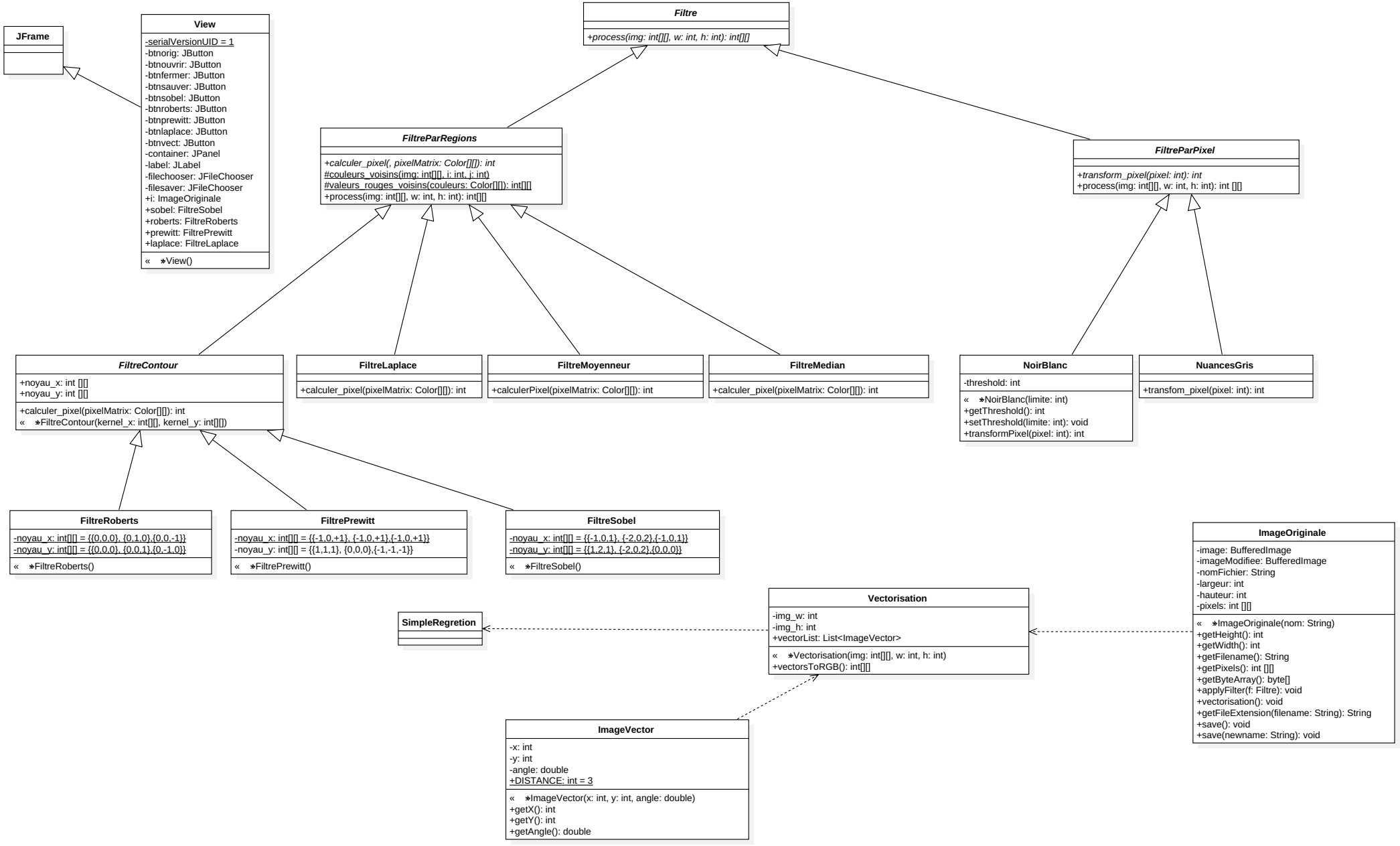
Licence 3 de Sciences, mention Informatique

Programmation Orientée Objets 2

Filtre de détection de contour

Gomes Fernandes Caty
Lamri Manal

Model::Main


JFrame
View
-serialVersionUID = 1
-btnorig: JButton
-btnouvrir: JButton
-btnfermer: JButton
-btnsauver: JButton
-btnsobel: JButton
-btnroberts: JButton
-btnprewitt: JButton
-btnlaplace: JButton
-btnvect: JButton
-container: JPanel
-label: JLabel
-filechooser: JFileChooser
-filesaver: JFileChooser
+i: ImageOriginale
+sobel: FiltreSobel
+roberts: FiltreRoberts
+prewitt: FiltrePrewitt
+laplace: FiltreLaplace
« constructor »View()
Filtre
+process(img: int[][], w: int, h: int): int[][]
FiltreParRegions
+calculer_pixel(, pixelMatrix: Color[][]): int
#couleurs_voisins(img: int[][], i: int, j: int)
#valeurs_rouges_voisins(couleurs: Color[]): int[][]
+process(img: int[][], w: int, h: int): int[][]
FiltreParPixel
+transform_pixel(pixel: int): int
+process(img: int[][], w: int, h: int): int [][]
FiltreContour
+noyau_x: int [][]
+noyau_y: int [][]
+calculer_pixel(pixelMatrix: Color[][]): int
« constructor »FiltreContour(kernel_x: int[][], kernel_y: int[][])
FiltreLaplace
+calculer_pixel(pixelMatrix: Color[][]): int
FiltreMoyenneur
+calculerPixel(pixelMatrix: Color[][]): int
FiltreMedian
+calculer_pixel(pixelMatrix: Color[][]): int
NoirBlanc
-threshold: int
« constructor »NoirBlanc(limite: int)
+getThreshold(): int
+setThreshold(limite: int): void
+transformPixel(pixel: int): int
NuancesGris
+transfom_pixel(pixel: int): int
FiltreRoberts
-noyau_x: int[][] = {{0,0,0}, {0,1,0},{0,0,-1}}
-noyau_y: int[][] = {{0,0,0}, {0,0,1},{0,-1,0}}
« constructor »FiltreRoberts()
FiltrePrewitt
-noyau_x: int[][] = {{-1,0,+1}, {-1,0,+1},{-1,0,+1}}
-noyau_y: int[][] = {{1,1,1}, {0,0,0},{-1,-1,-1}}
« constructor »FiltrePrewitt()
FiltreSobel
-noyau_x: int[][] = {{-1,0,1}, {-2,0,2},{-1,0,1}}
-noyau_y: int[][] = {{1,2,1}, {-2,0,2},{0,0,0}}
« constructor »FiltreSobel()
ImageOriginale
-image: BufferedImage
-imageModifiee: BufferedImage
-nomFichier: String
-largeur: int
-hauteur: int
-pixels: int [][]
« constructor »ImageOriginale(nom: String)
+getHeight(): int
+getWidth(): int
+getFilename(): String
+getPixels(): int [][]
+getByteArray(): byte[]
+applyFilter(f: Filtre): void
+vectorisation(): void
+getFileExtension(filename: String): String
+save(): void
+save(newname: String): void
SimpleRegretion
Vectorisation
-img_w: int
-img_h: int
+vectorList: List<ImageVector>
« constructor »Vectorisation(img: int[][], w: int, h: int)
+vectorsToRGB(): int[][]
ImageVector
-x: int
-y: int
-angle: double
+DISTANCE: int = 3
« constructor »ImageVector(x: int, y: int, angle: double)
+getX(): int
+getY(): int
+getAngle(): double

1 Choix de modélisation

L'application est écrite dans le langage Java.

Les classes se divisent en trois groupes : les classes nécessaires pour l'interface graphique, la classe contenant l'image en cours de traitement, et les classes qui définissent ces traitements.

Tout procédé qui transforme une image en une autre de même dimension hérite de la classe abstraite "Filtre". On distingue entre les filtres qui calculent la valeur de chaque pixel indépendamment de leurs voisins, et ceux qui ont besoin de connaître leur entourage. Ceci donne les deux sous-classes abstraites "FiltreParRegions" et "FiltreParPixel", qui dépendent des classes filles pour le calcul de chaque pixel. La classe "FiltreContour" sert à factoriser l'implémentation des filtres qui reposent sur des matrices de convolution.

Enfin, la classe "Vectorisation" effectue la conversion d'une image en pixels vers une image composée de vecteurs, et vice versa. L'image vectorisée est composée de segments, qui sont modélisés dans la classe "ImageVecteur".

2 Utilisation du programme

L'application nécessite la bibliothèque *Commons Math* de Apache et doit donc être compilé avec

```
javac filtres_contour/*.java -classpath commons-math3-3.6.1.jar
```

et lancé avec

```
java -cp commons-math3-3.6.1.jar:. filtres_contour/ApplicationFiltres
```

depuis le répertoire principal.

Le bouton "Ouvrir" permet de choisir l'image à traiter, qui s'affiche alors au milieu de la fenêtre.

Les boutons "Sobel", "Prewitt", "Roberts" et "Laplace" appliquent le filtre de contour correspondant à l'image originale. L'image résultante apparaît au milieu de la fenêtre, mais le fichier original de l'image reste inchangé. Cette dernière peut être visionnée avec le bouton "Originale".

Le bouton "Sauvegarder" permet de sauvegarder l'image affichée dans le fichier de votre choix.

Contrairement aux filtres de contour, le bouton "Vectoriser" vectorise l'image affichée à l'écran. Le résultat affiché sera la conversion de l'image vectorisée en image composée de pixels.

3 Explication des algorithmes

Une image est considérée comme une matrice de pixels en codage RVB, chaque pixel pouvant donc être décomposé en ses valeurs de rouge, vert et bleu. Le pixel d'indices (0,0) se trouve au coin supérieur gauche de l'image, les x correspondent aux colonnes et les y correspondent aux lignes.

FIGURE 1 – Orientation des axes

Avant l'application d'un filtre de contour, il faut se ramener à une image en nuances de gris,

ç.à.d. une image où pour chaque pixel on a R=V=B. Sachant que l'oeuil humain perçoit mieux les couleurs vertes mais est moins sensible au bleu, on effectue pour chaque pixel la moyenne pondérée suivante :

$$0.299 * R + 0.587 * G + 0.114 * B$$

Ceci donne la nouvelle valeur de R, V et B. Dans la suite on ne considérera que la composante rouge lors des traitements.

3.1 Filtres de contour

Un contour d'une image peut être vu comme une transition subite du niveau de gris. Pour détecter cette variation, on cherche le maximum du gradient d'intensité lumineuse pour chaque pixel. Les coordonnées de ce gradient sont les dérivées selon l'horizontale et la verticale. Concrètement, on appliquera deux masques à l'image pour obtenir une valeur approximative de ces deux dérivées en chaque pixel. Un masque peut être vu comme une matrice contenant des poids par lesquels on multiplie la valeur du pixel courant et de ses voisins.

Pour le filtre de Roberts, on a les masques suivants :

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

La valeur en rouge correspond à la position du pixel pour lequel on souhaite calculer le gradient. Ici, les dérivées pour le pixel à la position (x,y) se calculent de la manière suivante :

$$G_x = I(x, y) - I(x + 1, y + 1) \tag{1}$$

$$G_y = I(x + 1, y) - I(x, y + 1) \tag{2}$$

I(i,j) correspond à la valeur rouge du pixel à la position (i,j) de l'image originale. La valeur finale de R, V et B est l'amplitude du gradient :

$$G = \sqrt{G_x^2 + G_y^2} \tag{3}$$

Le principe reste identique pour le filtre de Sobel et le filtre de Prewitt, mais on applique les masques suivants (c = 2 pour Sobel, et c = 1 pour Prewitt), avant d'effectuer le calcul (3).

$$\begin{bmatrix} -1 & 0 & 1 \\ -c & 0 & c \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -c & -1 \\ 0 & 0 & 0 \\ 1 & c & 1 \end{bmatrix}$$

Le filtre de Laplace applique aussi un seul masque sans calculer la norme euclidienne :

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

3.2 Filtres de réduction de bruit

Trop de bruit numérique peut perturber la recherche des contours. Pour enlever les grains de l'image, on peut appliquer soit un filtre moyenneur, soit un filtre médian.

Pour chaque pixel, le filtre moyenneur calcule les moyennes des valeurs rouges, vertes et bleues de ce pixel et de ses 8 voisins. Le pixel de l'image non bruitée est composée de ces moyennes.

Le filtre médian prend aussi les valeurs RVB séparées du pixel courant et de ses huit voisins, puis il ordonne chaque composante par valeur croissante et choisit la médiane pour chacune.

3.3 Vectorisation

Une autre manière de représenter une image consiste à la décomposer en formes géométriques, qui sont caractérisées par une couleur, un point de départ, et une taille.

Pour simplifier la vectorisation, notre programme décompose une image en nuances de gris en un ensemble de segments de même taille et de couleur blanche. Les segments se trouveront sur un fond noir.

D'abord on applique le filtre "FiltreNoirBlanc" pour attribuer à chaque pixel un couleur blanche si sa valeur dépasse un seuil donné, ou noir sinon.

L'image noir et blanc sera ensuite "découpée" en carrés qui font la même taille que les segments. En effet, pour chaque carré, on cherche à trouver une droite qui passe par les pixels blancs. Ceci est effectué à l'aide des moindres carrés, où les positions des pixels sont nos observations. La bibliothèque *Commons Math* de Apache contient la classe "SimpleRegression", qui calcule la droite de régression à partir de ces observations. Pour chaque droite trouvée, on ajoute le vecteur à l'ensemble des vecteurs qui représente notre image.

Pour re-transformer l'ensemble de vecteurs en une image avec des pixels, on commence par une image noire. Pour chaque vecteur de l'ensemble, on dessine le segment en question sur l'image.

4 Répartition du travail

Manal était responsable pour l'interface utilisateur graphique ainsi que les diagrammes d'héritage et des classes, et le rapport. Caty était chargée de trouver les algorithmes nécessaires, qu'elle a expliqué ici dans le rapport.