DUT INFO2
2015/2016

Traitement d'images
TP6 : filtre NL-means

Benoît Naegel, Gregory Apou

Dans ce TP nous allons implanter le filtre des moyennes non-locales (Non-Local Means ou NL-Means) et le tester sur des images corrompues par du bruit gaussien.

1 Introduction

L'algorithme des moyennes non-locales, introduit par Buades et. al [1] est un procédé de filtrage non-linéaire visant à supprimer le bruit, et plus spécifiquement le bruit gaussien. Le principe de cette méthode est d'utiliser la redondance présente naturellement dans les images pour filtrer chaque point (voir figure 1).

Dans la version théorique, pour une image f, la valeur de chaque point x_i de l'image est obtenue en calculant une combinaison linéaire des valeurs de tous les autres points x_j ($j \neq i$) de l'image pondérés par une valeur de similarité entre le point filtré x_i et x_j :

$$NL(f)(x_i) = \sum_{x_j \in E} w(x_i, x_j) f(x_j).$$

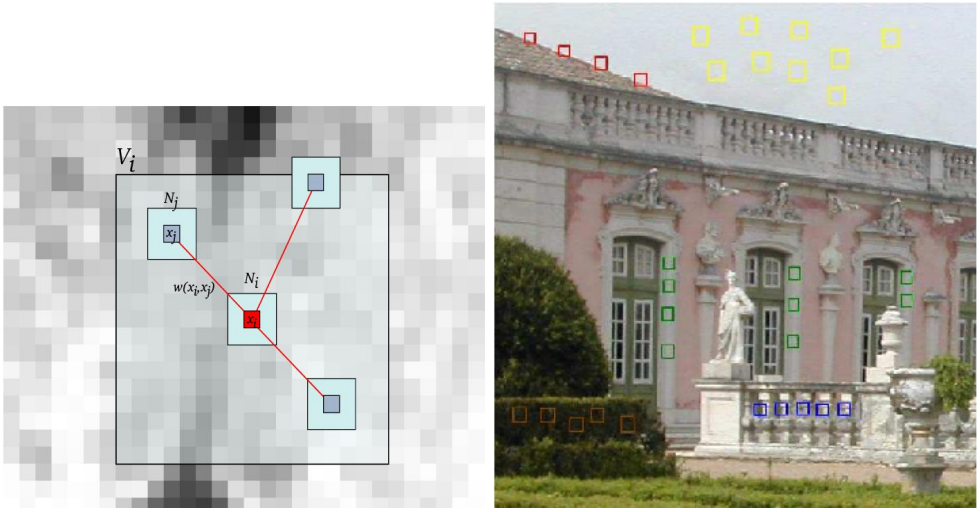


FIGURE 1 – a) La valeur du point central est calculée à partir d'une combinaison linéaire des valeurs de tous les points contenus dans l'espace de recherche. Plus la configuration du voisinage autour des deux points est similaire, plus la contribution (poids) du point est importante. b) Patchs d'images similaires (ayant la même configuration de voisinage) dans une image.

Le poids assigné à chaque valeur $f(x_j)$ pour la restauration du point x_i est défini par :

$$w(x_i, x_j) = \frac{1}{Z(i)} e^{-\frac{\|\mathbf{f}(N_i) - \mathbf{f}(N_j)\|^2_{2,a}}{h^2}},$$

où $\mathbf{f}(N_i)$ définit le vecteur contenant les intensités de f dans le voisinage carré centré en x_i de taille $2D+1$, $\|.\|^2_{2,a}$ est la distance euclidienne pondérée par un noyau gaussien d'écart-type a, $Z(i) = \sum_j e^{\frac{\|\mathbf{f}(N_i) - \mathbf{f}(N_j)\|^2_{2,a}}{h^2}}$ est

le facteur de normalisation. Le paramètre h contrôle la pente de la fonction exponentielle. La fenêtre utilisée pour calculer l'index de similarité est généralement prise carrée, de taille $(2D+1)^2$ et centrée.

Cette méthode est très coûteuse algorithmiquement : la complexité est de ($\mathcal{O}(N^4(2D+1)^2)$, avec N^2 la taille de l'image et $(2D+1)^2$ la taille de la fenêtre de similarité).

Pour être utilisable, cette méthode doit se fonder sur des heuristiques et certaines simplifications :

— Généralement, l'ensemble des points x_j utilisé pour filtrer le point x_i est restreint à une fenêtre carrée S de taille $(2M+1)^2$ centrée autour du point x_i, appelé espace de recherche. Réduire la taille de l'espace de recherche permet d'augmenter significativement la rapidité de l'algorithme, mais réduit la probabilité de trouver des patchs d'image similaires, ce qui réduit donc l'efficacité du débruitage.

— Une autre optimisation, introduite par Coupé et al. [2], consiste à supprimer de l'espace de recherche tous les points x_j dont la similarité avec le point x_i est trop faible. L'idée est de précalculer, avant le filtrage, les moments de premier ordre (moyenne et écart-type) de chaque point de l'image.

2 Implantation

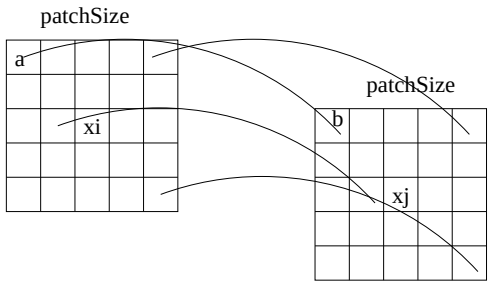
La fonction principale `computeNLMeans` pourra s'appuyer sur les fonctions intermédiaires suivantes :

— `double computeSimilarity(int x1, int y1, int x2, int y2)` qui retourne la similarité entre deux patchs d'image centrés en (x_1, y_1) et (x_2, y_2). Si les patchs sont parfaitement identiques, cette fonction retourne 0. On considère que les points situés en dehors de l'image ont une contribution nulle dans le calcule.

— `double computeWeight(int x1, int y1, int x2, int y2)` qui calcule le poids $w(x_i, x_j)$ avec $x_i = (x_1, y_1), x_j = (x_2, y_2)$ en utilisant la formule donnée plus haut. Cette méthode s'appuiera sur la fonction précédente pour calculer la similarité entre deux patchs.

2.1 Fonction *computeSimilarity*

Cette méthode calcule la similarité entre deux patchs (sous-images). Cette similarité est fondée sur la formule suivante : $\|\mathbf{f}(N_i) - \mathbf{f}(N_j)\|^2$, avec N_i et N_j les fenêtres de taille `patchSize` centrées sur les points x_i et x_j de coordonnées $(x_1, y_1), (x_2, y_2)$. Concrètement, cette méthode calcule la somme des différences au carré entre tous les points correspondants des deux sous-images :



similarity=$(a-b)^2 + ...$

2.2 Fonction *computeWeight*

Cette méthode calcule le poids défini par $w(x_i, x_j) = e^{-\frac{sim(x1,y1,x2,y2)}{h^2}}$, avec $sim(x1, y1, x2, y2)$ la valeur de retour de la fonction `computeSimilarity`.

2.3 Fonction principale *computeNLMeans*

Cette fonction calcule pour tous les points de l'image la nouvelle valeur affectée à l'image de sortie selon la formule donnée au début :

$$NL(f)(x_i) = \sum_{x_j \in S} w(x_i, x_j) f(x_j),$$

avec w la valeur de retour de la fonction `computeWeight` et pour tout $x_i \neq x_j$. S est la fenêtre de l'espace de recherche, centrée en x_i (point courant), de taille `windowSize`×`windowSize`. La valeur obtenue en chaque point doit être normalisée par la somme des poids obtenus (facteur Z) de la formule. Les paramètres d'entrée de cette fonction sont :

— l'image d'entrée ;

— `windowSize` : entier impair définissant la taille de l'espace de recherche ($windowSize = 2M+1$, cf. plus haut) ;

— `patchSize` : entier impair définissant la taille d'un patch ($patchSize = 2D + 1$, cf. plus haut) ;
— `double h` : paramètre h (cf. plus haut) définissant la "force" du filtrage. Plus l'écart-type du bruit gaussien est grand, plus ce paramètre doit-être élevé.

Un cas particulier existe lors du calcul de la distance entre le point courant (x, y) et lui-même. On affecte à ce poids la valeur maximale des poids calculés dans la fenêtre S. Ceci est résumé dans le pseudo-code ci-dessous.

2.4 Expérimentations

Tester votre filtre avec une image corrompue par du bruit gaussien d'écart-type $10, 15, 25$. Vous pourrez utiliser les paramètres suivants : `patchSize=7`, `windowSize=21`, `h=50.0` Plus l'écart-type est grand, plus il faudra augmenter ces paramètres.

Annexe

Voici l'algorithme du filtre en pseudo-code :

```
 pour tous les points (x,y) de l'image
    sumWeights=0;
    value=0;
    weightMax=0;
   pour tous les points (i,j) de la fenêtre S centrée en (x,y)
      si (x,y)!=(i,j)
         w=computeWeight(x,y,i,j);
         weightMax=max(weightMax,w);
         sumWeights+=w;
         value+=w * imageIn(i,j);  // valeur de l'image d'entrée au point (i,j)
      finsi
   finpour
      value+=weightMax*imageIn(x,y); // contribution du pixel courant (poids maximum cal
      value/=sumWeights;
      imageOut(x,y)=value; // écriture dans l'image de sortie
finpour
```

Références

[1] Buades, A., Coll, B., Morel, J., 2005. A review of image denoising algorithms, with a new one. Multiscale Modeling and Simulation 4 (2), 490–530.

[2] Coupé, P., Yger, P., Prima, S., Hellier, P., Kervrann, C., Barillot, C., 2008. An optimized blockwise non local means denoising filter for 3D magnetic resonance images. IEEE Transactions on Medical Imaging 27 (4), 425–441.