Ecole Nationale Supérieure
d'Informatique et de Mathématiques
Appliquées

Projet de Wavelets

Détection de contours basé sur la transformée en ondelettes directionnelles

Éliane Casassa

2021/2022

Table des matières

1 Introduction

La détection de contours constitue souvent une étape préliminaire à de nombreuses applications en traitement d'images comme la reconnaissance d'objets, de formes ainsi que la compression d'images. En effet les contours sont les parties les plus informatives d'une image. Il existe beaucoup d'algorithmes permettant la détection des contours mais les plus utilisés sont les algorithmes de Prewitt, de Sobel et de Canny. Ce sont des algorithmes basés sur le calcul des gradients des images. Mallat s'est également penché sur la question en utilisant la transformée en ondelettes.

Pour ma part, j'ai choisi d'étudier le papier *An edge detection approach based on directional wavelet transform* publié en 2009 par Zhen Zhang, Siliang Ma, Hui Liu et Yuexin Gong [2]. Ce n'est que plus tard que j'ai compris que ce papier s'est très fortement inspiré des papiers de Vladan Velisavljevic sur les Directionlets [1] et qu'il manquait d'explications. Malgré cela j'ai décidé de l'implémenter mais en y apportant des changements ainsi que davantage de contenu.

Les modifications apportés par ce papier consistent principalement à augmenter le nombre de directions pour lesquelles on calcule la transformée en ondelettes tout en gardant le caractère séparable de la transformée en ondelettes 2D (pour la simplicité des calculs).

La suite de ce rapport est constitué de deux parties. On développera dans un premier temps l'algorithme en lui-même ainsi que son implémentation informatique puis dans un second temps on comparera cet algorithme avec l'algorithme de Canny ainsi que la transformée en ondelettes classique (suivant les directions horizontales et verticales seulement).

2 Implémentation du papier

Dans cette partie j'ai souhaité présenter et développer les étapes clés de l'algorithme. Un algorithme type de détection de contours basé sur le gradient pourrait se décrire par quatre étapes principales :

1. Lissage de l'image
2. Calcul du gradient en chaque pixel
3. Détection des maximums locaux
4. Et enfin un seuillage des maximums locaux précédents

Les nouveautés de l'algorithme se répercutent sur les étapes 2 et 3 car le papier définit différemment le gradient en chaque point.

2.1 Lissage de l'image

Avant d'effectuer la détection de contours, la majorité des algorithmes lissent les images afin de réduire l'influence du bruit. Pour cela j'ai utilisé comme proposé par le papier un filtre gaussien. Il faut alors convoluer l'image avec le filtre choisit.

Rappelons tout d'abord la définition mathématique de la convolution discrète en 2 dimensions entre les fonctions A et B :

$$(A * B)(i,j) = \sum_{k=-\infty}^{+\infty} \sum_{l=-\infty}^{+\infty} A(i-k, j-l) \times B(k,l)$$

Cette formule devient :

$$\boxed{(A * B)[i,j] = \sum_{k=-\lfloor n/2 \rfloor}^{\lfloor n/2 \rfloor} \sum_{l=-\lfloor m/2 \rfloor}^{\lfloor m/2 \rfloor} A[i-k, j-l] \times B[k+\lfloor n/2 \rfloor, l+\lfloor m/2 \rfloor]}$$

si l'on considère que A et B sont 2 matrices (commençant à l'indice $(0,0)$) et que B, de dimension $n \times m$, est plus petite que A. Dans notre cas où A représenterait l'image et B le filtre gaussien, on peut interpréter cette convolution en un point (i,j) comme la moyenne des pixels autour de lui, pondéré par les coefficients du filtre.

La fonction gaussienne s'écrit :

$$f(x,y) = \frac{1}{2\pi\sigma^2} \exp \frac{-x^2-y^2}{2\sigma^2} \text{ avec } \sigma \text{ l'écart-type}$$

On remplit alors la matrice B avec le centre de la gaussienne au centre de la matrice en choisissant la taille de la matrice ainsi que l'écart-type de la gaussienne.

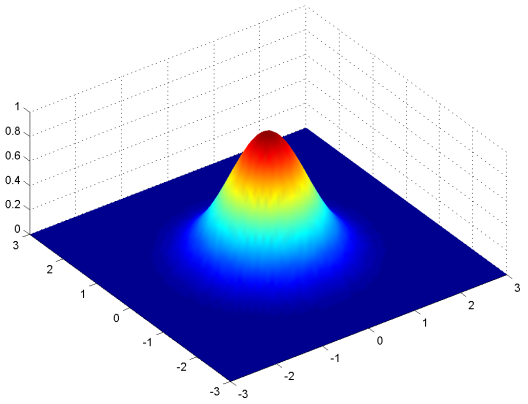

FIGURE 1 – Représentation visuelle de la matrice B

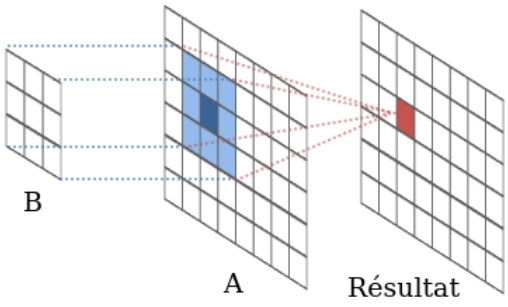

FIGURE 2 – Convolution en chaque pixel

Au plus l'écart-type sera grand au plus la gaussienne sera aplatie et donc au plus l'image sera floue car les pixels autour du pixel considéré auront davantage de poids.

Cependant la formule vu précédemment ne fonctionne pas sur les bords, il faut donc trouver des solutions pour y remédier. J'ai considéré et implémenté trois solutions possibles :

— Considérer une valeur constante (0 ou 1) en dehors de l'image.
— Considérer que la matrice A est périodique.
— Considérer que les pixels en dehors de l'image ont la même valeur que le pixel où nous appliquons la convolution.

Dans tous les cas, de petites perturbations sont visibles sur les limites de l'image. Il n'existe pas de "meilleures" solutions, cela dépend des images considérées. Ci-dessous un exemple avec l'utilisation de la troisième solution, répétition du pixel actuel en dehors de l'image.

FIGURE 3 – Image initiale

FIGURE 4 – Lissée, kernel de taille (7, 7), $\sigma = 5$

La complexité de cet algorithme est le nombre d'opérations que nous devons effectuer pour calculer la convolution. Nous considérons ici la matrice originale de l'image de dimensions (p, q) et le noyau B de dimension (n, m). Pour l'implémentation naïve décrite ci-dessus nous devons calculer la double somme pour chaque pixel et donc la complexité s'écrit $\boxed{\mathcal{O}(mnpq)}$.

2.2 Calcul du gradient de l'image

La deuxième étape est ensuite de calculer le gradient de l'image f en chaque point. Pour cela, les trois algorithmes que j'ai étudié effectue une convolution entre l'image et un opérateur de gradient dans différentes directions, on nomme par la suite ces résultats f_i. Enfin ils définissent l'amplitude du gradient comme :

$$||\nabla f||_2 = \sqrt{\sum_i f_i^2}$$

2.2.1 Algorithme de Canny

L'algorithme de Canny convolue simplement l'image en chaque point avec un filtre de type $[-1, 0, 1]$ suivant la direction horizontale et verticale de l'image. Cela signifie que le gradient en chaque point est calculé grâce à la méthode des différences finies d'ordre 1 dans la direction horizontale et verticale de l'image.

$$f_x = \frac{\partial f}{\partial x}(x, y) = f(x + 1, y) - f(x - 1, y)$$

$$f_y = \frac{\partial f}{\partial y}(x, y) = f(x, y + 1) - f(x, y - 1)$$

2.2.2 Transformée en ondelettes

Pour les 2 autres algorithmes, ceux-ci définissent des directions "privilégiées" pour lesquelles ils vont calculer la transformée en ondelettes. Au niveau théorique il faudrait alors introduire la théorie des *lattices* afin de pouvoir définir correctement et mathématiquement n'importe quelle ligne "numérique" à partir d'un pixel et d'une direction donnée. Dans notre cas actuel cela ne nous est pas utile car nous allons uniquement étudier les directions $[[1,0],[1,1],[0,1],[-1,1]]$, *ie* les lignes horizontales, verticales et les diagonales. Cependant si nous étudions n'importe quelle direction possible, il serait nécessaire de définir de façon plus concise les pixels à choisir dans une certaine direction donnée.

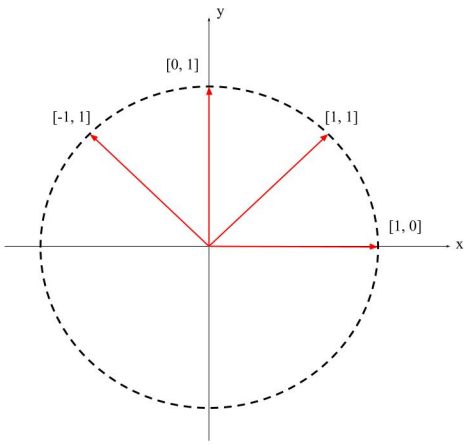

FIGURE 5 – Directions possibles

Afin d'appliquer la transformée en ondelette il faut voir chaque direction comme un signal 1D que l'on va convoluer avec une ondelette. L'ondelette choisie dans le papier est la dérivée de la gaussienne mais d'autres ondelettes peuvent être utilisées. Néanmoins la dérivée de la gaussienne s'avère être un bon choix ici car cela reprend l'idée de la méthode des différences finies en ne regardant pas uniquement les pixels voisins mais tous les pixels de la direction avec des poids différents.

L'ondelette mère considérée ici est :

$$\psi(t) = \frac{-t}{\sqrt{2\pi}} \exp\left(\frac{-t^2}{2}\right)$$

Par définition il s'agit bien d'une ondelette car :

$$\int_{-\infty}^{+\infty} \frac{-t}{\sqrt{2\pi}} \exp\left(\frac{-t^2}{2}\right) dt = 0 \text{ et } t\psi(t) \text{ est intégrable sur R}$$

On définit alors les ondelettes filles comme :

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}} \psi\left(\frac{t-b}{a}\right)$$

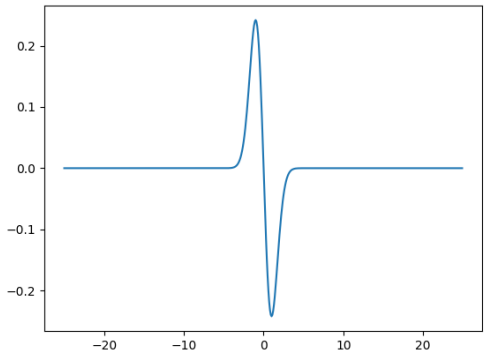

FIGURE 6 – Ondelette mère

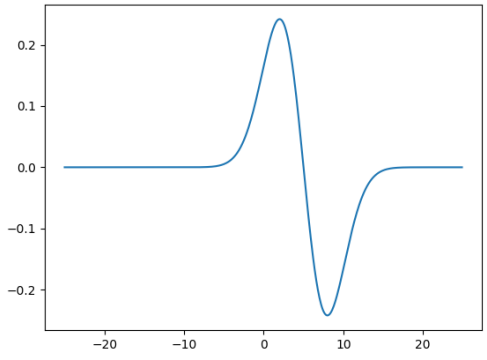

FIGURE 7 – Ondelette fille a = 3, b = 5

Les paramètres a et b correspondent respectivement au coefficient d'échelle et à la translation. Au plus a augmente, au plus son support va s'élargir (figures 6 et 7) donc le gradient va utiliser davantage de pixels autour.

La transformée en ondelettes continue en 1D est définit par :

$$WT_f(a,b) = \int_{-\infty}^{+\infty} f(t)\psi_{a,b}(t)dt = \int_{-\infty}^{+\infty} f(t)\frac{1}{\sqrt{a}}\psi(\frac{t-b}{a})dt = \frac{1}{\sqrt{a}}\int_{-\infty}^{+\infty} f(t)\psi(-\frac{b-t}{a})dt$$

Elle peut se réécrire sous la forme d'une convolution :

$$WT_f(a,b) = \frac{1}{\sqrt{a}} f * \bar{\psi_a}(b) \text{ avec } \bar{\psi_a}(b) = \psi(\frac{-b}{a})$$

Dans le cas discret on peut aussi le voir sous la forme :

$$\boxed{WT_f(a,b) = \frac{1}{\sqrt{a}} \sum_{t \in Pixels} f(t)\psi(\frac{t-b}{a})}$$

Avec a l'échelle voulue et b la position du pixel dans f pour lequel on veut calculer l'amplitude de gradient.

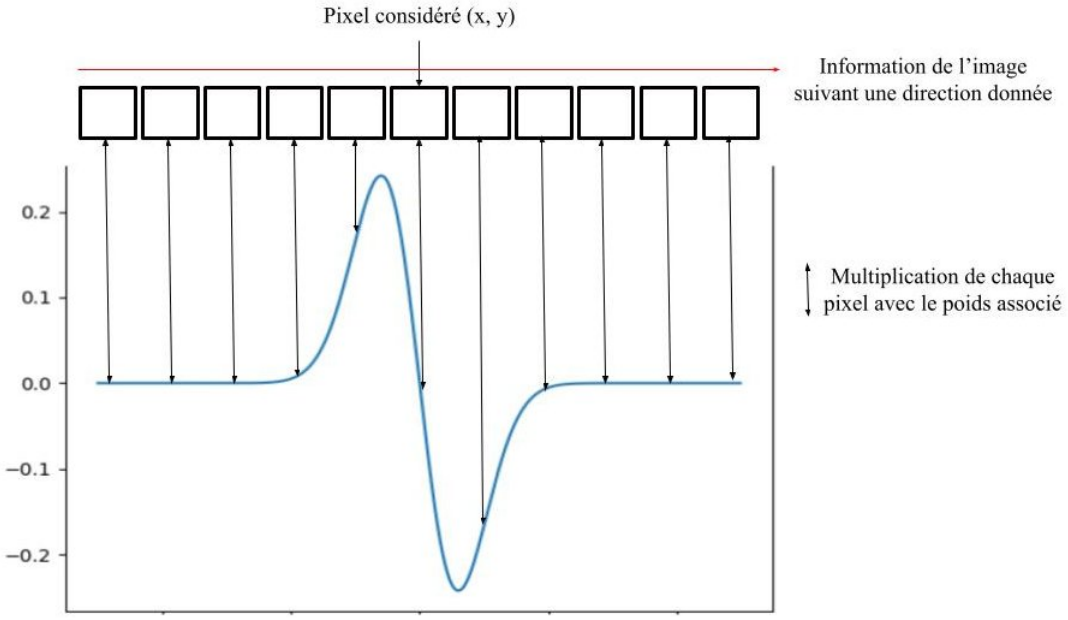

FIGURE 8 – Convolution entre l'image dans une direction donnée et l'ondelette

En appliquant cette transformée en ondelettes suivant les quatre directions en chaque pixel puis en calculant l'amplitude du gradient nous obtenons le résultat suivant :

FIGURE 9 – Exemple de résultat d'amplitude du gradient en chaque pixel pour l'algorithme implémenté par le papier

2.3 Détection des maximums locaux

La troisième étape consiste à regarder pour chaque pixel si l'amplitude du gradient calculé auparavant est un maximum local dans la direction du gradient. Si c'est le cas on le retient comme candidat pour la 4ème étape.

La définition de la direction du gradient change un peu suivant les algorithmes. Pour Canny ainsi que pour la transformée en ondelettes standard (voir figure 15), la direction du gradient est définie comme :

$$\boxed{\theta = \arctan(f_y/f_x)}$$

Cependant dans le papier étudié, ils redéfinissent la direction du gradient car ils ont davantage de directions à considérer. De plus la théorie disait que les deux amplitudes les plus grandes du gradient devaient être voisines (angle de 45° au maximum entre eux) mais ils se sont aperçu expérimentalement que ce n'est pas toujours vrai. Si on est dans un autre cas, la meilleure direction de gradient à prendre n'est pas forcément celle avec l'amplitude maximale. Dans certaines conditions la meilleure direction sera celle avec la deuxième meilleure amplitude. De ce fait, ils ont développé un algorithme adapté aux 4 directions calculé précédemment. Voici son fonctionnement :

Algorithm 1 Algorithme de suppression des non-maxima

Étant donné un pixel (x, y) on note f_1, f_2, f_3, f_4 les amplitudes du gradient en ce point suivant les 4 directions triés par ordre décroissant, et on appelle $\overrightarrow{f_i}$ la direction associée.

if les deux plus grandes amplitudes de gradient ont des vecteurs voisins **then**
Alors la direction du gradient est $\overrightarrow{f_1} + \overrightarrow{f_2}$ et pour savoir si c'est un maxima local on compare l'amplitude du gradient en (x, y) avec celle selon la direction du gradient (obtenu par interpolation).

else
La direction du gradient est soit $\overrightarrow{f_1}$ soit $\overrightarrow{f_2}$. Pour choisir on teste d'abord le petit algorithme suivant sur $\overrightarrow{f_1}$ puis sur $\overrightarrow{f_2}$ et on garde le 1er retenu :

Si f_i est supérieur à la moyenne des f_i, s'il s'agit bien du maximum local suivant la direction $\overrightarrow{f_i}$ et si l'amplitude du gradient est inférieure à la moyenne dans la direction perpendiculaire à $\overrightarrow{f_i}$ alors on le garde.

end if

Le premier cas de l'algorithme ("si les deux plus grandes amplitudes de gradient ont des vecteurs voisins") correspond à l'algorithme de Canny ainsi qu'à la transformée en ondelettes standard. Dans les autres algorithmes nous ne pouvons par contre pas rencontrer la seconde hypothèse.

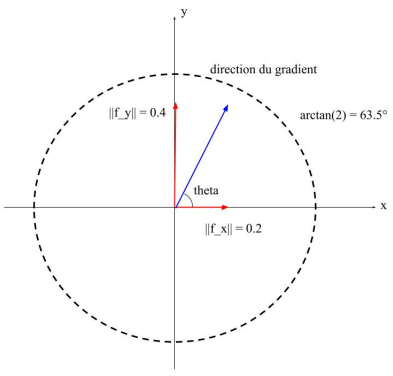

FIGURE 10 – Exemple de direction de gradient dans le cas de Canny

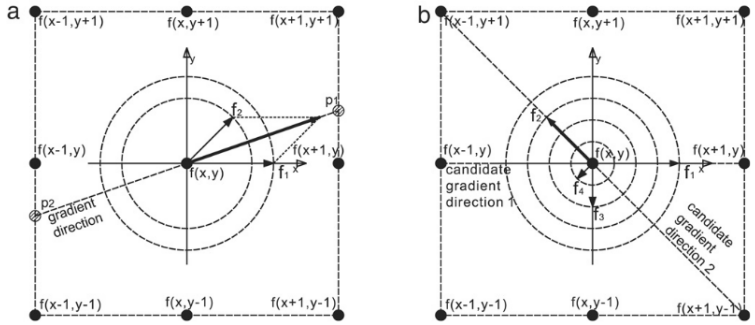

FIGURE 11 – Illustration de l'algorithme du papier

En reprenant les amplitudes de gradients obtenus dans l'étape précédente, les pixels gardés par cette détection des maximums locaux sont :

FIGURE 12 – Maximums locaux retenus

2.4 Double seuillage

Enfin la dernière étape qui est commune aux trois algorithmes s'appelle le "double seuillage".

On définit deux seuils τ_1 assez haut et τ_2 assez bas. Puis on associe à chaque pixel ayant été retenu précédemment trois états :

— si l'amplitude du gradient est plus grand que τ_1 alors il s'agit d'un pixel du contour
— si l'amplitude du gradient est entre τ_1 et τ_2 alors on le considère comme un potentiel candidat
— si l'amplitude du gradient est plus petit que τ_2 alors on ne garde pas le pixel

Enfin on décide si chaque candidat appartient ou non au contour en regardant si des pixels déjà classifié comme contour ne se trouverait pas à côté de lui dans la direction perpendiculaire au gradient.

Le choix des seuils est un problème difficile dans la littérature, surtout le choix de τ_2 c'est pourquoi j'ai également implémenté le même seuillage mais en considérant de pouvoir garder un certain pourcentage de pixels à la place des seuils. Cela me permet aussi de mieux pouvoir comparer les différents algorithmes.

FIGURE 13 – De gauche à droite : Pixels supérieur à τ_1, Pixels entre τ_1 et τ_2, Résultat final de l'algorithme

Ce résultat a été obtenu en prenant $\tau_1 = 40\%$ et $\tau_2 = 70\%$. On remarque qu'il est difficile à l'oeil nu de voir quels sont les pixels candidats qui ont bien été ajouté. Ici ce sont les contours du chapeau qui ont été le plus amélioré.

On peut également essayer en considérant différents paramètre d'échelle a :

FIGURE 14 – De gauche à droite : $a = 1$, $a = 2$, $a = 3$

Au plus a augmente au moins le contour détecté contient les petits détails de l'image. Cela est normal car dans le calcul du gradient on prend davantage en compte les pixels voisins donc on détecte les contours les plus significatifs.

2.5 Structure de données et implémentation informatique

Au niveau informatique, j'ai choisi d'utiliser le langage de programmation Python et d'implémenter par moi-même les trois algorithmes expliqué plus haut. J'ai néanmoins fait appel à la librairie *OpenCV* de Python afin d'importer les images souhaitées et de les passer en niveaux de gris. Pour coder les différents algorithmes, j'ai implémenté une classe *Mère* en y mettant toutes les fonctions reprises par plusieurs des algorithmes et j'ai fait trois classes filles en y inscrivant uniquement les fonctions qui changent entre elles.

Une fonction m'a demandé un peu plus de temps que les autres à coder. Il s'agit de la fonction qui décide si un candidat ($\tau_2 \leq ||\nabla f|| \leq \tau_1$) appartient ou non au bord. Pour ce faire j'ai utilisé un ensemble dans lequel j'ai mis tous les candidats possibles, puis je parcours chaque membre de l'ensemble en l'ajoutant ou non au contour (s'il satisfait la condition "proche d'un contour") et donc en l'enlevant ou non de l'ensemble des candidats. Je réitère cela tant qu'au moins un pixel est ajouté au contour à chaque parcours de l'ensemble. Ce choix est naïf et coûteux en complexité, une amélioration est sans doute possible.

Je me suis également penchée sur une optimisation de la transformée en ondelettes car ce calcul est coûteux s'il est effectué suivant toute une direction alors que peu de poids seront non nuls. En effet le support d'une ondelette dépend de a, l'échelle souhaitée, mais celui-ci n'est jamais très grand car puisque le gradient est une notion locale cela n'aurait pas d'intérêt de prendre un a trop élevé. J'ai alors choisi de manière empirique que l'ondelette mère s'étalait sur un support de 8 pixels et que les ondelettes filles s'étalait sur $8 * a$ pixels. J'aurais pu autrement ajouter une condition d'arrêt dès lors que le poids de l'ondelette devient trop petit.

Enfin pour plus de facilité de codage, je n'ai pas cherché de maximum locaux pour les pixels étant sur le bords de l'image afin de ne pas avoir à différencier les cas possible. Cela pourrait être ajouté moyennant un peu de réflection.

3 Comparaison des résultats avec les autres algorithmes

Dans cette deuxième partie j'ai voulu comparer les trois algorithmes développés auparavant afin de savoir si le gain du nouvel algorithme apporté par le papier était vraiment important. Pour ce faire j'ai dessiné par moi-même une image, celle-ci contient d'importants contours situé de façon diagonale.

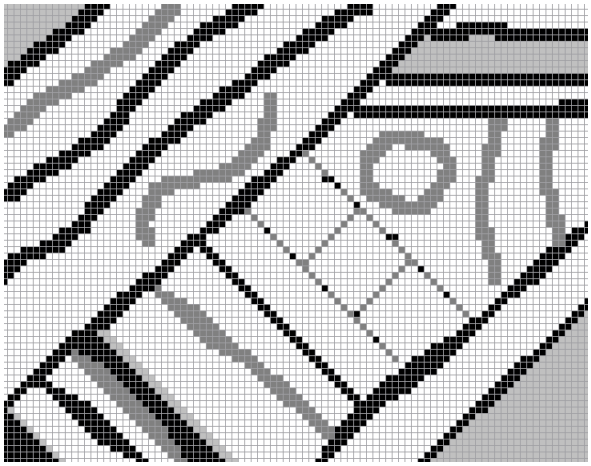

FIGURE 15 – Image originale avec pixels apparents

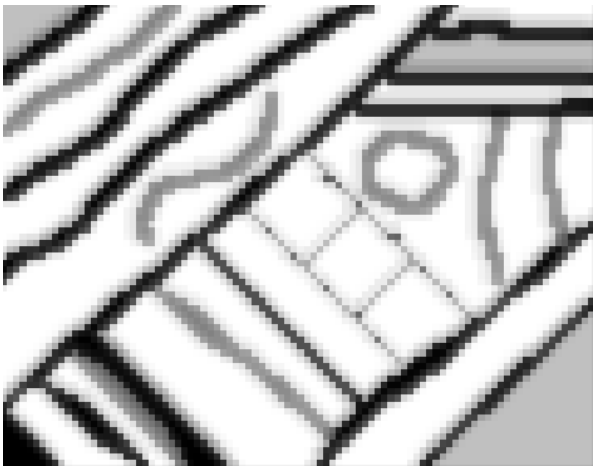

FIGURE 16 – Image lissée

Et voici ci-dessous les résultats des trois algorithmes sur cette image. Les paramètres choisis sont : $a = 1$, $\sigma = 0.5$, taille du kernel gaussien $= 3$, $\tau_1 = 30\%$ et $\tau_2 = 70\%$

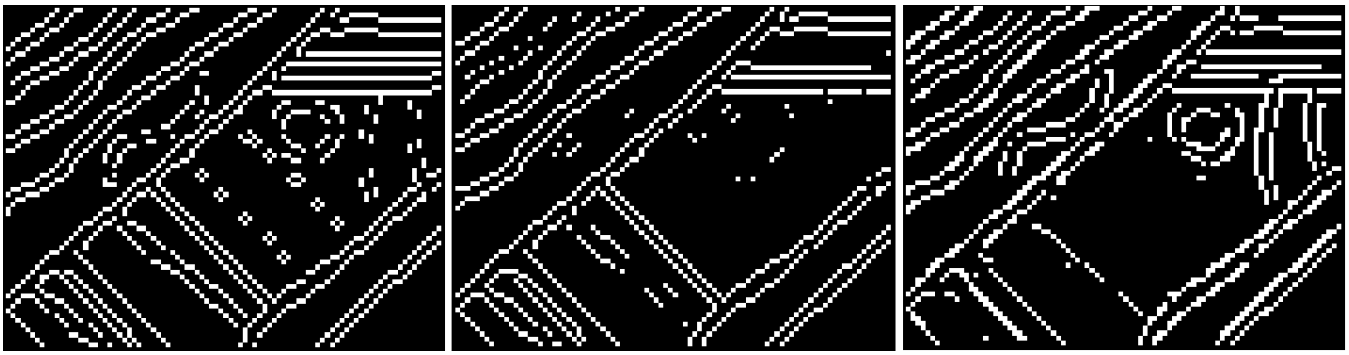

FIGURE 17 – De gauche à droite : algorithme de Canny, transformée standard, transformée suivant les 4 directions

On remarque plusieurs choses sur ces images. Tout d'abord les contours semblent plus mince sur Canny et la transformée standard que sur le nouveau algorithme du papiern, il permet d'avoir des contours plus épais. Cela peut être utile suivant l'utilisation de cette détection de contour comme par exemple si on est plus intéressé à "encadrer" les contours plutôt que de les placer précisément.

Cependant les contours ont l'air davantage continu pour l'algorithme de Canny. De plus le rendu est très bon en comparaison avec sa complexité. La seule question qu'on peut se poser c'est : mieux vaut-il détecter des pixels isolés comme ceux situé sur l'échelle dans le dessin ou bien mieux vaut-il ne pas détecter l'échelle car elle est de trop faible intensité ? Canny apporte une vision très localisée des contours.

Concernant la transformée en ondelettes standard, on observe ici que cela apporte beaucoup de considérer deux directions supplémentaires dans le papier. Les figures de la vague et du cercle n'apparaissent pas du tout ou quasiment pas. Cependant si on regarde de plus près ce cas là on remarque que ces figures apparaissent comme pixels candidats mais ils ne possèdent pas de proximité à des pixels déjà considérés comme contour.

4 Conclusion

En conclusion de ce projet et de ce cours, nous pouvons nous apercevoir que les ondelettes sont utilisés dans tous les domaines depuis leur développement par Stéphane Mallat. Ici on les utilise comme des poids à répartir sur les différents pixels suivant une direction, c'est donc une amélioration de la méthode des différences finies.

Néanmoins je pense que l'amélioration induite par ce papier n'apporte pas autant de gain que ce qu'elle le dit. Effectivement les contours se retrouvent être plus épais mais l'algorithme de Canny fait quasiment un aussi bon travail en bien moins de temps car il ne considère que les pixels directement voisins. C'est également pour cela que l'algorithme de Canny est encore couramment utilisé de part son efficacité et sa simplicité. Il y a aussi bien d'autres algorithmes qui se sont développés depuis comme les méthodes de level set et d'autres utilisant les réseaux de neurones.

Pour finir, et pour continuer ce projet, il faudrait se pencher sur les seuils dans le double seuillage. La méthode que j'ai développé en utilisant un pourcentage de pixel que l'on souhaite garder n'est peut être pas la meilleure solution pour comparer les différents algorithmes. En effet augmenter l'épaisseur de certains contours peut nuire à la détection d'autres contours. Cela reste à étudier plus en profondeur. De plus pour le choix des seuils j'ai vu qu'il existe différentes méthodes comme la *détection de vallées* utilisant des histogrammes ou encore le *seuillage entropique* mais je n'ai pas eu le temps de m'y intéresser.

Références

[1] Velisavljevic Vladan, Beferull-Lozano Baltasar, Vetterli Martin, and Dragotti Pier Luigi. Directionlets : Anisotropic multidirectional representation with separable filtering. 2006.

[2] Zhang Zhen, Ma Siliang, Liu Hui, and Gong Yuexin. An edge detection approach based on directional wavelet transform. 2009.