Traitement de l'Information et Vision Artificielle
Ecole des Ponts ParisTech, 2ème année IMI

TP 6: ONDELETTES 2D, COMPRESSION ET DÉBRUITAGE D'IMAGE

Pour commencer la séance

1. Créez un dossier `TIVA_TP6` sur le bureau.
2. Lancer ensuite `Matlab` et modifier le répertoire de travail en choisissant le répertoire `Bureau/TIVA_TP6` que vous avez créé.

1. Installation de Wavelab

1. Nous ferons ce TP avec le package Wavelab developpé par l'université de Stanford.
2. Créez un dossier `TIVA_TP6/` dans `Bureau/`
3. Téléchargez `Wavelab850.zip` et le fichier `all_shifts_denoising.m` et mettez les dans `TIVA_TP6/`.
4. Décompressez le dossier `Wavelab850.zip`.
5. Dans la ligne de commande de Matlab changez de dossier en utilisant la commande `cd Wavelab850`
6. Ensuite exécutez la commande `WavePath` dans la ligne de commande (qui permettra à toutes les fonctions de Wavelab d'être accessible depuis n'importe quel dossier).
7. Le code demande `Enter the correct path (type 0 to exit):`
8. En réponse à cette requête, donnez le chemin d'accès à Wavelab: normalement `C:/%USERPROFILE%/Bureau/TIVA_TP6/Wavelab850/` et validez avec la toucher enter. Si le code s'est exécuté sans message d'erreur vous pouvez retourner dans le dossier `TIVA_TP6` avec la commande `cd ..`
9. Nous allons utiliser un certain nombre de scripts et de fonctions du package. Pour rappel, vous pouvez taper `help myfunction` pour afficher l'aide de la fonction `myfunction.m` . Vous pouvez aussi taper `type myfunction` pour afficher directement dans la ligne de commande le code source de la fonction. Finalement, en tapant `open myfunction` vous demanderez à Matlab d'ouvrir le code source dans l'éditeur.

10. Le code contient un certain nombre de démos qui nous seront utiles, en particulier celles contenues dans
`/Wavelab850/Workouts/Toons/`
Pour aller dans ce dossier à partir de votre répertoire `Bureau/TIVA_TP6`, faites
`cd /Wavelab850/Workouts/Toons/`
La commande `ls` vous permet d'avoir la liste de tous le fichiers présents dans le dossier. Vous verrez un certain nombre de fichiers `toon_xyz.m` qui sont des petites démos. Si vous tapez `help Contents` dans ce répertoire, vous aurez le sommaire des démos. Je signalerai au fur et à mesure du TP quelles sont les démos utiles.
N'oubliez pas de retournez dans votre propre dossier si vous voulez imprimer des figures et sauver ou utiliser vos scripts.

2. Visualisation des ondelettes 2D

Le but de cette première partie est de comprendre comment est organisée la transformée en ondelettes (ou décomposition sur une base d'ondelettes) d'une image.

1. Commençons par présenter les fonctions principales de Wavelab que nous allons utiliser.

   - Ainsi que le montre les algorithmes de transformée en ondelettes et de transformée inverse une famille d'ondelette est encodée par son filtre miroir. Pour calculer la transformée en ondelette d'une image, il faut donc commencer par calculer ce filtre qu'on passera en argument aux différent algorithmes Le filtre miroir associé à un ondelette est obtenu avec la commande `MakeONFilter` (make orthonormal filter). Par exemple, le filtre miroir pour les ondelettes de Daubechies 4 s'obtient avec
     `qmf=MakeONFilter('Daubechies',4);`
   - Choisissez une famille d'ondelette en vous aidant de l'aide la fonction `MakeONFilter`.
   - Puis utilisez la commande `bar` pour afficher et imprimer le filtre pour cette ondelette. Mettez cette figure dans votre rapport et commentez: quelles différence principale y a-t-il entre les différents filtres.
   - La fonction `FWT2_PO` calcule la *transformée en ondelettes périodisée* (c'est-à-dire en considérant que l'image de départ est périodique comme pour la TFD). Il y a deux autres transformées qui construisent leurs ondelettes légèrement différement mais nous n'utiliserons que celle-ci.
     Sa syntaxe est
     `W=FWT2_PO(I,L,qmf);`
     où `I` est une matrice de niveaux de gris, `L` défini la résolution 2^{-L} la plus grossière de la TO et `qmf` est le filtre miroir associé à l'ondelette choisie.
   - La transformée inverse a quasiment la même syntaxe. En effet,
     `I=IWT2_PO(W,L,qmf);`

permet de reconstruire l'image `I` à partir de la transformée `W`.

2. Pour commencer visualisez les transformées en ondelettes des `toon0231`, `toon0232` et `toon0233`. A chaque fois pour avoir l'explication associée utilisez la commande `help toon023x`.

3. On considère la transformée en ondelettes `W` d'une image 256×256. `W` est aussi une matrice de taille 256×256. On suppose qu'on a calculé une transformée en ondelettes de l'image (donc une transformée 2D) en s'arrêtant à la résolution 16×16.

   Quelles sont dans ce cas les valeurs des coefficients d'échelle j pour lesquels il y aura dans `W` des coefficients associés à l'ondelette $\psi^k_{j,(n_1,n_2)}$? à l'ondelette $\phi_{j,(n_1,n_2)}$? Pour une valeur de j et de k fixée dans quel intervalle varient n_1 et n_2?

   A quelle coordonnée $(m_1, m_2) \in \{1, \ldots, 256\}^2$ se trouve dans `W` le coefficient associé à l'ondelette $\psi^k_{j,(n_1,n_2)}$?

4. Proposez une manière d'obtenir l'image 2D à la résolution 256×256 correspondant à la fonction $\psi^k_{j,(n_1,n_2)}$ en utilisant un simple et unique appel à la fonction qui calcule la transformée inverse de la transformée en ondelettes, c'est-à-dire la fonction `IWT2_PO` pour une matrice de coefficients bien choisie.

5. Qu'obtient-on si on execute la transformée inverse sur une matrice `W` ne contenant qu'un seul coefficient non nul égal à 1? Utilisez cette idée pour construire une image de la fonction $\phi_{j,(n_1,n_2)}$ à la résolution 256×256 pour une valeur de (j, n_1, n_2) qui vous convient (tout en veillant que l'ondelette choisie ne soit pas trop à cheval sur le bord de l'image).

   On pourra visualiser l'ondelette d'une part comme une image avec la fonction `imagesc` d'autre part comme une function en 2D avec la fonction `surf`.

   Précisément, si `I` est la matrice-image de l'ondelette, on pourra utiliser:

```
figure(2)
surf(I);
colormap('copper');
shading interp %donne a la surface un aspect lisse par interpolation
light %ajoute une source lumineuse pour illuminer la surface
lighting gouraud; %calcule le rendu de la diffusion/reflexion de la lumiere
```

6. Faites de même pour une ondelette $\psi^k_{j,(n_1,n_2)}$ pour $k \in \{1, 2, 3\}$ et pour deux valeurs de j: une petite et une grande. Incluez dans votre rapport des exemples d'images/surfaces d'ondelettes représentées pour Haar et Symmlet 8.

7. (Question rapide). Dans les toons `0521-0524`, on s'intéresse à la compression d'une image cérébrale. Quelles sont les conclusions qui ressortent de ces expériences? Est-ce que la reconstruction fonctionne mieux en prenant les coefficients associés aux es-

paces de détails les plus grossiers d'abord, ou en prenant seulement les coefficients d'ondelettes les plus grands? Pensez bien à exécuter `help toon0521` etc pour avoir la description de chaque démo.

8. (Question rapide). Dans les toons `0541-0548`, on s'intéresse à la compression d'une photo de visage. Que montre-t-on ? Quelles sont les conclusions qui ressortent de ces expériences?

3. Débruitage d'image.

On utilisera au choix pour cette partie soit l'image `Lenna` soit l'image `Canaletto` qui sont fournies avec Wavelab et que l'on peut ouvrir avec la fonction `ReadImage`. Voir l'aide de cette fonction pour plus de détails. Nous allons artificiellement bruiter ces images et essayer de les débruiter au mieux.

1. Il est très répandu en traitement d'image d'utiliser le SNR (*Signal to Noise Ratio*) et le PSNR (*Peak Signal to Noise Ratio*) comme des mesures de qualité d'approximation d'une image I par une image approchée I_1. Les définitions du SNR et du PSNR sont :

$$\mathrm{PSNR} = 10\log_{10}\frac{MN\max_{i,j} I(i,j)^2}{\sum_{i,j}|I(i,j)-I_1(i,j)|^2}, \qquad \mathrm{SNR} = 10\log_{10}\frac{\sum_{i,j} I(i,j)^2}{\sum_{i,j}|I(i,j)-I_1(i,j)|^2},$$

   où I et I_1 sont des images de taille $M \times N$ et $\sum_{i,j}$ désigne la somme pour $i \in \{1,\dots,M\}$ et $j \in \{1,\dots,N\}$.

   Écrire une fonction, qui prend comme entrée les deux images I et I_1 et qui calcule les deux valeurs SNR et PSNR correspondantes. Attention, comme le traitement des boucles est très lent en Matlab, il faudra utiliser exclusivement des opérations matricielles.

2. On ajoute à chaque pixel de l'image un bruit gaussien indépendant de celui présent aux autres pixels et d'écart-type $\sigma = 20$ (de sorte que le bruit soit suffisamment important pour dégrader l'image sans que le signal soit noyé dans le bruit). On pourra utiliser la fonction `randn` pour obtenir une matrice d'entrées gaussiennes indépendantes. (En fait le bruit des images a plutôt une distribution de Poisson, mais par souci de simplicité on travaillera avec un bruit gaussien). Afficher les images obtenues.

3. Comme première méthode pour débruiter on calcule la transformée en ondelette de l'image bruitée et on enlève les coefficients d'amplitudes inférieures à un seuil θ. Implémentez une fonction `wavelet_denoise` selon le format `denoised_I=wavelet_denoise(I,th)` utilisant l'ondelette de votre choix et testez avec différentes valeurs du seuil θ (correspondant à l'argument `th`). A chaque fois calculer le SNR et PSNR par rapport à l'image originale. Représentez les résultats sous-forme de courbes (on affichera les deux courbes sur le même graphique).

4. La transformée en ondelette n'est pas invariante par translation. En conséquence, si l'on applique une méthode débruitage à la transformée en ondelettes d'une image et à sa version translatée, les résultats peuvent être très différents. Pour pallier cet inconvénient, on utilise la transformée en ondelette *invariante par translations*. Il s'agit de stocker non-seulement les coefficients de la transformée en ondelette de l'image I, mais aussi d'un certain nombre d'images que l'on obtient à partir de I par une translation. Si l'on utilise une transformée en ondelette avec un niveau de résolution minimale J_0, alors il suffit de considérer les translations par un vecteur $(x, y) \in \{1, \dots, 2^{J_0} - 1\}^2$. En pratique, on peut éviter de considérer toutes les translations possibles et se limiter à un nombre de translations $m^2 \ll (2^{J_0})^2$. Utilisez la fonction `all_shifts_denoising` déjà implémentée et dans laquelle il ne reste qu'à définir la fonction `wavelet_denoise` (normalement construite à la question précédente).

   La fonction `all_shifts_denoising`

   - prend comme entrée une image (bruitée) I, un niveau de seuillage θ et un nombre de translation m,
   - pour chaque $i \in 1, \dots, m^2$,
     - détermine l'image translatée I_i (variable `I1`),
     - applique la fonction `wavelet_denoise` à I_i et ensuite la translation inverse pour obtenir l'image débruitée J_i (variable `denoised_I`),
   - fait la moyenne des images J_i:

$$J = \frac{1}{m^2} \sum_{i=1}^{m^2} J_i.$$

5. Pour différentes valeurs de m (exemple, $m = 1, 2, 4, 8$), calculer le SNR et le PSNR des images débruitées. Le résultat obtenu par la transformée invariante par translation est-il meilleur que celui obtenu par la transformée originale ?

6. (Question BONUS) Dans cette dernière question on essaie de faire du débruitage en suivant le même principe mais avec la transformée en cosinus discret (DCT) plutôt que la transformée en ondelette. Remplacez dans votre code la transformée en ondelette par la DCT. On pourra utiliser la commande `dct2_iv` de Wavelab qui est aussi la commande à utiliser pour inverser la DCT (Voir l'aide de la fonction).