MODULE 4

Binarisation et algorithme de traitement

Objectifs de ce module :

- ✓ ***Détection des couleurs.***
- ✓ ***Binarisation***
- ✓ ***Algorithme de traitement***

Table des matières

Sujets ***Page***

Binarisation

Cette étape consiste à segmenter l'image où les pixels obtiendront deux intensités possibles soit le noir ou le blanc.

Binarisation d'une image en noir et blanc

Cette méthode permet de binariser une image, c'est-à-dire, la rendre en deux ton de couleurs, le noir ou le blanc. L'idée est assez simple : pour chaque pixel de l'image, si la valeur de ce dernier est plus grande qu'un certain seuil, ce pixel prendra alors la valeur 0 ou 255.

Ainsi, on peut représenter ce principe par l'algorithme suivant :

```
SI (Valeur_Pixel_Courant > Seuil)

        Valeur_Pixel_Courant = 255

SINON

        Valeur_Pixel_Courant = 0;
```

Avec OpenCV, la fonction qui permet de réaliser cette opération se nomme « **threshold** ».

double **threshold**(InputArray **src**, OutputArray **dst**, double **thresh**, double **maxval**, int **type**)

InputArray src

Image source de type Mat à 1 octet ou de 4 octets.

OutputArray dst

Image destination de type Mat. Doit avoir la même dimension que la source.

double thresh

Valeur de seuil.

double maxval

Valeur maximum à donner au pixel.

int type

Le type de binarisation qui doit être effectuée. Une des trois constantes suivantes:

CV_THRESH_BINARY, CV_THRESH_BINARY_INV ou CV_THRESH_TRUNC

Chaque constante fait appel à une façon différente de traiter le pixel résultant (dst(x,y)).

- THRESH_BINARY

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- THRESH_BINARY_INV

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$$

- THRESH_TRUNC

$$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

- THRESH_TOZERO

$$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- THRESH_TOZERO_INV

$$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$$

Exemple :

```
#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\imgproc\imgproc.hpp>

using namespace cv;

int main()
{
	Mat ImgSource;
	Mat ImgGris;
	Mat ImgResultat;

	ImgSource = imread("shapes.jpg", CV_LOAD_IMAGE_COLOR);

	cvtColor(ImgSource, ImgGris, CV_BGR2GRAY);

	blur(ImgGris, ImgGris, Size(3, 3));  // Applique une matrice de convolution de 3 x 3 pour le blur

	threshold(ImgGris, ImgResultat, 185, 255, CV_THRESH_BINARY_INV);

	namedWindow("Image Source", CV_WINDOW_AUTOSIZE);
	namedWindow("Image Gris", CV_WINDOW_AUTOSIZE);
	namedWindow("Image Resultat", CV_WINDOW_AUTOSIZE);


	imshow("Image Source", ImgSource);
	imshow("Image Gris", ImgGris);
	imshow("Image Resultat", ImgResultat);

	waitKey(0);

	return 0;


}
```

Explication du code :

On procède à convertir une image en ton de gris.

On effectue un traitement de « blurring » pour éliminer le bruit dans l'image originale.

On applique le seuil de filtrage avec l'appel à « threshold ».

Renforcement des contours par la méthode de Sobel ou de Laplace

Matrice de convolution de Sobel

Le filtre de Sobel est un opérateur utilisé en traitement d'image pour la détection de contours. Il s'agit d'un des opérateurs les plus simples qui donne toutefois des résultats corrects.

Pour faire simple, l'opérateur calcule le gradient de l'intensité de chaque pixel. Ceci indique la direction de la plus forte variation du clair au sombre, ainsi que le taux de changement dans cette direction. On connaît alors les points de changement soudain de luminosité, correspondant probablement à des bords, ainsi que l'orientation de ces bords.

En termes mathématiques, le gradient d'une fonction de deux variables (ici l'intensité en fonction des coordonnées de l'image) est un vecteur de dimension 2 dont les coordonnées sont les dérivées selon les directions horizontale et verticale. En chaque point, le gradient pointe dans la direction du plus fort changement d'intensité, et sa longueur représente le taux de variation dans cette direction. Le gradient dans une zone d'intensité constante est donc nul. Au niveau d'un contour, le gradient traverse le contour, des intensités les plus sombres aux intensités les plus claires.

Voici les matrices :

$$\begin{matrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{matrix}$$

Matrice de Sobel pour les contours horizontaux

$$\begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

Matrice de Sobel pour contours verticaux

Exemple d'application de Sobel

```
#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\imgproc\imgproc.hpp>

using namespace cv;

int main()
{
	Mat ImgSource;
	Mat ImgGris;
	Mat ImgResultat;

	ImgSource = imread("sign.jpg", CV_LOAD_IMAGE_COLOR);

	 if (!ImgSource.data)
	{
	     cout << "Erreur dans ouverture du fichier source" << endl;
	     return -1;
	}

	//blur(ImgSource, ImgSource, Size(5, 5));  // Applique une matrice de convolution de 3 x 3 pour le blur

	cvtColor(ImgSource, ImgGris, CV_BGR2GRAY);

	Sobel(ImgGris, ImgResultat, CV_16S, 0, 1, 3); // On cherche les contours horizontaux y = 1
	// On utilise la fonction suivante pour les contours verticaux x = 1
	//Sobel(ImgGris, ImgResultat, CV_16S, 1, 0, 3); // On cherche les contours verticaux x = 1

	convertScaleAbs( ImgResultat, ImgResultat);
	blur(ImgResultat, ImgResultat, Size(3, 3));  // Pas absolument nécessaire mais peut être utile


	namedWindow("Image Source", CV_WINDOW_AUTOSIZE);
	namedWindow("Image Resultat", CV_WINDOW_AUTOSIZE);


	imshow("Image Source", ImgSource);
	imshow("Image Resultat", ImgResultat);

	waitKey(10000);

	return 0;
```

Résultat de l'application de Sobel

Image originale

Image après Sobel horizontal

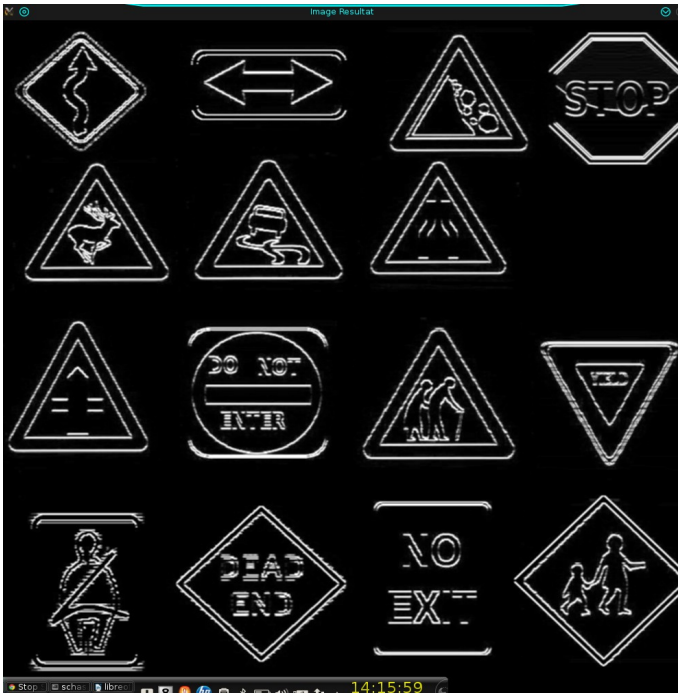
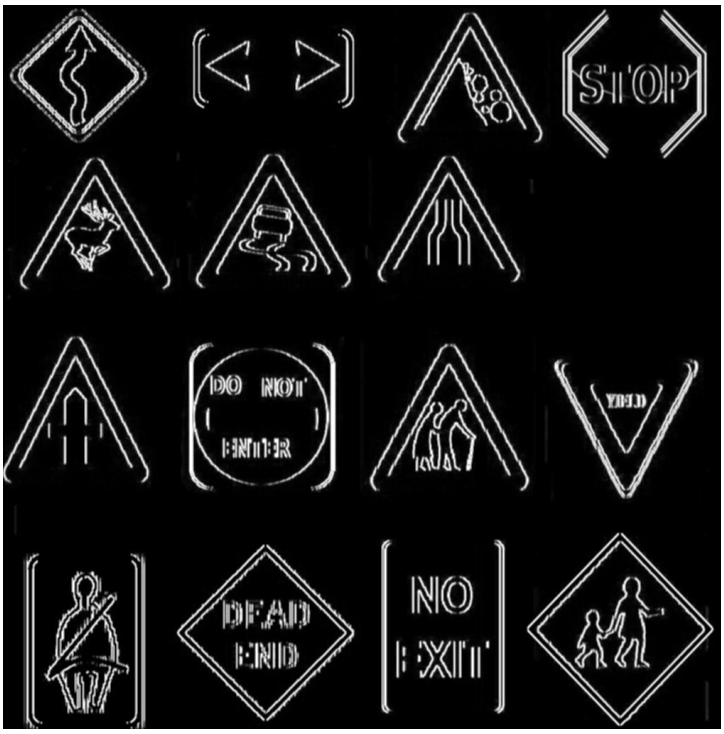


Image après Sobel vertical



Matrice de convolution de Laplace

Le filtre Laplacien est un filtre de convolution particulier utilisé pour mettre en valeur les détails qui ont une variation rapide de luminosité. Il est donc idéal pour rendre visible les contours des objets. D'un point de vue mathématique, le Laplacien est une dérivée d'ordre 2, à deux dimensions et se note $\triangle I(x,y)$ ou $\nabla 2I(x,y)$.

Dans le cas du traitement d'image, l'image de départ f(i,j) n'est pas une fonction continue, mais une fonction discrète à cause de la numérisation effectuée. Mais on peut tout de même obtenir la dérivée seconde (soit le laplacien) avec une bonne approximation grâce aux noyaux de convolution suivants (entre autres).

Matrice de Laplace :

$$\begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix}$$

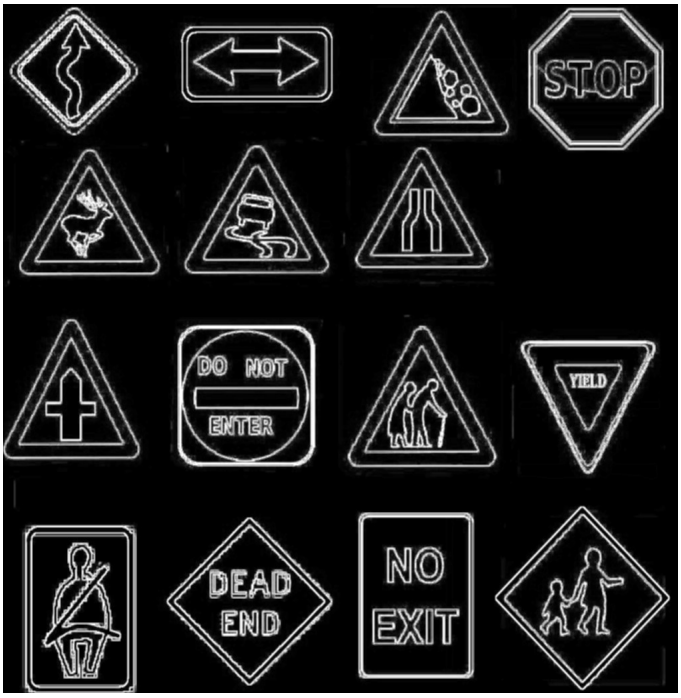
Exemple d'application de Laplace

```
#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\imgproc\imgproc.hpp>

using namespace cv;

int main()
{
	Mat ImgSource;
	Mat ImgGris;
	Mat ImgResultat;

	ImgSource = imread("sign.jpg", CV_LOAD_IMAGE_COLOR);

	if (!ImgSource.data)
	{
		cout << "Erreur dans ouverture du fichier source" << endl;
		return -1;
	}

	//blur(ImgSource, ImgSource, Size(5, 5));  // Applique une matrice de convolution de 3 x 3 pour le blur

	cvtColor(ImgSource, ImgGris, CV_BGR2GRAY);

	Laplacian(ImgGris, ImgResultat, CV_16S, 3);

	convertScaleAbs( ImgResultat, ImgResultat);

	blur(ImgResultat, ImgResultat, Size(3, 3));  // Pas absolument nécessaire mais peut être utile


	namedWindow("Image Source", CV_WINDOW_AUTOSIZE);
	namedWindow("Image Resultat", CV_WINDOW_AUTOSIZE);


	imshow("Image Source", ImgSource);
	imshow("Image Resultat", ImgResultat);

	waitKey(10000);

	return 0;
}
```

Résultat de l'application de Laplace

Image originale

Image après Laplace



Renforcement des contours par la méthode de Canny

- Ouvrir les fichiers d'image
- Convertir l'image originale en ton de gris (si nécessaire)
- Appliquer, au besoin, une binarisation (cvThreshold)
- Appliquer l'algorithme de Canny (cvCanny)

Exemple :

```
int main()
{
	Mat ImgSource;
	Mat ImgGris;
	Mat ImgResultat;

	ImgSource = imread("sign.jpg", CV_LOAD_IMAGE_COLOR);

	if (!ImgSource.data)
	{
		cout << "Erreur dans ouverture du fichier source" << endl;
		return -1;
	}

	//blur(ImgSource, ImgSource, Size(5, 5));  // Applique une matrice de convolution de 3 x 3 pour le blur

	cvtColor(ImgSource, ImgGris, CV_BGR2GRAY);

	blur(ImgGris, ImgGris, Size(3, 3));  // Pas absolument nécessaire mais peut être utile

	threshold(ImgGris, ImgGris, 140, 255, CV_THRESH_BINARY_INV);
	Canny(ImgGris, ImgResultat, 10, 20);

	namedWindow("Image Source", CV_WINDOW_AUTOSIZE);
	namedWindow("Image Resultat", CV_WINDOW_AUTOSIZE);


	imshow("Image Source", ImgSource);
	imshow("Image Resultat", ImgResultat);

	waitKey(10000);

	return 0;
}
```

Détection de la couleur

La couleur est quelquefois utile dans certaines applications. On peut penser qu'une application effectue un contrôle de qualité au niveau de l'intensité de couleur sur un produit. On pourrait aussi utiliser la couleur pour permettre à un robot de s'arrêter lorsqu'il arrive à un panneau de couleur rouge.

Pour ce faire, on convertit l'image d'un format « BGR » en format « HSV ». On est plus intéressé ici à travailler avec les valeurs des teintes et de la saturation qui nous permet d'obtenir la quantité d'une certaine couleur contenu dans un objet.

Transformation de BGR à HSV

On utilise la fonction cvtColor pour transformer le format BGR en un format HSV. La procédure ressemble à celle-ci :

```
Mat ImgSource;

Mat ImgHSV;

ImgSource = imread(« monimage.jpg », CV_LOAD_IMAGE_COLOR);

cvtColor(ImgSource, ImgHSV, CV_BGR2HSV);
```

Binarisation de l'image obtenu en HSV

Cette étape est très similaire à celle que vous aviez programmé pour binariser une image en ton de gris. Vous obteniez ainsi une image noir et blanc où les pixels des objets désirés étaient en blanc et le reste en noir. L'étape que nous allons faire ici pour traiter l'image en HSV est exactement la même à la différence que maintenant nous binarisons les pixels qui se trouvent dans un certain intervalle de couleur.

Malheureusement, à moins d'avoir une étendue de pixel dont la couleur est uniforme, ce qui est rarement le cas surtout si l'image provient d'un dispositif de capture comme une caméra, la binarisation ne donnera pas un bon résultat si on utilise la technique du seuil.

Rappelez-vous que dans cette technique, le seuil est une valeur qui nous permet de départager les pixels selon la relation suivante :

Pixel < Valeur du Seuil alors Pixel = 0

Pixel >= Valeur du Seuil alors Pixel = 255

Pour la couleur, il faut plutôt se donner un intervalle dans lequel le seuil se situera. Ainsi la relation est plutôt la suivante :

H – tolérance <= Hpixel < H + tolérance && S - tolérance <= Spixel < S + tolérance.

Où H : valeur de la teinte (Hue)

S : Valeur de la saturation (Saturation)

La valeur de tolérance nous permet d'obtenir cet intervalle dont nous parlions précédemment.

La valeur pour la teinte et la saturation dépend de la couleur que l'on veut rechercher.

Fonction InRange

Voici la syntaxe de cette fonction :

```
void inRange(InputArray src,
             InputArray lowerb,
             InputArray upperb,
             OutputArray dst);
```