Travaux pratiques de traitement d'image numérique

Année 2014-2015

SEANCE 6

I Outils généraux de traitement d'image

Les parties suivantes nécessitent d'avoir à disposition les fonctions suivantes : `dunzip`, `dzip`, `inte` et `optim`

Quelques indications sur Matlab

(1)

Cours 1 *Les images binaires sont formées de* 0 *et de* 1. *Ils permettent de stocker des images formées des teintes noires* (0) *et blanches* (1).

```
im1=imread('text.png');
figure(1); imshow(im1);
```

Dans cet exemple `im1` *indique qu'en mémoire est stockée une matrice dont les composantes sont* 0 *ou* 1. *Le fait qu'il n'y ait que deux valeurs est la raison pour laquelle on appelle cela une image binaire. Les commandes suivantes permettent d'abord de mettre sous la forme d'un vecteur vertical les composantes de l'image binaire puis de supprimer les doubles dans ce vecteur (i.e. les valeurs identiques n'apparaissent qu'une fois).*

```
val=im1(:);
unique(val),
```

Les images en niveaux de gris permettent d'afficher des images qui contiennent différentes teintes entre le noir et le blanc.

```
im2=imread('trees.tif');
figure(1); imshow(im2);
```

Elles sont mémorisées sous la forme d'une matrice dont les composantes peuvent des entiers entre 0 *et* 255. *Les commandes suivantes permettent d'afficher une courbe des les valeurs prises sont les valeurs des pixels de l'image lorsqu'on parcourt cette image de haut en bas puis de gauche vers la droite.*

```
val=im2(:);
figure(1); plot(val);
```

Les images en niveaux de gris peuvent aussi être composées de réels entre 0 *et* 1.

```
im2Bis=double(im2)/255;
figure(1); imshow(im2Bis);
val=im2Bis(:);
figure(2); plot(val);
```

Les images colorées sont parfois mémorisées sous la forme d'un tableau de chiffres chaque chiffre correspondant à un index dans une table de couleurs. Les commandes suivantes permettent par exemple de récupérer l'image `trees.tif`.

```
[im3,map]=imread('trees.tif');
im3Bis=ind2rgb(im3,map);
figure(1); imshow(im3Bis);
```

`im3` *désigne une matrice dont les composantes ne sont pas des teintes. En effet si on cherche à les afficher on obtient une image qui n'a pas de sens (pourquoi les parties rouges auraient-elles la même teinte que les parties noires ?).*

```
figure(2); imshow(im3);
```

`map` *désigne un tableau composé de trois colonnes, la première correspond aux composantes de rouge, la deuxième aux composantes de vert et la troisième aux composantes de bleu. La commande suivante donne la taille de* `map`*, la première valeur correspond au nombre de lignes et la deuxième valeur correspond au nombre de colonnes.*

```
size(map),
```

Si on annule les composantes de la première colonne, l'image reconstruite n'aura plus de rouge.

```
mapBis=map;
mapBis(:,1)=0;
im3Ter=ind2rgb(im3,mapBis);
figure(3); imshow(im3Ter);
```

Ces images couleurs sont stockées sous la forme d'une matrice (ici `im3`*) dont les composantes indiquent pour chaque position des pixels un index vers une table (ici* `map`*) où sont référencées les couleurs associées à cet index. Les commandes suivantes illustrent ce fonctionnement en transférant l'index 127 de* `im3` *à l'index* 0*.*

```
im3Qua=im3; im3Qua(im3==127)=0;
im3Qui=ind2rgb(im3Qua,map);
figure(4); imshow(im3Qui);
```

Pour comprendre ce que l'on observe, on constate d'abord que la première ligne de `map` *référence la teinte noire tandis que la ligne* 128 *référence une teinte blanche.*

```
map(1,:),
map(128,:),
```

Cela permet de comprendre pourquoi les parties de l'image couleur qui étaient très blanches (associé à l'index 127*) sont alors devenues noires (associées à l'index* 255*).*

Les images peuvent aussi être mémorisées sous la forme d'une matrice-3D formées chacune d'entiers de 0 *à* 255 *(par exemple* `image(1,1,1)` *désigne l'intensité du rouge [3ème 1] présent dans le pixel qui est sur la colonne 1 [1er 1] et sur la ligne 1 [2ème 1]). Les commandes suivantes permettent par exemple de charger et visualiser l'image* `autumn.tif`*, elles permettent aussi de lire sur une courbe les valeurs des composantes couleurs en parcourant l'image de haut en bas, puis de gauche vers la droite puis du rouge vers le vert et enfin le bleu.*

```
im4=imread('autumn.tif');
figure(1); imshow(im4);
val=im4(:);
figure(2); plot(val),
```

La taille d'une image est le nombre de lignes et le nombre de colonnes. La commande suivante indique dans son premier chiffre le nombre de ligne, dans son deuxième chiffre le nombre de colonnes et dans son troisième chiffre 3 *s'il y a effectivement trois composantes couleurs.*

```
size(im4),
```

Le traitement d'image amène à faire des opérations sur les valeurs de chaque pixel. Il faut donc que ces valeurs soit un type adapté appelé `double` *en Matlab. La convention est alors que les valeurs doivent être comprises entre* 0 *et* 1*.*

```
im4Bis=double(im4)/255;
figure(3); imshow(im4Bis);
val=im4Bis(:);
figure(4); plot(val),
```

Pour vérifier si une image est au format entier entre 0 *et* 255 *(noté ici* `uint8`*) ou au format* `double`*, on peut utiliser la commande* `class`

```
class(im4),
class(im4Bis),
```

On peut aussi vérifier les valeurs prises par les images avec

```
max(max(max(im4))),
min(min(min(im4))),
max(max(max(im4Bis))),
min(min(min(im4Bis))),
```

On peut former une image plus petite en ne considérante qu'un ligne sur deux et une colonne sur deux (c'est-à-dire un pixel sur quatre).

```
im7=im4(1:2:end,1:2:end,:);
figure(1); imshow(im4);
figure(2); imshow(im7);
```

Une façon plus générale de modifier la taille d'une image peut se faire avec les commandes suivantes :

```
im8=imresize(im2,[256 256]);
figure(1); imshow(im2);
figure(2); imshow(im8);
```

Cette commande ne fonctionne pas en tant que telle pour les images couleurs. Pour réaliser cette transformation sur une image couleur, il suffit que cette image couleur soit d'abord convertie de façon à avoir des valeurs entre 0 *et* 1*, puis de définir avec* `zeros` *une image couleur de la taille souhaitée, et enfin pour chaque composante de la remplir avec le résultat de l'application de la commande* `imresize` *sur chaque composante couleur.*

```
im10=zeros(256,256,3);
im10(:,:,1)=imresize(im4Bis(:,:,1),[256 256]);
im10(:,:,2)=imresize(im4Bis(:,:,2),[256 256]);
im10(:,:,3)=imresize(im4Bis(:,:,3),[256 256]);
figure(1); imshow(im4);
figure(2); imshow(im10);
```

On peut aussi considérer une partie de l'image :

```
im9=im2(1:128,1:128);
figure(1); imshow(im2);
figure(2); imshow(im9);
```

Matlab dispose d'une aide en ligne sur la fenêtre de commandes : `help` *permet d'afficher les différentes sections,* `help` *nom d'une section permet d'avoir la liste des commandes dans cette section, en particulier* `help images` *donne la liste des commandes relatives à l'image.* `help` *nom d'une commande donne une explication sur la commande et souvent un exemple qui peut vraiment être essayé. La liste des images disponibles sous Matlab est* `help imdemos`*, ou*

```
dir([which('autumn.tif'),'\..'])
```

On peut aussi réaliser une image en lui donnant des valeurs. Ainsi les commandes suivantes permettent de réaliser une image rouge et bleue.

```
im5=zeros(10,10,3);
im5(:,:,1)=[ones(10,5) zeros(10,5)];
im5(:,:,3)=[zeros(10,5) ones(10,5)];
figure(1); imshow(im5);
```

Autant il est en général difficile de colorer une image en niveaux de gris, il est simple de transformer une image en niveaux de gris.

```
im6=rgb2gray(im4);
figure(1); imshow(im4);
figure(2); imshow(im6);
```

Les fonctions Matlab pour lire et enregistrer les images sont `imread` *et* `imwrite`. *Les fonctions disponibles pour afficher les images sont* `image`, `imagesc` *et* `imshow`. *Attention à* `imshow` *qui peut étendre la palette de couleur ou de niveau de gris sans prévenir. Plusieurs images peuvent être affichées sur une seule figure grâce aux fonctions* `figure` *et* `subplot`. *Une image en niveau de gris ou n'importe quelle matrice peut être considérée comme une surface : à chaque coefficient de la matrice, on associe un point dont l'abscisse et l'ordonnée sont déterminés par la position de ce coefficient et la cote est déterminée par la valeur de ce coefficient. Les fonctions Matlab disponibles sont* `surf` *et* `mesh`, *la première fonction colore les éléments de surface tandis que la deuxième colore seulement les bords des éléments de surface.*

En ce qui concerne l'interface Matlab et la disposition des fenêtres, si celle-ci n'est pas satisfaisante, il est possible de revenir à la disposition par défaut en actionnant successivement dans les menus déroulant : `Desktop, Desktop Layout, Default`.

II Présentation d'une application, la restauration d'une image bruitée

(59) Dans cette partie, vous choisirez une image en niveau de gris de taille carré, (par exemple en prélevant une partie carré). Cette image est notée `im`. Cette image doit être convertie en double à valeurs dans l'intervalle $[0, 1]$.

II.a Simulation d'un bruit

(60) On considère l'effet du bruit gaussien d'écart-type $\sigma = 0.2$ appliqué sur l'image noté `im`.

```
bruit=0.2*randn(size(im));
im_bruit=im+bruit;
figure(1); imshow(im);
figure(2); imshow(im_bruit);
```

II.b Applications de filtres pour réduire le bruit

(61)
On cherche maintenant à réduire le bruit ainsi ajouté au moyen d'un filtre.

II.1. On considère d'abord le filtre moyenneur, son masque est une matrice dont toutes les composantes valent 1. Choisissez la taille du masque à appliquer notée `taille_bruit`. L'image restaurée, notée ici `im_rest1` est donnée par :

```
taille_bruit=10;
filtre_moy=ones(taille_bruit)/taille_bruit/taille_bruit;
im_rest1=filter2(filtre_moy,im_bruit);
figure(1); imshow(im_rest1);
```

II.2. On applique maintenant un filtre gaussien d'écart-type σ et défini sur une matrice de taille `taille_gaus`. Choisissez des valeurs pour ces deux paramètres.

```
taille_gaus=5;
sigma=1;
filtre_gaus=fspecial('gaussian',[taille_gaus taille_gaus],sigma);
im_rest2=filter2(filtre_gaus,im_bruit);
figure(1); imshow(im_rest2);
```

II.c Mesure de l'efficacité de la restauration

(62)

Le taux d'amélioration de l'erreur quadratique moyenne (*improved mean square normalized error* en anglais) :

$$TAEQM = \frac{\sum_m \sum_n (g^o_{m,n} - g^r_{m,n})^2}{\sum_m \sum_n (g^o_{m,n} - g^b_{m,n})^2}$$

où $[g^o_{m,n}]$, $[g^b_{m,n}]$, $[g^r_{m,n}]$ sont l'image origine, bruitée et restaurée.

```
TAEQM=std2(image_o-image_r)/std2(image_o-image_b);
```

En pratique on calcule plutôt la différence entre le PSNR de g^r avec g^o et le PSNR de g^b avec g^o.

II.3. Montrez que ces deux calculs, TAEQM et différence de PSNR, sont reliés entre eux par une relation non-linéaire.

Le calcul du PSNR en traitement d'image pour des images en niveaux de gris à valeurs sur $[0, 1]$.

```
PSNR=@(im1,im2)-10*log10(mean2((im1-im2).^2))*...
 (max(max(max(im1,im2)))<=1.5);
```

La deuxième partie de la formule est juste pour tenter d'avertir si jamais la formule est appliquée à des données qui n'auraient pas le bon format.

Un programme permet d'aider à choisir les paramètres de façon optimale, chaque paramètre à déterminer doit être sur un champ particulier de la variable option.

```
filtre_moy=@(option)ones(option.taille_bruit)/option.taille_bruit...
  /option.taille_bruit;
im_rest1=@(option)filter2(filtre_moy(option),im_bruit);
option.taille_bruit=2:25;
cout=@(option)-(PSNR(im_rest1(option),im)-PSNR(im_bruit,im));
[option_choix,val]=optim(option,cout);
figure(1); imshow(im_rest1(option_choix));
```

II.4. Choisissez les valeurs optimales pour les paramètres pour les questions II.1 et II.2.

II.d Visualisation de l'impact du bruit sur le spectre

(63)

On souhaite maintenant regarder le spectre de l'image, du bruit et des filtres utilisés afin de justifier le choix des paramètres faits à la question II.4.

II.5. Rappelez (en faisant l'analogie avec le traitement du signal), comment trouver les caractéristiques du filtre permettant au mieux de se rapprocher de l'image de originale.

On peut représenter sur une image le spectre d'une image de la façon suivante :

```
S=@(im)abs(fftshift(fft2(im))).^2/prod(size(im));
figure(1); imshow(mat2gray(S(im)));
Sln=@(im)10*log10(S(im));
figure(2); imshow(mat2gray(Sln(im)));
```

On peut représenter les zones où le bruit est plus important que le signal et les zones où le signal est plus important que le bruit.

```
figure(3); imshow((S(im)>1.5*S(bruit))+0.5*(S(im)>0.5*S(bruit)));
```

Les fréquences des spectres dessinées sont données par

```
[f1,f2]=ndgrid(-1/2:1/size(im,1):(1/2-1/size(im,1)),...
  -1/2:1/size(im,2):(1/2-1/size(im,2)));
```

II.6. Les commandes ci-dessus, correspondent-elles à une échelle en cycle par image ?

A partir du masque M, on peut représenter la réponse fréquentielle sur les fréquences `f1,f2`

```
masque=ones(4)/16;
H=freqz2(masque,f1,f2);
figure(4); imshow(H);
```

II.7. Représentez les filtres choisies de façon optimale aux questions II.1 et II.2.

II.8. Quel défaut avait les filtres précédents. Essayez de proposer un meilleur filtre.

III Présentation d'une application particulière : la compression

(59) Dans cette partie, vous choisirez une image en niveau de gris de taille carré, (par exemple en prélevant une partie carré). Cette image est notée `im`. Cette image doit être convertie en double à valeurs dans l'intervalle $[0, 1]$.

III.a Compression sans perte

(64)

Les programmes `dzip` et `dunzip` permettent de réaliser une compression sans perte.

On peut d'abord vérifier qu'on retrouve l'image de départ

```
figure(1); imshow(dunzip(dzip(uint8(255*im))));
```

III.1. Quelle est la différence entre une compression sans perte et une compression avec perte ?

Dans la mesure où la sortie de la fonction `dzip` sont un ensemble d'entiers entre 0 et 255, on peut en déduire le nombre de bits par pixel nécessaire au stockage de l'image quand on utilise ces programmes faire une compression sans perte.

III.2. On note L la longueur du vecteur fourni par `dzip` et $N_X \times N_Y$ la taille de l'image considérée, que vaut le nombre de bits par pixel pour compresser cette image ? Expliquez pourquoi ce nombre ne devrait pas dépasser 8 ?

III.b Compression avec pertes des images en niveaux de gris

(65)

La fonction `imwrite` et `imread` permettent de faire une compression JPEG et une décompression JPEG avec perte.

Choisissez un répertoire où il est possible d'enregistrer un fichier. On peut observer l'impact d'une dégradation liée à une compression JPEG.

```
imwrite(im,'tmp.jpg','Quality',50);
figure(1); imshow(imread('tmp.jpg'));
```

Chaque valeur du paramètre `Quality` entre 0 et 100 permet d'obtenir un fichier particulier `tmp.jpg` qui occupe sur le disque dur un certain nombre d'octets donné par

```
S=dir('tmp.jpg'); S.bytes,
```

Une fois décompressée avec `imread('tmp.jpg')`, on retrouve une version déformée de l'image, dont on peut mesurer le `PSNR` définie lors de la question II.4.

III.3. Tracez la courbe débit/distorsion, le PSNR en dB en fonction du débit en bits par pixel.

III.c Compression avec pertes des images en couleur, espace YUV

(66)

Dans cette partie on considère à nouveau une image couleur notée ici `im`.

Ce qu'on appelle l'espace YUV correspond en général à l'espace `YCbCr` dont les fonctions `rgb2ycbcr` et `ycbcr2rgb` assurent les conversions

Les formules, lorsqu'on cherche une conversion d'entier à entier, sont :

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix} + \frac{1}{256} \begin{bmatrix} 66 & 129 & 25 \\ -38 & -74 & 112 \\ 112 & -98 & -18 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{1}$$

III.4. Montrez que (1) est plus ou moins équivalent à dire que $Y = 0.3R + 0.59G + 0.11B$ et que $Cb = B - Y$ et $Cr = R - Y$. En réalité (1) tient aussi compte d'une normalisation (addition et multiplication) de façon utiliser pleinement le fait que les images avant et après transformation sont stockées en tant qu'entiers de 0 à 255.

On considère l'expérimentation suivante :

```
[X,Y]=ndgrid(0:255,0:255);
figure(1); imshow(ycbcr2rgb(uint8(cat(3,zeros(256),X,Y))));
figure(2); imshow(ycbcr2rgb(uint8(cat(3,128*ones(256),X,Y))));
figure(3); imshow(ycbcr2rgb(uint8(cat(3,255*ones(256),X,Y))));
```

III.5. Que représentent les trois figures générées par les commandes ci-dessus ?

Les fonctions `imwrite` et `imread` fonctionnent très bien pour des images couleurs. Cependant on ne va supposer ici que cela ne fonction que pour les images en niveau de gris.

III.6. Avec l'image en couleur `im`, convertissez la en format YCbCr puis compressez la partie Y, la partie Cb et la partie Cr avec les fonctions `imwrite` et `imread`. Calculez le débit en bits par pixel en ajoutant le nombre d'octets des trois fichiers `tmp.jpg` générés. A partir des trois fichiers `tmp.jpg`, retrouvez une image couleur relativement similaire `im`. Calculez pour chacune des composantes Y, Cb et Cr un PSNR particulier appelé PSNR_Y, PSNR_U et PSNR_V.

III.d Visualisation des performances bloc par bloc

(67)

On considère ici une image en niveaux de gris notée `im`.

Les codeurs généralement découpe l'image en bloc contigüs. Ici les blocs sont tous de la même taille. On cherche ici à coder séparément chaque bloc avec une compression JPEG, implémentée par les fonctions `imwrite` et `imread`.

Comme la taille de l'image n'est pas nécessairement un multiple de la taille des blocs, il est nécessaire d'agrandir l'image

```
bloc=[8 8];
z=@(x,y)x*(mod(y,x)~=0)-mod(y,x);
m1=@(im,bloc)z(bloc(1),size(im,1));
m2=@(im,bloc)z(bloc(2),size(im,2));
agrandir=@(im,bloc)[im zeros(size(im,1),m2(im,bloc));...
 zeros(m1(im,bloc),size(im,2)+m2(im,bloc))];
```

Le nombre de lignes et de colonnes de blocs est donné par

```
I=@(im,bloc)ceil(size(im,1)/bloc(1));
J=@(im,bloc)ceil(size(im,2)/bloc(2));
```

On définit une nouvelle fonction `extrait`

```
d=@(x,y)1+(x-1)*y; f=@(x,y)x*y;
extrait=@(im,bloc,i,j)im(d(i,bloc(1)):f(i,bloc(1)),...
  d(j,bloc(2)):f(j,bloc(2)));
```

On souhaite disposer d'un affichage à l'écran obtenu qui fasse un zoom.

```
inte=@(im)im([1:end;1:end],[1:end;1:end]);
inte_z=@(im)inte(inte(inte(im)));
imshow_z=@(im)imshow(inte_z(im));
```

On accède au bloc `(i,j)` de la façon suivante :

```
extrait(agrandir(im,bloc),bloc,i,j)
```

On peut afficher successivement l'ensemble des blocs.

```
for i=1:I(im,bloc)
  for j=1:J(im,bloc)
    figure(1);
    imshow_z(extrait(agrandir(im,bloc),bloc,i,j));
    title(['(',num2str(i),',',num2str(j),')']),
    drawnow,
    pause(0.1)
  end
end
```

On appelle `w`, `bpp` et `dB` les fonctions qui sauvegardent, donnent le coût et la qualité de la compression de chaque bloc d'une image. Vous pouvez réaliser ces fonctions sous la forme de fichier `.m`. Voici aussi un exemple de ces fonctions.

```
w=@(im,Q)imwrite(im,'tmp.jpg','Quality',Q);
bpp=@(im)eval(['subsref(dir(''tmp.jpg''),substruct(''.'',''bytes''))']),...
  *8/numel(im);
dB=@(im)PSNR(double(imread('tmp.jpg'))/255,im);
```

On peut alors afficher les résultats obtenus sur les différents blocs. La résolution est d'autant meilleur que les blocs sont de petites tailles, cependant l'implémentation de JPEG n'est pas du tout adaptée aux images petites et donc le taux de compression augmente fortement lorsque les blocs sont petits.

```
Q=50;
im_bpp=zeros(I(im,bloc),J(im,bloc));
im_dB=zeros(I(im,bloc),J(im,bloc));
for i=1:I(im,bloc)
  for j=1:J(im,bloc)
    ex=extrait(agrandir(im,bloc),bloc,i,j);
    w(ex,Q);
    im_bpp(i,j)=bpp(ex);
    im_dB(i,j)=dB(ex);
  end
end
figure(1); imshow(im);
figure(2); imshow_z(mat2gray(im_bpp));
figure(3); imshow_z(mat2gray(im_dB));
```

III.7. Décrivez les blocs qui sont facile à compresser et ceux qui sont difficiles à compresser.

III.e Non-robustesse au bruit du fichier compressé

(68)

La compression sans perte présentée plus haut est un exemple de compression non-robuste à la moindre erreur sur le fichier compressé :

```
M=dzip(uint8(255*im));
M(ceil(rand(1)*numel(M)))=ceil(rand(1)*255);
figure(10); imshow(dunzip(M));
```

III.8. Expliquez ce que font ces lignes de code ?

SEANCE 7

IV Des outils pour analyser les images

IV.a Profil d'intensité d'une image

(69)

Cours 2 *Le profil d'intensité d'une image en niveaux de gris correspond à une ligne souvent horizontale traversant l'image, il s'agit de la courbe des intensités de chaque pixel présent sur cette ligne. Par exemple le profil montré sur la figure 2 d'une image en niveaux de gris montrée sur la figure 1. Les pixels concernés par le profil sont montrés sur la figure 3. L'instruction* `axis` *permet de s'assurer de la maîtrise de l'axe des ordonnées (i.e. ne pas laisser qu'il y a de grandes fluctuations, alors qu'en réalité il n'y en a pas).*

```
im=double(imread('coins.png'))/255;
figure(1); imshow(im);
figure(2); plot(1:size(im,1),im(:,50)); axis([-inf inf 0 1]);
im1=im; im1(:,50)=max(max(im));
figure(3); imshow(im1);
```

Un profil peut être fait suivant une ligne horizontale ou verticale.
Il est possible de choisir un extrait de l'image avec les commandes suivantes

```
figure(1); imshow(im); A=round(ginput(2)); ex=im(A(3):A(4),A(1):A(2));
figure(2); imshow(ex);
```

IV.1. Donnez un exemple de profil soit en utilisant une image extraite à partir d'un morceau choisi soit à partir des données issues de III.7. En quoi le profil permet de préciser ce qu'on voit dans l'image ?

IV.b Histogramme d'une image

Cours 3 *(70) Pour une image en* **niveaux de gris**, *on appelle histogramme le fait de représenter le nombre (ou la proportion) de pixels ayant tels niveaux de gris en fonction du niveau de gris. Sous Matlab, les commandes suivantes permettent de tracer un histogramme.*

```
[y,x]=hist(im(:),50);
figure(1); plot(x,y);
axis([0 1 0 inf]); %si im est à valeurs dans [0,1]
```

où 50 *est ici le nombre de plages de niveaux de gris considérés.*

IV.2. Donnez un exemple d'histogramme soit en utilisant une image extraite à partir d'un morceau choisi soit à partir des données issues de III.7. En quoi l'histogramme permet de préciser ce qu'on voit dans l'image ?

IV.c Visualisation du spectre d'une image

(71)
Pour visualiser le spectre d'un extrait d'une image, il est préférable que cet extrait soit de taille carré

```
figure(1); imshow(im);
A=round(ginput(2)); A([2 4])=A([1 3])+min(diff(A));
ex=im(A(3):A(4),A(1):A(2));
figure(2); imshow(ex);
```

Les commandes permettant d'afficher un spectre ont été données pour la question II.6. Afin de pouvoir mieux visualiser vous pourrez aussi utiliser `imshow_z` défini lors de la question III.7 (p. 9).

IV.3. En choisissant des extraits obtenus soit avec les instructions ci-dessus, soit à partir des données issues de III.7 (p. 9), expliquez comment l'analyse d'un spectre peut permettre de retrouver des propriétés d'anisotropie (existence ou non d'une ou plusieurs directions principales et angles de directions).

Donnez un exemple d'histogramme soit en utilisant une image extraite à partir d'un morceau choisi soit à partir des données issues de III.7. En quoi l'histogramme permet de préciser ce qu'on voit dans l'image ?

IV.4. En choisissant d'autres extraits, montrez comme l'analyse d'un spectre permet de retrouver l'existence de petites ou grandes variations dnas l'extrait ?

V Action sur l'image

V.a Simulation de l'effet de quantification

(72) On cherche à simuler l'effet de quantification. Voici un exemple de commande matlab permettant de quantifier une image.

```
im=double(imread('coins.png'))/255;
imQ=floor(5*im)/5
figure(1); imshow(im);
figure(2); imshow(imQ);
```

V.1. Dans la quantification présentée, combien y a-t-il de niveaux de quantifications ? Quel est le pas de quantification ? L'erreur de quantification est-elle a priori symétrique ou garde-t-elle un signe constant ?

Les commandes précédentes permettent de modifier le nombre de niveau de quantification. Vous pouvez aussi considérer l'image suivante :

```
[J,I]=meshgrid((0:255)/255,(0:255)/255);
im=2*I.*(I<=0.5)+2*(1-I).*(I>0.5);
```

V.2. Expliquez comment on peut reconnaître une image qui a été quantifiée, d'une image qui a subi un autre type de déformation, par exemple celle de la section II.a ou celles des questions II.1 et II.2.

V.3. La quantification déforme l'image mais elle permet aussi de stocker l'image avec moins d'octets. En combinant la quantification et la compression sans perte de la section III.a, calculez quelques points de la courbe débit-distorsion, comment se situent ces points par rapport aux autres points déjà calculés ?

V.b Simulation de différents effets de flou

(73) On considère l'effet de flou produit par un filtre moyenneur.

```
masque=ones(5,5)/25;
im=double(imread('coins.png'))/255;
imMoy=filter2(masque,im);
figure(1); imshow(im);
figure(2); imshow(imMoy);
```

V.4. Pourquoi dans les commandes Matlab, y a-t-il une division par 25 ?

V.5. `masque` est une matrice. Trouver un moyen de changer la taille sans que l'impact de ce filtre sur l'image n'assombrisse ou n'éclaircisse l'image filtrée. Commentez sur la relation entre la taille de `masque` et l'impact sur l'image filtrée.

V.6. En modifiant le masque, en fait en lui faisant ressembler à un rectangle orienté dans une certaine direction, montrez qu'il est possible de simuler un effet de bougé dans une direction.

V.7. Expliquez comment on peut distinguer une image qui a été déformé par un effet de flou par rapport aux autres déformations vues précédemment.

V.8. Le fait de rajouter du flou sur une image permet de la stocker avec moins de place. Combinez l'application du flou avec une compression JPEG pour une certaine qualité donnée et commentez la position du point ainsi obtenu par rapport à la courbe débit-distorsion obtenu avec la compression JPEG.

V.9. Expliquez comment on peut distinguer une image qui a été déformé par un effet de bougé par rapport aux autres déformations vues précédemment.

V.c Simulation de l'effet de sous-échantillonnage

(74)

On cherche ici à sous-échantillonner une image puis à partir de l'image sous-échantillonnée à retrouver l'image de départ en sur-échantillonnant l'image de taille réduite ; ceci en s'inspirant de ce qui a été vu en traitement du signal. Pour sous-échantillonner la première technique, notée `deci0` consiste à prélever une ligne sur deux et une colonne sur deux (c'est-à-dire un pixel sur quatre), la deuxième technique, notée `deci1` consiste à appliquer une filtre passe-bas avant de faire le prélèvement. Ces deux techniques transforment une image de taille $M \times N$ en une image de taille $\frac{M}{2} \times \frac{N}{2}$.

Pour sur-échantillonner, la première technique, notée `inte2`, consiste à répéter chaque ligne et chaque colonne, c'est-à-dire que chaque pixel se trouve entouré de trois nouveaux pixels qui ont la même valeur que ce pixel. La deuxième technique `inte3` consiste à appliquer un filtre passe-bas après avoir le rajout des pixels supplémentaires, ce faisant les pixels rajoutés auront des valeurs intermédiaires entre les pixels qui les entourent. Ces deux dernières techniques appliquées à une image de taille $M' \times N'$ en une image de taille $2M' \times 2N'$.

L'objectif est d'appliquer l'une des deux premières techniques `deci0` ou `deci1` trois fois de suite de façon à ce que le nombre de pixels de l'image soit divisé par 64, figure 1. Parmi les huit images que l'on pourrait obtenir, on considère seulement les images notées `im000` et `im111`. Ensuite on applique à ces deux (`im000` et `im111`) l'une des deux autres techniques `inte2` et `inte3` de façon à retrouver pour les différentes images la taille initiale. Lorsqu'on considère initialement l'image `im000`, les images obtenues avec le schéma de la figure 2 sont `im000222` et `im000333`. Lorsqu'on considère initialement l'image `im000`, les images obtenues avec le schéma de la figure 3 sont `im111222` et `im111333`.

Du point de vue de l'implémentation Matlab, les transformations `deci0`, `deci1`, `inte2`, `inte3` peuvent s'implémenter sous la forme de fonctions en ligne.

```
deci0=@(im)im(1:2:end,1:2:end);
deci1=@(im)deci0(filter2(ones(2)/4,im));
inte2=@(im)im([1:end;1:end],[1:end;1:end]);
inte3=@(im)filter2(ones(2)/4,inte2(im));
```

V.10. Pourquoi l'image `im111333` est-elle équivalente à une déformation obtenue avec du flou ? Quel est le masque du filtre correspondant ?

V.11. Réalisez et observez les images `im000222` et `im111333` à partir de l'image `im`. La première est ce qu'on appelle un effet de **pixellisation**. Expliquez comment on peut distinguer cette déformation des autres déformations vues précédemment ?

Une technique classique pour la compression consiste à appliquer un sous-échantillonnage avant de compresser une image et appliquer un sur-échantillonner juste après la décompression.

V.12. Combinez cette technique avec la compression JPEG sur l'image `im` et comparez les performances obtenues en terme de débit-distorsion avec la courbe associée à la compression JPEG.

Pour les images couleurs, il est fréquent de sous-échantillonner plus la chrominance (i.e. les composantes `Cr` et `Cb`) que la luminance (i.e. la composante `Y`).

- 4 :4 :4 signifie aucun sous-échantillonnage sur les composantes `Y`, `Cb` et `Cr`.
- 4 :2 :2 signifie pleine résolution pour `Y`, et sous-échantillonnage d'un facteur 2 horizontalement pour `Cb` et `Cr`.
- 4 :2 :0 signifie pleine résolution pour `Y`, et sous-échantillonnage d'un facteur 2 horizontalement et verticalement pour `Cb` et `Cr`.

V.13. Implémentez la compression 4 :2 :0 avec la compression JPEG sur chacune des composantes `Y`, `Cb` et `Cr`. Observez le résultat.

V.d Reconstitution partielle des blocs avec les coefficients de Fourier

Cours 4 *(75) La transformée de Fourier adaptée aux images est la transformée de Fourier discrète bidimensionnelle :*

$$G_{k,l} = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} g_{m,n} e^{-j2\pi(\frac{km}{M}+\frac{ln}{N})}$$

où $g_{m,n}$ est l'intensité (valeur entre 0 et 1) du pixel à la position (m,n) et $G_{k,l}$ est le coefficient associé à la fréquence dont la composante horizontale u est $\frac{k}{M} f_e$ et la composante verticale v est $\frac{l}{N} f_e$, f_e étant la fréquence d'échantillonnage qui s'exprime dans une unité. Les variables associées aux fréquences spatiales sont notées u et v. La taille de l'image est $M \times N$. Le choix du coefficient $\frac{1}{N^2}$ est dans une certaine mesure arbitraire, il garantit ici que la composante constante $G_{0,0}$ est la moyenne du signal image.

La transformée de Fourier discrète ne tient pas compte du fait que les images sont à valeurs réelles et que par suite elles sont en plus d'être périodiques, symétriques par rapport à la moitié de la fréquence d'échantillonnage. Il faut donc comme en traitement du signal utiliser l'instruction `fftshift` *pour placer la fréquence nulle au centre de l'image et ainsi mieux visualiser les propriétés de symétrie.*

La transformée de Fourier discrète inverse est :

$$g_{m,n} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} G_{k,l} e^{j2\pi(\frac{km}{M}+\frac{ln}{N})}$$

Cette formule exprime le fait que les images sont des combinaisons linéaires d'images formées de sinusoïdes, il suffit pour cela d'ajouter les exponentielles complexes qui sont conjuguées avec des coefficients qui sont conjugués ($G_{k,l} = \overline{G_{M-k,N-l}}$ si $k \neq 0$ et $l \neq 0$).

On considère une image `im` et carrée dont la taille est paire et petite et dont les valeurs sont dans $[0,1]$. Vous pouvez par exemple choisir cette image

```
im=zeros(32); im(8:23,8:23)=1;
```

Les instructions suivantes montrent comment construire ces différentes sinusoïdes qui permettent par approximation successive de reconstruire l'image d'origine. L'image de gauche sont les points qu'on a sélectionnés et les composantes de la transformée de Fourier retenues. L'image du milieu est l'image d'origine et l'image de droite est sa reconstruction à partir des composantes retenues. Pour arrêter la simulation il suffit de cliquer en dehors de la première image. Le premier point déjà sélectionné est celui de la fréquence nulle.

```
estOk=@(I)(I(2)>=1&I(2)<=size(im,1)&I(1)>=1&I(1)<size(im,2));
points=zeros(size(im)); im1=zeros(size(im));
I=1+size(im)/2; vert=0.5*ones(size(im,1),2);
while(estOk(I))
  points(I(2),I(1))=1;
  sp=fftshift(fft2(im)); sp1=points.*sp;
  im1=real(ifft2(fftshift(sp1)));
  figure(1); imshow_z([points vert im vert im1]);
  I=round(ginput(1)/8);
end;
```

V.14. Proposez un schéma de compression en ne retenant pour chaque bloc que quelques composantes de la transformée de Fourier, en quantifiant ces composantes et en les transmettant avec une compression sans perte. Evaluez les performances de cette méthode de compression.

Utilisation de bloc de tailles diverses

(20)

Cours 5 *La segmentation en région consiste à découper l'image suivant un critère en différentes régions connexes (c'est-à-dire composées de pixels qui sont voisins). Ainsi pour l'image* `coins.png`*, une segmentation en région possible consisterait à définir une région pour chaque pièce de monnaie ainsi qu'une région pour le fond de l'image.*

On considère ici une image `im` carrée de taille 256×256 à valeurs dans $[0, 1]$. Une telle image peut être par exemple

```
im1=double(imread('coins.png'))/256;
im=[0.5*ones(10,256); im1(:,1:256)];
```

La segmentation en région proposée ici est une segmentation en `quadtree`. Les régions ont été rendues visibles en donnant aux pixels de la région la valeur moyenne des pixels de l'image originale.

```
qtim3=qtdecomp(im,0.5);
im4=im;
for dim=[2 4 8 16 32 64 128]
  [vals,r,c]=qtgetblk(im,qtim3,dim);
  if (false==isempty(vals))
    newvals=[];
    for cpt1=1:size(vals,3)
      newvals=cat(3,newvals,mean2(vals(:,:,cpt1)*ones(dim)));
    end
    im4=qtsetblk(im4,qtim3,dim,newvals);
  end
end;
imshow(im4);
```

Les deux premières instructions rend l'image carrée. Les autres fonctions ont la signification suivante :

- `qtdecomp` réalise le découpage en régions carrés ;
- `full` convertit une matrice `sparse` (matrice représentée seulement par les indices de ses coefficients non-nuls) en une matrice `full` (représentation classique) ;
- `qtgetblk` donne l'ensemble des régions d'une certaine taille, les valeurs sont stockées sous la forme d'un ensemble de matrices `vals` et leur localisation sont dans deux vecteurs `r` et `c`.
- `qtsetblk` permet d'affecter des valeurs aux différentes régions d'une certaine taille.

La segmentation en `quadtree` ne permet certes pas d'atteindre l'objectif de la segmentation (i.e. isoler les objets dans une image), mais elle est efficace du point de vue de la compression. En effet elle permet dans une certaine mesure d'avoir un traitement différencié pour les différentes zones de l'image et elle nécessite peu d'octets pour être codé et transmis au décodeur.

V.15. Sachant qu'une décomposition en `quadtree` peut se voir comme un arbre pour lequel chaque noeud a 4 fils, expliquez pourquoi le coût du codage de cette décomposition est le nombre de noeuds de cet arbre ?

A partir d'une décomposition, on peut retrouver le nombre de noeud de l'arbre en comptant le nombre de pixels qui sont limitrophe de trois autres régions situées à gauche, à gauche en haut et en haut, elles-mêmes étant différentes les unes des autres. Voici une implémentation pour compter ces pixels.

```
cpt=0;
for dim=[2 4 8 16 32 64 128]
  [vals,r,c]=qtgetblk(im,qtim3,dim);
  if (false==isempty(vals))
    newvals=[];
    for cpt1=1:size(vals,3)
      newvals=cat(3,newvals,(cpt+cpt1)*ones(dim));
    end
    im4=qtsetblk(im4,qtim3,dim,newvals);
```

```
    cpt=cpt+size(vals,3);
  end
end
estDiff=@(x)x(4)~=x(1)&x(4)~=x(2)&x(4)~=x(3)&...
  x(1)~=x(3)&x(1)~=x(2)&x(2)~=x(3);
nb_regions=cpt;
nb_bits=sum(sum(blkproc(im4,[2 2],estDiff)));
```

Cette nouvelle simulation permet aussi de visualiser les régions avec une autre coloration (sombre pour les petites régions et claire pour les grandes régions)

```
figure(10); imshow(mat2gray(im4));
```

V.16. Adaptez le schéma de compression que vous avez proposé lors de la question V.14 à ces blocs qui sont de taille variable. Tenez compte aussi du nombre de bits supplémentaires requis pour transmettre la décomposition en blocs de taille variable. En terme de débit-distorsion, comment se situe les performances de ce nouveau schéma par rapport à celui de la question V.14.

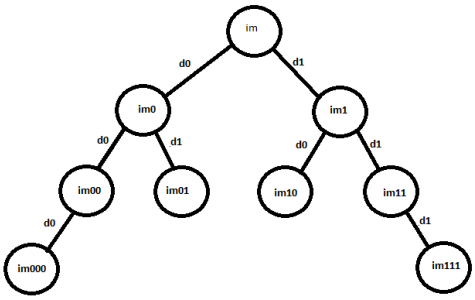

FIGURE 1 – Schéma pour les sous-échantillonnages successifs relatif à la question V.11 ; im désigne soit l'image synthétique choisie soit l'image naturelle choisie ; d0 et d1 désignent respectivement `deci0` et `deci1`.

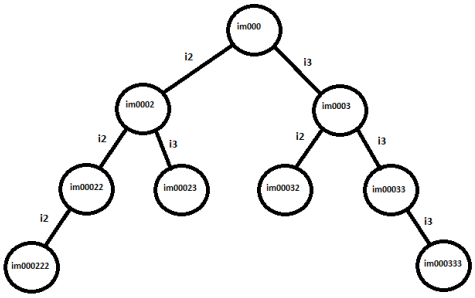

FIGURE 2 – Schéma pour les sur-échantillonnages successifs relatif à la question V.11 ; i2 et i3 désignent respectivement `inte2` et `inte3`.

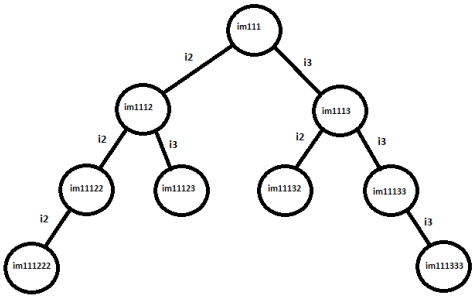

FIGURE 3 – Schéma pour les sur-échantillonnages successifs relatif à la question V.11 ; i2 et i3 désignent respectivement `inte2` et `inte3`.