

CR : Traitement d'image

Introduction

On a travaillé sous l'IDE, MS Visual Studio. La base de projet fournie utilisait la librairie openCV et on pouvait l'utiliser. Mais, pour mettre en pratique la connaissance acquise en cours, on implémentait nous-même les différentes techniques du traitement d'image.

Objectif

On a pour objectif d'un algorithme de reconnaissance de panneaux routiers pour l'assistance à la conduite. Les images à traiter proviennent de différentes caméras avec des caractéristiques de capteurs différentes, leur taille et leur qualité est variable. La caméra étant embarquée dans un véhicule. Les conditions d'acquisition peuvent donc varier. Finalement, suivant la distance du panneau, sa taille est variable dans l'image et le contenu de l'image également.

La deuxième application concerne le comptage de bâtiments vus du ciel.

Contenu

Introduction

Objectif

Exercice 1 : Histogrammes et espaces couleurs

Exercice 2 : Modification d'histogramme

Exercice 3 : Représentation fréquentielle

Exercice 4 : Convolution et filtrage dans l'image

Exercice 5 : Classification

Exercice 6 : Morphologie mathématique

Exercice 7 : Etiquetage en composantes connexes

Exercice 8 : détection de contours

Exercice 9 : Analyse de formes, reconnaissance de panneaux routiers

Annexes

Exercice 1 : Histogrammes et espaces couleurs

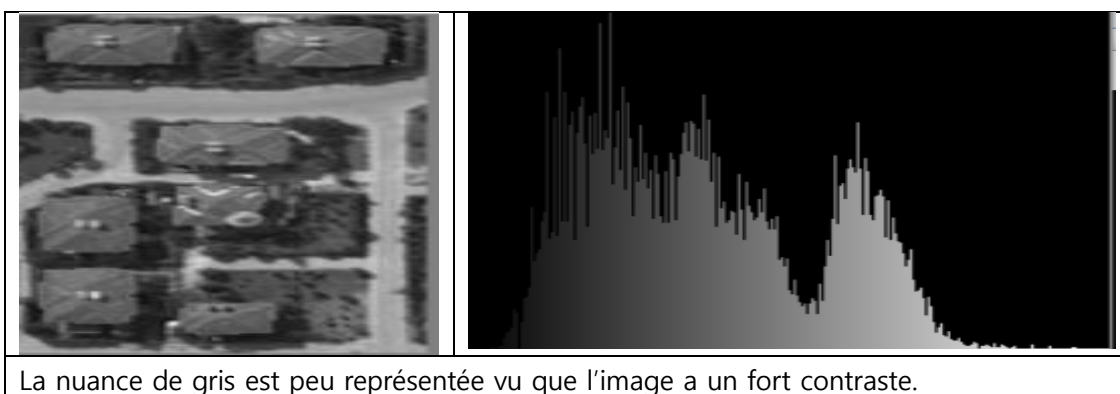
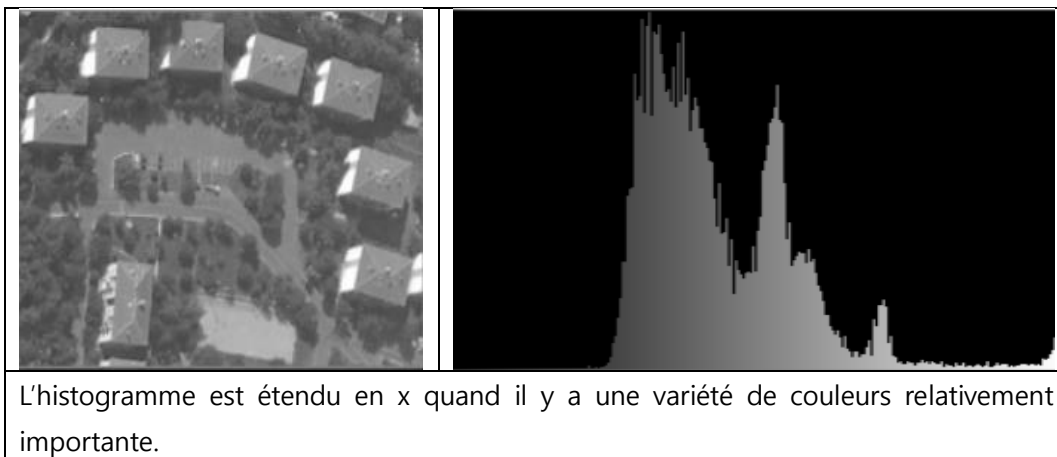
La fonction `histo_1D` calcule l'histogramme d'une composante couleur d'une image d'entrée.

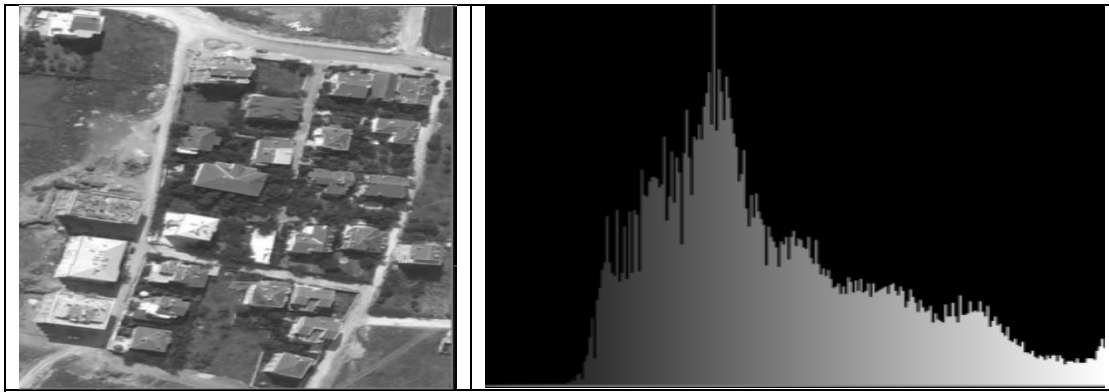
On a testé les images de différentes natures, voici le résultat obtenu

Image	Histogramme
Peu de variétés de couleurs	L'axe des abscisses peu étendue
Une grande variété de couleurs	L'axe des abscisses très étendue
Très contrastée	Peu de valeurs élevées sur l'axe y
Peu contrastée	Comporte des « pics » sur l'axe y

*le code de cette fonction est donné en annexe.

Le cas d'un histogramme à dimension 1





La nuance de gris est relativement bien représentée vu que l'image a un contraste faible.

image initiale :	dans l'espace Lab :	la composante L
dans l'espace RGB :	dans l'espace rgb	dans l'espace HSV :
la teinte	la saturation	la luminance

->À quoi correspondent les images L (de l'espace Lab) et V de l'espace (HSV) ?

L correspond à la clarté (la [clarté](#) L^* dérive de la [luminance](#) de la surface ; les deux paramètres a^* et b^* expriment l'écart de la couleur par rapport à celle d'une surface grise de même clarté.). V correspond à l'intensité

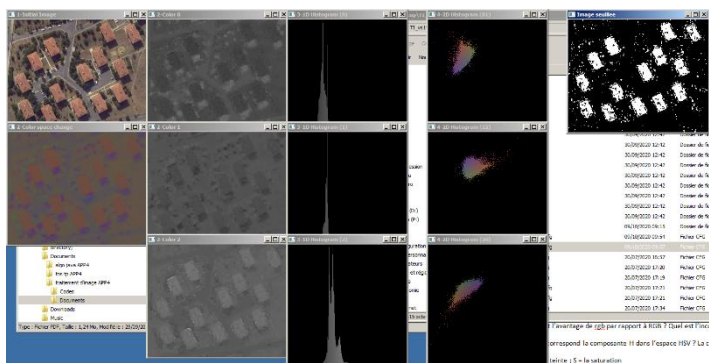
-> À quoi correspond la composante H dans l'espace HSV ? La composante S ?

H = hue = teinte ; S = la saturation ; V = la luminosité ou la valeur

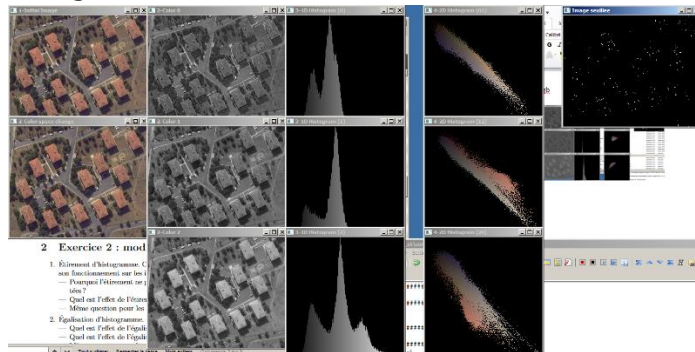
-> Observez également les histogrammes 2D. Quels espaces couleur sont à priori les plus adaptés pour répondre à notre problème, c'est-à-dire qui permet d'isoler les pixels de toit ?

D'après l'observation des histogramme 2D, les espaces de couleurs rgb, et Lab sont les mieux adaptés pour l'isolation des pixels de toit. Il faut choisir l'espace de couleur où la couleur de toit (qui est un peu orange) est bien séparée des autres couleurs.

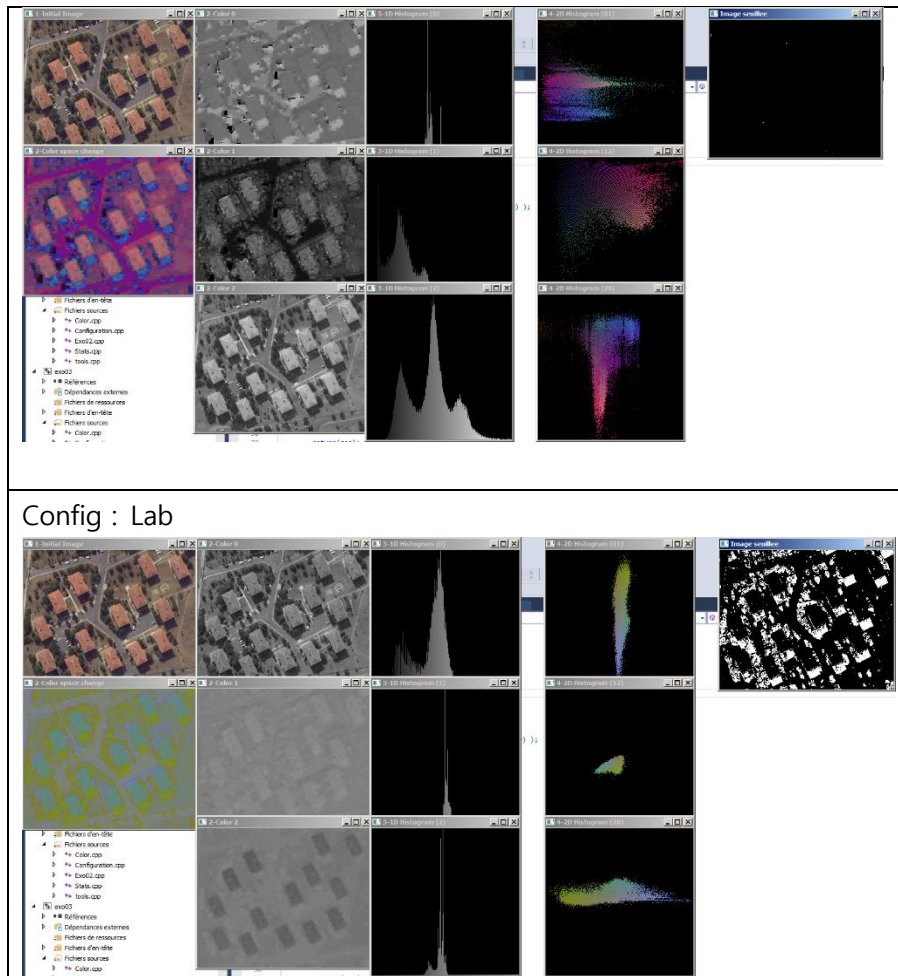
Config : rgb



Config : RGB



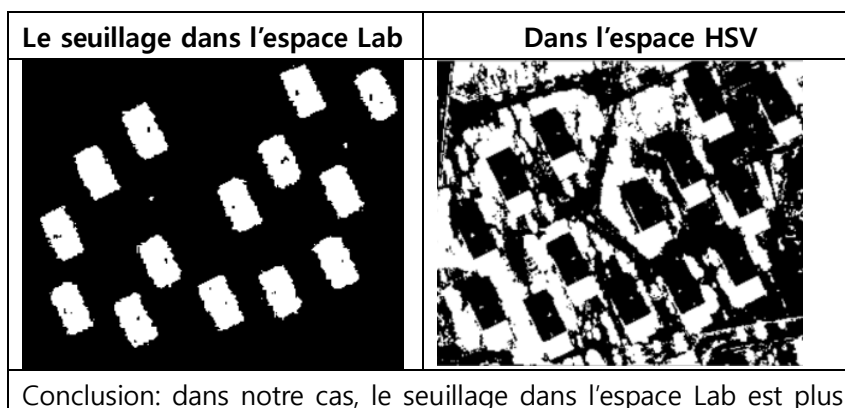
Config : HSV



Le seuillage :

* la fonction « threshold » se trouve dans l'annxe.

La fonction « threshold » parcourt les images en lignes avant de les parcourir en colonnes. Chaque pixel de la matrice de sortie ne peut prendre que 2 valeurs, 0 ou 255. Cette valeur de sortie dépend de la comparaison de la nuance de gris.


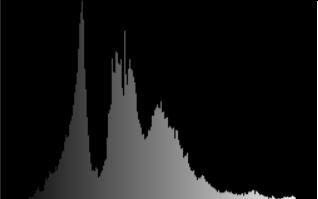

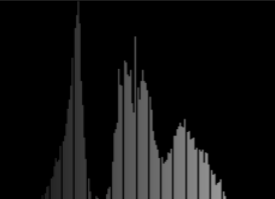
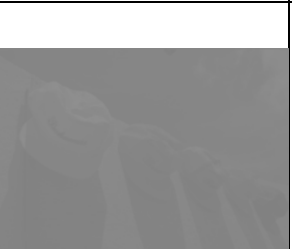


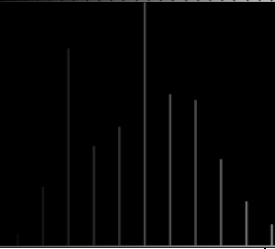

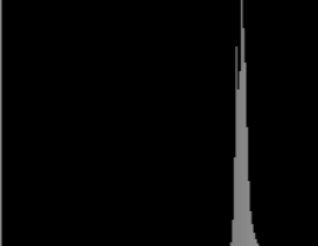

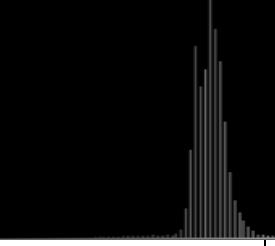


efficace.

Exercice 2 : Modification d'histogramme

* la fonction « HistoStretching » permet l'étirement d'histogramme. Il se trouve dans l'annexe.

Comparons quelques images avec des caractéristiques variées :

avant étirement		Après étirement	
			
			
			

Conclusion :

L'étirement d'histogramme est le plus efficace quand l'image contient très peu de couleurs.

Dans le cas d'une image bruitée, il réduit un peu le bruit, mais il n'est pas très efficace.



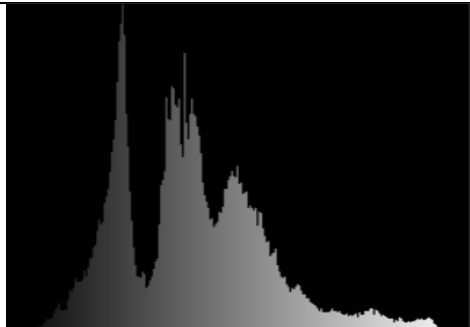
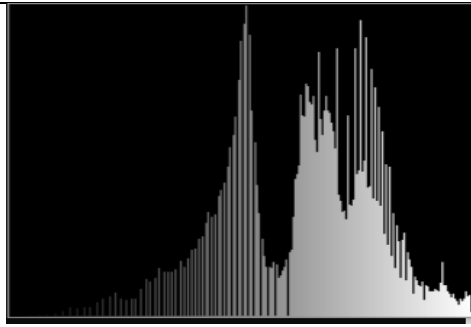
-> Pourquoi l'étirement ne permet-il pas d'améliorer la qualité de certaines images peu contrastées ?



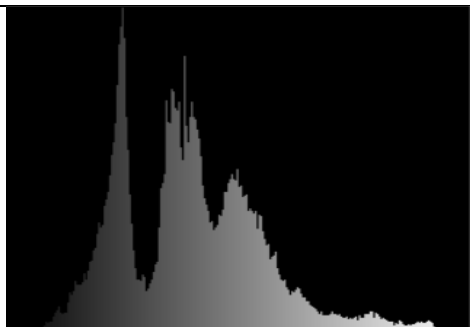
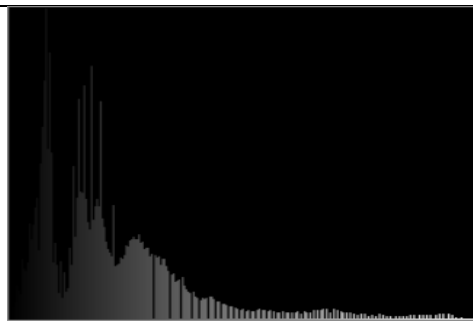
Si l'image ne contient que des noirs et des blancs. L'étirement ne peut pas faire grande chose vu qu'ils sont déjà aux intensités maximas. 0 et 255.

Le cas des transformations non-linéaires d'histogramme :

* Il y a les transformations logarithmiques et exponentielles. Leurs codes sources se trouvent dans

l'annexe.

Avant transformation log	Après la transformation
	
	
remarque : l'image est éclaircie ! l'histogramme est décalé vers la droite.	

Avant transformation exp	Après la transformation
	
	
remarque : l'image est plus sombre après la transformation. l'intensité d'histogramme est baissée partout, et il est décalé vers la gauche.	

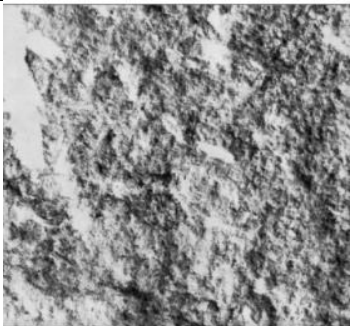
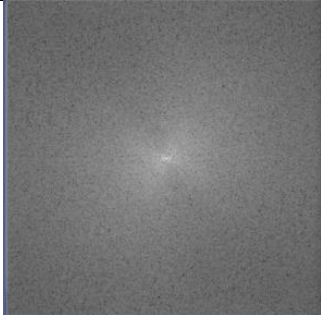
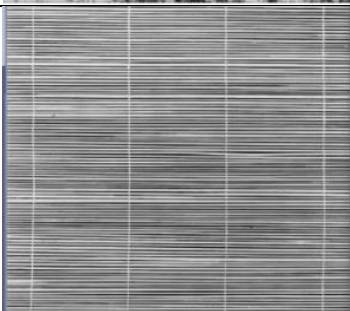
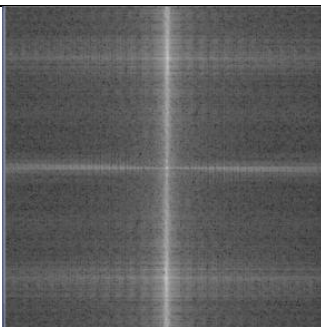
Conclusion : la transformation $\exp()$ amplifie le contraste des valeurs de gris élevés et la transformation $\log()$ amplifie celles de gris faibles.

Exercice 3 : Représentation fréquentielle

* Le FFT en z est une fonction qui récupère la partie réelle Re et imaginaire Im de la transformée de Fourier que l'on stocke dans une matrice.

* Le FFT shift est une fonction qui permet d'obtenir la transformée de Fourier.

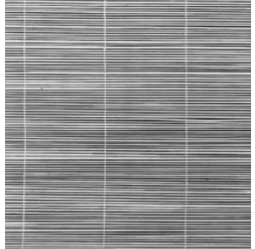
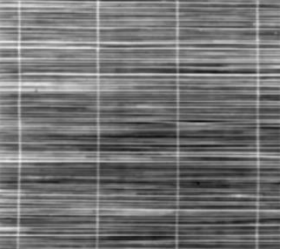
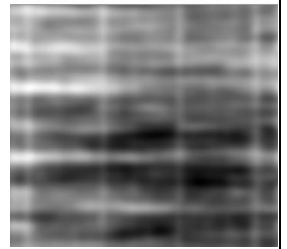
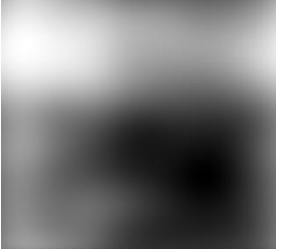
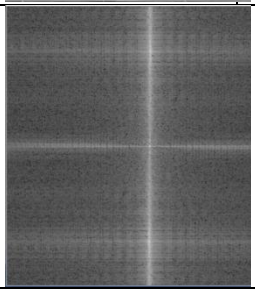
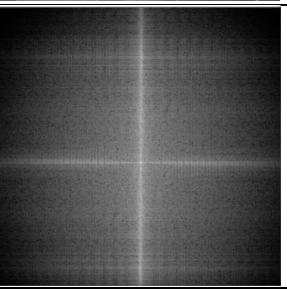
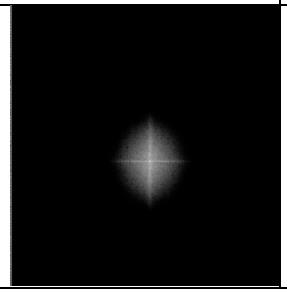
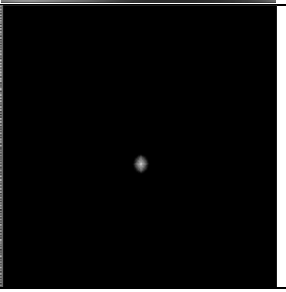
* Leurs codes sources se trouvent dans l'annexe.

image		spectre	
			
			
<p>Conclusion :</p> <p>Dans le premier image, on a une image non structure, donc son spectre ne montre pas ses fréquences.</p> <p>Dans le deuxième image, on a une image qui est structurée, donc on retrouve les différentes fréquences correspondant aux rayures de l'image dans son spectre.</p>			

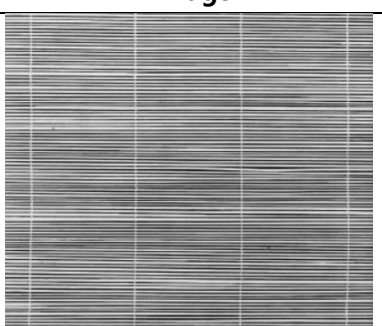
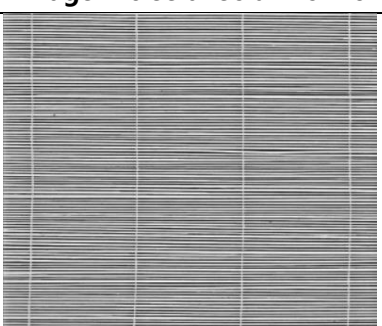
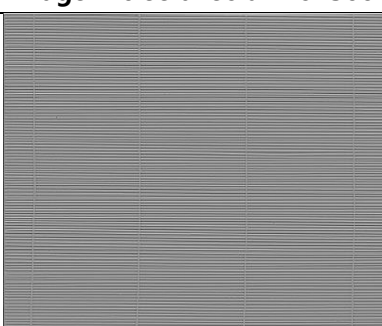
On effectuera maintenant un filtre passe-bas pour filtrer les hautes fréquences pour caractériser les basses fréquences. Ce filtre est un filtre gaussien PB en 2 dimensions. Cette fenêtre n'est pas carrée, on peut ainsi éviter le phénomène de Gibbs.

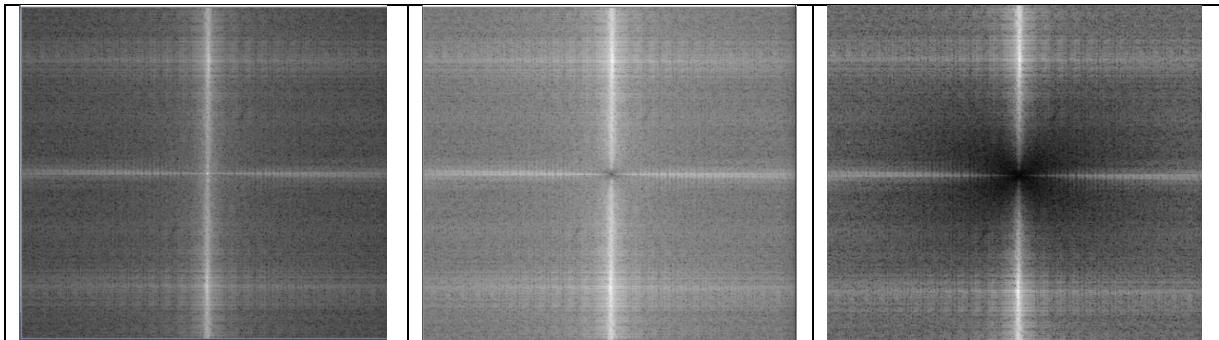
* la fonction GaussLowPass se trouve dans l'annexe

On fait varier la fréquence de coupure f_c que l'on applique aux mêmes images, on peut donc voir l'impact de la fréquence de coupure f_c et caractériser les basses fréquences de l'image.

image	image filtrée avec $P_b = 50$	avec $P_b = 10$	avec $P_b = 2$
			
			
<p>Conclusion :</p> <p>Ce filtre lise les contours et les bruits impulsionnels.</p>			

On va maintenant réaliser un filtre passe-haut pour caractériser les hautes fréquences. Pour créer ce filtre on fait : 1-Gabarit d'un filtre passe-bas. On fera varier la fréquence de coupure que l'on applique aux mêmes images, on pourra ainsi voir l'impact de la fréquence de coupure f_c et caractériser les hautes fréquences de l'image.

image	image filtrée avec un $f_c=10$	image filtrée avec un $f_c=500$
		



Conclusion :

Les images filtrées par un filter passe haut sont mieux définies ainsi que leurs contours.

Donc, la **troncature** peut être **utile** pour **mettre en avant** les **contours** d'un **objet**.

Exercice 4 : Convolution et filtrage dans l'image

On effectuera maintenant une convolution. C'est une opération mathématique entre deux signaux, qui permet une convolution de l'image avec un noyau pour l'extraction de certaines caractéristiques.

Attention : La formule qu'on utilise pour un signal en 1D et en 2D n'est pas la même.

* la fonction convolution se trouve dans l'annexe.

On applique d'abord un filtre de Sobel à l'image initiale suivante et cette image après application de la fonction présentée précédemment :



Explication : Ce filtre fait la norme de gradient ce qui induit le blanc au niveau des contours et noir ailleurs.

* l'algo pour effectuer la convolution gaussienne pour le filtrage médian se trouve dans l'annexe

image filtrée en gaussien avec $w = 6$	avec $w = 48$
	
Remarque : Plus le noyau w d'un filter gaussien est grand plus l'image est floue.	

Comparaison entre un filtre médian et un filtre gaussien, $w = 6$:

Image	Image avec filtrage gaussien	Image avec filtrage médian
		
		
		



Exercice 5 : Classification

On fera maintenant une classification. C'est le fait de ranger les données d'une image dans différents groupes avec des caractéristiques similaires comme la couleur, ce qui va nous permettre de détecter les panneaux de circulation dans les domaines RGB et rgb.

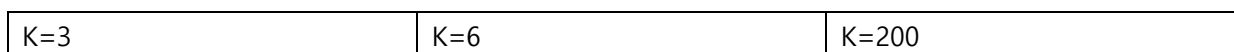


Figure : image initiale

Dans l'espace RGB :



Dans l'espace rgb :





Conclusion :

Dans notre cas, l'espace rgb est mieux adapté pour l'identification des panneaux de circulation.

Exercice 6 : Morphologie mathématique

On commence cet exercice en complétant les fonctions de dilatation et d'érosion dans la classe morpho, puis on vérifie sur les quatre images initiales suivantes (que l'on notera de 1 à 4 de gauche à droite) le fonctionnement des deux fonctions :




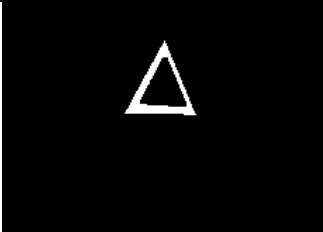
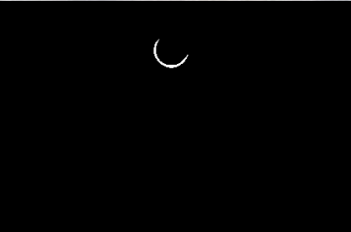

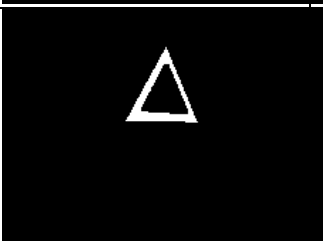
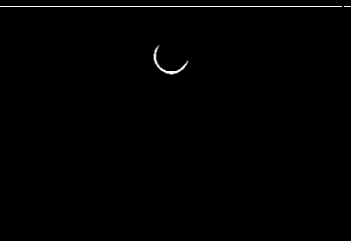
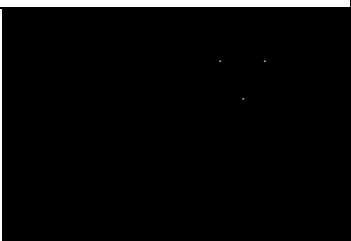
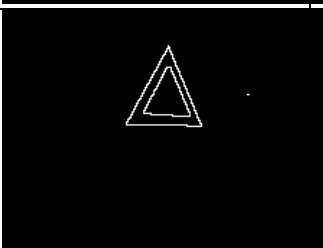
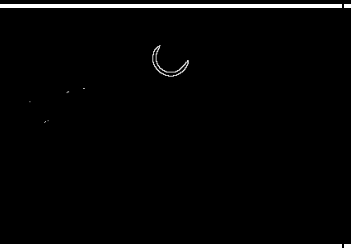
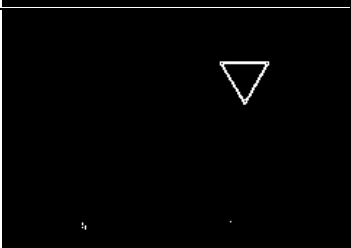
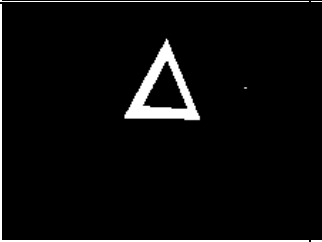

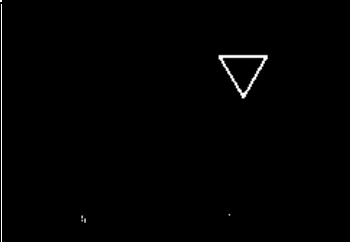
image			
érosion			
dilatation			
<p>Conclusion :</p> <p>La dilatation est le mieux adapté dans notre cas.</p>			

L'**érosion** permet d'étudier les relations « est inclus dans l'ensemble » $\epsilon_B(X) = X \ominus B = \{x \mid B_x \subset X\}$

La **dilatation** permet d'étudier les relations « intersecte avec l'ensemble ». La dilatation morphologique avec l'élément structurant B est définie comme la somme de Minkowski :

$$\delta_B(X) = X \oplus B = \{x + b \mid b \in B, x \in X\} = \cup_{x \in X} B_x$$

Ensuite en utilisant les fonctions d'érosion et de dilatation, on complète les fonctions d'**ouverture** (qui est une érosion puis une dilatation), **fermeture** (qui est une dilatation puis une érosion), et contours.


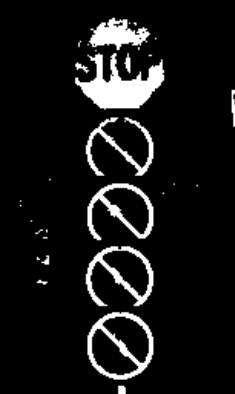
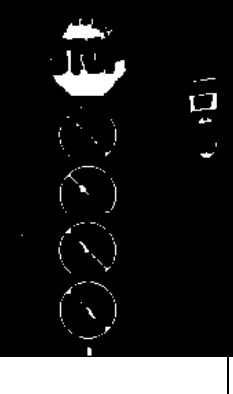



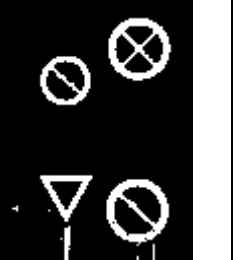
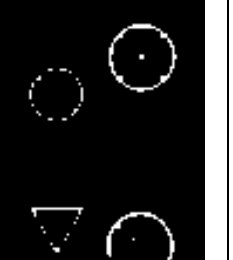
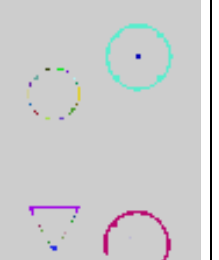

image			
ouverture			
fermeture			
contour			
binarisation			

Conclusion :

Dans le cas de détection des panneaux, la binarisation est le mieux adapté.

Exercice 7 : Etiquetage en composantes connexes

Avant de réaliser une analyse de formes à partir des images binaires obtenues préalablement, il est nécessaire de faire un étiquetage en composantes connexes. Pour le faire, on a complété la méthode « régions » dans la classe « segmentation », cette méthode effectue l'étiquetage en composantes connexes à partir d'une image binaire.

Image	Image binaire	Image après ouverture	Image segmenté avec ouverture	Image segmenté sans ouverture
				
				

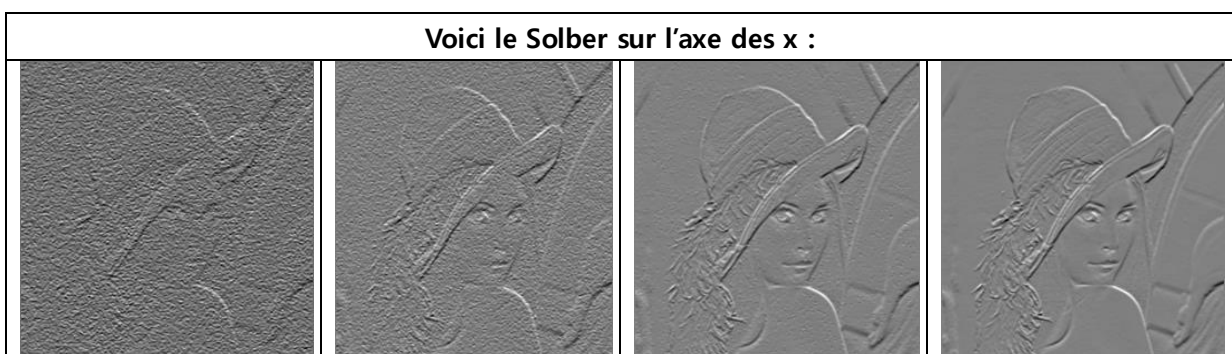


On peut voir que pour la détection des panneaux avec la segmentation, le fait d'utiliser l'ouverture avant la segmentation dégrade la reconnaissance, il est donc préférable d'utiliser la segmentation seule pour la reconnaissance de panneaux.

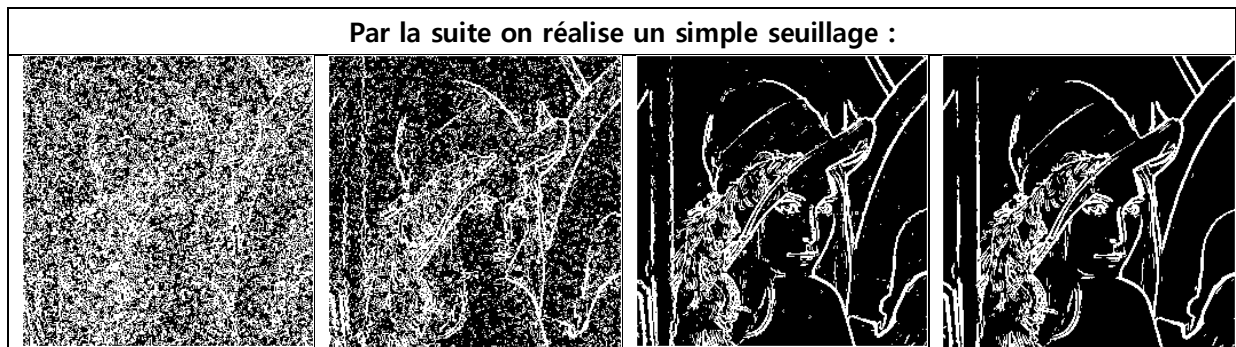
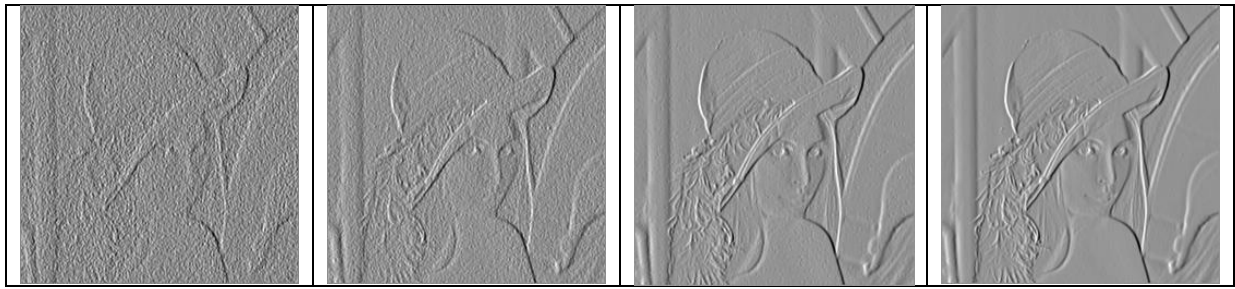
Exercice 8 : détection de contours



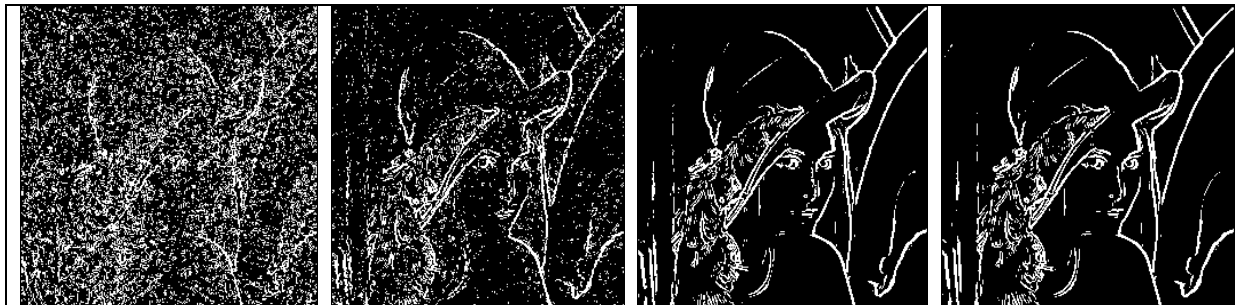
On commence par appliquer un Sobel sur les axes x et y des images.



Puis sur l'axe des y :



* la fonction de reconstruction géodésique (qui est au cœur de la fonction hystérésis) se trouve dans l'annexe. Cette fonction va nous permettre de réaliser un seuillage par hystérésis



On peut voir que le « seuillage par hystérésis » est une méthode plus efficace que le « seuillage simple », les contours sont plus marqués.

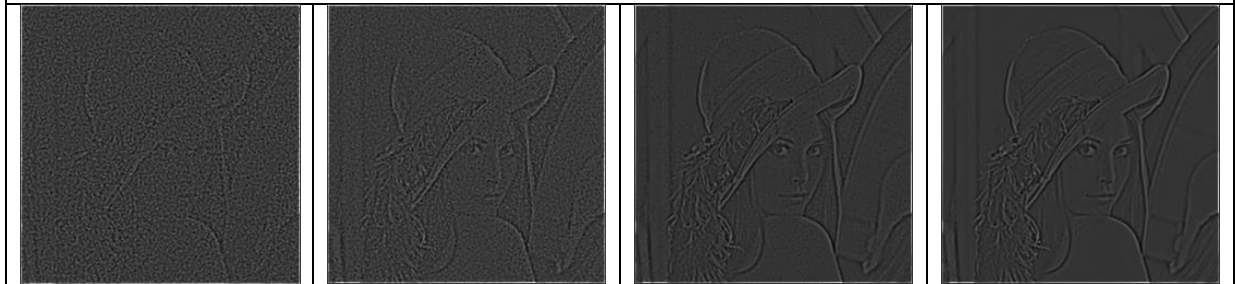
* la fonction zeros de la classe Morpho se trouve dans l'annexe.

On va maintenant détecter les contours avec les passages par zéro du laplacien.

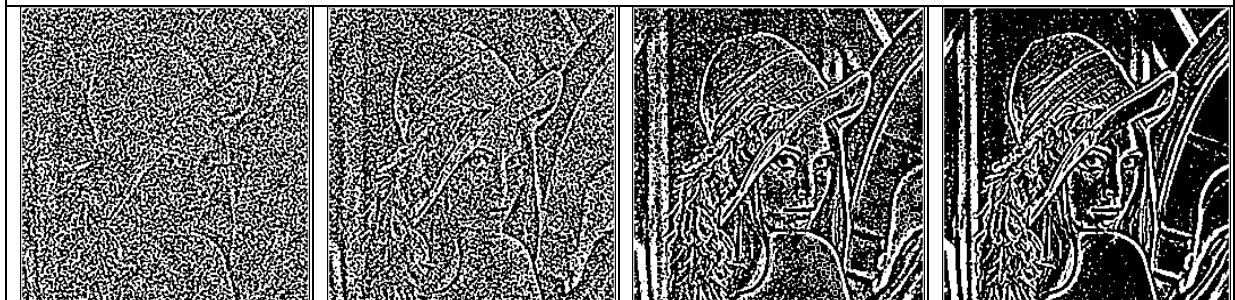
Voici les images après application du filtre gaussien :



Nous avons donc ici les images avec l'application du Laplacien au filtre gaussien :



On applique ensuite la détection des passages à 0 :



On observe que cette méthode n'est pas efficace.

Exercice 9 : Analyse de formes, reconnaissance de panneaux routiers

Dans cet exercice on a pour objectif de reconnaître les formes suivantes : rond, triangle et hexagone. La méthode compute de la classe Shape calcule, à partir d'une image d'étiquettes de régions (ou de contours), les descripteurs de forme suivants :

Descripteur	attribut de Shape
aire de la région	t_area
périmètre	t_perim
centre d'inertie	t_center
limites de la boîte englobante	t_limits
moments spatiaux, centrés et normalisés ¹	t_moments
les 7 moments invariants de Hu ²	t_hu
la signature du contours ³	t_sig

* la classe compute de la classe Shape se trouve dans l'annexe.

Dans cet exercice la méthode compute récupère la liste des distances entre chaque point du contour d'une région et le centre de la région, ce qui lui permet de déterminer le nombre de maximums. Pour déterminer les maximums, on détermine les maximums locaux, puis on vérifie si ces maximums locaux sont supérieurs à 80% de la distance maximale au centre, cette dernière dépend de chaque forme (triangle...).

S'il y en a trois, la région c'est un triangle, s'il y en a 4, il s'agit soit d'un carré soit d'un rectangle, 6 un hexagone, et enfin si le nombre est élevé il s'agit d'un cercle.

Eventuelle amélioration :

On pourra utiliser la « transformée de Hough ». Cette méthode est souvent utilisée pour détecter les lignes d'une image, mais elle pourra également permettre de détecter les formes géométriques.

Annexe

Exercice 1 : fonction histo_1D

```

for (int i = 0; i < im.rows; i++)
    for (int j = 0; j < im.cols; j++) {
        uchar val = tmp.at<uchar>(i, j);
        H1D.at<float>(val) += 1.0;
        float h = H1D.at<float>(val);
        if (h > hist_max) hist_max = h;
    }

```

Exercice 2 : les fonctions threshold, histostretching, et logTransform

<pre> Mat Clustering::threshold(Mat I, int t1, int t2) { Mat res(I.rows, I.cols, CV_8U); for(int i=0; i< I.rows; i++) for(int j=0; j< I.cols; j++) { if (I.at<uchar>(i,j)>=t1 && (I.at<uchar>(i,j)<t2)) res.at<uchar>(i,j)=255; else res.at<uchar>(i,j)=0; } return(res); } </pre>	<pre> Mat Color::HistoStretching(Mat It) { Mat Jt; double minVal, maxVal; minMaxLoc(It, &minVal, &maxVal); float b = (-minVal * 255) / (maxVal - minVal); float a = 255 / (maxVal - minVal); It.convertTo(Jt, (CV_8U), a, b); return(Jt); } </pre>
<pre> Mat Color::LogTransform(Mat It, int R) { double minVal, maxVal; float c = 1/cv::log(1+R); Mat Jt(It.rows, It.cols, CV_32F); Mat out(It.rows, It.cols, CV_8U); It.convertTo(Jt, (CV_32F), 1,1); cv::log(Jt, Jt); Jt = Jt * c; minMaxLoc(Jt, &minVal, &maxVal); Jt.convertTo(out, (CV_8U), 255.0/(maxVal - minVal), -minVal * 255.0/(maxVal - minVal)); return(out); } </pre>	<pre> Mat Color::ExpTransform(Mat It, int R) { double minVal, maxVal; float c = 1/cv::exp(1+R); Mat Jt(It.rows, It.cols, CV_32F); Mat out(It.rows, It.cols, CV_8U); It.convertTo(Jt, (CV_32F), 1/255.,0); minMaxLoc(Jt, &minVal, &maxVal); cout << "min " << minVal << " max:" << maxVal << endl; pow(Jt, 1.5, Jt); cv::exp(Jt, Jt); minMaxLoc(Jt, &minVal, &maxVal); cout << "min " << minVal << " max:" << maxVal << endl; Jt = Jt * c; minMaxLoc(Jt, &minVal, &maxVal); Jt.convertTo(out, (CV_8U), 255.0/(maxVal - minVal), -minVal * 255.0/(maxVal - minVal)); return(out); } </pre>

Exercice 3 :

```

Mat Frequency::fft_z(Mat I)
{
    Mat Ipad;
    copyMakeBorder(I, Ipad, 0, DFT_rows - I.rows, 0, DFT_cols - I.cols, BORDER_CONSTANT, Scalar::all(0));

    Mat planes[] = {Mat_<float>(Ipad), Mat::zeros(Ipad.size(), CV_32F)};
    Mat complexI;
    merge(planes, 2, complexI);

    dft(complexI, complexI);
    cout << " " << complexI.type() << " " << complexI.channels() << endl;
    split(complexI, imTFD); //split source

    return(complexI);
}

Mat Frequency::fftshift(Mat im)
{
    Mat imF;
    im.copyTo(imF);
    imF = im(Rect(0, 0, im.cols & -2, im.rows & -2));
    int cx = imF.cols/2;
    int cy = imF.rows/2;
    Mat q0(imF, Rect(0, 0, cx, cy)); // Top-Left - Create a ROI per quadrant
    Mat q1(imF, Rect(cx, 0, cx, cy)); // Top-Right
    Mat q2(imF, Rect(0, cy, cx, cy)); // Bottom-Left
    Mat q3(imF, Rect(cx, cy, cx, cy)); // Bottom-Right
    Mat tmp; // swap quadrants (Top-Left with Bottom-Right)
    q0.copyTo(tmp);
    q3.copyTo(q0);
    tmp.copyTo(q3);
    q1.copyTo(tmp);
    q2.copyTo(q1);
    tmp.copyTo(q2);

    split(imF, imTFD); //split source
    return(imF);
}

```



```
Mat imFFT_f;
imFFT0.copyTo(imFFT_f);
Mat tmp[2];
int nrows = imFFT0.rows;
int ncols = imFFT0.cols;
int cr = nrows/2;
int cc = ncols/2;
Mat G;
G.create(nrows, ncols, CV_32F);

for(int fr=0; fr<nrows; fr++)
    for(int fc=0; fc<ncols; fc++) {
        float dr = (fr - cr) ;
        float dc = (fc - cc) ;

        G.at<float>(fr,fc) = exp( -dr * dr / (2*sigma_r*sigma_r) -dc* dc / (2*sigma_c*sigma_c) );
    }

split(imFFT0, imTFD);

multiply(imTFD[0], G, imTFD[0]);
multiply(imTFD[1], G, imTFD[1]);

merge(imTFD, 2, imFFT_f);

return(imFFT_f);
```

Filtre PB

```
Mat tmp[2];
int nrows = imFFT0.rows;
int ncols = imFFT0.cols;
int cr = nrows/2;
int cc = ncols/2;

Mat G;
G.create(nrows, ncols, CV_32F);

for(int fr=0; fr<nrows;fr++)
    for (int fc = 0; fc < ncols; fc++) {
        float dr = (fr - cr);
        float dc = (fc - cc);

        G.at<float>(fr, fc) = 1- exp(-dr * dr / (2 * sigma_r * sigma_r) - dc * dc / (2 * sigma_c * sigma_c));
    }

split(imFFT0, imTFD);

multiply(imTFD[0], G, imTFD[0]);
multiply(imTFD[1], G, imTFD[1]);

merge(imTFD, 2, imFFT0);

return(imFFT0);
```

Filtre PH

```
Mat J;
J.create(I.rows, I.cols, CV_32F);
Zeros(J);

for(int i=(K.rows/2); i<I.rows-(K.rows/2); i++)
    for (int j = (K.cols/2); j < I.cols - (K.cols/2); j++) //parcours de l'image ligne puis colonne sans aller sur les bords
        for(int k=0;k<K.rows;k++)
            for (int h = 0; h < K.rows; h++) { //parcours du noyau ligne puis colonne
                J.at<float>(i, j) = J.at<float>(i, j) + (float)I.at<uchar>(i + k - (K.rows / 2), j + h - (K.rows / 2)) *
                    K.at<float>(k, h);
            }

return(J);
```

Convolution gaussienne pour effectuer un filtrage médian

```
Mat g(W,W, CV_32F);
Zeros(g);
int sigma= W/2;
float A = 1. / (2*PI*sigma*sigma);

for(int i = -sigma; i <= sigma; i++)
    for (int j = -sigma; j <= sigma; j++) {
        g.at<float>(i + sigma, j + sigma) = (float)A * (exp(((double)-((double)i * i + (double)j * j) / (2 * (double)sigma * sigma))));
    }
return(g);
```

```
Mat Morpho::reconstruction(const Mat& src1, const Mat& src2)
{
    Mat A, B, C;
    int n;
    src1.copyTo(A);
    do {
        B = dilate(A, 1);
        bitwise_and(B, src2, C);
        n = cv::countNonZero(C - A);
        C.copyTo(A);
    } while (n > 0);

    return(C);
}
```

```
Mat Morpho::zeros(const Mat& src)
{
    Mat I;
    src.convertTo(I, (CV_8U), 1, 0);
    threshold(I, I, 0, 255, CV_8U);
    Mat J = contours(I, 1);
    return(I);
}
```

```
void Shape::compute(const Segmentation& s)
{
    cout << "debut shape " << endl;
    if(!s.Iregions.empty()) {
        Mat r(s.rows, s.cols, CV_8U);
        double hu[7];
        // processing of all regions
        for(int k=1; k<=s.nr; k++) {
            t_area[k]=0; //aire
            t_limits.at<int>(k,0)= s.rows; //row min
            t_limits.at<int>(k,1)= s.cols; // col min
            t_limits.at<int>(k,2)= 0; //row max
            t_limits.at<int>(k,3)= 0; //col max

            for(int i=0; i<s.rows; i++)
                for(int j=0; j<s.cols; j++){
                    r.at<uchar>(i,j)=0;
                    if(s.Iregions.at<int>(i,j)==k){
                        r.at<uchar>(i,j)=255;
                        //aire
                        t_area[k] ++;
                        //boite englobante
                        t_limits.at<int>(k,0)= testMin(i, t_limits.at<int>(k,0));
                        t_limits.at<int>(k,1)= testMin(j, t_limits.at<int>(k,1));
                        t_limits.at<int>(k,2)= testMax(i, t_limits.at<int>(k,2));
                        t_limits.at<int>(k,3)= testMax(j, t_limits.at<int>(k,3));
                    }
                }
        }
    }
}
```