (à droite).

transformée inverse de la "transformée" formée de la phase de la transformée de l'image `mit` avec le module de la transformée de l'image `camman`. Les commandes pour obtenir ce résultat sont les suivantes :

```
>> Fcamman=fft2(camman);
>> Fmit=fft2(mit);
>> Fmix=abs(Fcamman).*exp(i*angle(Fmit));
>> mix=ifft2(Fmix);
>> imshow(real(mix),[0,255]);
```

L'information portée par la phase de l'image semble nettement plus significative que celle portée par le module (cf. fig 4). En effet, la phase informe sur l'emplacement et l'orientation de tous les contours de l'image, alors que le module ne renseigne que sur "l'intensité" de ces contours.

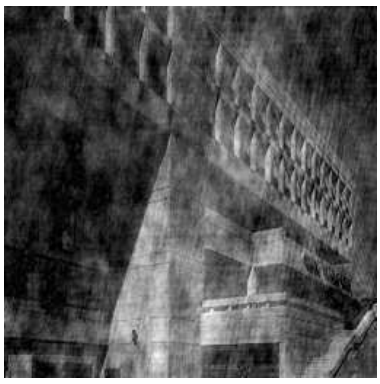

FIG. 4 – Importance de l'information contenue dans la phase de la transformée de Fourier

5. ***Manipulations dans le domaine de Fourier :***

Écrire deux versions des fonctions suivantes, une qui travaille dans le domaine de Fourier, l'autre dans le domaine spatial.

(a) Le calcul de la moyenne de l'image peut évidemment s'effectuer via `mean2()`. On peut également réecrire cette fonction de diverses manières.

Dans le domaine spatial, avec une double boucle `for` :

```
function m = moyenne(X)
# MOYENNE
#   MOYENNE(X) retourne la valeur moyenne de l'image;

somme = 0;
for a=1:size(X,1)
  for b=1:size(X,2)
    somme = somme + X(a,b);
  end
end
m = somme / ( size(X,1) * size(X,2) );
```

Les boucles `for` sont malheureusement très inefficaces sous matlab. Dans le domaine spatial, on préfèrera :

```
function m = moyenne(X)
# MOYENNE
#   MOYENNE(X) retourne la valeur moyenne de l'image;

m = sum(sum(X))/prod(size(X));
```

Dans le domaine de Fourier, le coefficient à l'origine de la FFT correspond à la somme des valeur de l'image. On peut donc écrire la fonction comme suit :

```
function m = moyenne(X)
# MOYENNE
#   MOYENNE(X) retourne la valeur moyenne de l'image;

FX = fft2(X);
m = FX(1,1)/prod(size(X));
```

(b) Pour effectuer une symétrie centrale dans le domaine spatial, on pourrait de nouveau appliquer une double boucle `for`. On peut être beaucoup plus efficace comme suit :

```
function Y = symcentre(X)
# SYMCENTRE
#   SYMCENTRE(X) inverse l'image par sym{\'e}trie centrale

Y(size(X,1):-1:1,size(X,2):-1:1) = X;
```

Pour effectuer cette symétrie dans le domaine de Fourier, on note que la symétrie centrale revient appliquer une transformation `y(m,n) = x(-m,-n)` à une translation près. Dans l'expression de la transformée de Fourier, inverser le signe des coordonnées spaciales entraîne une inversion de la phase du coefficient de Fourier. Dès lors, la FFT de `x(-m,-n)` est simplement le conjugué complexe de la FFT de `x`. La fonction s'écrit comme suit :

```
function Y = symcentre(X)
# SYMCENTRE
```

```
#   SYMCENTRE(X) inverse l'image par sym{\'e}trie centrale

Y = real(ifft2(conj(fft2(X))));
```

(c) Dans le domaine spatial :

```
function Y = contraste(X,alpha)
# CONTRASTE
#   CONTRASTE(X,alpha) change le contraste d'un facteur alpha

Y = ( ( X - mean2(X) ) * alpha ) + mean2(X);
```

Dans le domaine de Fourier :

```
function Y = contraste(X,alpha)
# CONTRASTE
#   CONTRASTE(X,alpha) change le contraste d'un facteur alpha

FX = fft2(X);
CC = FX(1,1);
FY = FX * alpha;
FY(1,1) = CC;
Y = real(ifft2(FY));
```

(d) Dans le domaine spatial :

```
function Y = translation(X,da,db)
# TRANSLATION
#   TRANSLATION(X,da,db) translate circulairement l'image X par un vecteur (da,db

for a=1:size(X,1)
   for b=1:size(X,2)
        na = mod(a+da-1,size(X,1)) + 1;
    nb = mod(b+db-1,size(X,2)) + 1;
    Y(na,nb) = X(a,b);
   end
end
```

Si on considère l'image `x` et sa translation `y`, on a

$$\begin{aligned} X(f,g) &= \sum_{m=1}^{M}\sum_{n=1}^{N} x(m,n).e^{-2.\pi.i.(\frac{m.f}{M}+\frac{n.g}{N})} \\ y(m,n) &= x(m-a,n-b) \end{aligned}$$

Par substitution et changement de variable `m'=m-a, n'=n-b`, et en tenant compte de ce que `x` peut être conmsidéré comme périodique, on trouve

$$Y(f,g) = \sum_{m=1}^{M}\sum_{n=1}^{N} y(m,n).e^{-2.\pi.i.(\frac{m.f}{M}+\frac{n.g}{N})}$$

$$
\begin{aligned}
&= \sum_{m=1}^{M}\sum_{n=1}^{N} x(m-a,n-b).e^{-2.\pi.i.(\frac{m.f}{M}+\frac{n.g}{N})} \\
&= \sum_{m'=1-a}^{M-a}\sum_{n'=1-b}^{N-b} x(m',n').e^{-2.\pi.i.(\frac{(m'+a).f}{M}+\frac{(n'+b).g}{N})} \\
&= \sum_{m'=1}^{M}\sum_{n'=1}^{N} x(m',n').e^{-2.\pi.i.(\frac{(m'+a).f}{M}+\frac{(n'+b).g}{N})} \\
&= (\sum_{m'=1}^{M}\sum_{n'=1}^{N} x(m',n').e^{-2.\pi.i.(\frac{m'.f}{M}+\frac{n'.g}{N})}).e^{-2.\pi.i.(\frac{a.f}{M}+\frac{b.g}{N})} \\
&= X(f,g).e^{-2.\pi.i.(\frac{a.f}{M}+\frac{b.g}{N})}
\end{aligned}
$$

Dès lors, dans le domaine de Fourier, on peut écrire `translation` avec une double boucle `for` :

```
function Y = translation(X,da,db)
# TRANSLATION
#   TRANSLATION(X,da,db) translate circulairement l'image X par un vecteur (da,db

s = size(X);
FX = fft2(X);
for k=1:s(1)
   for l=1:s(2)
      FY(k,l) = FX(k,l) * exp( -2*pi*i*(da*k/s(1)+db*l/s(2)) );
   end
end
Y = abs(ifft2(FY));
```

Et de même, sans boucle `for` :

```
function Y = translation(X,da,db)
# TRANSLATION
#   TRANSLATION(X,da,db) translate circulairement l'image X par un vecteur (da,db

FX = fft2(X);
k = transpose(ones(1,256)) * [1:256];
l = transpose([1:256]) * ones(1,256);
FY = FX .* exp(-2*pi*i*(k*da/size(X,1)+l*db/size(X,2)));
Y = abs(ifft2(FY));
```

FIG. 5 – Image Lena originale (à gauche), résultat de `contraste(X,0.4)` (centre) et de `contraste(X,2.5)` (droite)

FIG. 6 – Image Lena originale (à gauche), résultat de `symcentre(X)` (centre) et de `translation(X,50,100)` (droite)