Traitement du signal 1D et 2D
Année académique 2022-2023

Jean-Marc Wagner

Projet
Développement d'une librairie Java de traitement d'images
Application « IsilImageProcessing »

Etape 1 : Filtrage linéaire global (traitements fréquentiels) et local

Construire une classe **FiltrageLinaireGlobal** (dans un sous-package de **ImageProcessing.Lineaire**) contenant les méthodes statiques suivantes :

- public static int[][] **filtrePasseBasIdeal**(int[][] image,int frequenceCoupure) : réalisant le filtrage passe-bas de l'image en passant par sa transformée de Fourier et en utilisant une fenêtre de forme circulaire de rayon égal à la fréquence de coupure.
- public static int[][] **filtrePasseHautIdeal**(int[][] image,int frequenceCoupure) : réalisant le filtrage passe-haut de l'image en passant par sa transformée de Fourier et en utilisant une fenêtre de forme circulaire de rayon égal à la fréquence de coupure.
- public static int[][] **filtrePasseBasButterworth**(int[][] image,int frequenceCoupure,int ordre) : réalisant le filtrage passe-bas de l'image en passant par sa transformée de Fourier et en utilisant la fonction de transfert de Butterworth correspondante.
- public static int[][] **filtrePasseHautButterworth**(int[][] image,int frequenceCoupure, int ordre) : réalisant le filtrage passe-haut de l'image en passant par sa transformée de Fourier et en utilisant la fonction de transfert de Butterworth correspondante.

Ajouter un menu à l'application, appelé « Filtrage linéaire ». Dans ce menu, un sous-menu « Global » doit apparaître. Celui-ci contiendra les 4 opérations citées ci-dessus. La fréquence de coupure pourra être choisie par l'utilisateur (ainsi que l'ordre du filtre dans le cas de Butterworth).

Construire une classe **FiltrageLineaireLocal** (dans le sous-package **ImageProcessing.Lineaire**) contenant les méthodes statiques suivantes :

- public static int[][] **filtreMasqueConvolution**(int[][] image, double [][] masque) : réalisant le filtrage local de l'image en utilisant le masque de convolution, celui-ci étant une matrice carrée quelconque de dimension nxn, avec n impair.
- public static int[][] **filtreMoyenneur**(int[][] image, int tailleMasque) : réalisant un filtrage moyenneur de l'image avec un masque de convolution de dimension tailleMasque x tailleMasque. Cette méthode peut appeler la méthode filtreMasqueConvolution avec le masque adéquat.

Ajouter au menu « Filtrage linéaire » un sous-menu « Local ». Celui-ci contiendra les deux opérations ci-dessus. Pour la première, le masque de convolution pourra être encodé par l'utilisateur.

Etape 2 : Traitement non-linéaire (opérateurs morphologiques)

Pour tous les opérateurs morphologiques à implémenter, on se limitera à des éléments structurants carrés de taille nxn, où n est impair.

Construire une classe **MorphoElementaire** (dans le sous-package **ImageProcessing.NonLineaire**) contenant les méthodes statiques suivantes :

- public static int[][] **erosion**(int [][] image,int tailleMasque) : réalisant l'érosion de l'image.
- public static int[][] **dilatation**(int [][] image,int tailleMasque) : réalisant la dilatation de l'image.
- public static int[][] **ouverture**(int [][] image,int tailleMasque) : réalisant l'ouverture de l'image. Evidemment, cette méthode peut appeler les deux premières.
- public static int[][] **fermeture**(int [][] image,int tailleMasque) : réalisant la fermeture de l'image. Evidemment, cette méthode peut appeler les deux premières.

Ces méthodes doivent être valables pour des images binaires (0 → niveau de gris 0, 1 → niveau de gris 255) **et** des images en niveaux de gris.

Ajouter un menu « Traitement non-linéaire » contenant un sous-menu « Elementaire ». Celui-ci contiendra les 4 opérations ci-dessus. La taille de l'élément structurant pourra être choisie par l'utilisateur.

Construire une classe **MorphoComplexe** (dans le sous-package **ImageProcessing.NonLineaire**) contenant les méthodes suivantes :

- public static int[][] **dilatationGeodesique**(int[][] image,int[][] masqueGeodesique, int nbIter) : réalise la dilatation géodésique (de taille nbIter) de l'image conditionnellement au masque géodésique, nbIter étant plus grand ou égal à 1.
- public static int[][] **reconstructionGeodesique**(int[][] image, int[][] masqueGeodesique) : réalise la reconstruction géodésique de l'image conditionnellement au masque géodésique.
- public static int[][] **filtreMedian**(int[][] image, int tailleMasque) : réalisant le filtrage médian de l'image.

Les deux premières méthodes s'appliquent essentiellement à des images binaires tandis que le filtre médian ne s'applique qu'à des images en niveaux de gris.

Ajouter au menu « Traitement non-linéaire » un sous-menu « Complexe » (rien à voir avec les nombres complexes bien sûr !). Celui-ci contiendra les 3 opérations ci-dessus. Le masque géodésique et la taille de l'élément structurant pourront être choisis par l'utilisateur.

Etape 3 : Rehaussement par manipulation de l'histogramme

Dans le sous-package **ImageProcessing.Histogramme** se trouve une classe **Histogramme** contenant déjà une méthode statique **Histogramme256** permettant de calculer l'histogramme d'une image en 256 niveaux de gris. On vous demande d'ajouter à cette classe les méthodes statiques suivantes (pour des images en 256 niveaux de gris) :

- public static int **minimum**(int[][] image) : permettant de calculer la valeur minimum de l'image.
- public static int **maximum**(int[][] image) : permettant de calculer la valeur maximum de l'image.
- public static int **luminance**(int[][] image) : permettant de calculer la luminance de l'image en utilisant la formule (1.28) des notes de cours.
- public static double **contraste1**(int[][] image) : permettant de calculer le contraste de l'image en utilisant la formule (1.29) des notes de cours (il s'agit en fait de l'écart-type).
- public static double **contraste2**(int[][] image) : permettant de calculer le contraste de l'image en utilisant la formule (1.30) des notes de cours.

Dans le menu « Histogramme » de l'application, ajouter un sous-menu « Afficher les paramètres de l'image » qui permettra d'ouvrir une boîte de dialogue contrant les diverses informations calculées ci-dessus.

Ajouter à la classe **Histogramme** les méthodes statiques suivantes :

- public static int[][] **rehaussement**(int[][] image, int[] courbeTonale) : permettant de rehausser l'image en utilisant la courbe tonale fournie. Celle-ci est représentée par un vecteur de 256 entiers compris entre 0 et 255. Un niveau de gris i de l'image de départ est remplacé par la valeur du vecteur courbeTonale situé à l'indice i.
- public static int[] **creeCourbeTonaleLineaireSaturation**(int smin, int smax) : permettant de créer une courbe tonale correspondant à la transformation linéaire avec saturation (formule 1.32 des notes de cours). Si on veut créer une courbe tonale pour une transformation linéaire sans saturation, il suffira d'appeler la méthode avec smin = minimum de l'image et smax = maximum de l'image.
- public static int[] **creeCourbeTonaleGamma**(double gamma) : permettant de créer une courbe tonale correspondant à la correction gamma (formule 1.33 des notes de cours)
- public static int[] **creeCourbeTonaleNegatif**() : permettant de créer la courbe tonale correspondant au négatif de l'image.
- public static int[] **creeCourbeTonaleEgalisation**(int[][] image) : permettant de créer la courbe tonale correspondant à l'égalisation de l'histogramme.

Dans le menu « Histogramme » de l'application, ajouter des sous-menus correspondant aux transformations linéaire, linéaire avec saturation, gamma et à l'égalisation de l'histogramme de l'image, ainsi que le négatif de l'image Arrangez-vous pour que l'on puisse visualiser les histogrammes de l'image avant et après rehaussement.

Etape 4 : Détection de contours et segmentation

Dans le sous-package **ImageProcessing.Contours**, construire une classe **ContoursLineaire** contenant les méthodes statiques suivantes :

- public static int[][] **gradientPrewitt**(int[][] image,int dir) : calculant le gradient de Prewitt de l'image. Si dir=1, le calcul du gradient se fera dans la direction horizontale. Si dir=2, le calcul du gradient se fera dans la direction verticale.
- public static int[][] **gradientSobel**(int[][] image,int dir) : calculant le gradient de Sobel de l'image. Si dir=1, le calcul du gradient se fera dans la direction horizontale. Si dir=2, le calcul du gradient se fera dans la direction verticale.
- public static int[][] **laplacien4**(int[][] image) : calculant le laplacien (formule 1.52 des notes de cours) de l'image.
- public static int[][] **laplacien8**(int[][] image) : calculant le laplacien (formule 1.53 des notes de cours) de l'image.

Ajouter à l'application un menu « Contours ». Dans ce menu, ajouter un sous-menu « Linéaire » proposant les 4 opérations précédentes.

Aristide

Dans le sous-package **ImageProcessing.Contours**, construire une classe **ContoursNonLineaire** contenant les méthodes statiques suivantes :

- public static int[][] **gradientErosion**(int[][] image) : calculant le gradient d'érosion de l'image.
- public static int[][] **gradientDilatation**(int[][] image) : calculant le gradient de dilatation de l'image.
- public static int[][] **gradientBeucher**(int[][] image) : calculant le gradient de Beucher de l'image.
- public static int[][] **laplacienNonLineaire**(int[][] image) : calculant le laplacien non-linéaire de l'image (formule 1.60 des notes de cours).

Dans le menu « Contours », ajouter un sous-menu « Non-linéaire » proposant les 4 opérations précédentes.

Dans le sous-package **ImageProcessing.Seuillage**, construire une classe **Seuillage** contenant les méthodes statiques suivantes :

- public static int[][] **seuillageSimple**(int[][] image, int seuil) : réalisant le seuillage simple de l'image en niveaux de gris. La sortie est donc une image binaire.
- public static int[][] **seuillageDouble**(int[][] image,int seuil1, int seuil2) : réalisant le seuillage multiple à deux seuils de l'image. La sortie est donc une image comportant 3 niveaux de gris distincts (au choix).
- public static int[][] **seuillageAutomatique**(int[][] image) : réalisant le seuillage automatique de l'image selon l'algorithme des notes de cours (section 1.6.4).

Ajouter à l'application un menu « Seuillage » proposant les 2 (ou 3) opérations précédentes.

Etape 5 : Applications

Pour les exercices qui suivent, vous pouvez (ce serait très préférable !) utiliser les différentes méthodes des 4 premières étapes. Néanmoins, vous pouvez introduire d'autres classes et méthodes statiques si cela vous est nécessaire (par exemple : soustraction de deux images).

1 et 2 : Ari
3 et 4 : clara
5 et 6 : moi

1. Réduire au maximum le bruit présent dans l'image **lenaBruit.png**. *Suggestions* : décomposition RGB, filtrage linéaire, filtrage non-linéaire.

2. Rehausser l'image **lenaAEgaliser.jpg** en réalisant une égalisation d'histogramme. Tester les deux possibilités : (a) Réaliser l'égalisation de l'histogramme des 3 images RGB séparément. (b) Egaliser l'histogramme de l'image « luminance » de l'image et appliquer cette même égalisation aux 3 composantes RGB de l'image (même courbe tonale). Quelle méthode donne le meilleur résultat ?

3. A partir de l'image **petitsPois.png**, créer deux images binaires, l'une comportant les poids bleus, l'autre les pois rouges. *Suggestions* : érosion, dilatation, ouverture, fermeture, seuillage, opérations géodésiques.

4. A partir de l'image **balanes.png**, créer deux images en niveaux de gris, l'une comportant les balanes de grande taille, l'autre les balanes de petite taille. *Suggestions* : érosion, dilatation, ouverture, fermeture, seuillage, opérations géodésiques.

5. A partir de l'image **tools.png**, réaliser une segmentation binaire afin d'en extraire les outils. Le résultat doit donc être une image binaire valant 1 à l'endroit des outils et 0 à l'extérieur. *Suggestions* : segmentation, érosion, dilatation, ouverture, fermeture.

6. Extraire le petit vaisseau spatial de l'image **vaisseaux.jpg** et le coller dans l'image **planete.jpg** (au même endroit). Sauver le résultat sous le nom **synthese.png**. Créer l'image **synthese2.png** ajoutant un contour rouge (épaisseur d'un pixel) au petit vaisseau. *Suggestions* : érosion, dilatation, ouverture, fermeture, seuillage, opérateurs géodésiques, détection de contours.

7. Détecter les contours des tartines de l'image **Tartines.jpg**. Une fois détectés, tracer ces contours en vert sur l'image de départ.

Ajouter à l'application principale un menu « Applications » comportant 8 items permettant de lancer la résolution de ces exercices et d'afficher les résultats.