TI : TP 9 : Détection de contours par approches du premier ordre

binôme : Benjamin Ruytoor et Aurore Allart
date : 22 mars 2013

Introduction

Dans ce TP, nous allons manipuler une image d'un phare pour récupérer les contours des bâtiments grâce à 3 méthodes. Tout d'abord, en utilisant 2 convolutions avec les masques de Sobel dans les directions x et y, cela nous permettra de calculer la norme et la direction du gradient. Ensuite, on effectue une suppression des pixels qui ne sont pas les maxima locaux de l'image. Enfin, on utilise le seuillage par hystérésis sur les maxima locaux, pour restituer une image binarisée contenant les contours de l'image de départ.

1. Seuillage de la norme d'un gradient

1. Représentation de la norme d'un gradient

1. Calcul de la norme d'un gradient par convolution

Voici le code qui nous a permis de calculer l'approximation des dérivées partielles de la fonction image :

```
//macro
run("Conversions...", " ");
// Calcul de la norme du gradient par masque de Sobel
requires("1.41i");       // cause substring(string, index)
//permet d'éviter de voir les calculs fait sur les images
setBatchMode(true);

//image initiale
imageDeBase = getImageID();
filename = getTitle();
filenameWithoutExt = substring(filename, 0, lastIndexOf(filename, "."));
extension = substring(filename, lastIndexOf(filename, "."));

//nom des images pour le calcul des dérivees
filenameGradX = filenameWithoutExt+"_grad_x"+extension;
filenameGradY = filenameWithoutExt+"_grad_y"+extension;
run("32-bit"); // Conversion en Float avant calcul des derivees !!

//duplicata de l'image initiale pour le calcul de la derivees suivant X
run("Duplicate...", "title="+filenameGradX);
run("Convolve...", "text1=[-1 0 1\n-2 0 2\n-1 0 1\n] normalize");
imageGradX = getImageID();

//duplicata de l'image initiale pour le calcul de la derivees suivant Y
selectImage(imageDeBase);
run("Duplicate...", "title="+filenameGradY);
run("Convolve...", "text1=[-1 -2 -1\n0 0 0\n1 2 1\n] normalize");
imageGradY = getImageID();
```

```
/****** Calcul de la norme du gradient ******/
// recuperation de la taille de l'image
w = getWidth();
h = getHeight();

//creation de la nouvelle image qui contiendra pour chaque pixel la norme du gradient
newImage("imageNorme", "32-bit", w, h,1) ;
imageNorme = getImageID();

// Calculs pour chaque pixel
norme = 0;
px = 0;
py = 0;

//calcul pour chaque pixel de la norme et de la direction du gradient
for (j=0; j<h; j++) {
        for (i=0; i<w; i++) {
                selectImage(imageGradX);
                px=getPixel(i,j);
                selectImage(imageGradY);
                py=getPixel(i,j);
                norme=sqrt(px*px+py*py);
                selectImage(imageNorme);
                setPixel(i,j,norme);
                }
}
```

La norme nous permet d'avoir tous les contours de l'image initiale. Malheureusement, il y a encore du bruit par exemple dans l'herbe. En voici le résultat :

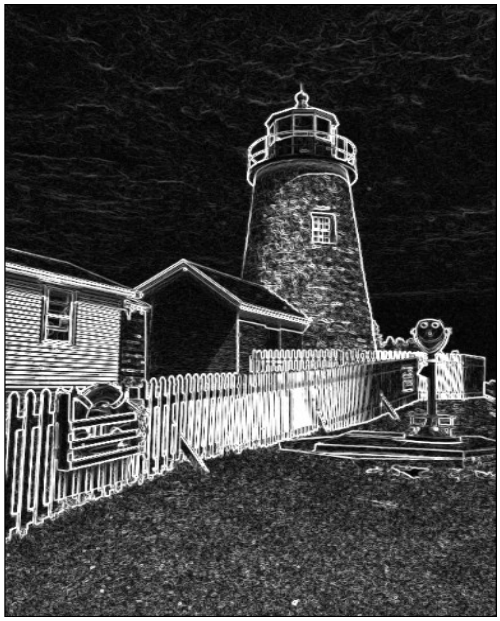


Figure 1 : image résultante du calcul de la norme du gradient.

2. Comparaison avec le résultat fourni par ImageJ

Pour l'image de la norme, la valeur minimale est de 0 et la valeur maximale est de 904,896. Cela est du aux calculs des convolutions qui ne sont pas normalisés par le programme.

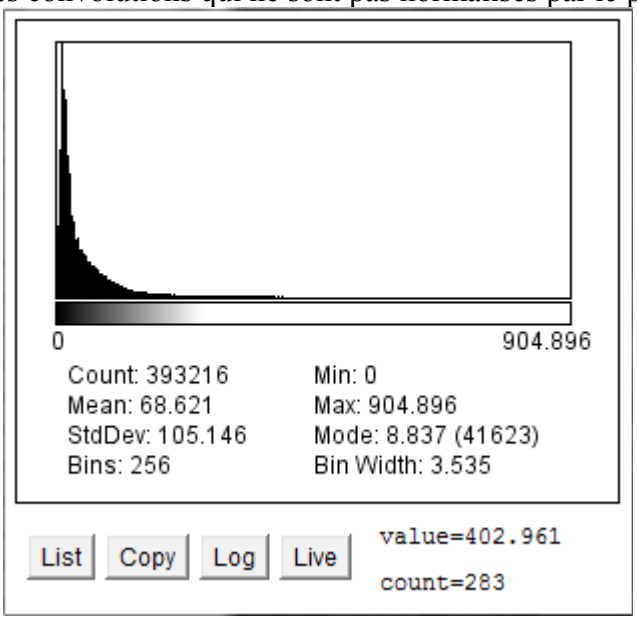
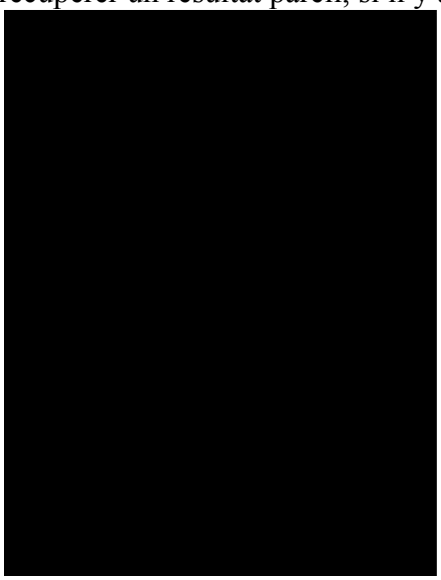


Figure 2 : histogramme de l'image résultant du calcul de la norme.

L'image obtenue par le calcul de la norme fourni est identique à celle obtenue par le calcul de notre macro. Donc nous ne voyons pas quelle manipulation faire pour obtenir la même. Voici le résultat de la comparaison entre les 2 images ainsi que son histogramme. Même avec toute la volonté du monde et avec une manipulation avec les outils d'imageJ, nous n'aurions pas pu récupérer un résultat pareil, si il y avait eu une différence.



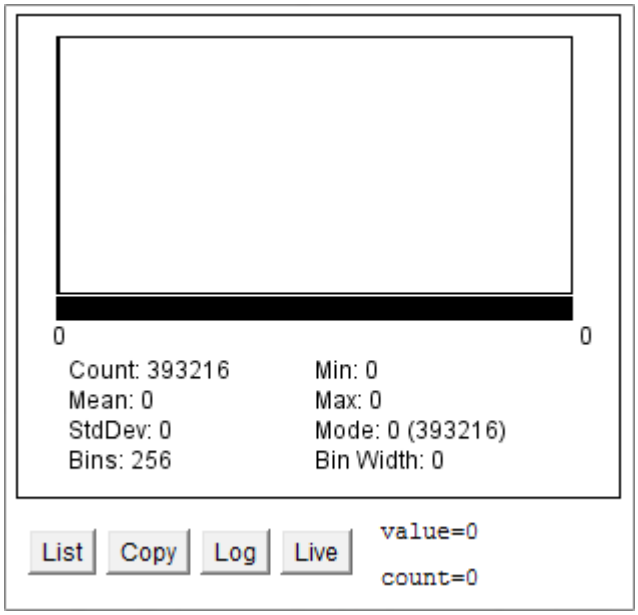
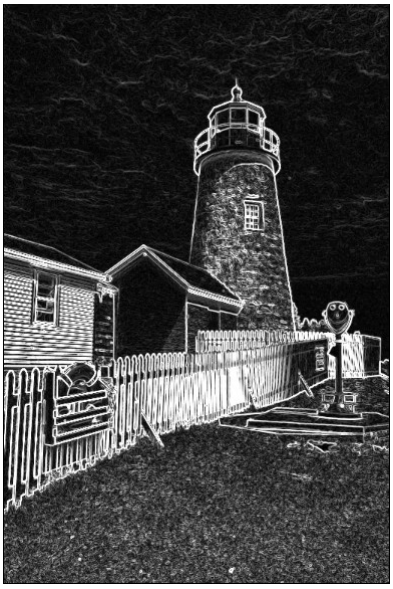


Figure de gauche : résultat de la comparaison.
Figure de droite : histogramme de l'image de gauche.

2. Seuillage de la norme du gradient précédemment calculée

Il est possible d'utiliser la fonction Threshold pour obtenir un seuillage de la norme du gradient calculée en voici un exemple en image. Avec comme valeur 0 et 84 pour le Brightness, nous obtenons l'image de droite à partir de l'image de gauche. On peut voir que le bruit du ciel et de l'herbe est atténué.



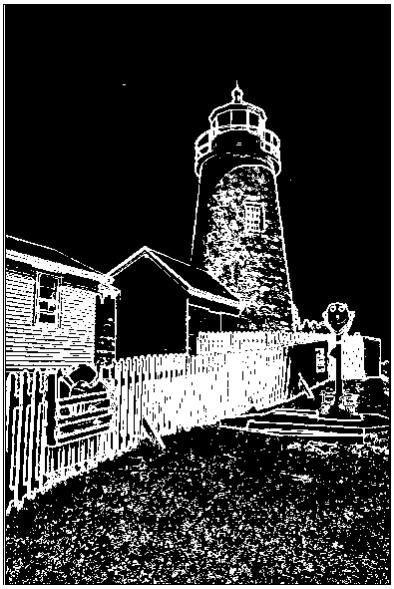


Figure de gauche : l'image de la norme calculée.
Figure de droite : l'image obtenue par le seuillage automatique.

Il est aussi possible d'utiliser une macro pour calculer le seuil de manière semi-automatique grâce à l'histogramme cumulé. L'histogramme cumulé de la norme du gradient est celui-ci :

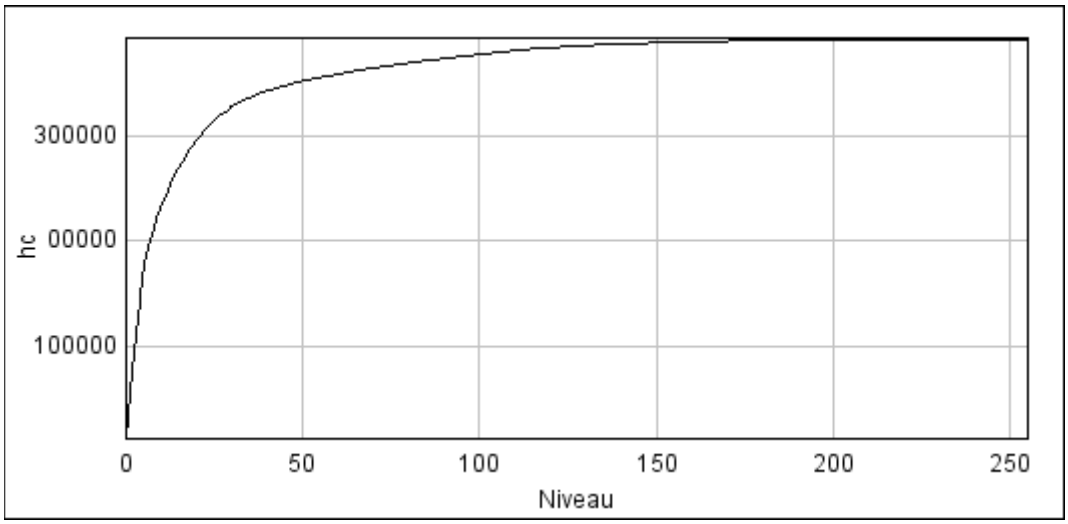


Figure 7 : histogramme cumulée de la norme du gradient calculée.

2. Détection des maxima locaux

1. Calcul de la direction du gradient

Voici le code pour calculer la macro de départ, pour obtenir la direction du gradient pour chaque pixel :

```
//macro
run("Conversions...", " ");
// Calcul de la norme du gradient par masque de Sobel
requires("1.41i");       // cause substring(string, index)
//permet d'éviter de voir les calculs fait sur les images
setBatchMode(true);

//image initiale
imageDeBase = getImageID();
filename = getTitle();
filenameWithoutExt = substring(filename, 0, lastIndexOf(filename, "."));
extension = substring(filename, lastIndexOf(filename, "."));

//nom des images pour le calcul des dérivees
filenameGradX = filenameWithoutExt+"_grad_x"+extension;
filenameGradY = filenameWithoutExt+"_grad_y"+extension;
run("32-bit"); // Conversion en Float avant calcul des derivees !!

//duplicata de l'image initiale pour le calcul de la derivees suivant X
run("Duplicate...", "title="+filenameGradX);
run("Convolve...", "text1=[-1 0 1\n-2 0 2\n-1 0 1\n] normalize");
imageGradX = getImageID();

//duplicata de l'image initiale pour le calcul de la derivees suivant Y
selectImage(imageDeBase);
run("Duplicate...", "title="+filenameGradY);
run("Convolve...", "text1=[-1 -2 -1\n0 0 0\n1 2 1\n] normalize");
imageGradY = getImageID();

// recuperation de la taille de l'image
w = getWidth();
h = getHeight();

//creation de la nouvelle image qui contiendra pour chaque pixel la direction du gradient
newImage("imageDirection", "32-bit", w, h, 1) ;
imageDirection = getImageID();

// Calculs pour chaque pixel
px = 0;
py = 0;
```

```
//calcul pour chaque pixel de la direction du gradient
for (j=0; j<h; j++) {
        for (i=0; i<w; i++) {
                selectImage(imageGradX);
                px=getPixel(i,j);
                selectImage(imageGradY);
                py=getPixel(i,j);
                selectImage(imageDirection);
                direction = atan2(py,px)*180/PI;
                setPixel(i,j,direction);
                }
}
```

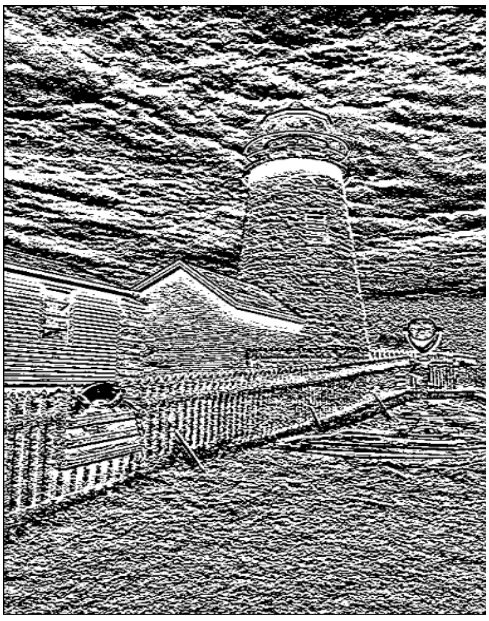


Figure 8 : image résultante du calcul de la direction du gradient.

2. Détection des maxima locaux

Voici le code permettant de calculer les maxima locaux :

```
//      x2  x3  x4
//      x1 CC x1
//      x4  x3  x2
// xi -> valeurs possibles pour les pixels autours
// x1<22.5<x2<67.5<x3<112.5<x4<157.5<x1
for(y=0;y<h;y++){
        for(x=0;x<w;x++){
                selectImage(imageDirection);
                direction=getPixel(x,y);
                direction=abs(direction);
                selectImage(imageNorme);
                if(direction>112.5){
                        if(direction<157.5){ //x4
                                if(getPixel(x,y)<getPixel(x-1,y+1)){
                                        selectImage(imageFinal);
                                        setPixel(x,y,0);
                                        selectImage(imageNorme);
                                        if(getPixel(x-1,y+1)<getPixel(x+1,y-1)){
```

```
                        selectImage(imageFinal);
                        setPixel(x-1,y+1,0);
                    }else{
                        selectImage(imageFinal);
                        setPixel(x+1,y-1,0);
                    }
                }else{
                    selectImage(imageFinal);
                    setPixel(x-1,y+1,0);
                    selectImage(imageNorme);
                    if(getPixel(x+1,y-1)<getPixel(x,y)){
                        selectImage(imageFinal);
                        setPixel(x+1,y-1,0);
                    }else{
                        selectImage(imageFinal);
                        setPixel(x,y,0);
                    }
                }//fin x4
            }else{//x1
                if(getPixel(x,y)<getPixel(x+1,y)){
                    selectImage(imageFinal);
                    setPixel(x,y,0);
                    selectImage(imageNorme);
                    if(getPixel(x+1,y)<getPixel(x-1,y)){
                        selectImage(imageFinal);
                        setPixel(x+1,y,0);
                    }else{
                        selectImage(imageFinal);
                        setPixel(x-1,y,0);
                    }
                }else{
                    selectImage(imageFinal);
                    setPixel(x+1,y,0);
                    selectImage(imageNorme);
                    if(getPixel(x-1,y)<getPixel(x,y)){
                        selectImage(imageFinal);
                        setPixel(x-1,y,0);
                    }else{
                        selectImage(imageFinal);
                        setPixel(x,y,0);
                    }
                }
            }//fin x1
        }else{
            if(direction>67.5){ //x3
                if(getPixel(x,y)<getPixel(x,y+1)){
                    selectImage(imageFinal);
                    setPixel(x,y,0);
                    selectImage(imageNorme);
                    if(getPixel(x,y+1)<getPixel(x,y-1)){
                        selectImage(imageFinal);
                        setPixel(x,y+1,0);
```

```
                }else{
                        selectImage(imageFinal);
                        setPixel(x,y-1,0);
                }
        }else{
                selectImage(imageFinal);
                setPixel(x,y+1,0);
                selectImage(imageNorme);
                if(getPixel(x,y-1)<getPixel(x,y)){
                        selectImage(imageFinal);
                        setPixel(x,y-1,0);
                }else{
                        selectImage(imageFinal);
                        setPixel(x,y,0);
                }
        }//fin x3
}else if(direction<22.5){ //x1
        if(getPixel(x,y)<getPixel(x+1,y)){
                selectImage(imageFinal);
                setPixel(x,y,0);
                selectImage(imageNorme);
                if(getPixel(x+1,y)<getPixel(x-1,y)){
                        selectImage(imageFinal);
                        setPixel(x+1,y,0);
                }else{
                        selectImage(imageFinal);
                        setPixel(x-1,y,0);
                }
        }else{
                selectImage(imageFinal);
                setPixel(x+1,y,0);
                selectImage(imageNorme);
                if(getPixel(x-1,y)<getPixel(x,y)){
                        selectImage(imageFinal);
                        setPixel(x-1,y,0);
                }else{
                        selectImage(imageFinal);
                        setPixel(x,y,0);
                }
        } //fin x1
}else{ //x2
        if(getPixel(x,y)<getPixel(x+1,y+1)){
                selectImage(imageFinal);
                setPixel(x,y,0);
                selectImage(imageNorme);
                if(getPixel(x+1,y+1)<getPixel(x-1,y-1)){
                        selectImage(imageFinal);
                        setPixel(x+1,y+1,0);
                }else{
                        selectImage(imageFinal);
                        setPixel(x-1,y-1,0);
                }
```

```
                    }else{
                        selectImage(imageFinal);
                        setPixel(x+1,y+1,0);
                        selectImage(imageNorme);
                        if(getPixel(x-1,y-1)<getPixel(x,y)){
                            selectImage(imageFinal);
                            setPixel(x-1,y-1,0);
                        }else{
                            selectImage(imageFinal);
                            setPixel(x,y,0);
                        }
                    } //fin x2
                }
            }
        }
}
```

Et voici l'image obtenue :

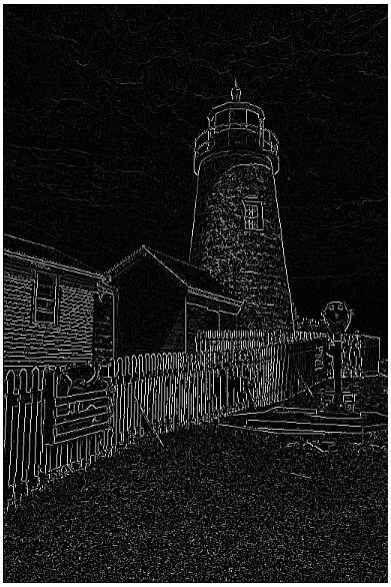


Figure 9 : image obtenue en utilisant les maxima locaux.

On peut constater que les contours sont beaucoup plus nets et moins épais qu'avec l'image de la norme précédemment calculée. Cette technique est plus fiable que cette de la norme. Mais il reste du bruit au niveau de l'herbe.

3. Seuillage des maxima locaux par hystérésis

1. Analyse d'un plugin existant

La méthode du seuillage par hystérésis nous permet de réduire le bruit et de fermer des coutours à trous.Voici le code commenté de la méthode hystIter :

```
public ByteProcessor hystIter(ImageProcessor imNormeG, int seuilBas, int seuilHaut) {
   //taille de l'image
   int width = imNormeG.getWidth();
   int height = imNormeG.getHeight();
   ByteProcessor maxLoc = new ByteProcessor(width,height);
   List<int[]> highpixels = new ArrayList<int[]>();
```

```
    // parcours de chaque pixel
    // si la valeur du pixel est superieur au seuilHaut => on ajoute un pixel à 255
    for (int y=0; y<height; y++) {
      for (int x=0; x<width; x++) {
        int g = imNormeG.getPixel(x, y)&0xFF;
        if (g<seuilBas) continue;
        if (g>seuilHaut) {
          maxLoc.set(x,y,255);
          highpixels.add(new int[]{x,y});
          continue;
        }
        maxLoc.set(x,y,128);
      }   }
    //création de 2 tableaux contenant les indices des pixels autours du pixel choisi
    int[] dx8 = new int[] {-1, 0, 1,-1, 1,-1, 0, 1};
    int[] dy8 = new int[] {-1,-1,-1, 0, 0, 1, 1, 1};
    List<int[]> newhighpixels = new ArrayList<int[]>();
    // parcours de la liste des pixels qui ont une valeur > seuilHaut
    while(!highpixels.isEmpty()) {
      newhighpixels.clear();
      for(int[] pixel : highpixels) {
        int x=pixel[0], y=pixel[1];
        //parcours des pixels autours du pixel choisi
        //on regarde la valeur de chaque pixel autour et on prend les + grandes valeurs
        // on le définit comme le gradient
        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;
          if (maxLoc.get(xk, yk)==128) {
            maxLoc.set(xk, yk, 255);
            newhighpixels.add(new int[]{xk, yk});
          }//fin if        }//fin for     }//fin deuxieme for
      List<int[]> swap = highpixels; highpixels = newhighpixels; newhighpixels = swap;
    }
    //tous les pixels avec comme valeur!= 255, on les met à 0
    for (int y=0; y<height; y++) {
      for (int x=0; x<width; x++) {
        if (maxLoc.get(x, y)!=255) maxLoc.set(x,y,0);
      }//fin for    } //fin for
    return maxLoc;
}
```

Conclusion

En conclusion, on peut dire que la méthode du seuillage par hystérésis demande beaucoup plus de calcul mais est beaucoup plus représentatif de la détection des contours dans une image. Malheureusement, cette méthode n'est pas parfaite car il reste encore des « trous » pour fermer des contours.

Par contre, la méthode des maxima locaux restitue une image avec des contours fermés tout en aillant encore des zones de bruit comme l'herbe.

Enfin, la méthode en ne tenant compte que de la norme du gradient n'est pas suffisante pour restituer un contour net. Il faut la couplet avec la méthode des maxima locaux.