

TP: Détection de contours par approches du second ordre

Kasperek Gautier
Sparrow Alice

L'objectif de ce TP est de manipuler le Laplacien de différentes images afin de détecter les contours. Dans un premier temps, nous aborderons une approximation discrète du Laplacien par masque de convolution. Dans un second temps, nous implémenterons un algorithme de seuillage afin d'extraire au mieux les contours de l'image. Enfin, nous utiliserons les résultats obtenus avec d'autres approches dans le but d'affiner nos résultats. Toutes les manipulations sont réalisées sous ImageJ. Vous trouverez dans ce rapport une explication de notre méthode de travail, nos résultats et leurs interprétations. Les parties du code intéressantes seront expliquées dans les parties correspondantes, le plugin complet est disponible en annexe du rapport.

TP: Détection de contours par approches du second ordre	1
1. Calcul du Laplacien	2
Calcul et représentation du Laplacien	2
2. Seuillage des passages par 0 du Laplacien	6
Seuillage manuel des passages par 0 du Laplacien	6
Seuillage automatique des passages par 0 du Laplacien (Optionnel)	9
3. Utilisation du filtre LoG et détection multi-échelles	9
Génération de masques pour l'opérateur LoG	9
Détection multi-échelles (Optionnel)	10
Utilisation de la norme du gradient	13
Conclusion	15
Annexe	16

1. Calcul du Laplacien

L'objectif de cette première partie est d'implémenter et d'utiliser un calcul simple du Laplacien d'une image. Tout au long de ce TP nous utiliserons deux images de travail.

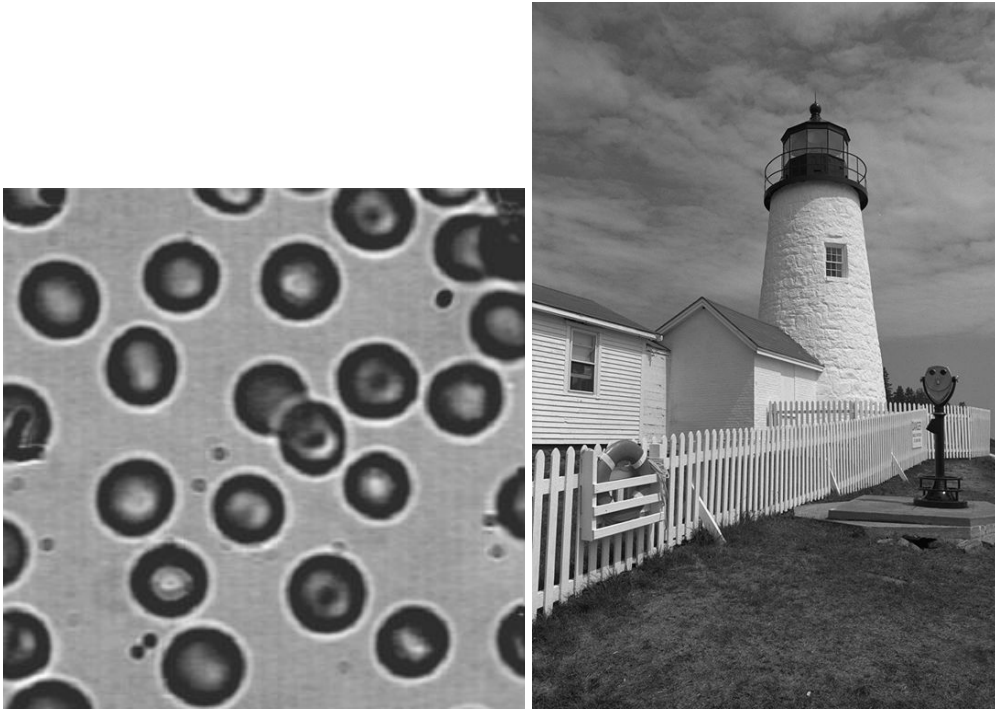


Figure 1 - Images de travail, spores.png à gauche, phare.png à droite

Calcul et représentation du Laplacien

Nous avons donc implémenté un plugin permettant de calculer le Laplacien d'une image sous forme de plugin ImageJ. L'environnement de développement d'ImageJ permet de réaliser ce calcul très facilement, il suffit dans un premier temps de créer un masque de convolution sous forme de tableau de float :

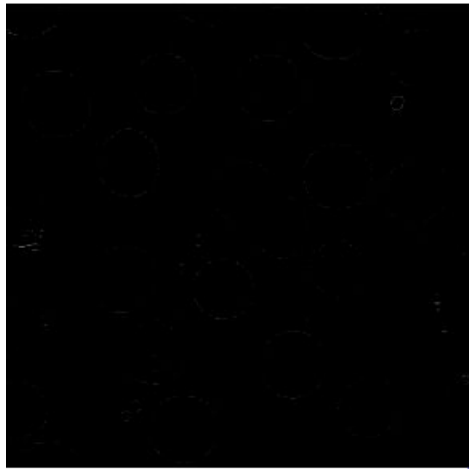
```
private final static float[][] MASQUES_LAPLACIENS3x3 = {  
    {0,1,0, 1,-4,1, 0,1,0}  
}
```

Puis une méthode nous permet d'appliquer ce masque directement à l'image choisie, ici *fpLaplacian* :

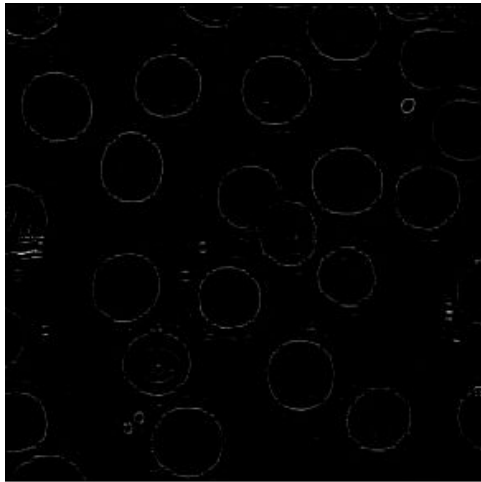
```
fpLaplacian.convolve(MASQUES_LAPLACIENS3x3[filtre],3,3);
```

Nous avons ensuite appliqué différentes approches du Laplacien afin de constater les différences et déterminer le plus adéquat à notre problème.

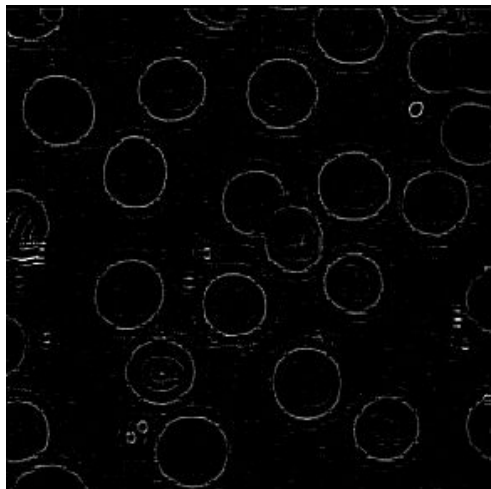
0	1	0
1	-4	1
0	1	0



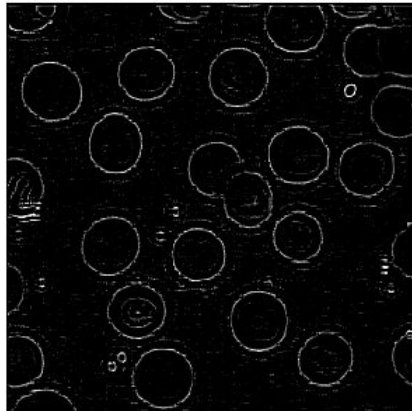
1	0	1
0	-4	0
1	0	1



1	1	1
1	-8	1
1	1	1



1	2	1
2	-12	2
1	2	1



1	4	1
4	-20	4
1	4	1

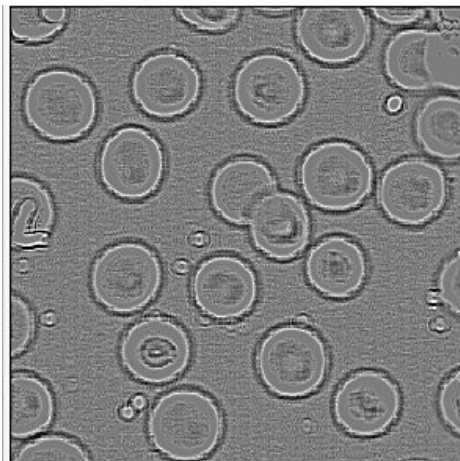
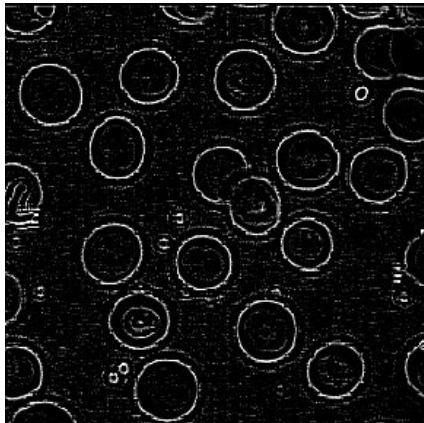


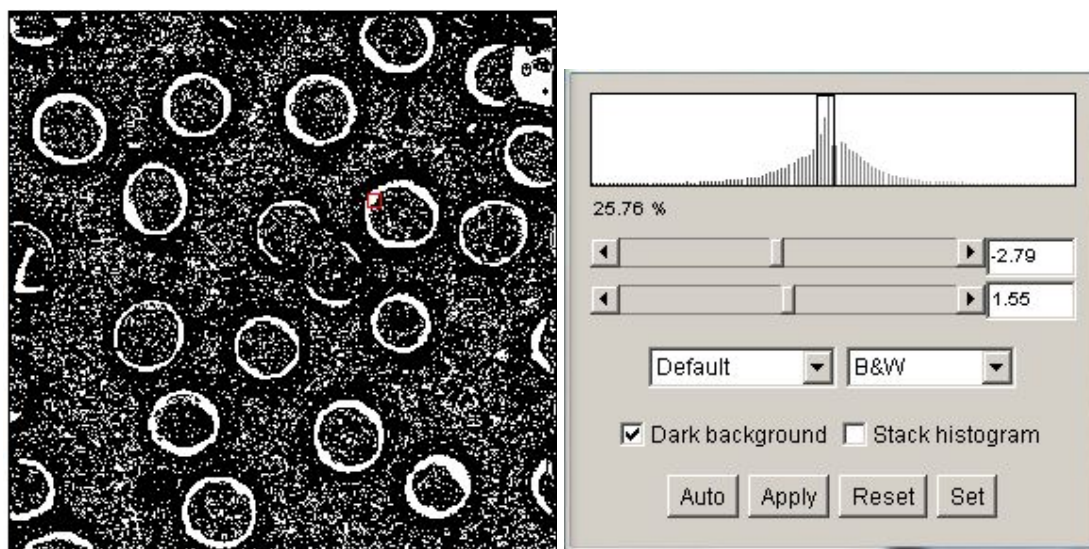
Figure 2 - Résultat de la convolution du Laplacien sur l'image spores.png

La figure 2 est le résultat du calcul par convolution du Laplacien de l'image d'entrée *spores.png*.



Figure 3 - Histogramme du résultat de la convolution du Laplacien

On remarque que l'application par convolution du masque Laplacien fait apparaître les contours de l'image spores.png. Après le filtrage de l'image avec un masque Laplacien, il faut détecter les passages par zéros les plus significatifs car cette technique est sensible au bruit ce qui peut se traduire par la détection de contours parasites. Les pixels ayant des valeurs positives ou négatives éloignées de 0 appartiennent au fond de l'image.



Prefs	189	190	191	192	193	194	195
96	16.00	-2.00	0.00	0.00	0.00	-1.00	3.00
97	3.00	-1.00	0.00	0.00	0.00	-1.00	3.00
98	-2.00	0.00	0.00	0.00	0.00	-2.00	6.00
99	-1.00	0.00	0.00	0.00	-1.00	4.00	5.00
100	0.00	0.00	0.00	0.00	-1.00	6.00	17.00
101	0.00	0.00	-1.00	-1.00	2.00	12.00	7.00
102	0.00	-2.00	3.00	4.00	5.00	-4.00	2.00

Figure 4 - Inspection d'un pixel de l'image résultante

Les pixels valant 0 représentent les contours de l'image.

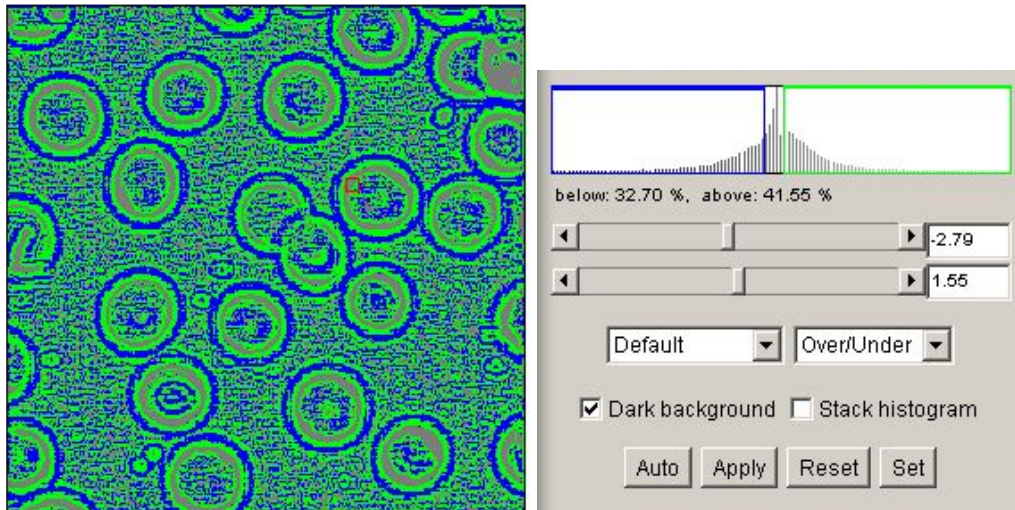


Figure 5 - Mise en évidence des pixels négatifs et positifs

On peut remarquer sur la figure ci-dessus que les pixels négatifs (en bleu) et les pixels positifs (en vert) ne font pas partie des contours de l'image. Cette représentation met en évidence les pixels gris comme étant ceux constituant les contours de l'image car ils s'agit des pixels proches de 0.

2. Seuillage des passages par 0 du Laplacien

Après avoir calculer le Laplacien d'une image nous allons tester l'apport d'un seuillage dans la détection d'un contour.

Seuillage manuel des passages par 0 du Laplacien

Dans la partie précédente nous avons calculer le Laplacien d'une image, il faut maintenant traiter ces données pour appliquer un seuillage défini manuellement par l'utilisateur. L'algorithme parcourt alors chaque pixel du Laplacien. Pour chaque pixel, on s'intéresse aux voisins dans une limite de 3*3 autour du pixel courant. Les coordonnées x_k et y_k représentent les coordonnées des voisins pris un à un.

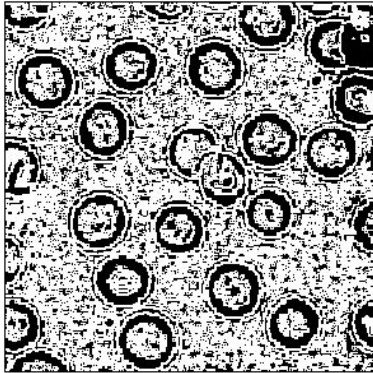
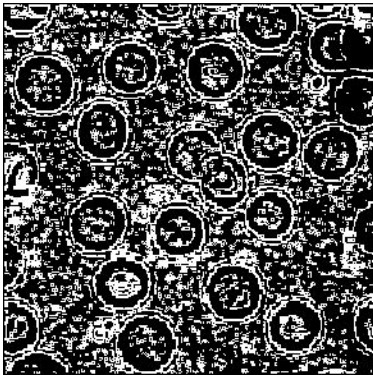
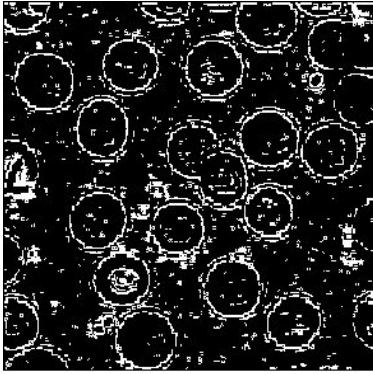
```
// Détermine les valeurs voisines max et min
    if(max < imLaplacien.getf(xk,yk)){
        max = imLaplacien.getf(xk,yk);
    }
    if(min > imLaplacien.getf(xk,yk)){
        min = imLaplacien.getf(xk,yk);
    }
```

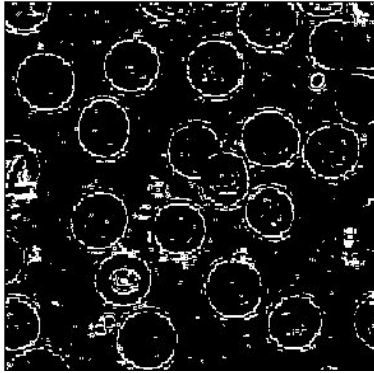
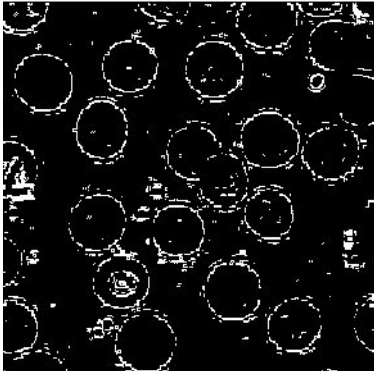
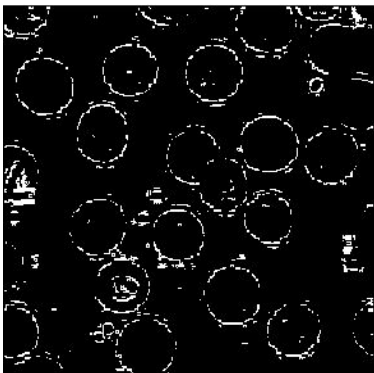
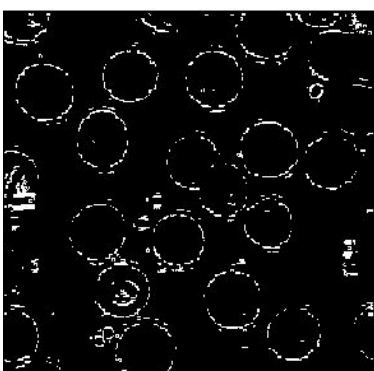
Le but de la boucle parcourant les voisins est de déterminer la valeur maximum et minimum du au voisinage du pixel. Ainsi, si le maximum est supérieur au seuil et le minimum est inférieur à l'inverse du seuil, on considère le pixel courant comme contour de l'image.

```
// Seuillage
if(max >= seuil && min < (-1 * seuil)) {
    imZeros.set(x,y,255);
}else {
    imZeros.set(x,y,0);
}
```

On applique notre plugin à l'image *spores.png* avec différent seuil afin de déterminer lequel est le plus efficace pour extraire les contours de l'image.

Voici un tableau récapitulatif des seuils que nous avons testé pour l'image *spores.png*

Seuil = 5		Beaucoup de bruit sur cette image et les contours extérieurs des spores sont presque indiscernables. Cependant, elle permet de différencier le noyau des spores.
Seuil = 10		Le fond est moins bruité mais les contours restent peu exploitable. De plus on perd en précision sur la représentation des noyaux des spores.
Seuil = 15		A partir de ce seuil, les contours extérieurs sont mieux définis mais on ne peut plus exploiter les contours intérieurs.

Seuil = 18		Les noyaux des spores ont presque totalement disparu. Cependant les bruits touchant le fond ont presque tous disparu. Les contours extérieurs des spores sont nettement visibles.
Seuil = 20		Les contours extérieurs des spores commencent à se dissiper et on peut constater des interruptions de contour sur certains spores.
Seuil = 23		Il n'y a presque plus de bruit parasite dans le fond et à l'intérieur des spores. Cependant les contours extérieurs sont fortement interrompu.
Seuil = 25		Il y a beaucoup d'interruption dans les contours et rendent certains spores difficilement discernables.

Comme nous pouvons le constater sur ces différentes images seuillé, le choix du seuil peut se faire en fonction de l'usage que l'on a de l'image. En effet, si par exemple nous souhaitons compter ou repérer les spores sur une image nous pouvons utiliser un seuil entre

20 et 23. Cependant si nous souhaitons par exemple comparer la taille d'un spore à celle de son noyau, un faible seuil, de 5 à 10 sera plus adéquat.

Après avoir tester l'apport du seuillage dans la détection de contour grâce au Laplacien d'une image, nous allons maintenant définir ce seuil de manière automatique grâce à l'image de base.

Seuillage automatique des passages par 0 du Laplacien (Optionnel)

Nous avons effectué le même type d'expérimentation sur l'image *lighthouse.png*, le résultat le plus satisfaisant est un seuillage à 35. Cependant les contours sont très épais et nous perdons en précisions dans certaines zones comme les barrières par exemple.



Figure 6 - Laplacien de *lighthouse.png* seuillé à 35

3. Utilisation du filtre LoG et détection multi-échelles

Nous avons pu remarquer que la détection de contours pouvait être perturbé par le bruit de l'image. C'est pourquoi dans cette partie nous proposons d'étudier l'application et l'impact d'un filtre gaussien en amont de la détection de contours.

Génération de masques pour l'opérateur LoG

Lors de l'application d'un filtre Gaussien il faut être attentif à la relation entre sigma et la taille du masque. En effet, sigma représente l'écart-type d'une fonction Gaussienne, c'est pourquoi plus sigma est grand plus le masque doit être grand puisque la fonction est plus "aplati". De plus, la taille du masque doit être impair avec pour centrer le pixel courant.

```
int sizeM = (int)(sigma*6);
```

```

if(sizeM%2 == 0){
    sizeM++;
}
float[] masque = masqueLoG(sizeM,sigma);

```

Après avoir défini le masque , on l'applique à l'image par la fonction suivante.

```

conv.convolve(fpLaplacien,masque,sizeM,sizeM);

```

Nous avons appliqué le filtre gaussien avec $\sigma = 2$ avant le calcul du Laplacien de l'image. Voici le résultat obtenu.

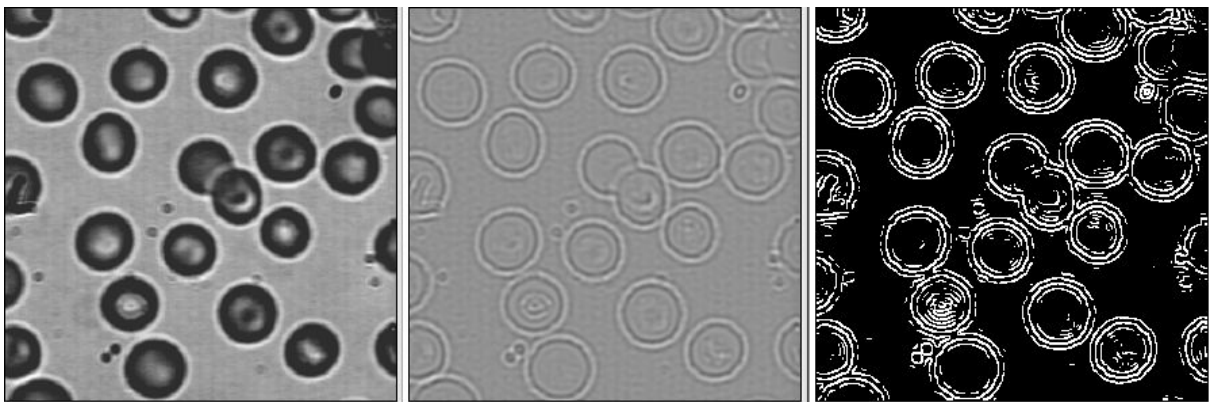


Figure 7 - spores.png à gauche, spores.png lissé avec $\sigma = 2$, Laplacien seuiller à 1 à droite

Comme nous pouvons le constater l'image finale des contours obtenu est bien plus exploitable que les résultats obtenus dans la partie précédente.

Détection multi-échelles (Optionnel)

L'algorithme implémenté dans la partie précédente a pour inconvénient de générer des bords trop épais. C'est pourquoi nous le modifions par le code ci-dessous.

```

// Pour chaque pixel (x,y),
float pixelC = imLaplacien.getPixelValue(x,y);           // Pixel central
float pixelD = imLaplacien.getPixelValue(x+1,y);         // Pixel droit
float pixelB = imLaplacien.getPixelValue(x,y+1);         // Pixel bas
float pixelBD = imLaplacien.getPixelValue(x+1,y+1);      // Pixel bas droit
// Détection des transitions Horizontales
if (pixelC<-seuil && pixelD>seuil) { // Transition horizontale -|+
    imZeros.set(x,y,255); }
if (pixelC>seuil && pixelD<-seuil) { // Transition horizontale +|-
    imZeros.set(x+1,y,255);}
// Détection des transitions Verticales

```

```

if (pixelC<-seuil && pixelB>seuil) { // Transition verticale -|+
    imZeros.set(x,y,255);}
if (pixelC>seuil && pixelB<-seuil) { // Transition verticale +|-
    imZeros.set(x,y+1,255);}
// Détection des transitions Diagonales
if (pixelC<-seuil && pixelBD>seuil) { // Transition diagonale -|+
    imZeros.set(x,y,255);}
if (pixelC>seuil && pixelBD<-seuil) { // Transition diagonale +|-
    imZeros.set(x+1,y+1,255);}

```

La principale différence avec l'implémentation précédente est la manière dont est calculé le seuil et la manière dont sont parcouru les pixels. Dans la version précédente, nous cherchions des minimums dans le voisinage entier du pixel courant et effectuons le test du seuil seulement pour ces maxima locaux. Alors que, dans cette version on teste le seuil sur plus de configuration mais on ne prend que le voisinage bas droite du pixel courant.

Nous avons effectué plusieurs essais en fixant différentes valeurs à sigma. Plus l'écart type du filtre Gaussien est important, plus on atténue les variations c'est à dire plus on lisse.

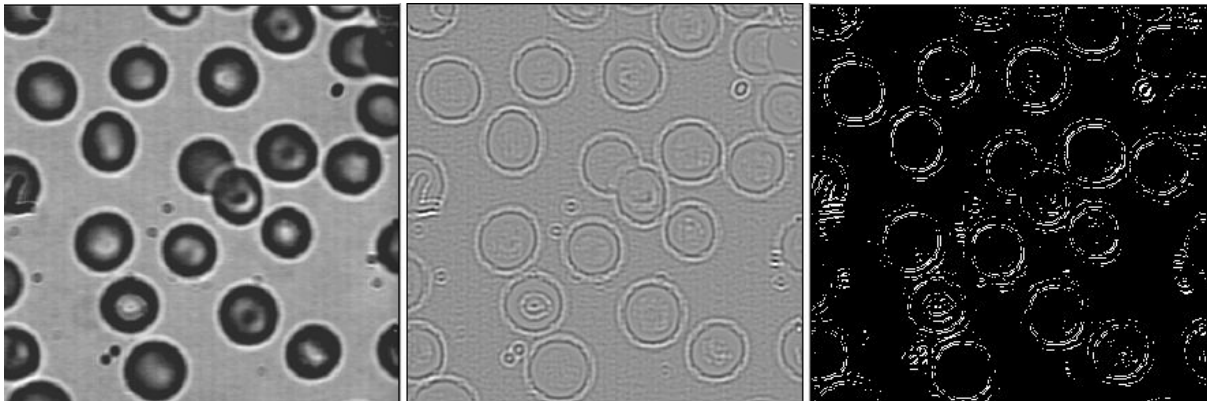


Figure 8 - Détection de contour par filtre LoG avec Sigma 1.4, seuillage des passages par zéros avec le seuil égal à 1

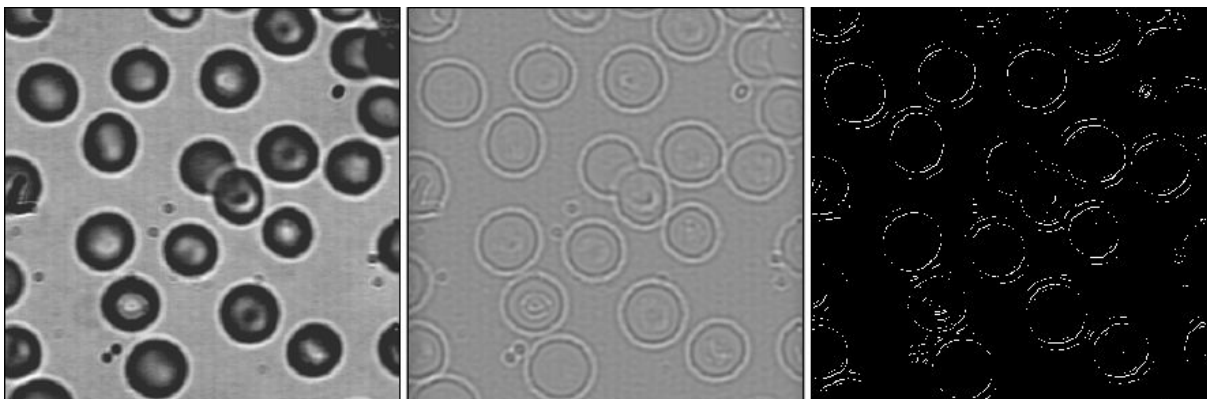


Figure 9 - Détection de contour par filtre LoG avec Sigma 2, seuillage des passages par zéros avec le seuil égal à 1

On peut comparer le seuillage des passages par zéro de la figure 9 à celui de la figure 7. On remarque que les contours de la figure 9 sont plus fin que ceux de la figure 7.

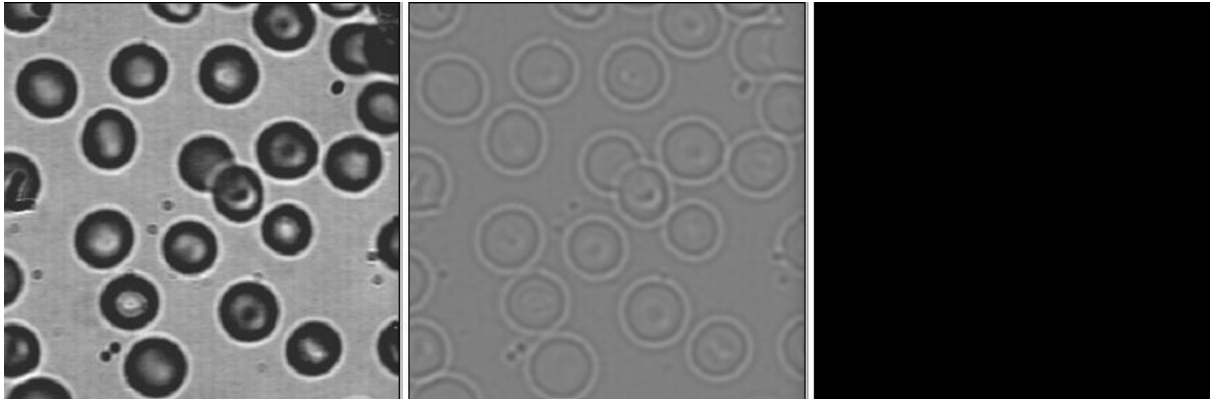


Figure 10 - Détection de contour par filtre LoG avec Sigma 3, seuillage des passages par zéros avec le seuil égal à 1

On remarque que pour une valeur $\sigma = 1.4$ (Figure 8), le seuillage des passages par zéro fait apparaître les contours avec des points parasites, l'image n'est donc pas assez lissée. La figure 9 met en évidence les contours avec un lissage Gaussien d'écart type $\sigma = 2$. Le résultat est meilleur, on distingue les contours et moins de points parasites. En revanche avec une valeur $\sigma = 3$, l'image est trop lissée et on ne distingue plus aucun contour.

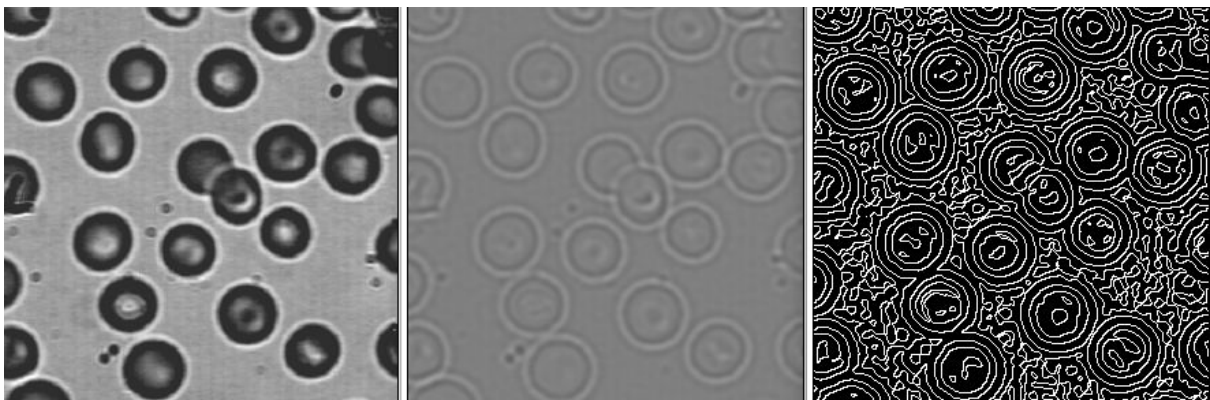


Figure 11 - Détection de contour par filtre LoG avec Sigma 3, seuillage des passages par zéros avec le seuil égal à 0



Figure 12 - Détection de contour par filtre LoG avec Sigma 1.4, seuillage des passages par 0 avec le seuil égal à 1

La figure 12 ci-dessus montre le résultat du filtre de lissage d'écart type $\sigma = 1.4$ sur l'image *phare.png* ainsi que le seuillage des passages par 0 du Laplacien. On distingue les contours du phare cependant on remarque un nombre beaucoup trop important de points parasites sur le sol et à l'intérieur du phare.

Utilisation de la norme du gradient

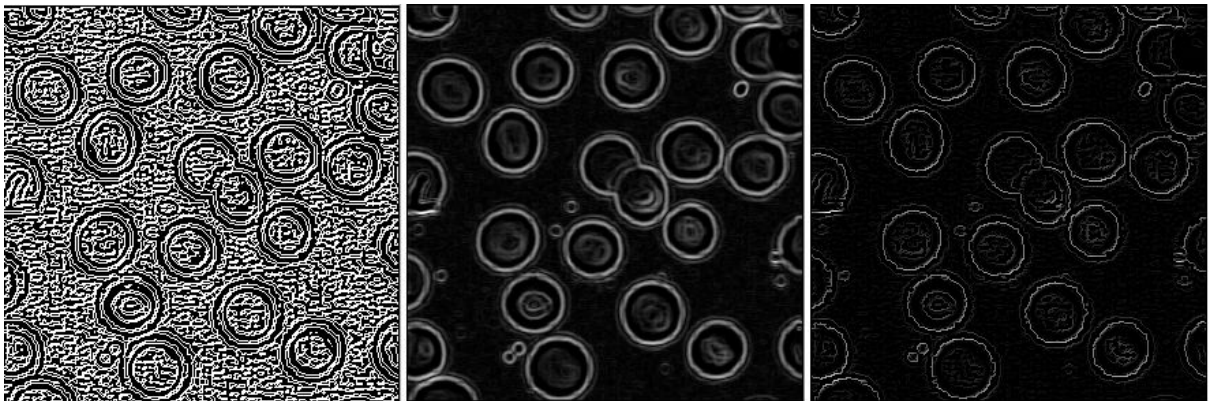


Figure 13 - Laplacien seuil 0, Norme du gradient, Norme du gradient masquée par le Laplacien seuil 0

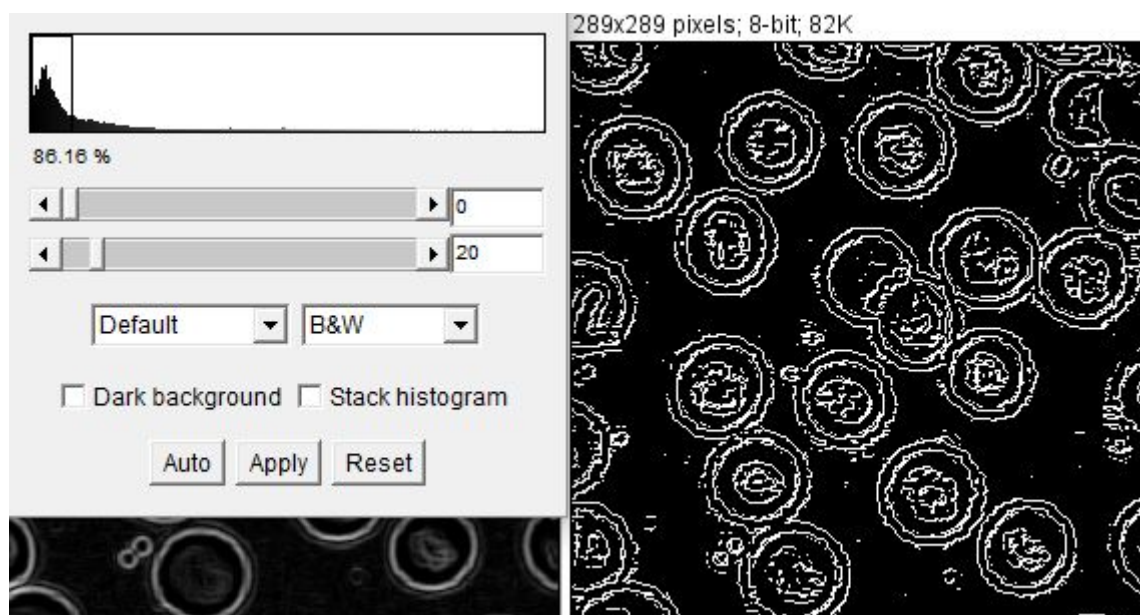


Figure 14 - Norme du gradient masquée par le Laplacien seuil 0, Seuil à 20

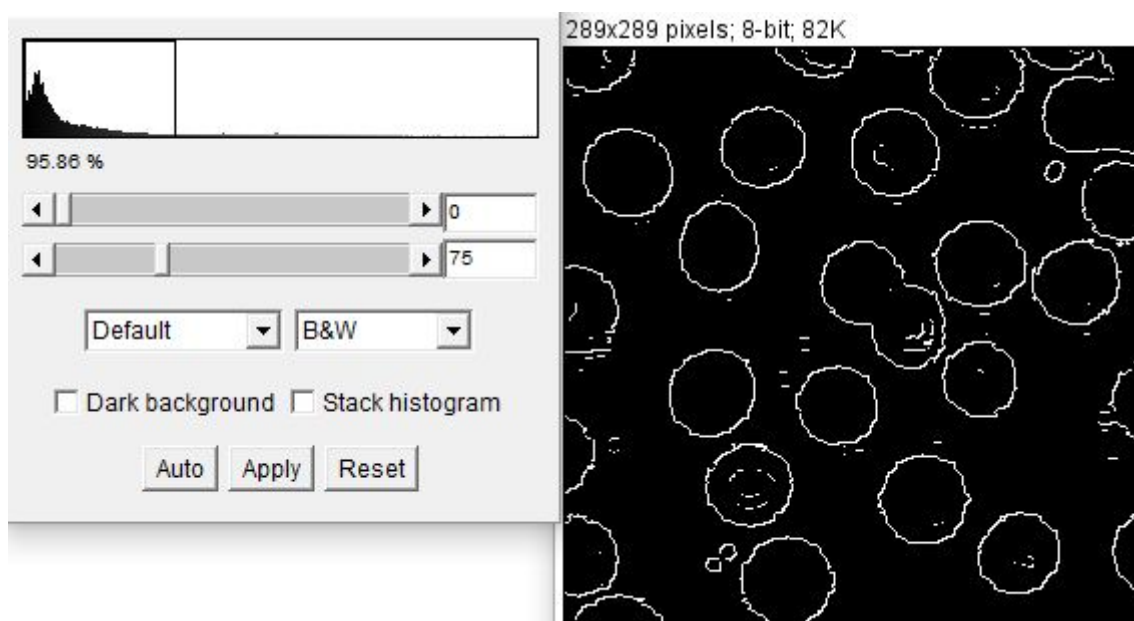


Figure 15 - Norme du gradient masquée par le Laplacien seuil 0, Seuil 75

Nous avons masquée la norme du gradient par le Laplacien seuil 0 et avons obtenu la troisième image de la figure 13. Lorsqu'on applique un seuil à cette image résultante nous obtenons les figures 14 et 15. La figure 15 présente un résultat satisfaisant puisqu'on distingue correctement les contours des spores sans trop de points parasites.

Conclusion

Au cours de ce tp nous avons manipulé le Laplacien d'une image afin de détecter ses contours. Après le filtrage de l'image avec un masque Laplacien, il faut détecter les passages par zéros les plus significatifs car cette technique est sensible au bruit. Nous avons dans un second temps cherché à seuiller les passages par 0 manuellement et automatiquement afin d'obtenir des contours satisfaisant. Pour finir nous avons cherché à mettre en évidence l'importance de l'application d'un lissage Gaussien avant l'utilisation du Laplacien. En effet le lissage préalable de l'image permet d'atténuer le bruit dans la détection des passages par 0 du Laplacien.

Annexe

```
public void run(ImageProcessor ip) {

    // Affichage de la fenêtre de configuration
    if (!showDialog())
        return;

    // Titre et extension de l'image source
    String titre = imp.getTitle();
    String extension="";
    int index = titre.lastIndexOf('.');
    if (index>0)
        extension = titre.substring(index);
    else
        index = titre.length();
    titre = titre.substring(0,index);

    // Génération d'un masque LoG de taille impaire (exercice 3)
    int sizeM = (int)sigma*5;
    if(sizeM%2 == 0){
        sizeM++;
    }
    float[] masque = masqueLoG(sizeM,sigma);

    // Calcul et représentation du Laplacien (exercice 1, puis à modifier
    // dans le 3)
    FloatProcessor fpLaplacian =
    (FloatProcessor)(ip.duplicate().convertToFloat());

    conv.convolve(fpLaplacian,masque,sizeM,sizeM);
    if(sigma!=0) {
        Convolver conv = new Convolver();
        conv.setNormalize(false);
        conv.convolve(fpLaplacian,masque,sizeM,sizeM);
    }
    fpLaplacian.convolve(MASQUES_LAPLACIENS3x3[filtre], 3, 3);

    // Détection et affichage des passages par 0 du Laplacien par
    // seuillage du Laplacien (exercice 2)
    if (seuillageZeroCross) {
```

```

        ByteProcessor laplacienZeroBP =
        laplacienZero(fpLaplacian,seuilZeroCross);
        ImagePlus resLaplacianZero = new ImagePlus("Resultat laplacien seuil
        : "+ seuilZeroCross, laplacienZeroBP);
        resLaplacianZero.show();
    }
    ImagePlus resLaplacian = new ImagePlus("Resultat
    laplacian",fpLaplacian);
    resLaplacian.show();
}

```

```

/**
 * Detection des passages par 0 du Laplacien (algo min<-seuil &&
max>seuil)
 *
 * @param imLaplacien Image du Laplacien (ImageProcessor 32 bits)
 * @param seuil Seuil sur les valeurs du Laplacien
 * @return imZeros Carte des passages par 0 (image binaire)
 */
public ByteProcessor laplacienZero(ImageProcessor imLaplacien,float
seuil) {
    int width = imLaplacien.getWidth();
    int height = imLaplacien.getHeight();
    float max = 0;
    float min = 0;

    // Image binaire résultat des points contours
    ByteProcessor imZeros = new ByteProcessor(width,height);

    int[] dx8 = new int[] {-1, 0, 1,-1, 1,-1, 0, 1};
    int[] dy8 = new int[] {-1,-1,-1, 0, 0, 1, 1, 1};
    // Parcours des valeurs du Laplacien
    for(int x = 0; x < width; x++) {
        for(int y = 0; y < height; y ++) {

            // Parcours des voisins de la valeurs courrante
            for(int k=0;k<8;k++) {
                int xk=x+dx8[k], yk=y+dy8[k];
                if (xk<0 || xk>=width) continue;
                if (yk<0 || yk>=height) continue;

```

```

        // Détermine les valeurs voisines max et min
        if(max < imLaplacien.getf(xk,yk)){
            max = imLaplacien.getf(xk,yk);
        }
        if(min > imLaplacien.getf(xk,yk)){
            min = imLaplacien.getf(xk,yk);
        }
    }
    // Seuillage
    if(max >= seuil && min < (-1 * seuil)) {
        imZeros.set(x,y,255);
    }else {
        imZeros.set(x,y,0);
    }
    max = 0;
    min = 0;
}
return imZeros;
}

```