Le traitement d'images

Avec OpenCV

Sommaire

Le traitement d'image avec opencv

1) Présentation

OpenCV (pour Open Computer Vision) est une bibliothèque graphique libre, initialement développée par Intel, spécialisée dans le traitement d'images en temps réel.

2) Les bases de openCV

Toutes les classes et les méthodes d'OpenCV en C++ ont un espace de travail (namespace) nommée cv. Ainsi on doit utiliser la directive de compilation suivante après avoir déclaré les fichiers d'en tête

```
using namespace cv;
```

Une image peut être mémorisée à l'intérieur d'une structure en C du type Cvmat ou Ipimage. Ces structures sont issues des versions 1.x de Opencv. La structure Ipimage est un vieux format d'image original compatible intel IPP.
Depuis la version 2.1 d'OpenCV l'accent a été mis sur les matrices et les opérations sur celles-ci. En effet, la structure de base est la matrice. Une image peut être considérée comme une matrice de pixel. Ainsi, toutes les opérations de bases des matrices sont disponibles La classe Mat est un nouveau format qui est né depuis la version 2.x de openCV. La classe Cv ::mat est la version C++ de Cvmat, son avantage est de ne pas associer des pointeurs et les problèmes inhérents à ces pointeurs.

2.1) les types basiques sous OpenCV

On retrouvera dans beaucoup de programme les types de données suivantes issues des premières version d'opencv (langage C) et des évolutions en langage C++

CvPoint : point (pixel) du plan
typedef struct CvPoint {
int x; // *coordonnée* x
int y; // *coordonnée* y
} CvPoint;
CvPoint2D32f : idem avec membres de type *double*
CvPoint3D32f : 3 membres réels (dont coordonnée *z*)

La classe Point en C++

```
Template <typename _Tp> class CV_EXPORTS Point_
{
public:
   typedef _Tp value_type;

   // différents constructeurs
   Point_();
   Point_(_Tp _x, _Tp _y);
   Point_(const Point_& pt);
.....................
 _Tp x, y; //< les coordonnées du point
};
```

exemples d'utilisation

```
Point a(0.3f, 0.f), b(0.f, 0.4f);
Point pt = (a + b)*10.f;
cout << pt.x << ", " << pt.y << endl;
```

CvSize : taille d'une zone rectangulaire

```
typedef struct CvSize {
int width; // largeur du rectangle
int height; // hauteur du rectangle
} CvSize;
```

La classe Size en c++

```
Template <typename _Tp> class CV_EXPORTS Size_
{
public:
   typedef _Tp value_type;

   //! various constructors
   Size_();
   Size_(_Tp _width, _Tp _height);
   Size_(const Size_& sz);
   .....................
   _Tp width, height; // membres  la largeur et la hauteur
};
```

Cette classe spécifie par la hauteur ou la largeur la dimension d'une image ou d'un rectangle

CvRect : zone rectangulaire (décalage et taille) : int *x, y, width, height*

La classe Rect en c++

```
template <typename _Tp> class CV_EXPORTS Rect_
{
public:
   typedef _Tp value_type;

   //! différents constructeur et opérateurs
   Rect_();
   Rect_(_Tp _x, _Tp _y, _Tp _width, _Tp _height);
   Rect_(const Rect_& r);

   //! cette méthode contrôle si le rectangle contient un point
   bool contains(const Point_<_Tp>& pt) const;

   _Tp x, y, width, height; //< les coordonnées du sommet du rectangle, la largeur et la hauteur
du rectangle
};

Rect R1= Rect(1,2,3,4)
```

CvScalar : conteneur pour 1, 2, 3 ou 4 flottants (= vecteur !)

```
typedef struct CvScalar {
double val[4]; // exemple d'utilisation : RGBa
} CvScalar;
```

La classe Scalar en c++

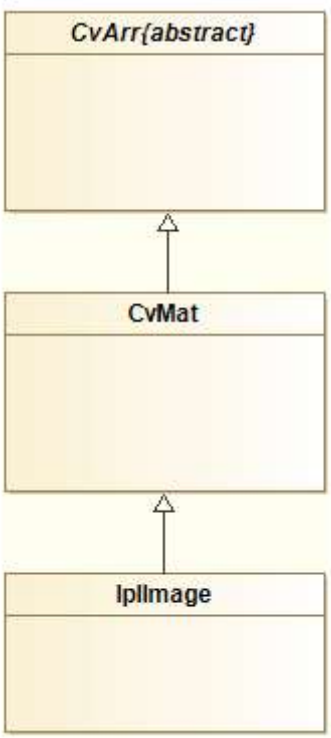
```
template<typename _Tp> class CV_EXPORTS Scalar_ : public Vec<_Tp, 4> // hérite d'un
vecteur à 4 dimensions
{
public:
   //! différents constructeurs
   Scalar_();
   Scalar_(_Tp v0, _Tp v1, _Tp v2=0, _Tp v3=0);// construction d'un scalaire à 4 dimensions
```

```
    Scalar_(const CvScalar& s);
    Scalar_(_Tp v0);// construction d'un scalaire à une dimension
        autres constructeurs méthodes et opérateurs


    // retourne vrai si v1 == v2 == v3 == 0
    bool isReal() const;
};
```

Le type scalaire est largement utilisé pour passer les valeurs de pixel

2.2) Le type CvMat

La structure CVMat est un vieux format issu du langage C et de la première version de opencv 1.x. Ce format a donné naissance au format IplImage en utilisant le principe de l'héritage en C++. Il est intéressant de comprendre la classe de base avant de passer à la classe dérivée IplImage. Une troisième classe appelée CvArr (classe abstraite) possède comme classe dérivée la classe CvMat.



Le type matrice CvMat *(simplifié)*

```
typedef struct CvMat {
```

```
        int type; // type des éléments
        int step; // taille d'une ligne en octets
        union {
        uchar* ptr; short* s; int* i; float* fl; double* db;
        } data; // pointeur vers données (le membre utilisé est déterminé par le type)
        union { int rows; int height; }; // nombre de lignes (ou hauteur)
        union { int cols; int width; }; // nombre de colonnes (ou largeur)
} CvMat;
```

Types des éléments : constante CV_<profondeurEnBits>{S|U|F}C<nbCanaux>

Exemple :

CV_32FC1 : flottants sur 32 bits (1 seul canal)
CV_8UC3 : triplets d'entiers non signés sur 8 bits (3 canaux)

Principales fonctions

CvMat* **cvCreateMat**(int *nbLignes*, int *nbCols*, int *type*)// créé une matrice
void **releaseMat**(CvMat** *mat*) // libère l'espace mémoire pour la matrice
CvScalar **cvGet2D**(const cvArr* *tab*, int *iLig*, int *iCol*) // lit une donnée dans la matrice
CvScalar **cvSet2D**(const cvArr* *tab*, int *iLig*, int *iCol*, cvScalar *valeur*)// écrit une donnée dans la matrice

Exemple d'utilisation

```
CvMat* M = cvCreateMat(2, 2, CV_8UC1); // 1 entier sur 8 bits
cvSet2D(M, 0, 0, cvScalar(1));
cvSet2D(M, 0, 1, cvScalar(2));
cvSet2D(M, 1, 0, cvScalar(3));
cvSet2D(M, 1, 1, cvScalar(4));
unsigned short i = cvGet2D(M, 0, 0).val[0]; //1
unsigned short j = cvGet2D(M, 0, 1).val[0]; //2
cvReleaseMat(&M);
```

$$M = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

2.3) La structure IPlImage

Iplimage est la structure de base pour encoder les images. Ces images peuvent être en échelle de gris , en couleur, en 4 canaux (RGB + alpha)), et chaque canal peut contenir plusieurs types d'entier et de flottants. chaque canal a une certaine « profondeur »

typedef struct _IplImage {
int nSize; // correspond à sizeof(IplImage)
int **nChannels**; // Nombre de canaux (1,2,3 ou 4)
int **depth**; // profondeur de pixel en bits, parmi : IPL_DEPTH_8U, IPL_DEPTH_8S, IPL_DEPTH_16U, IPL_DEPTH_16S, IPL_DEPTH_32S, IPL_DEPTH_32F, IPL_DEPTH_64F
int dataOrder; // =IPL_DATA_ORDER_PIXEL ici (canaux couleur entrelacés)
int origin; // origine : IPL_ORIGIN_TL=H à G, IPL_ORIGIN_BL=B à G
int width; // largeur d'image en pixels (nombre de colonnes)
int height; // hauteur d'image en pixels (nombre de lignes)
struct _IplROI ***roi**; // région d'intérêt (zone traitée si ¹ NULL)
int imageSize; // taille de l'image en octets
char* imageData; // pointeur vers les données
int **widthStep**; // taille d'une ligne en octets (=*step* dans CvMat)
...
} IplImage;

La profondeur

La profondeur d'une image est le type de données qui représente un pixel ou un scalaire. IplImage étaient toujours des entiers positifs (U = Unsigned) codés sur 8 bits : c'est-à-dire des nombres compris entre 0 et 255.
Elle est contenue dans la variable *depth*, dans notre IplImage.
Les valeurs que l'on peut donner sont sous forme de constantes, en voici la liste exhautive :

- ✓ IPL_DEPTH_8U : nous l'utilisons depuis le début du tutoriel, elle signifie « entiers positifs sur 8 bits » ;
- ✓ IPL_DEPTH_8S : entiers positifs ou négatifs sur 8 bits ;
- ✓ IPL_DEPTH_16U : entiers positifs sur 16 bits ;
- ✓ IPL_DEPTH_16S : entiers positifs ou négatifs sur 16 bits ;
- ✓ IPL_DEPTH_32S : entiers positifs ou négatifs sur 32 bits ;
- ✓ IPL_DEPTH_32F : float ;
- ✓ IPL_DEPTH_64F : double.

2.4) La classe Mat

La classe Mat et un conteneur d'image. Ce dernier format est apparu avec la version 2 de opencv
Une image peut être enregistrée dans un objet du type Mat. Mat est donc un conteneur d'image

Déclaration de la classe

```
class CV_EXPORTS Mat {
public:    // ... beaucoup de méthodes ...    ...
 //! La dimension du tableau, >= 2
 int dims;
//! Le nombre de rangées et de colonnes quand la dimension est > à 2
int rows, cols;
//! Pointeur sur une donnée
uchar* data;
};
```

Les constructeurs

La classe Mat propose un nombre important de constructeur, ceux qui vous seront le plus utile :
Mat::Mat(int rows, int cols, int type)
Mat::Mat(Size size, int type)

Paramètres :

- rows – nombre de rangées dans un tableau à deux dimensions.
- cols – nombre de colonnes dans un tableau à deux dimensions..
- size – 2D array size: Size(cols, rows) . dans le constructeur Size(), le nombre de rangées et de colonnes est inversé.
- type – type de données dans le tableau. utilise CV_8UC1, ..., CV_64FC4 pour créer des matrices de 1-4 canaux.

La matrice

Avec Mat qui est très proche de l'objet matrice
Une valeur pour un élément d'une matrice a le type suivant :

CV_8U (8 bit unsigned integer)

CV_8S (8 bit signed integer)

CV_16U (16 bit unsigned integer)

CV_16S (16 bit signed integer)

CV_32S (32 bit signed integer)

CV_32F (32 bit floating point number)

CV_64F (64 bit float floating point number)

Matrice un canal

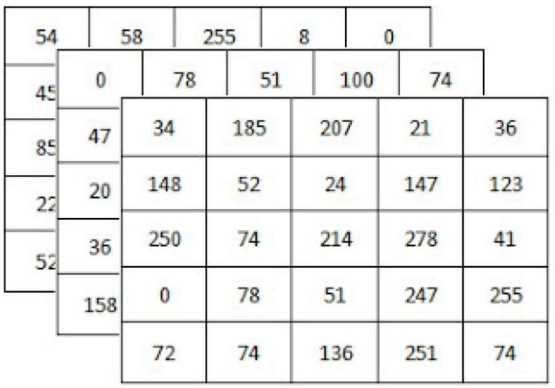
54	58	255	8	0
45	24	25	214	23
85	124	85	23	55
22	78	25	21	0
52	52	36	127	47

Single Channel Array

Exemple de déclaration pour une matrice :

Mat img1(3, 5, CV_32F); Simple matrice 3x5 avec un format flottant de 32bits

Le réseau de matrices



3 Channel Arrays

Exemple de déclaration pour un réseau de matrice :

Mat img2(23, 53, CV_64FC(5)); //réseau de matrice avec des flottants de format 64 bits

Une valeur pour un élément d'un réseau de matrice a le type suivant :

CV_8UC1 (réseau simple matrice avec 8 bit unsigned integers)
CV_8UC2 (réseau de 2 matrices avec 8 bit unsigned integers)
CV_8UC3 (réseau de 3 matrices avec 8 bit unsigned integers)
CV_8UC4 (réseau de 4 matrices avec 8 bit unsigned integers)
CV_8UC(n) (réseau de n matrices avec 8 bit unsigned integers (n peut être de 1 à 512))

exemple

```
#include "opencv2/core/core.hpp"
using namespace cv;
int main()
{
Mat frame = cvQueryFrame( capture ); //
imshow( "Video", frame );
....
```

Accès aux pixels

L'accès aux pixels d'une image sous OpenCV peut se faire de nombreuses façons différentes. Nous nous concentrons ici sur les fonctions d'accès haut niveau qui sont plus agréables à manipuler que les fonctions bas-niveau qui demandent à faire de l'arithmétique de pointeurs et donc de connaitre les détails d'implémentation des structures de données.

Les images à niveau de gris ne possède qu'un canal ce qui facilite leur manipulation. Le type de données stocké sera généralement des uchar (8 bit non signé) ou des float (32 bit)

La méthode générique at :

```
template<typename _Tp> _Tp& Mat::at(int i, int j)
template<typename _Tp> _Tp& Mat::at(Point pt)
```

Paramètres :

- i – l'index des rangées (démarre à 0)
- j – L'index de colonnes (démarre à 0)
- pt – position de l'élément spécifié Point(j,i)

Exemple :

```
Mat img(100, 100, CV_64FC1);
for(int i = 0; i < img.rows; i++)
for(int j = 0; j < img.cols; j++)
img.at<double>(i,j) = 1./(i+j+1);   // écriture d'un pixel du type double 64 bits
Mat img2(100,100,CV_8UC1);
for(int i = 0; i < img2.rows; i++)
 for(int j = 0; j < img2.cols; j++)
 img.at<uchar>(i,j) = saturate_cast<uchar>(img.at<double>(i,j)*1000.0);
```

exemples d'utilisation

```
Mat F = A.clone(); // clonage d'un objet du type Mat
Mat G;
A.copyTo(G); // copie d'un objet du type Mat
```

```
IplImage* img = cvLoadImage("greatwave.png", 1);
Mat mtx(img); // conversion  IplImage* -> Mat
```

Création d'une matrice la fonction Create():

```
M.create(4,4, CV_8UC(2));
cout << "M = "<< endl << " " << M << endl << endl;
```

<u>Lecture d'une image avec la classe Mat</u>

La fonction imread

Mat imread(const string& filename, int flags=1)

Paramètres :

- filename – nom du fichier à lire.
- flags – spécifie le type de couleur chargé:
  - ✓ >0 l'image chargée est forcée à une couleur 3 canaux
  - ✓ =0 l'image chargée est forcée à une échelle de gris

<u>Exemples d'utilisation:</u>

```
#include <cv.h>
#include <cvaux.h>
#include <highgui.h>
using namespace cv;
int main(int argc, char* argv[ ]){
Mat image = imread(argv[1]);
  namedWindow("Sample Window");
imshow("Sample Window",image);
waitKey(0);
return 0; }
```

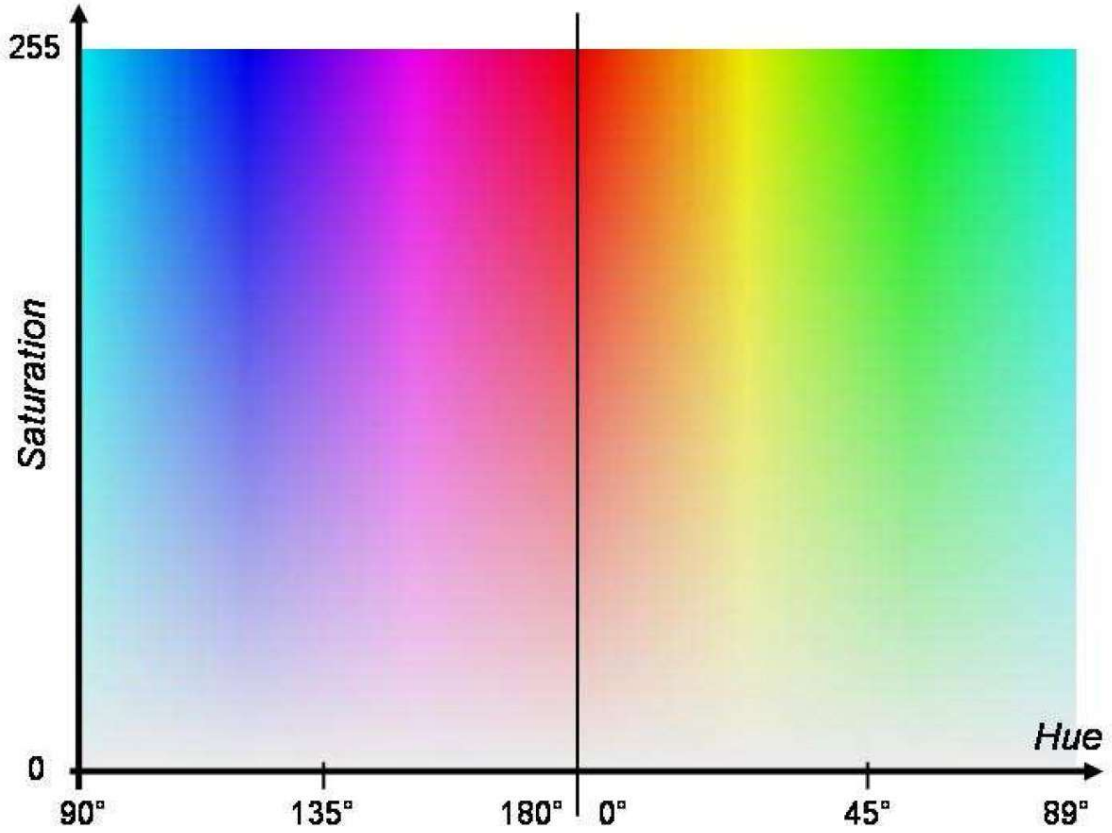
```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <cv.h>
 #include <highgui.h>
using namespace std;
using namespace cv;
int main(int argc, char *argv[]) {     // load an image
Mat img=imread("camera.png",0);
 if(img.empty()){
 printf("Could not load image file\n");
exit(0);   }
  // get the image data
int height    = img.rows;
int width     = img.cols;
printf("Processing a %dx%d image\n",height,width);
  // invert the image
 for(int i=0;i<height;i++)
for(int j=0;j<width;j++)
 img.at<uchar>(i,j)=255-img.at<uchar>(i,j);
  // create a window
namedWindow("mainWin", CV_WINDOW_AUTOSIZE);
  // show the image
imshow("mainWin", img );
  // wait for a key
waitKey(0);
  return 0; }
```

2.5) Espace de couleur HSV

Bien souvent les couleurs sont codés en Rouge vert bleue on peut représenter les couleurs par le format HSV. L'espace HSV (Hue, Saturation, Value) ou TSV (Teinte Saturation Valeur) est un espace colorimétrique, défini en fonction de ses trois composantes :

- ✓ Teinte (H) : le type de couleur. La valeur varie entre 0 et 360.
- ✓ Saturation (S) : l'« intensité » de la couleur. La valeur varie entre 0 et 100 %. Plus la saturation d'une couleur est faible, plus l'image sera « grisée » et plus elle apparaîtra fade, il est courant de définir la « désaturation » comme l'inverse de la saturation.
- ✓ Valeur (V) : la « brillance » de la couleur, elle varie entre 0 et 100%.

Dans OpenCV, la valeur H est normalisée en 0–180; les valeurs S et V sont normalisées en 0–255



L'espace de couleur HSV sépare les informations de couleurs en teinte, saturation et valeur.

3) Lecture, écriture, affichage d'une image

3.1) Lecture d'une image sous Mat

La fonction imread

Mat imread(const string& filename, int flags=1)

Paramètres :

- filename – nom du fichier à lire.
- flags – spécifie le type de couleur chargé:
  - ✓ >0 l'image chargée est forcée à une couleur 3 canaux
  - ✓ =0 l'image chargée est forcée à une échelle de gris

Exemples d'utilisation:

```
#include <cv.h>
#include <cvaux.h>
#include <highgui.h>
using namespace cv;
int main(int argc, char* argv[ ]){
Mat image = imread(argv[1]);
  namedWindow("Sample Window");
imshow("Sample Window",image);
waitKey(0);
return 0; }
```

3.2) Création d'image avec IplImage

Déclaration et création :

```
#include <cv.h>
IplImage *im;
im = cvCreateImage(cvSize(512,256), IPL_DEPTH_8U, 3);
/* Image 512 x 256 de unsigned char avec 3 canaux pour les couleurs R V B*/
```

Remplissage : (codage par défaut = BGR)

```
pteur = (unsigned char*)im->imageData;
fin = im->width*im->height;
for(i=0; i<fin; i++)
{
*(pteur++) = 255; /* Bleu */
```

```
*(pteur++) = 0; /* Vert */
*(pteur++) = 0; /* Rouge */
}
```

Libération de l'espace mémoire

```
cvReleaseImage(&im);
```

```
IplImage* image=cvCreateImage(cvSize(130, 150), IPL_DEPTH_8U, 1);
```

Détail des paramètres

- ✓ **imageSize** : taille de l'image.

C'est en fait une structure CvSize ("size" = taille), que l'on crée avec la fonction cvSize(int, int).
Pour une image de 130 pixels de large, et 150 de haut, on lui donne cvSize(130,150).

- ✓ **depth** : la "profondeur" de l'image.

on utilisera toujours la constante IPL_DEPTH_8U, qui signifie que nos pixels sont des entiers naturels codés sur un octet.

- ✓ **channels** : le nombre de canaux de nos images.

On y reviendra quand on fera de la couleur. Retenez simplement qu'une image en niveaux de gris n'a qu'un seul canal.

3.3) Lire et afficher une image avec IPlImage

```
#include <highgui.h>
IplImage *im;
im = cvLoadImage("mon_image.jpg", 1); /* (1) */
/* 1 => 3 canaux (0 => 1 seul, -1 => automatique) */
im = cvLoadImage("mon_image.jpg", 0); /* (2) */

//Afficher une image
cvNamedWindow("Ma fenetre", CV_WINDOW_AUTOSIZE);
cvShowImage("Ma fenetre", im);
cvWaitKey(0); /* Attendre qu'une touche soit pressée */
```

Pour charger une image, on utilise tout simplement la fonction cvLoadImage dont voici la signature :

```
IplImage* cvLoadImage(const char *filename, int flags = CV_LOAD_IMAGE_COLOR);
```

En gros, on lui passe en paramètre le chemin de notre fichier image (la chaîne de caractères) et éventuellement des constantes parmi les suivantes :

- CV_LOAD_IMAGE_COLOR : L'image est chargée en couleur
- CV_LOAD_IMAGE_GRAYSCALE : l'image est chargée en niveaux de gris.

3 fonctions à connaître pour créer une fenètre :

void cvNamedWindow(const char *titre_fenetre, int flag);
void cvShowImage(const char *titre_fenetre, IplImage *image);
void cvDestroyWindow(constchar *titre_fenetre);

Comme leurs noms l'indiquent :

- la première sert à **créer une fenêtre**. Dans le champ "flag" on mettra toujours la constante CV_WINDOW_AUTOSIZE afin que la fenêtre s'adapte automatiquement au contenu ;
- la seconde sert à **afficher une image dans une fenêtre** ;
- la dernière sert à **détruire une fenêtre**.

La fonction char cvWaitKey(int delay);

Cette fonction :

- bloque le programme pendant un certain temps (paramètre delay = nombre de millisecondes) ;
- attend pendant ce temps que l'utilisateur ait appuyé sur une touche du clavier ;
- retourne le caractère associé à la touche du clavier.

La function cvNamedWindow
Creates window
int cvNamedWindow(const char* name, int flags);

name:
Name of the window which is used as window identifier and appears in the window caption.

Flags:
Flags of the window. Currently the only supported flag is CV_WINDOW_AUTOSIZE. If it is set, window size is automatically adjusted to fit the displayed while user can not change the window size manually.
The function cvNamedWindow creates a window which can be used as a placeholder for images and trackbars. Created windows are referred by their names.

La fonction cvDestroyWindow
Destroys a window
void cvDestroyWindow(const char* name);

name :
Name of the window to be destroyed.
The function cvDestroyWindow destroys the window with a given name.

La fonction cvResizeWindow
Sets window size
void cvResizeWindow(const char* name, int width, int height);

name
Name of the window to be resized.

width
New width

height
New height
The function cvResizeWindow changes size of the window.

La fonction cvMoveWindow
Sets window position
void cvMoveWindow(const char* name, int x, int y);

name:
Name of the window to be resized.
x
New x coordinate of top-left corner
y
New y coordinate of top-left corner
The function cvMoveWindow changes position of the window.

La fonction cvShowImage
Shows the image in the specified window
void cvShowImage(const char* name, const CvArr* image);

name
Name of the window.

image
Image to be shown.
The function cvShowImage shows the image in the specified window. If the window was created with CV_WINDOW_AUTOSIZE flag then the image is shown with its original size, otherwise the image is scaled to fit the window.

La fonction cvLoadImage

Loads an image from file

```
#define CV_LOAD_IMAGE_COLOR 1
#define CV_LOAD_IMAGE_GRAYSCALE 0
#define CV_LOAD_IMAGE_UNCHANGED -1
IplImage* cvLoadImage( const char* filename, int iscolor=CV_LOAD_IMAGE_COLOR );
```

Filename:
Name of file to be loaded.

iscolor
Specifies colorness of the loaded image:
if >0, the loaded image is forced to be color 3-channel image;
if 0, the loaded image is forced to be grayscale;
if <0, the loaded image will be loaded as is (with number of channels depends on the file).
The function cvLoadImage loads an image from the specified file and returns the pointer to the loaded image. Currently the following file formats are supported:
* Windows bitmaps - BMP, DIB
* JPEG files - JPEG, JPG, JPE

3.4) *Lire et afficher une image avec Mat*

La fonction imread

Mat **imread**(const string& filename , int flags)

Paramètres:

filename – nom du fichier à charger.
flags –Flags spécifie le type de couleur de l'image à charger:
CV_LOAD_IMAGE_ANYDEPTH –
CV_LOAD_IMAGE_COLOR(>0) – convertit l'image en couleur
CV_LOAD_IMAGE_GRAYSCALE (0)- convertit l'image en échelle de gris
CV_LOAD_IMAGE_UNCHANGED (<0) charge l'image telle qu'elle

La fonction imwrite

bool **imwrite**(const string& filename, InputArray image, const vector<int>&params=vector<int>())

paramètres:

filename – Nom du fichier
image – image à sauvegarder
params – Format
pour une image JPEG, on peut avoir une qualité (CV_IMWRITE_JPEG_QUALITY) de 0 à 100. La valeur par défaut est 95.
pour une image PNG, c'est le niveau de compression (CV_IMWRITE_PNG_COMPRESSION) de 0 to 9. La valeur par défaut est 3.
Pour les formats PPM, PGM, or PBM, c'est simplement un indicateur de format (CV_IMWRITE_PXM_BINARY), 0 ou 1. La valeur par défaut est 1.

La fonction imshow

void **imshow**(const string& winname, InputArray mat image)

Paramètres:

winname – nom de la fenètre.

image Image à visualiser.

Exemples:

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>

using namespace cv;
using namespace std;

int main( )
{

	Mat image;

	// LOAD image
	image = imread("image1.jpg", CV_LOAD_IMAGE_COLOR);   // Read the file "image.jpg".
		//This file "image.jpg" should be in the project folder.
		//Else provide full address : "D:/images/image.jpg"

	if(! image.data )  // Check for invalid input
	{
		cout <<  "Could not open or find the image" << std::endl ;
		return -1;
	}

	//DISPLAY image
	namedWindow( "window", CV_WINDOW_AUTOSIZE ); // Create a window for display.
	imshow( "window", image ); // Show our image inside it.

	//SAVE image
	imwrite("result.jpg",image);// it will store the image in name "result.jpg"

	waitKey(0);                    // Wait for a keystroke in the window
	return 0;
}
```

Lire et afficher une image avec la classe Mat

Mat est une structure de données pour stocker des images. Imread() est une fonction déclarée dans **"opencv2/highgui/highgui.hpp"** elle charge une image à partir d'une source video et la stocke dans la structure de données Mat

```
Mat img = imread(const string& filename, int flags=CV_LOAD_IMAGE_COLOR)
```

Paramètres:

filename – Position du fichier

flags – Il y a quatre entrées possibles

CV_LOAD_IMAGE_UNCHANGED – profondeur d'image=8 bits par pixel pour chaque canal, numéro des canaux inchangés
¡ CV_LOAD_IMAGE_GRAYSCALE profondeur d'image=8 bits par pixel, nombre de canaux =1
¡ CV_LOAD_IMAGE_COLOR – nombre de canaux=3
¡ CV_LOAD_IMAGE_ANYDEPTH – profondeur d'images inchangée ,
¡ CV_LOAD_IMAGE_ANYCOLOR – nombre de canaux inchangés

On peut combiner les paramètres :

CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR
CV_LOAD_IMAGE_COLOR | CV_LOAD_IMAGE_ANYDEPTH

Afin de contrôler si l'image a été chargée avec succès on utilise la méthode empty(). Cette méthode retourne vraie si l'image n'est pas chargée ou dans le cas contraire faux

```
bool Mat::empty()
```

Cette fonction crée une fenêtre :
.

```
void namedWindow(const string& winname, int flags = WINDOW_AUTOSIZE);
```

Paramètres:

winname – le titre de la fenètre

flags- détermine la dimension de la fenêtre :

WINDOW_AUTOSIZE l'utilisateur ne peut pas redimensionner l'image.L'Image sera affichée dans sa dimension d'origine
CV_WINDOW_NORMAL – l'image sera redimensionnée si tu redimensionnes la fenêtre

Cette fonction montre l'image qui a été stocké dans l'objet Mat à l'intérieur d'une fenêtre spécifiée par winname. Si la fenêtre est créée avec

```
void imshow(const string& winname, InputArray mat);
```

Paramètres -

winname – nom de la fenêtre.

mat – L'image à afficher

Cette fonction ferme la fenêtre ouverte baptisée winname et désalloue la mémoire

void **destroyWindow**(const string& winname)

exemples

```
#include <iostream>
using namespace cv;
using namespace std;
int main( int argc, const char** argv )
{
Mat img = imread("MyPic.JPG", CV_LOAD_IMAGE_UNCHANGED); //read the image data in the file "MyPic.JPG"
and store it
in 'img'
if (img.empty()) //check whether the image is loaded or not
{
cout << "Error : Image cannot be loaded..!!" << endl;
//system("pause"); //wait for a key press
return -1;
}
namedWindow("MyWindow", CV_WINDOW_AUTOSIZE); //create a window with the name "MyWindow"
imshow("MyWindow", img); //display the image which is stored in the 'img' in the "MyWindow" window
waitKey(0); //wait infinite time for a keypress
destroyWindow("MyWindow"); //destroy the window with the name, "MyWindow"
return 0;
}
```

```
#include "opencv2/highgui/highgui.hpp"
#include <iostream>
using namespace cv;
using namespace std;
int main( int argc, const char** argv )
{
Mat img(500, 1000, CV_8UC3, Scalar(0,0, 100)); //create an image ( 3 channels, 8 bit image depth, 500 high, 1000
wide, (0, 0, 100) assigned for Blue, Green and Red plane respectively. )
if (img.empty()) //check whether the image is loaded or not
{
cout << "Error : Image cannot be loaded..!!" << endl;
//system("pause"); //wait for a key press
return -1;
}
namedWindow("MyWindow", CV_WINDOW_AUTOSIZE); //create a window with the name "MyWindow"
imshow("MyWindow", img); //display the image which is stored in the 'img' in the "MyWindow" window
waitKey(0); //wait infinite time for a keypress
destroyWindow("MyWindow"); //destroy the window with the name, "MyWindow"
return 0;}
```

3.5) Modification d'une image avec IPLImage

Lire et écrire un pixel

```
#include <opencv/cv.h>
#include <opencv/highgui.h>


int main()
{
IplImage *img = cvCreateImage(cvSize(800, 200), IPL_DEPTH_8U, 3); //3 canaux
cvNamedWindow("img", CV_WINDOW_AUTOSIZE);
CvScalar scalaire;
scalaire.val[0] = 255; //val[0] = Bleu
scalaire.val[1] = scalaire.val[2] = 0; //val[1] = Vert, val[2] = Rouge
for(int x=0; x < img->width; x++)
{
for(int y=0; y < img->height; y++)
{
cvSet2D(img, y, x, scalaire);
}
}
cvShowImage("img", img);
cvWaitKey(0);
cvReleaseImage(&img);
return 0;
}
```

Cloner une image

Pour cloner une image, on utilisera la fonction :

IplImage* **cvCloneImage**(IplImage* src);

Cette fonction copie simplement une image dans une autre.
Exemple :
Code : C++

```
IplImage *image=cvLoadImage("monimage.jpg");
IplImage *copie=cvCloneImage(image);
```

Détail des paramètres
src : l'image que l'on clone.

Les fonctions de modification de pixels

Un pixel d'une image est représenté par la structure CvScalar (scalaire).
Cela signifie clairement que pour OpenCV, **une image est une matrice**. D'ailleurs, la structure IplImage est en fait tellement similaire à la structure CvMat (matrice), que l'on peut les utiliser indifféremment dans la plupart des fonctions d'OpenCV (elles sont alors désignées par le type CvArr, comme « *Array* » - « Tableau » en anglais).
Mais laissons ça de côté pour le moment, et intéressons-nous au CvScalar.
Un scalaire pour OpenCV, ce n'est pas **un** nombre, mais **un tableau** de nombres (dont les champs peuvent être nuls si on n'a pas besoin d'eux).

Donc, en fait, pour accéder à la valeur d'un pixel en niveau de gris dans une image, on doit :

- accéder au scalaire dans l'image ;
- accéder à la valeur dans le scalaire.

Cela se fait en réalité très simplement grâce à la fonction :

```
CvScalar cvGet2D([IplImage*] image, int y, int x);
```

Remarquez que l'on doit donner "y" avant "x"... comme pour accéder à un scalaire dans une matrice ! (C'est en fait la même fonction
pour les images et les matrices dans OpenCV, c'est pour ça que j'ai mis le IplImage* entre crochets.)

```
//J'accède ici au pixel de coordonnées (30,20) dans mon image. ;)
CvScalar pixel = cvGet2D(mon_image, 20, 30);
//Dans "valeur" je mets la valeur du niveau de gris du pixel.
int valeur = pixel.val[0];
```

Maintenant, pour modifier un pixel dans mon image, je peux utiliser la fonction :
void **cvSet2D**([IplImage*] image, int y, int x, CvScalar scalaire);

Cette fonction modifie la valeur du pixel de coordonnées (x, y) dans l'image, en la remplaçant par le scalaire donné en paramètre.

3.6) Rajouter un curseur

La fonction cvCreateTrackbar

La fonction créé le curseur et l'attache à une fenêtre spécifié

```
int cvCreateTrackbar( const char* trackbar_name, const char* window_name,
int* value, int count, CvTrackbarCallback on_change );
```

Paramètres
trackbar_name : nom du curseur créé
window_name **:** nom de la fenètre qui sera utilize comme parent pour créer le curseur.
Value **:** pointeur sur une variable entière don't la valeur est l'image de la position du curseur
count **:** Position maximal du curseur
on_change **:** pointeur sur la fonction qui sera appelée toutes les fois que le curseur changera de position
Cette fonction aura comme prototype
void Foo(int); peut être NULL si l'appel n'est pas nécessaire.

exemple

```
#include <opencv/cv.h>
#include <opencv2/highgui/highgui.hpp>

using namespace cv;

/// Global Variables
const int alpha_slider_max = 100;
int alpha_slider;
Mat src1,src2,dst;

void on_trackbar( int, void* )
{
 double alpha;
 double beta;

 alpha = (double) alpha_slider/alpha_slider_max ;
 beta = ( 1.0 - alpha );

 addWeighted( src1, alpha, src2, beta, 0.0, dst);

 imshow( "Linear Blend", dst );
}
```

```
int main( )
{
 // Read image ( same size, same type )
 src1 = imread("image2.jpg",0);
 src2 = imread("image1.jpg",0);

 // Create Windows
 namedWindow("Linear Blend", 1);

 // Initialize values
 alpha_slider = 0;
 createTrackbar( "Alpha ", "Linear Blend", &alpha_slider,  alpha_slider_max, on_trackbar );

 // Initialize trackbar
 on_trackbar( 0, 0 );

 waitKey(0);
 return 0;
}
```

Rajouter un curseur issu de la classe Mat

Cette fonction créée un trackbar et attache ce trackbar dans une fenêtre spécifiée

int **createTrackbar**(const string& trackbarname, const string& winname, int* value, int count, TrackbarCallback onChange = 0, void* userdata = 0)

Paramètres:

trackbarname – le nom de la trackbar
winname – Le nom de la fenêtre dans laquelle la fenêtre est attachée
value – Cet entier maintient la valeur associée avec la position du curseur
count – la valeur maximum du curseur. La valeur minimum est toujours 0.
onChange – Cette fonction sera appellee toutes les fois que la position du curseur sera changée
userdata – ce pointeur sera passé comme le secon paramètre de la fonction vue précédemment

```
#include "opencv2/highgui/highgui.hpp"
#include <iostream>
using namespace std;
using namespace cv;
int main( int argc, char** argv )
{
// Read original image
Mat src = imread("MyPic.JPG");
//if fail to read the image
if (!src.data)
{
cout << "Error loading the image" << endl;
return -1;
}
// Create a window
namedWindow("My Window", 1);
//Create trackbar to change brightness
int iSliderValue1 = 50;
createTrackbar("Brightness", "My Window", &iSliderValue1, 100);
//Create trackbar to change contrast
int iSliderValue2 = 50;
createTrackbar("Contrast", "My Window", &iSliderValue2, 100);
while (true)
{
Mat dst;
int iBrightness = iSliderValue1 - 50;
double dContrast = iSliderValue2 / 50.0;
src.convertTo(dst, -1, dContrast, iBrightness);
//show the brightness and contrast adjusted image
imshow("My Window", dst);
// Wait until user press some key for 50ms
int iKey = waitKey(50);
//if user press 'ESC' key
{
break;
}
}
return 0;
}
```

```
#include "opencv2/highgui/highgui.hpp"
#include <iostream>
using namespace std;
using namespace cv;
Mat src;
void MyCallbackForBrightness(int iValueForBrightness, void *userData)
{
Mat dst;
int iValueForContrast = *( static_cast<int*>(userData) );
//Calculating brightness and contrast value
int iBrightness = iValueForBrightness - 50;
double dContrast = iValueForContrast / 50.0;
//Calculated contrast and brightness value
cout << "MyCallbackForBrightness : Contrast=" << dContrast << ", Brightness=" << iBrightness <<
endl;
//adjust the brightness and contrast
src.convertTo(dst, -1, dContrast, iBrightness);
//show the brightness and contrast adjusted image
imshow("My Window", dst);
}
void MyCallbackForContrast(int iValueForContrast, void *userData)
{
Mat dst;
int iValueForBrightness = *( static_cast<int*>(userData) );
//Calculating brightness and contrast value
int iBrightness = iValueForBrightness - 50;
double dContrast = iValueForContrast / 50.0;
//Calculated contrast and brightness value
cout << "MyCallbackForContrast : Contrast=" << dContrast << ", Brightness=" << iBrightness <<
endl;
//adjust the brightness and contrast
src.convertTo(dst, -1, dContrast, iBrightness);
//show the brightness and contrast adjusted image
imshow("My Window", dst);
}
int main(int argc, char** argv)
{
// Read original image
src = imread("MyPic.JPG");
//if fail to read the image
if (src.data == false)
{
```

```
cout << "Error loading the image" << endl;
return -1;
}
// Create a window
namedWindow("My Window", 1);
int iValueForBrightness = 50;
int iValueForContrast = 50;
//Create track bar to change brightness
createTrackbar("Brightness", "My Window", &iValueForBrightness, 100,
MyCallbackForBrightness, &iValueForContrast);
//Create track bar to change contrast
createTrackbar("Contrast", "My Window", &iValueForContrast, 100, MyCallbackForContrast,
&iValueForBrightness);
imshow("My Window", src);
// Wait until user press some key
waitKey(0);
return 0;
}
```

3.7) Dessiner et écrire du texte sur une image

Tracer une ligne

Tracer une ligne, c'est simplement relier deux points entre eux.
Voici donc la fonction qui nous permet de le faire avec OpenCV :

```
void cvLine( CvArr* img, CvPoint pt1, CvPoint pt2, CvScalar color, int thickness=1, int
line_type=8, int shift=0 );
```

Détail des paramètres

- **img** : l'image (ou la matrice, en fait), sur laquelle on trace la ligne.
- **pt1, pt2** : les deux points à relier par une ligne droite.
- **color** : la couleur (ou le niveau de gris) de la ligne.
- **thickness** : l'épaisseur du "crayon", en pixels. Par défaut, 1 pixel.
- **line_type** : la *connexité* de la ligne : 4 ou 8, ou CV_AA pour "Anti-Aliased" (anti-crénelée). Nous la laisserons par défaut.
- **shift** : Nombre de chiffres après la virgule dans le calcul des points appartenant à la ligne, dans le cas où elle serait "antialiasée".

// Je ne remplis que les 5 premiers champs, je laisse la connexité par défaut.

cvLine(image, cvPoint(50,60), cvPoint(110,120), cvScalar(120), 2);

Tracer un cercle

Pour tracer un cercle, la fonction est similaire à la précédente : la seule différence est qu'au lieu de préciser deux points, nous donnons le centre et le rayon du cercle.

void **cvCircle**(CvArr* img, CvPoint center, int radius, CvScalar color, int thickness=1, int line_type=8, int shift=0);

Vous reconnaissez bien sûr la plupart des paramètres, aussi les deux seuls qui varient sont :

- **center** : le centre du cercle ;
- **radius** : le rayon du cercle (en pixels).

cvCircle(image, cvPoint(100,150), 1, cvScalar(0), -1);

Tracer un rectangle

Le rectangle suit très logiquement la lancée des deux précédentes figures.
L'astuce, c'est qu'on le définit par **deux points** : classiquement, le point le plus **en haut à gauche** (donc de "petites" coordonnées), et le point le plus **en bas à droite** (de "grandes" coordonnées).
Mais il faut savoir qu'OpenCV se débrouillera toujours pour vous afficher un rectangle dont l'une des deux diagonales relie les deux points spécifiés, quels qu'ils soient.

void cvRectangle(CvArr* img, CvPoint pt1, CvPoint pt2, CvScalar color, int thickness=1, int line_type=8, int shift=0);

Comme pour le cercle : une épaisseur négative ou égale à la constante CV_FILLED vous coloriera l'intérieur du rectangle, avec la couleur spécifiée.

La structure CvFont

Pour écrire, il faut que nous disposions d'une police de caractères (= d'une "fonte").
Celle-ci est décrite par la structure CvFont.
Pour initialiser une police, on se sert de la fonction suivante :

void **cvInitFont**(CvFont* font, int font_face, double hscale, double vscale, double shear=0, int thickness=1, int line_type=8);

Détail des paramètres

- **font** : pointeur sur la structure CvFont à initialiser.
- **font_face** : le type de police. Par exemple : CV_FONT_HERSHEY_SIMPLEX.
- **hscale** : la taille "horizontale". La taille "normale" est 1.
- **vscale** : la taille "verticale".
- **shear** : "plus ou moins" italique. 0 = police droite, et 1 = police penchée à 45°.
- **thickness** : épaisseur du crayon, en pixels.

- **line_type** : la connexité, comme pour les fonctions précédentes.

```
CvFont font;
cvInitFont( &font , CV_FONT_HERSHEY_SIMPLEX, 0.5, 0.5, 0.0, 1);
```

Nom de la police	Description
CV_FONT_HERSHEY_SIMPLEX	Police de taille "normale", sans sérifications.
CV_FONT_HERSHEY_PLAIN	Police de petite taille, sans sérifications.
CV_FONT_HERSHEY_DUPLEX	Police de taille normale, sans sérifications. Un peu plus complexe que CV_FONT_HERSHEY_SIMPLEX.
CV_FONT_HERSHEY_COMPLEX	Police de taille normale, à sérifications.
CV_FONT_HERSHEY_TRIPLEX	Police de taille "normale", à sérifications. Plus complexe que CV_FONT_HERSHEY_COMPLEX.
CV_FONT_HERSHEY_COMPLEX_SMALL	Police de petite taille, à sérifications.
CV_FONT_HERSHEY_SCRIPT_SIMPLEX	Police de taille normale, de type "écriture manuelle".
CV_FONT_HERSHEY_SCRIPT_COMPLEX	Police de taille normale, de type "écriture manuelle", mais plus complexe que la précédente.

```
cvInitFont(&font, CV_FONT_HERSHEY_SCRIPT_COMPLEX | CV_FONT_ITALIC, 1, 1, 0, 1);
```

Écrire du texte sur l'image

Maintenant que nous avons une police, nous pouvons nous en servir pour placer notre texte sur une image... voici la fonction magique :

```
void cvPutText( CvArr* img, const char* text, CvPoint org, const CvFont* font, CvScalar color );
```

Détail des paramètres

- **img** : l'image sur laquelle on écrit.
- **text** : la chaîne de caractères (type "C") à écrire.
- **org** : le point d'origine du texte. C'est-à-dire le coin en bas à gauche de la première lettre.
- **font** : un pointeur sur la police de caractères utilisée.
- **color** : la couleur (ou le niveau de gris) du texte.

```
cvPutText( image, "Salut ", cvPoint(10,10), &font, cvScalar(127));
```

```
#include "cv.h"
#include "highgui.h"
#include "stdio.h"

int main( void )
{
    int i, j;
    IplImage* img;   // Image data structure

    //----------------- Programme -----------------

    //----------------- Afficher une image de 640 x 480 noire -----------------

    img = cvCreateImage( cvSize(640,480), IPL_DEPTH_8U, 3 ); // Creation d'une image en
                                                             // 640*480 en RGB, chaque
                                                             // encre codée sur un octet
    //----------------- Créer un rectangle -----------------------------------

    cvRectangle(img, cvPoint (25,3), cvPoint (200,200), cvScalar (0,255,0), 12, 8, 0 );

    //----------------- Créer une ligne --------------------------------------

    cvLine(img, cvPoint(50,60), cvPoint(300,300), cvScalar(0,0,255), 2);

    //----------------- Créer un cercle --------------------------------------

    cvCircle(img, cvPoint(450,150), 45, cvScalar(255,0,0), 1);

    // Création d'une fenêtre

    cvNamedWindow( "image", CV_WINDOW_AUTOSIZE );

    // Affichage de l'image dans la fenêtre

    cvShowImage( "image", img );

    // Attente infinie d'une touche clavier

    cvWaitKey(0);

    // La mémoire utilisée par l'image est libérée

    cvReleaseImage( &img );

    return 0;
}
```

3.8) Lecture d'une vidéo AVI

Il est possible de lire une séquence d'images à partir d'un fichier vidéo. Pour chaque étape, une image de type *IplImage* est automatiquement allouée (puis libérée) en mémoire.

```
IplImage *im;
CvCapture *avi;
/* Ouverture de la video */
avi = cvCaptureFromAVI("ma_video.AVI");
while(cvGrabFrame(avi))
{
im = cvRetrieveFrame(avi);
/* Traitement de l'image */
}
```

Fonction

Initialise la capture vidéo à partir d'un fichier

```
CvCapture* cvCaptureFromFile( const char* filename );
```

La fonction cvCaptureFromFile positionne et initialise le pointeur CvCapture

Lecture d'une vidéo AVI au format Mat

Cette fonction est le constructeur dans la classe_VideoCapture. Ce constructeur ouvre la vidéo et initialise l'objet VideoCapture

VideoCapture::**VideoCapture**(const string& filename)

const string& filename: nom du fichier vidéo à ouvrir
Le destructeur de cette classe désalloue la mémoire ce destructeur est explicite, on n' a pas besoin de l'appeler

Si l'objet VideoCapture s'est bien créé cette méthode retourne vraie ou sinon faux. Elle permet de contrôler si l'ouverture s'est bien réalisée

bool VideoCapture::**IsOpened()**

On peut changer les propriétés d'un objet VideoCapture. Si tout se passe bien cette méthode retourne vraie

bool VideoCapture::**set**(int propId, double value)

Paramètres :

int propID – Cet argument spécifie la propriété que tu veux changer

il en existe beaucoup :

CV_CAP_PROP_POS_MSEC – position courante de la video en millisecondes
CV_CAP_PROP_POS_FRAMES – position courante de la vidéo en trame
CV_CAP_PROP_FRAME_WIDTH – largeur de l'image du flux vidéoam
CV_CAP_PROP_FRAME_HEIGHT – Hauteur de l'image du flux vidéo
CV_CAP_PROP_FPS – nombre de trames par secondes
CV_CAP_PROP_FOURCC – code quatres caractères du CODEC

double value – C'est la nouvelle valeur que tu souhaites assigner à la nouvelle propriété

Cette fonction retourne la valeur de la propriété qui est spécifié par propId

doubleVideoCapture::**get**(int propId)

Cette fonction capture la prochaine trame de la vidéo, la décode et la stocke dans l'objet image

bool VideoCapture::**read**(Mat& image);

Si l'opération est un succès elle retournera vraie sinon faux

La fonction attend pendant n millisecondes. Si une touche a été pressée avant le temps spécifié la fonction retourne la valeur ASCII de la touche pressée

Char **waitKey**(int n)

Si la touche escape est pressée le programme continue, si aucune touche n'est pressée pendant n millisecondes la fonction retourne -1

```
#include <iostream>
using namespace cv;
using namespace std;
int main(int argc, char* argv[])
{
VideoCapture cap("C:/Users/SHERMAL/Desktop/SampleVideo.avi"); // open the video file for
reading
if ( !cap.isOpened() ) // if not success, exit program
{
cout << "Cannot open the video file" << endl;
return -1;
}
//cap.set(CV_CAP_PROP_POS_MSEC, 300); //start the video at 300ms
double fps = cap.get(CV_CAP_PROP_FPS); //get the frames per seconds of the video
cout << "Frame per seconds : " << fps << endl;
namedWindow("MyVideo",CV_WINDOW_AUTOSIZE); //create a window called "MyVideo"
while(1)
{
Mat frame;
bool bSuccess = cap.read(frame); // read a new frame from video
if (!bSuccess) //if not success, break loop
{
cout << "Cannot read the frame from video file" << endl;
break;
}
imshow("MyVideo", frame); //show the frame in "MyVideo" window
if(waitKey(30) == 27) //wait for 'esc' key press for 30 ms. If 'esc' key is pressed, break loop
{
cout << "esc key is pressed by user" << endl;
break;
```

```
}
return 0;
}
```

3.9) Ecrire une image ou une vidéo avec la classe Mat

La fonction sauvegarde une image dans un fichier baptisé filename. Si la fonction échoue, elle retourne faux si le fichier a été écrit sur le disque dur elle retourne vraie

```
bool imwrite( const string& filename, InputArray img, const vector<int>& params=vector<int>())
```

Paramètres-

filename – spécifie la position et le nom du fichier à sauvegarder
img – image que tu souhaites sauvegarder
params – Il spécifie le format de l'image specifying the format of the image

- ✓ format JPEG – tu dois spécifier la variable CV_IMWRITE_JPEG_QUALITY et un chiffre entre 0 et 100. Si tu veux la meilleure qualité il faut prendre 100
- ✓ format PNG – tu dois spécifier la variable CV_IMWRITE_PNG_COMPRESSION et un chiffre entre 0 et 9 (9 représente la meilleure compression).

Seulement des images avec 8 ou 16 bits non signés et un canal ou 3 canaux peuvent être sauvegardées (
CV_8UC1, CV_8UC3, CV_8SC1, CV_8SC3, CV_16UC1, CV_16UC3). Si la profondeur ou l'ordre des canaux est différents on utilisera les méthodes ::convertTo()' ou 'cvtColor' afin d'avoir un format correct pour la méthode imwrite

Créé un objet Size avec une largeur et une hauteur données.

```
Size frameSize(static_cast<int>(dWidth), static_cast<int>(dHeight))
```

C'est le constructeur de la classe VideoWriter. Il initialise l'objet VideoWriter

```
VideoWriter::VideoWriter(const string& filename, int fourcc, double fps, Size frameSize, bool isColor=true)
```

Paramètres:

const string& filename – spécifie le nom et la position du fichier à écrire. Le flux video sera écrit dans ce fichier

int fourcc – spécifie le code quatre caractère pour le CODEC qui est utilisé pour compresser la vidéo. Les CODEC supportés sont:

CV_FOURCC('D', 'I', 'V', '3') pour DivX MPEG-4 codec
CV_FOURCC('M', 'P', '4', '2') pour MPEG-4 codec
CV_FOURCC('D', 'I', 'V', 'X') pour DivX codec
CV_FOURCC('P','I','M','1') pour MPEG-1 codec
CV_FOURCC('I', '2', '6', '3') pour ITU H.263 codec
CV_FOURCC('M', 'P', 'E', 'G') pour MPEG-1 codec
double fps – trames par secondes du flux vidéo.
Size frameSize – dimension de l'objet qui spécifie la largeur et la hauteur de chaque trame.
bool isColor – si tu veux sauvegarder la vidéo en couleur passer la valeur à vraie

Cette méthode contrôle si l'objet VideoWriter a bien été créé, vraie en cas de succès.

bool **isOpened**(void)

écrire une trame dans le flux video . La dimension de l'image sera la même que celle que tu as spécifié dans l'objet VideoWriter

void **write**(const Mat& image)

```
#include "opencv2/highgui/highgui.hpp"
#include <iostream>
using namespace cv;
using namespace std;
int main( int argc, const char** argv )
{
Mat img(650, 600, CV_16UC3, Scalar(0,50000, 50000)); //create an image ( 3 channels, 16 bit
image depth, 650 high, 600 wide, (0,
50000, 50000) assigned for Blue, Green and Red plane respectively. )
if (img.empty()) //check whether the image is loaded or not
{
cout << "ERROR : Image cannot be loaded..!!" << endl;
//system("pause"); //wait for a key press
return -1;
}
vector<int> compression_params; //vector that stores the compression parameters of the
image
compression_params.push_back(CV_IMWRITE_JPEG_QUALITY); //specify the compression
technique
compression_params.push_back(98); //specify the compression quality
```

```
bool bSuccess = imwrite("D:/TestImage.jpg", img, compression_params); //write the image to
file
if ( !bSuccess )
{
cout << "ERROR : Failed to save the image" << endl;
//system("pause"); //wait for a key press
}
namedWindow("MyWindow", CV_WINDOW_AUTOSIZE); //create a window with the name
"MyWindow"
imshow("MyWindow", img); //display the image which is stored in the 'img' in the "MyWindow"
window
waitKey(0); //wait for a keypress
destroyWindow("MyWindow"); //destroy the window with the name, "MyWindow"
return 0;
}
```

```
#include "opencv2/highgui/highgui.hpp"
#include <iostream>
using namespace cv;
using namespace std;
int main(int argc, char* argv[])
{
VideoCapture cap(0); // open the video camera no. 0
if (!cap.isOpened()) // if not success, exit program
{
cout << "ERROR: Cannot open the video file" << endl;
return -1;
}
namedWindow("MyVideo",CV_WINDOW_AUTOSIZE); //create a window called "MyVideo"
double dWidth = cap.get(CV_CAP_PROP_FRAME_WIDTH); //get the width of frames of the
video
double dHeight = cap.get(CV_CAP_PROP_FRAME_HEIGHT); //get the height of frames of the
video
cout << "Frame Size = " << dWidth << "x" << dHeight << endl;
Size frameSize(static_cast<int>(dWidth), static_cast<int>(dHeight));
VideoWriter oVideoWriter ("D:/MyVideo.avi", CV_FOURCC('P','I','M','1'), 20, frameSize,
true); //initialize the VideoWriter object
if ( !oVideoWriter.isOpened() ) //if not initialize the VideoWriter successfully, exit the
program
```

```
{
cout << "ERROR: Failed to write the video" << endl;
return -1;
}
while (1)
{
Mat frame;
bool bSuccess = cap.read(frame); // read a new frame from video
if (!bSuccess) //if not success, break loop
{
cout << "ERROR: Cannot read a frame from video file" << endl;
break;
}
VideoWriter.write(frame); //writer the frame into the file
imshow("MyVideo", frame); //show the frame in "MyVideo" window
if (waitKey(10) == 27) //wait for 'esc' key press for 10ms. If 'esc' key is pressed, break loop
{
cout << "esc key is pressed by user" << endl;
break;
}
}
return 0;
}
```

```
#include <opencv2\core\core.hpp> //Module pour les structures de base et la classe d'image Mat
#include <opencv2\highgui\highgui.hpp> // Module pour les fonctions d'affichage à l'écran
using namespace cv; // Pour éviter de "trainer" le cv:: devant les noms.
int main( int argc, char** argv ) // argc et argv uniquement nécessaire si vous prévoyez envoyer des
// paramètres à la ligne de commande.
// Exemple : MonProgramme.exe Param1 Param2
// Param1 serait disponible dans argv[1] et Param2 dans argv[2].
{
Mat ImgSource; // Déclaration d'un objet de la classe Mat. C'est un objet qui représente
l'image en mémoire.
Mat ImgResultat; // Objet Mat pour l'image résultante. Ici, elle sera en ton de gris.
ImgSource = imread("Image.jpg", CV_LOAD_IMAGE_COLOR );
If (! ImgSource.data )
{
cout << "Erreur dans l'ouverture du fichier" << endl;
return -1;
}
Mat ImgResultat;
cvtColor( ImgSource, ImgResultat, CV_BGR2GRAY );
imwrite( "../../images/Image_Gris.jpg", ImgResultat );
namedWindow( "Image source", CV_WINDOW_AUTOSIZE );
namedWindow( "Image Resultat", CV_WINDOW_AUTOSIZE );
imshow( "Image source", ImgSource );
imshow( "Image Resultat", ImgResultat );
waitKey(0);
return 0;
}
```

3.10) Utilisation de la WEBCAM

On crée une image au format IplImage* (format d'image pour OpenCV) et on utilise la structure CvCAPTURE. Celle-ci sert notamment à capturer les frames de la vidéo et de les convertir en IplImage*.

On crée une image au format IplImage qui s'appelle ::< frame >>

```
IplImage* frame = 0

//On initialise une structure CvCapture nominee << capture >>

CvCapture* capture = 0

//On définit capture comme étant la capture de frame de la camera 0 (notre
webcam)
capture = cvCaptureFromCAM(0) ;
```

Pour afficher le flux d'image de la Webcam, il faut faire en sorte d'afficher les images converties en IplImage en boucle (dans un `while infinie'), c'est-a-dire :

```
while(1)        //boucle infinie
{
Trame= cvQueryFrame(capture) ;
cvShowImage("video ",frame) ;
}
```

On traite ensuite l'image comme on veut dans cette boucle infinie, on fait toutes les opérations nécessaires au traitement d'image souhaité.

Il faut ensuite des allouer la mémoire occupée par nos variables :

```
cvReleaseImage(&frame);
cvReleaseCapture( &capture );
```

Initialise la capture à partir d'une WEBCAM

```
CvCapture* cvCaptureFromCAM( int index );
```

index
Index de la camera à utiliser. Si tu ne le connais pas utilise -1

cvQueryFrame

Recueil et retourne une trame à partir de la caméra

```
IplImage* cvQueryFrame( CvCapture* capture );
```

cvReleaseCapture

Relache la capture

```
void cvReleaseCapture( CvCapture** capture );
```

capture : Pointeur sur une structure de la vidéo capturée

cvSetCaptureProperty

Positionne les propriétés de la capture

```
int cvSetCaptureProperty( CvCapture* capture, int property_id, double value );
```

capture : Pointeur sur une structure de la vidéo capturée

property_id

Identificateur des propriétés qui put être de la forme suivante :

CV_CAP_PROP_POS_MSEC
Position en milliseconds à partir du début de fichier

CV_CAP_PROP_POS_FRAMES
Position dans la trame seulement pour les fichiers vidéo

CV_CAP_PROP_FRAME_WIDTH
Largeur des trames capturées pour la webcam

CV_CAP_PROP_FRAME_HEIGHT
Hauteur des trames capturées pour la webcam

CV_CAP_PROP_FPS
Nombre d'images par seconde

CV_CAP_PROP_BRIGHTNESS

Luminosité des images capturées

CV_CAP_PROP_CONTRAST

Contraste des images capturées

CV_CAP_PROP_SATURATION
Saturation des images capturées

Utilisation de la webcam avec la classe Mat

Cette fonction est le constructeur dans la classe_VideoCapture. Ce constructeur ouvre la webcam indexée par l'argument et initialise l'objet VideoCapture

VideoCapture::**VideoCapture**(int device)

Les autres méthodes vues dans le chapitre précèdent sur la capture vidéo reste valable

```
#include <iostream>
using namespace cv;
using namespace std;
int main(int argc, char* argv[])
{
VideoCapture cap(0); // open the video camera no. 0
if (!cap.isOpened()) // if not success, exit program
{
cout << "Cannot open the video cam" << endl;
return -1;
}
double dWidth = cap.get(CV_CAP_PROP_FRAME_WIDTH); //get the width of frames of the video
double dHeight = cap.get(CV_CAP_PROP_FRAME_HEIGHT); //get the height of frames of the video
cout << "Frame size : " << dWidth << " x " << dHeight << endl;
namedWindow("MyVideo",CV_WINDOW_AUTOSIZE); //create a window called "MyVideo"
while (1)
{
Mat frame;
bool bSuccess = cap.read(frame); // read a new frame from video
if (!bSuccess) //if not success, break loop
{
cout << "Cannot read a frame from video stream" << endl;
break;
```

```
}
imshow("MyVideo", frame); //show the frame in "MyVideo" window
if (waitKey(30) == 27) //wait for 'esc' key press for 30ms. If 'esc' key is pressed, break loop
{
cout << "esc key is pressed by user" << endl;
break;
}}return 0;}/
```

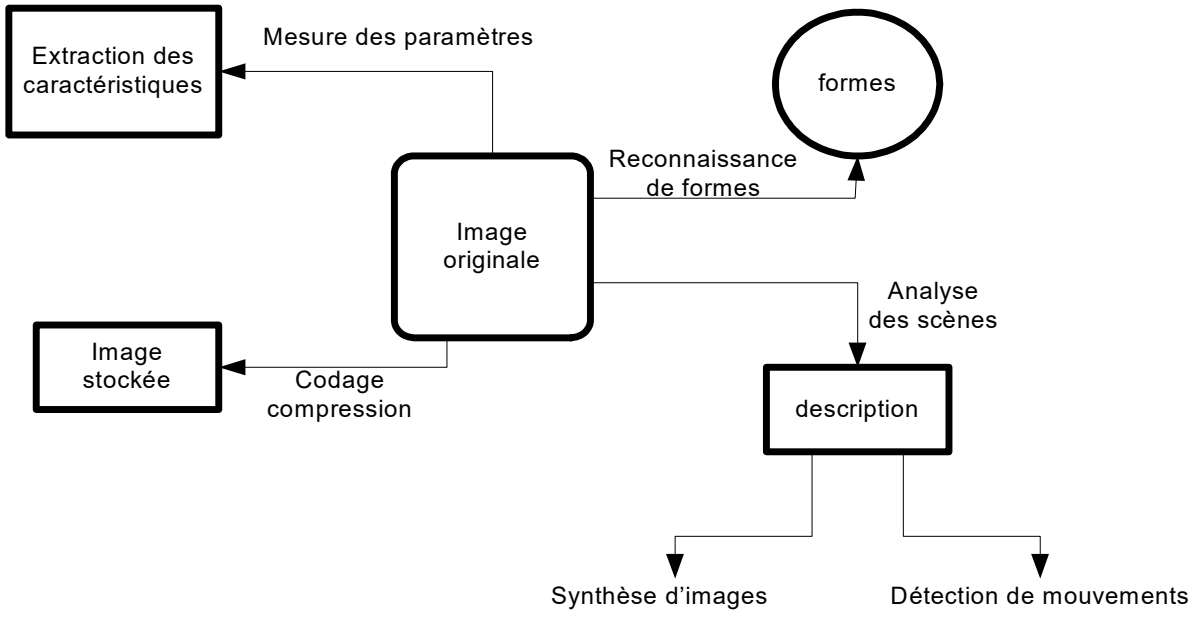
4) Les principaux traitements sous OPENCV

4.1) Introduction à la vision assistée par ordinateur

a) Définitions

V.A.O: Chaîne visuelle partant de la perception d'une image, interprétant son contenu pour agir sur l'environnement.

Traitement de l'image: Modification de l'image en vue d'en interpréter son contenu. Le traitement de l'image s'intègre comme un des outils de la chaîne de VAO.



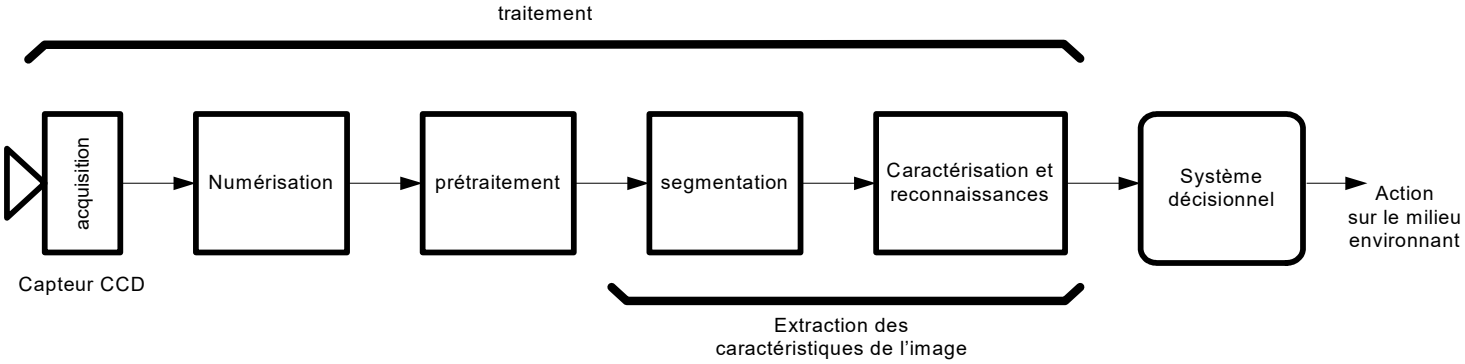
b) Chaîne fonctionnel du traitement d'image pour la reconnaissance de formes

Une image en reconnaissance de forme est une somme d'objets (caractérisés par des contours) et des fonds que l'on appelle texture

b.1/ Domaines d'application du traitement d'image

- Imagerie aérienne et spatiale (images satellites, cartographie, météorologie ...)
- techniques biomédicales (scanner, échographie ...)
- robotique (assemblage, contrôle de qualité ...)

- astronomie, physique nucléaire, ….



c) Le prétraitement des images

c.1/ Introduction

Le prétraitement d'une image a pour but d'améliorer la qualité des images (LUT , Filtrage).
Le but de cette partie est de mettre en œuvre des outils permettant d'effectuer des traitements et prétraitements sur l'image affichée précédemment avant d'effectuer la phase de décision.

c.2/ Intérêt du prétraitement

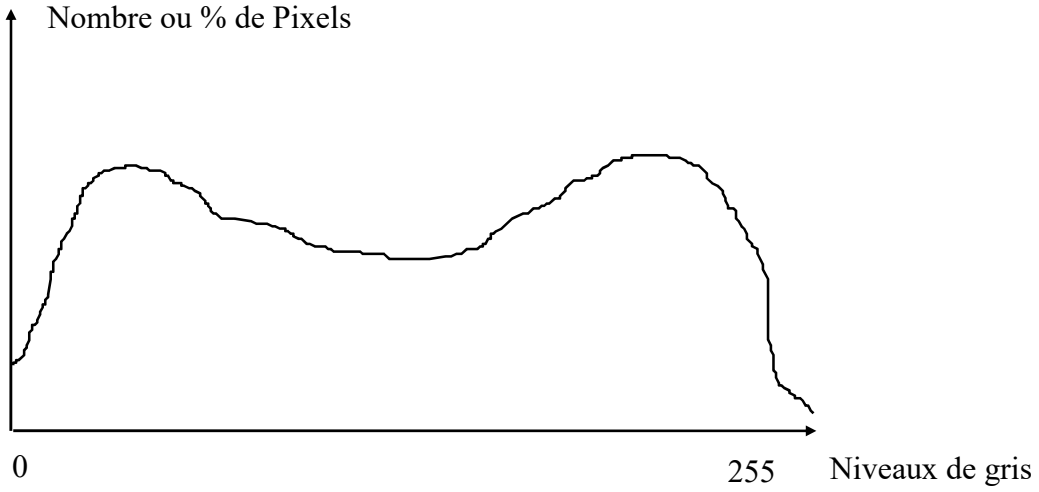
Il s'effectue sur une image brute, et permet:

- la restauration de l'image en supprimant un certain nombre de défauts tels que le bruit (pixels isolés de valeur erronée) ou le flou de mise au point.
- l'amélioration de l'image, par amélioration de l'aspect visuel ou meilleure adéquation aux besoins par exemple en assombrissant une image trop claire.

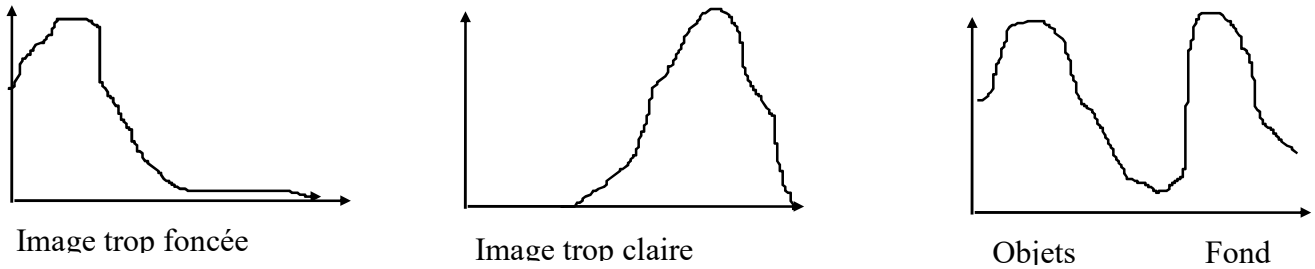
c.3/ Les outils du prétraitement

c.3.1/ L'histogramme

C'est un graphique permettant de connaitre la population de chaque niveau de gris de l'image.



L'observation des histogrammes permet de déceler les défauts d'une image (trop claire, trop foncée, manque de contraste) ou de déduire certaines propriétés (séparation des objets du fond).



c.3.2/ Modification des histogrammes avec une LUT

Utilisation des LUT (Look Up Table)

C'est une table de conversion qui réalise une fonction de transfert entre les données numériques d'entrée et sa sortie numérique. A un niveau de gris en entrée elle fait correspondre un niveau de gris en sortie.

En informatique, une LUT peut donc être modélisée par un tableau dont l' index représente les niveaux de gris en entrée et le contenu du tableau les niveaux de gris en sortie.

Exemple:

```
unsigned char LutInversion[256];
int i;
for (i=0;i<256;i++)
        LutInversion[i] = 255-i;
```

c.3.3/ Le Filtrage

Les opérations précédentes étaient ponctuelles, chaque pixel étant transformé indépendamment de la valeur de ses voisins. Dans le filtrage, au contraire chaque pixel est traité en fonction de sa propre valeur et de celle de ses voisins. On utilise souvent un masque de filtrage (matrice de convolution) qui prend en compte les 8 voisins immédiats du point traité.

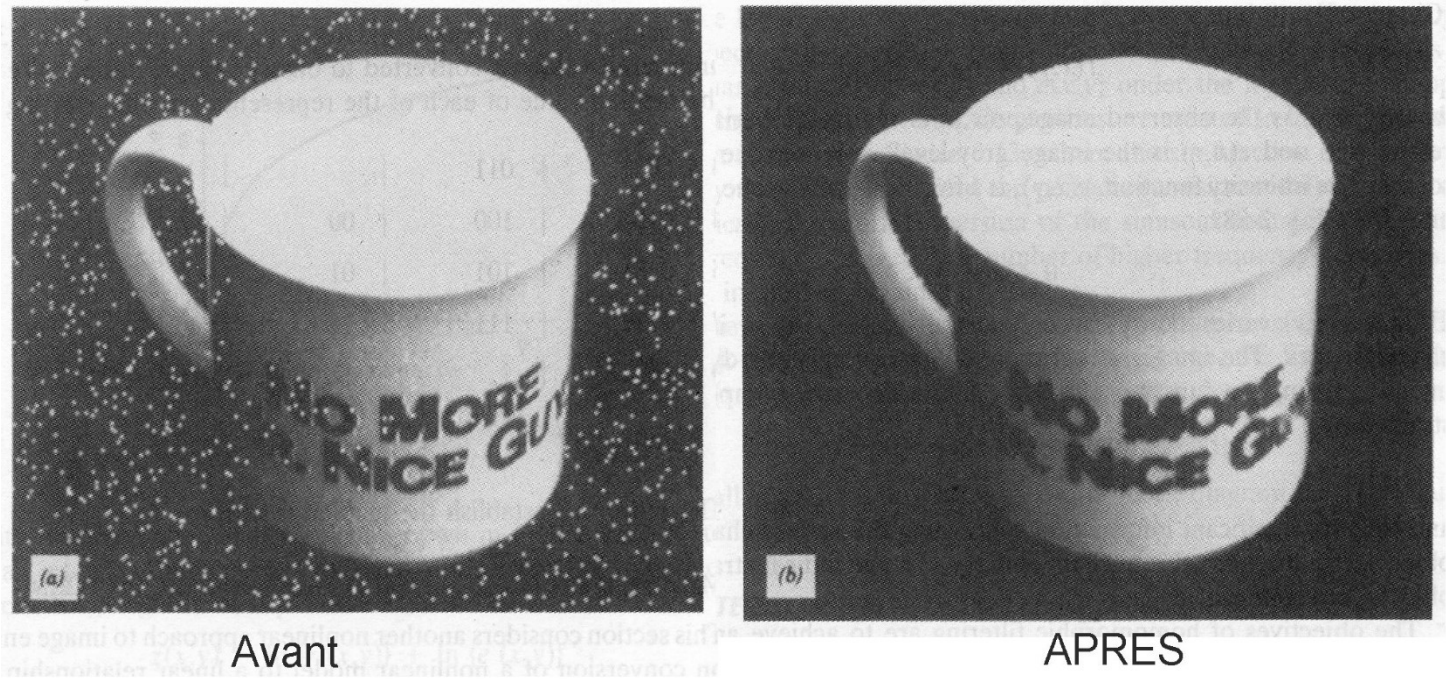
$$H = \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}$$

$$\boxed{I(x,y)= [\ a^* I(x-1,y-1) + b\ ^*I(x-1,y) + c^* I(x-1,y+1) + ...]\ /K}$$

Exemples de filtres:

Le moyenneur : Ce filtre à pour but d'adoucir une image en diminuant les variations brusques d'intensité. Il est notamment utilisé pour diminuer les effets du bruit.

$$H = \begin{vmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{vmatrix}$$



c.4/ La segmentation

c.4.1) Définition

La segmentation regroupe toutes les **techniques permettant de faire apparaître sur l'image les objets ou les entités d'intérèts**. Ces techniques sont la détection de contour et le seuillage.

Dans une image un contour est un ensembles de points qui correspond à un changement rapide (vis à vis du voisinage des points du contour) du niveau.

Une approche classique repose sur le traitement en 2 étapes :

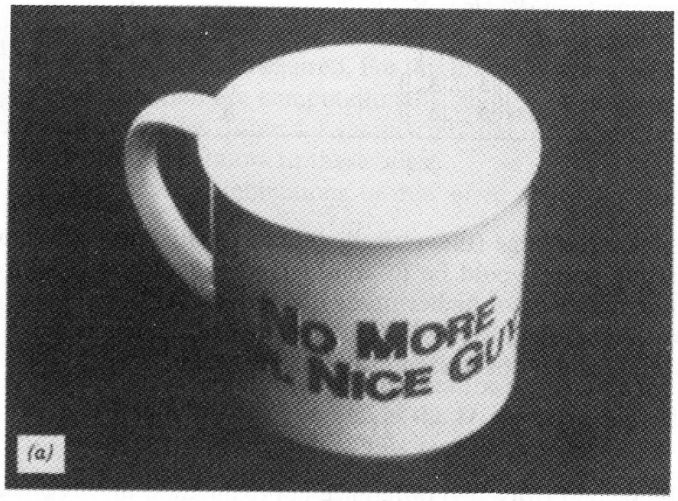
- Une accentuation des contours
- Un seuillage qui permet d'obtenir une image binaire.

c.4 .2) Le filtrage Laplacien

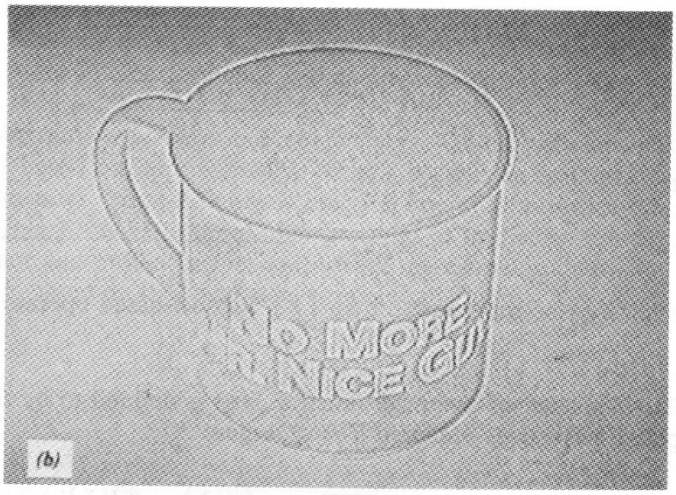
Une première approche très simple pour accentuer les contours est d'utiliser le filtre laplacien. Le gabarit du filtre est le suivant :

Le laplacien : Ce filtre a pour but de faire apparaître toute variation de niveau de pixel dans une image

$$H = \begin{vmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{vmatrix}$$



avant filtrage Laplacien



après filtrage Laplacien

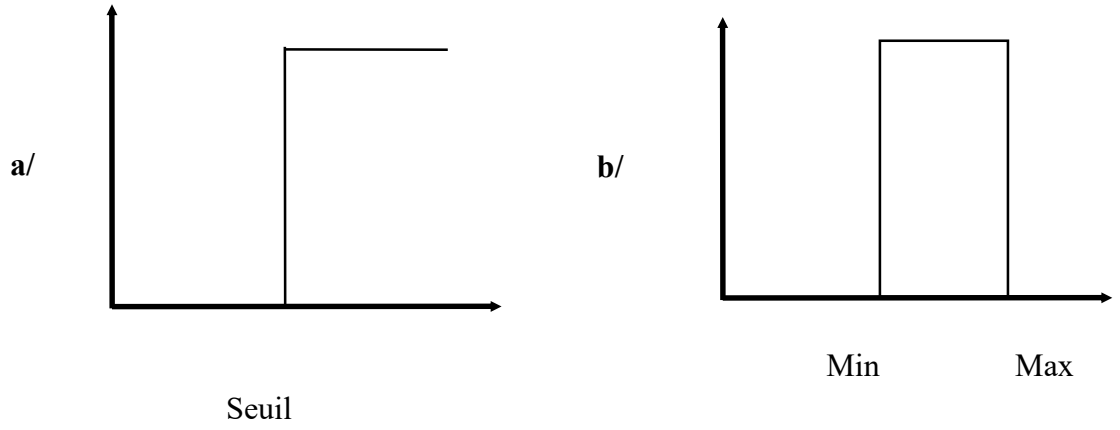
c.4.3/ Binarisation d'une image

La binarisation est une opération de traitement simple qui consiste à ne conserver dans une image que 2 niveaux de gris (0=noir 1=blanc);

- soit en convertissant en noir tous les niveaux de gris inférieurs à un seuil et en blanc tous ceux qui dépassent le seuil
- soit en convertissant en blanc toutes les valeurs comprises entre deux seuils, et en noir tout le reste.

Le choix d'un seuil de binarisation à partir d'un histogramme permet par exemple de différencier des objets (noir) d'un fond (blanc).

LUT de binarisation:



4.2) Convertir une image en niveaux de gris et retourner une image verticalement

void **cvConvertImage**(IplImage *src, IplImage *dst, int flags=0);

Cette fonction convertit l'image "src" en image "dst", avec des options facultatives.

Exemple :

Détail des paramètres

- **src** et **dst** : images source et destination de la conversion.
- **flags** : laisser à 0 si on fait une conversion simple, ou bien lui donner la constante CV_CVTIMG_FLIP pour retourner l'image en même temps.

Code : C++

```
IplImage *image_couleur = cvLoadImage("monimage.jpg");
//Image en niveaux de gris :
//Même taille que image_couleur, même profondeur aussi, mais 1 canal
IplImage *image_nvg = cvCreateImage(cvGetSize(image_couleur), image_couleur->depth, 1);
//Convertir en niveaux de gris :
cvConvertImage(image_couleur, image_nvg);
//Convertir en niveaux de gris en retournant verticalement :
cvConvertImage(image_couleur, image_nvg, CV_CVTIMG_FLIP);
```

Conversion en niveaux de gris

Là nous allons convertir notre image en niveaux de gris.
Mais **comme nous sommes prudents**, nous nous souvenons que l'image de base peut ne pas avoir une origine conventionnelle.
Donc nous allons d'abord le vérifier, et s'il le faut, on retournera l'image.

IPL_ORIGIN_TL est une constante qui signifie "Origine de l'IplImage en haut à gauche" (TL = "Top Left").
Si l'origine n'est pas bonne, flip contient l'instruction "retourner l'image", sinon, il contient 0 pour "ne rien faire".
On utilise ensuite la fonction cvConvertImage pour convertir notre image en niveaux de gris.

```
//On vérifie l'origine de l'image
//Si elle n'est pas en haut à gauche, il faut la corriger
int flip=0;
if(img->origin!=IPL_ORIGIN_TL)
{
flip=CV_CVTIMG_FLIP;
}
```

```
//Conversion en niveaux de gris
cvConvertImage(img, img_nvg, flip);
```

La conversion en niveau de gris et la binarisation d'une image sous Mat

La conversion en niveau de gris

La méthode s'appelle cvtColor et permet de passer d'un espace couleur à un autre

C++: void **cvtColor**(InputArray **src**, OutputArray **dst**, int **code**, int **dstCn**=0)¶

- **src** – image d'entrée sous le format Mat 8-bit unsigned, 16-bit unsigned (`CV_16UC...`), ou single-precision floating-point.
- **dst** – image traitée de la même dimension et profondeur que **code** – color space conversion code (see the description below).
- **Code** Paramètre qui définit le type de conversion de couleur
- **dstCn** – Nombre de canaux dans l'image de destination si le parameter est 0 le nombre de canaux est dérivé de src

Pour une conversion de RGB en HSV on pourra utiliser les codes :

RGB ↔HSV (`CV_BGR2HSV, CV_RGB2HSV, CV_HSV2BGR, CV_HSV2RGB`)

Exemple d'utilisation

```
cv::Mat gray;
cv::cvtColor(src, gray, CV_BGR2GRAY); // conversion en niveau de gris
```

Conversion d'une image binarisée

La méthode threshold permet d'obtenir entre autre une image binaire elle est décrite dans le chapitre précédent :

Exemple d'utilisation

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

Pour le mode CV_THRESH_BINARY

si (pixel imgGris > 185) alors pixel=255

sinon pixel =0

4.3) Les filtres de base et OPENCV

4.3.1) Le filtre moyenneur

void **medianBlur**(InputArray src, OutputArray dst, int ksize)

Les paramètres:
•src – image à traiter 1-, 3-, or 4-canaux; CV_8U, CV_16U, or CV_32F, CV_8U.
•dst – image traitée qui a la même dimension que l'image traitée.
•ksize – dimension linéaire d'ouverture; il doit être impair et plus grand que 1

```
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui_c.h>

using namespace cv;

int main(int argc, char** argv)
{
  namedWindow("Before" , CV_WINDOW_AUTOSIZE);

  // Load the source image
  Mat src = imread( "/home/khong/OpenCV/workspace/images/zebra.jpg", 1);

  // Create a destination Mat object
  Mat dst;

  // display the source image
  imshow("Before", src);

  for (int i=1; i<51; i=i+2)
  {
    // smooth the image in the "src" and save it to "dst"
    // blur(src, dst, Size(i,i));

    // Gaussian smoothing
    // GaussianBlur( src, dst, Size( i, i ), 0, 0 );

    // Median smoothing
    medianBlur( src, dst, i );
```

```
    // show the blurred image with the text
    imshow( "Median filter", dst );

    // wait for 5 seconds
    waitKey(5000);
  }
}
```

4.3.2) Le filtre laplacien

void **Laplacian**(InputArray src, OutputArray dst, int ddepth, int ksize=1, double scale=1, double delta=0, int borderType=BORDER_DEFAULT)

src – image Source
dst – image de destination
ddepth – profondeur désirée de l'image de destination
ksize –dimension de l'ouverture.
scale – facteur d'échelle
delta –
borderType

```
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <stdlib.h>
#include <stdio.h>

using namespace cv;

int main( int argc, char** argv )
{
    Mat src, gray, dst, abs_dst;
    src = imread( "lena.jpg" );

    /// Remove noise by blurring with a Gaussian filter
    GaussianBlur( src, src, Size(3,3), 0, 0, BORDER_DEFAULT );
    cvtColor( src, gray, CV_RGB2GRAY );

    /// Apply Laplace function
    Laplacian( gray, dst, CV_16S, 3, 1, 0, BORDER_DEFAULT );
    convertScaleAbs( dst, abs_dst );

    imshow( "result", abs_dst );

    waitKey(0);
    return 0;}
```

4.4) L' histogramme sous Opencv

Cette fonction permet d'obtenir l'histogramme d'une image. Le nombre de paramètres est très important et cette fonction peut devenir très vite complexe. Pour un exemple d'histogramme à une dimension on se limitera à la déclaration suivante :

```
void calcHist(const Mat* arrays, int narrays, const int* channels, InputArray mask,
OutputArray hist, int dims, const int* histSize, const float** ranges, bool uniform=true,
bool accumulate=false )
```

arrays : l'image source
narrays : le nombre d'image en entrée le plus souvent une
channels : 0
mask :
hist : l'histogramme de sortie qui est un tableau de dimension dims
dims : dimension de l'histogramme bien souvent à 1
histSize : la taille de l'histogramme en x
ranges: l'échelle pour chaque dimension de l'histogramme
bool uniform true par défaut
bool Accumulate false par défaut

Cette fonction extrait chacun des n canaux en n tableau multi de canaux séparés et les stockent dans un vecteur.

```
void split(const Mat& m, vector<Mat>& mv )
```

Normalise une gamme de valeur dans un tableau

```
void normalize(const SparseMat& src, SparseMat& dst, double alpha, int normType)
```

Paramètre:

- **src** – tableau d'entrée.
- **dst** – tableau de sortie de la même dimension que src .
- **alpha** Plus basse echelle dans le cas d'une normalization normale.
- **beta** – Plus haute echelle dans le cas d'une normalization normale
- **normType** – type de normalisation (`NORM_MINMAX`, `NORM_INF`, `NORM_L1`, or `NORM_L2`).
- **dtype** – quand négatif , le tableau de sortie a le même type que l'entrée
- **mask** – optionnel.

Pour une image en niveau de gris

```
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>

using namespace std;
using namespace cv;
```

```
int main(int, char**)
{
    Mat gray=imread("image.jpg",0);
    namedWindow( "Gray", 1 );    imshow( "Gray", gray );

    // Initialize parameters
    int histSize = 256;    // bin size
    float range[] = { 0, 255 };
    const float *ranges[] = { range };

    // Calculate histogram
    MatND hist;
    calcHist( &gray, 1, 0, Mat(), hist, 1, &histSize, ranges, true, false);

    // Show the calculated histogram in command window
    double total;
    total = gray.rows * gray.cols;
    for( int h = 0; h < histSize; h++ )
         {
            float binVal = hist.at<float>(h);
            cout<<" "<<binVal;
         }

    // Plot the histogram
    int hist_w = 512; int hist_h = 400;
    int bin_w = cvRound( (double) hist_w/histSize );

    Mat histImage( hist_h, hist_w, CV_8UC1, Scalar( 0,0,0) );
    normalize(hist, hist, 0, histImage.rows, NORM_MINMAX, -1, Mat() );

    for( int i = 1; i < histSize; i++ )
    {
      line( histImage, Point( bin_w*(i-1), hist_h - cvRound(hist.at<float>(i-1)) ) ,
                       Point( bin_w*(i), hist_h - cvRound(hist.at<float>(i)) ),
                       Scalar( 255, 0, 0), 2, 8, 0  );
    }

    namedWindow( "Result", 1 );    imshow( "Result", histImage );

    waitKey(0);
    return 0;
}
```

Pour une image en couleur

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
#include <stdio.h>

using namespace std;
using namespace cv;

/**
 * @function main
 */
int main( int argc, char** argv )
```

```
{
  Mat src, dst;

  /// Load image
  src = imread( argv[1], 1 );

  if( !src.data )
    { return -1; }

  /// Separate the image in 3 places ( B, G and R )
  vector<Mat> bgr_planes;
  split( src, bgr_planes );

  /// Establish the number of bins
  int histSize = 256;

  /// Set the ranges ( for B,G,R) )
  float range[] = { 0, 256 } ;
  const float* histRange = { range };

  bool uniform = true; bool accumulate = false;

  Mat b_hist, g_hist, r_hist;

  /// Compute the histograms:
  calcHist( &bgr_planes[0], 1, 0, Mat(), b_hist, 1, &histSize, &histRange, uniform, accumulate );
  calcHist( &bgr_planes[1], 1, 0, Mat(), g_hist, 1, &histSize, &histRange, uniform, accumulate );
  calcHist( &bgr_planes[2], 1, 0, Mat(), r_hist, 1, &histSize, &histRange, uniform, accumulate );

  // Draw the histograms for B, G and R
  int hist_w = 512; int hist_h = 400;
  int bin_w = cvRound( (double) hist_w/histSize );

  Mat histImage( hist_h, hist_w, CV_8UC3, Scalar( 0,0,0) );

  /// Normalize the result to [ 0, histImage.rows ]
  normalize(b_hist, b_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat() );
  normalize(g_hist, g_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat() );
  normalize(r_hist, r_hist, 0, histImage.rows, NORM_MINMAX, -1, Mat() );

  /// Draw for each channel
  for( int i = 1; i < histSize; i++ )
  {
     line( histImage, Point( bin_w*(i-1), hist_h - cvRound(b_hist.at<float>(i-1)) ) ,
                   Point( bin_w*(i), hist_h - cvRound(b_hist.at<float>(i)) ),
                   Scalar( 255, 0, 0), 2, 8, 0  );
     line( histImage, Point( bin_w*(i-1), hist_h - cvRound(g_hist.at<float>(i-1)) ) ,
                   Point( bin_w*(i), hist_h - cvRound(g_hist.at<float>(i)) ),
                   Scalar( 0, 255, 0), 2, 8, 0  );
     line( histImage, Point( bin_w*(i-1), hist_h - cvRound(r_hist.at<float>(i-1)) ) ,
```

```
                Point( bin_w*(i), hist_h - cvRound(r_hist.at<float>(i)) ),
                Scalar( 0, 0, 255), 2, 8, 0  );
  }

  /// Display
  namedWindow("calcHist Demo", CV_WINDOW_AUTOSIZE );
  imshow("calcHist Demo", histImage );

  waitKey(0);

  return 0;
}
```

5) La détection de formes sur une image

5.1) Detection de contours

Le seuillage

Applique un seuil fixe à chaque élément de l'image src écrit l'image correspondante dans src

cvThreshold(const Mat& src, Mat& dst, double threshVal, double max, int thresholdType)

Paramètres:

const Mat& src – image source simple canal
Mat& dst – image de destination
double threshVal – valeur du seuil
double max – valeur maximal à utilizer avec 'THRESH_BINARY' et 'THRESH_BINARY_INV' qui sont les types de seuillage
int thresholdType – Type de seuillage

- THRESH_BINARY

dst(x,y)=max, if src(x,y) > ThreshVal
dst(x,y)=0, if src(x,y) < ThreshVal

- THRESH_BINARY_INV

dst(x,y)=max, if src(x,y) < ThreshVal

- THRESH_TOZERO

dst(x,y)=src(x,y), if src(x,y) > ThreshVal
dst(x,y)=0, if src(x,y) < ThreshVal

- THRESH_TOZERO_INV

dst(x,y)=0, if src(x,y) > ThreshVal
dst(x,y)=src(x,y), if src(x,y) < ThreshVal

- THRESH_TRUNC

dst(x,y)=threshVal, if src(x,y) > ThreshVal
dst(x,y)=src(x,y), if src(x,y) < ThreshVal

5.1.1) Renforcement des contours par la méthode de Sobel, Laplace et Canny

5.1.1.1) Filtre de Sobel

Le filtre de Sobel est un operateur 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 connait alors les points de changement soudain de luminosite, correspondant probablement à des bords, ainsi que l'orientation de ces bords. En termes mathematiques, le gradient d'une fonction de deux variables (ici l'intensite en fonction des coordonnees 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 :

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

Matrice de Sobel pour les contours horizontaux

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

Matrice de Sobel pour les contours verticaux

```
#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)
{

return -1;
}

cout << "Erreur dans ouverture du fichier source" << endl;


//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;
```

5.1.1.2) 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 $\triangle I(x,y)$.

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).

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

```
#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;
}
```

5.1.1.3) Renforcement des contours par la méthode de Canny

void **Canny**(InputArray **image**, OutputArray **edges**, double **threshold1**, double **threshold2**, int **apertureSize**=3, bool **L2gradient**=false)¶

- **image** – image sur 8 bits
- **edges** –les fronts; il a la même dimension et type que l'`image` .
- **threshold1** – premier seuil pour la procédure d' hysteresis.
- **threshold2** – second seuil pour la procédure d' hysteresis.

  apertureSize – dimension de l'ouverture pour l'opérateur de **`Sobel()`** .

- Ouvrir l'image à traiter
- Convertir l'image originale en ton de gris (si nécessaire)
- Appliquer, au besoin, une binarisation (threshold)
- Appliquer l'algorithme de Canny (Canny)

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;
}
```

5.2) Analyse des contours

Après avoir procéder à améliorer les contours et les isoler adéquatement du fond de l'image, nous sommes maintenant prêt à trouver les contours des objets dans l'image.

Trouver les contours des objets (findContours)

Cette fonction permet de trouver les contours en remplissant un vecteur de point qui correspond aux points qui sont définies sur le contour extérieur de l'objet. En fait, plusieurs constantes sont définies dans OpenCV et permettent de trouver les points extérieurs de l'objet mais aussi ceux qui font parties des « trous » internes de l'objet. On peut aussi vouloir créer une hiérarchie d'objet en passant au travers une liste des contours extérieurs jusqu'aux contours intérieurs.

Dans le module précédent, nous avons appliqué un filtre de Canny pour ressortir les pixels faisant partie du contour d'un objet.On peut ensuite, avec la fonction « findContours » trouver les points sur ce contour. Cette séquence de point est essentielle dans la reconnaissance de forme. Nous allons donc appliquer cette fonction aux objets de façon à déterminer leur forme.

Cette fonction trouve tous les contours dans une image binaire

cvFindContours(CvArr* img, CvMemStorage* str, CvSeq** first_contour, int header_size, int mode,int method, CvPoint offset)

Arguments -

CvArr* img – image source (celle-ci doit être sur 8bits simple canal). L'image sera binarisée.
CvMemStorage* str – Bloc de mémoire pour stocker tous les coutours obtenus
CvSeq** first_contour – pointeur sur le premier contour de la mémoire bloc, 'str'
int header_size – dimension de l'en tête de séquence
int mode – mode de reconnaissance des contours
CV_RETR_LIST – Reconnait tous les contours et les mets dans une liste
CV_RETR_EXTERNAL – Reconnait seulement tous les contours externes
CV_RETR_CCOMP – Reconnaissance de tous les contours et les organisent dans une hiérarchie à deux niveaux
CV_RETR_TREE – Reconnait tous les contours et les reconstruit dans une pleine hiérarchie
int method – methods d'approximation
CV_CHAIN_CODE -
CV_CHAIN_APPROX_NONE
CV_CHAIN_APPROX_SIMPLE
CV_CHAIN_APPROX_TC89_L1,
CV_CHAIN_APPROX_TC89_KCOS
CV_LINK_RUNS

CvPoint offset – offset par lequel tous les points des contours seront décalés. Ceci est habituel quand nous avons une région d'intérêt dans l'image.

Cette fonction approxime les courbes polygonales avec une précision spécifiée

cvApproxPoly (const void* src, int header_size, CvMemStorage* storage, int method, double para1,int para2)

arguments -

const void* src – Séquence de points
int header_size – dimension de l'entête de séquence
CvMemStorage* storage – mémoire bloc qui contient tous les contours
int method – méthode d'approximation
double para1 – précision de l'approximation
int para2 – Détermine si une simple séquence sera approximé ou toutes les séquences

Cette fonction retourne l'index de l'élément de la séquence

cvGetSeqElem(const CvSeq* seq, int index)

Désalloue la mémoire qui a été positionné par cvCreateMemStorage()

cvReleaseMemStorage(CvMemStorage** storage)

Cette function créé une mémoire de stockage qui a la capacité spécifiée par le paramètre byteSize. Si byteSize=0 la capacité allouée est de 64Kb

cvCreateMemStorage(int byteSize)

exemple :

```
#include <cv.h>
#include <highgui.h>
using namespace std;
int main()
{
IplImage* img = cvLoadImage("C:/Users/SHERMAL/Desktop/FindingContours.png");
//show the original image
cvNamedWindow("Raw");
cvShowImage("Raw",img);
//converting the original image into grayscale
IplImage* imgGrayScale = cvCreateImage(cvGetSize(img), 8, 1);
cvCvtColor(img,imgGrayScale,CV_BGR2GRAY);
cvThreshold(imgGrayScale,imgGrayScale,128,255,CV_THRESH_BINARY);
CvSeq* contours; //hold the pointer to a contour in the memory block
CvSeq* result; //hold sequence of points of a contour
CvMemStorage *storage = cvCreateMemStorage(0); //storage area for all contours
//finding all contours in the image
cvFindContours(imgGrayScale, storage, &contours, sizeof(CvContour), CV_RETR_LIST,
CV_CHAIN_APPROX_SIMPLE, cvPoint
(0,0));
//iterating through each contour
while(contours)
{
//obtain a sequence of points of contour, pointed by the variable 'contour'
result = cvApproxPoly(contours, sizeof(CvContour), storage, CV_POLY_APPROX_DP,
cvContourPerimeter(contours)*0.02, 0);
//if there are 3 vertices in the contour(It should be a triangle)
if(result->total==3 )
{
//iterating through each point
CvPoint *pt[3];
for(int i=0;i<3;i++){
pt[i] = (CvPoint*)cvGetSeqElem(result, i);
}
//drawing lines around the triangle
cvLine(img, *pt[0], *pt[1], cvScalar(255,0,0),4);
cvLine(img, *pt[1], *pt[2], cvScalar(255,0,0),4);
cvLine(img, *pt[2], *pt[0], cvScalar(255,0,0),4);
}
//if there are 4 vertices in the contour(It should be a quadrilateral)
else if(result->total==4 )
{
//iterating through each point
CvPoint *pt[4];
for(int i=0;i<4;i++){
pt[i] = (CvPoint*)cvGetSeqElem(result, i);
}
//drawing lines around the quadrilateral
```

```
cvLine(img, *pt[0], *pt[1], cvScalar(0,255,0),4);
cvLine(img, *pt[1], *pt[2], cvScalar(0,255,0),4);
cvLine(img, *pt[2], *pt[3], cvScalar(0,255,0),4);
cvLine(img, *pt[3], *pt[0], cvScalar(0,255,0),4);
}
//if there are 7 vertices in the contour(It should be a heptagon)
else if(result->total ==7 )
{
//iterating through each point
CvPoint *pt[7];
for(int i=0;i<7;i++){
pt[i] = (CvPoint*)cvGetSeqElem(result, i);
}
//drawing lines around the heptagon
cvLine(img, *pt[0], *pt[1], cvScalar(0,0,255),4);
cvLine(img, *pt[1], *pt[2], cvScalar(0,0,255),4);
cvLine(img, *pt[2], *pt[3], cvScalar(0,0,255),4);
cvLine(img, *pt[3], *pt[4], cvScalar(0,0,255),4);
cvLine(img, *pt[4], *pt[5], cvScalar(0,0,255),4);
cvLine(img, *pt[5], *pt[6], cvScalar(0,0,255),4);
cvLine(img, *pt[6], *pt[0], cvScalar(0,0,255),4);
}
//obtain the next contour
```

Recherche de contours par la méthode findContours

void findContours**(InputOutputArray** image**, OutputArrayOfArrays** contours**, OutputArray** hierarchy**, int** mode**, int** method**, Point** offset**=Point())**¶

- **image** – Image source est une image simple canal à 8 bits. Source, an 8-bit single-channel image. L'image sera traitée comme une image binaire.
- **contours** – contours détectés. Chaque contour est stocké comme un vecteur de points
- **hierarchy** – vecteurs de sorties optionnels, contenant des informations à propos de la topologie de l'image. Pour chaque i^{éme} contour `contours[i]` , si les éléments `hierarchy[i][0]` , `hiearchy[i][1]` , `hiearchy[i][2]` , et `hiearchy[i][3]` sont prépositionnés avec un indice 0 ce sont les contours précédents et suivants dans le même niveau hiérarchique. Si pour le contour d'indice I, il n'y a pas de précèdent et suivant l'élément correspondant de `hierarchy[i]` sera négative. pour un contour i, hierarchy[i][0] est le contours extérieurs, hierarchy[i][1] est le deuxième contours

intérieurs de l'objet et ainsi de suite. Si les objets n'ont pas de contours intérieurs, ce paramètre n'est pas utilisé.

- **mode** – de recherche de contours

- CV_RETR_LIST – Reconnait tous les contours et les mets dans une liste
- CV_RETR_EXTERNAL – Reconnait seulement tous les contours externes
- CV_RETR_CCOMP – Reconnaissance de tous les contours et les organisent dans une hiérarchie à deux niveaux
- CV_RETR_TREE – Reconnait tous les contours et les reconstruit dans une pleine hiérarchie

- **method** (méthode d'approximation des contours)
  - **CV_CHAIN_APPROX_NONE** stocke tous les points du contours
  - **CV_CHAIN_APPROX_SIMPLE** Compresse tous les segments diagonaux, verticaux, horizontaux, et laisse seulement les points finaux. Un contour rectangulaire est encodé avec 4 points
  - **CV_CHAIN_APPROX_TC89_L1,CV_CHAIN_APPROX_TC89_KCOS**

- **offset** – offset par lequel tous les points des contours seront décalés. Ceci est habituel quand nous avons une région d'intérêt dans l'image.

```
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
using namespace cv;
int main(int argc, char *argv[])
{
Mat ImgSource;
Mat ImgGris;
Mat ImgResultat;
vector<vector<Point> > Contours;
ImgSource = imread("chessboard.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);
threshold(ImgGris, ImgGris, 230, 255, CV_THRESH_BINARY_INV);
```

```
Canny(ImgGris, ImgResultat, 10,20);
findContours(ImgResultat, Contours, CV_RETR_LIST ,CV_CHAIN_APPROX_SIMPLE);
Mat ImgResultat2 = Mat::zeros( ImgResultat.size(), CV_8UC3 );
for( int i = 0; i< Contours.size(); i++ )
{
Scalar color = Scalar(0, 255, 255);
drawContours( ImgResultat2, Contours, i, color, 2, 8 );
}
namedWindow("Image Source", CV_WINDOW_AUTOSIZE);
namedWindow("Image Resultat", CV_WINDOW_AUTOSIZE);
imshow("Image Source", ImgSource);
imshow("Image Resultat", ImgResultat2);
waitKey(0);
```

void **drawContours**(InputOutputArray **image**, InputArrayOfArrays **contours**, int **contourIdx**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, InputArray **hierarchy**=noArray(), int **maxLevel**=INT_MAX, Point **offset**=Point())¶

Parametre de ≪ drawContours ≫ :

• Image qui recevra les résultats : ImgResultat

• Le vecteur de contour : Contours

• L'indice de l'objet contenu dans le vecteur de contours. Ainsi, la valeur 0, signifie Contours[0].

• La couleur utilisée pour dessiner le contour : color. Ce paramètre est de type ≪ Scalar ≫ qui correspond a un triplet qui représente la couleur (B,G,R).

• Les deux derniers paramètres correspondent a l'épaisseur du trait et au type de ligne respectivement. Leur valeur par défaut est de 1 pour l'épaisseur et de 8 pour le type de ligne.

5.3) isoler une couleur sous Opencv

Avant d'isoler la couleur on utilise la fonction cvtColor pour transformer le format BGR en un format HSV. La procédure ressemble à celle-ci :

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

Binarisation de l'image HSV

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 – tolerance <= **Hpixel** < H + tolerance) && (S - tolerance <= **Spixel** < S + tolerance)

Ou H : valeur de la teinte (Hue)
S : Valeur de la saturation (Saturation)

La méthode inrange

Cette function est composée de 3 arguments elle va permettre de détecter la couleur entre une limite basse et une limite haute

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

- **src** – image à traiter
- **lowerb** – scalaire qui définit le plus petit seuil.
- **upperb** – scalaire qui définit le plus grand seuil.

- **dst** – image de destination au format `CV_8U`.

Exemple d'utilisation dans le domaine HSV:

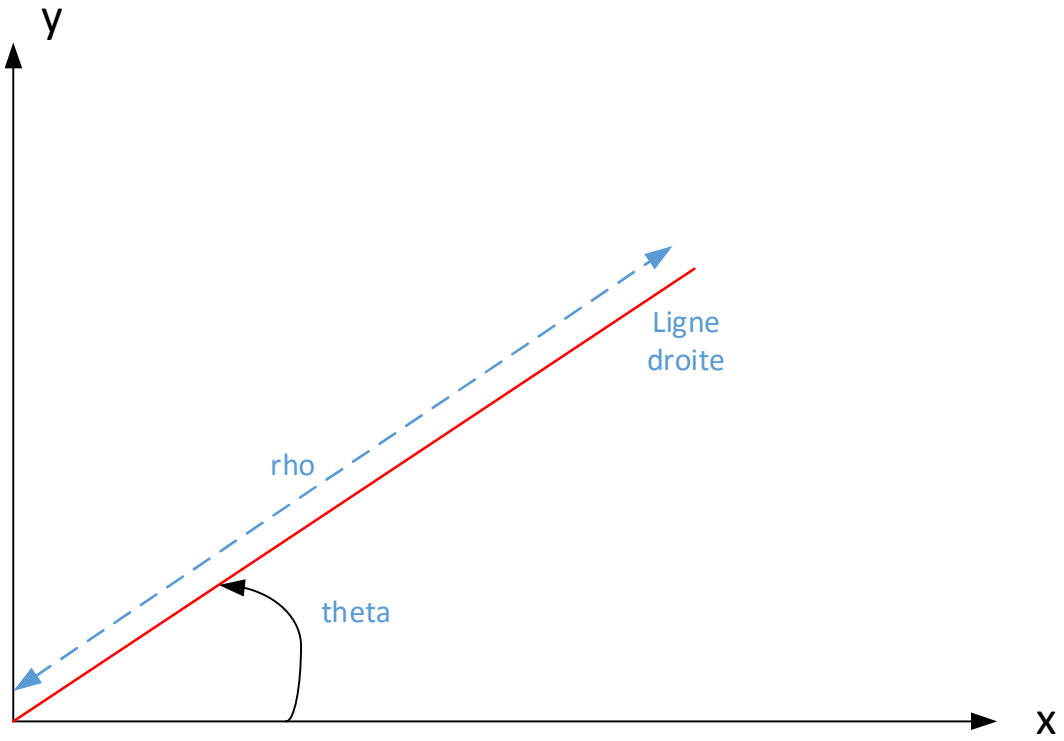
```
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
using namespace cv;

int main(){
   Mat src = imread("qq.jpg");
   if (src.empty())
      return -1;
   blur( src, src, Size(3,3) );
   Mat hsv;
   cvtColor(src, hsv, CV_BGR2HSV);
   Mat bw;
   inRange(hsv, Scalar(0, 10, 60), Scalar(20, 150, 255), bw);
   imshow("src", src);
   imshow("dst", bw);
   waitKey(0);
   return 0;
}
```

Exemple d'utilisation dans le domaine RGB:

```
int main()
{
   // Paint a blue square in image
   cv::Mat img = cv::Mat::zeros(100,100,CV_8UC3);
   cv::Scalar blue(255,0,0);

   img(cv::Rect(20,20,50,50)) = blue;
   cv::imshow("Original Image", img);


   // Detect this blue square
   cv::Mat img2;
   cv::inRange(img, blue, blue, img2);
   cv::imshow("Specific Colour", img2);

   cv::imwrite("Input.png",img);
   cv::imwrite("Output.png",img2);
   cv::waitKey(0);

   return 0;
}
```

5.4) Détection de lignes droites sous OPENCV

La transformation de Hough permet d'extraire des lignes droites d'une image binaire. Cette méthode est basée sur un accumulateur (matrice qui correspond à un domaine rectangulaire du plan), cet accumulateur permettra de détecter les lignes droites. Hough utilisé le paramétrage suivant des droites, en utilisant les coordonnées polaires :



Sous OPENCV la fonction HoughLines effectue la transformation de Hough. **rho et theta** sont les résolutions de l'accumulateur respectivement en pixel et en radian. **Threshold** est le seuil de détection des maxima dans l'accumulateur.

```
vector<Vec2f> lines;
HoughLines(dst, lines, rho, theta, threshold, 0, 0 );
```

- dst: Image à traiter en echelle de gris (de préférence binarisée)
- lines: un vecteur qui stockera les paramètres rho et theta des lignes détectés
- rho : la resolution du parametre rho en pixel
- theta: la resolution du parametre theta en pixel
- threshold: Le nombre minimum d'intersection pour détecter une ligne

Le résultat est renvoyé sous forme d'un tableau (vecteur de lignes) dont chaque ligne comporte une paire de paramètres *(ρ,θ)*.

Prenons l'exemple suivant :

```
cvtColor( image, gray, CV_BGR2GRAY );
equalizeHist(gray, gray );
Mat contours
Canny(gray,contours,50, 200, 3);

Mat contoursInv;
vector<cv::Vec2f> lines; // création d'un vecteur à deux composantes
HoughLines(contours,lines,1,3.14/180,150, 0, 0 );
cv::Mat result(contours.rows,contours.cols,CV_8U,cv::Scalar(255));
//cv::Mat result();
image.copyTo(result);
std::vector<cv::Vec2f>::const_iterator it= lines.begin();
while (it!=lines.end()) {
float rho= (*it)[0];   // le premier élément est la distance rho
float theta= (*it)[1]; // le premier élément est la distance theta
// affichage des lines quasi verticales
                if ((theta < (75*PII)) || (theta > (115*PII)))  // ~vertical line
                        if ((theta > (55*PII)) || (theta < (135*PII)))
                        {
                            Point pt1, pt2;
                            double a = cos(theta), b = sin(theta); //conversion polaire cartésien
                            double x0 = a*rho, y0 = b*rho;
                            pt1.x = cvRound(x0 + 1000*(-b));
                            pt1.y = cvRound(y0 + 1000*(a));
                            pt2.x = cvRound(x0 - 1000*(-b));
                            pt2.y = cvRound(y0 - 1000*(a));
                            printf ("pt1.x %d pt1.y %d theta %f \n",pt1.x,pt1.y, (theta*360/(2*PI)) );
                            printf ("pt2.x %d pt2.y %d \n",pt2.x,pt2.y );
                            line( image, pt1, pt2, Scalar(0,0,255), 3, CV_AA);


                }
```

5.5) La détection d'objets sous OPENCV

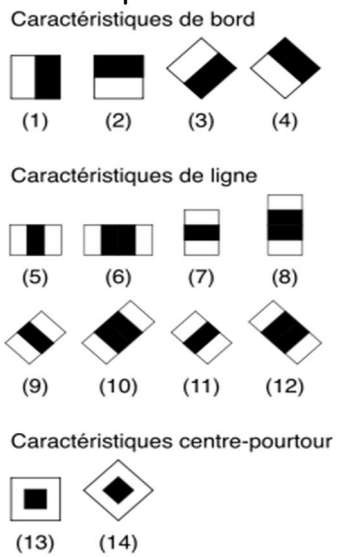
La méthode de Haar : détection d'objet

Les chercheurs Paul Viola et Michael Jones en 2001 ont proposés une méthode de détection d'objet dans une image numérique. Elle fait partie des toutes premières méthodes capables de détecter efficacement et en temps réel des objets dans une image. Inventée à l'origine pour détecter des visages, elle peut également être utilisée pour détecter d'autres types d'objets comme des voitures ou des avions. La méthode de Viola et Jones est l'une des méthodes les plus connues et les plus utilisées, en particulier pour la détection de visages et la détection de personnes. Une étape préliminaire et très importante est l'apprentissage du classificateur. Il s'agit d'entraîner le classificateur afin de le sensibiliser à ce que l'on veut détecter. Pour cela, il est mis dans deux situations.
La première où une énorme quantité d'images positives (avec l'objet à détecter) lui sont présentés et la deuxième où, à l'inverse, une énorme quantité de d'images négatives lui sont présentés.
Concrètement, une banque d'images contenant des objets à détecter est passée en revue afin d'entraîner le classificateur. Ensuite, une banque d'images ne contenant pas d'objets est passée.

5.5.1) L'apprentissage du classificateur

5.5.1.1) Caractéristiques d'une image

a) Extraction des caractéristiques d'une image
Viola et Jones proposent d'utiliser des caractéristiques, c'est à dire une représentation synthétique et informative, calculée à partir des valeurs des pixels. Viola et Jones définissent des caractéristiques très simples, les caractéristiques pseudo-Haar, qui sont calculées par la différence des sommes de pixels de deux ou plusieurs zones rectangulaires adjacentes.



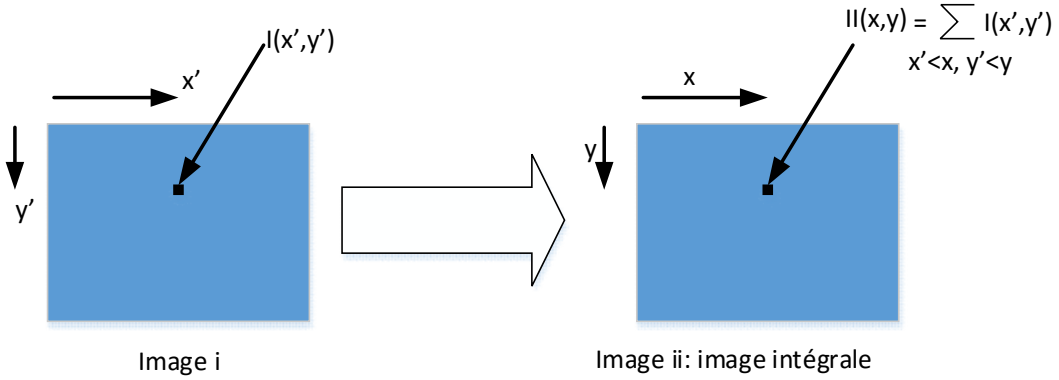
La figure ci-dessus donne des exemples des caractéristiques proposées par Viola et Jones à 2, 3 ou 4 rectangles, dans lesquelles la somme des luminances de pixels sombres est soustraite de la somme des luminances des pixels blancs.
Ces caractéristiques ont été choisies de façon à détecter des motifs. Par exemple, la reconnaissance des visages est rendue possible par:

- La variation de l'intensité de la lumière entre les yeux et le nez (caractéristique n°2)
- la variation de l'intensité de la lumière entre les yeux et les pommettes (caractéristique n°3)

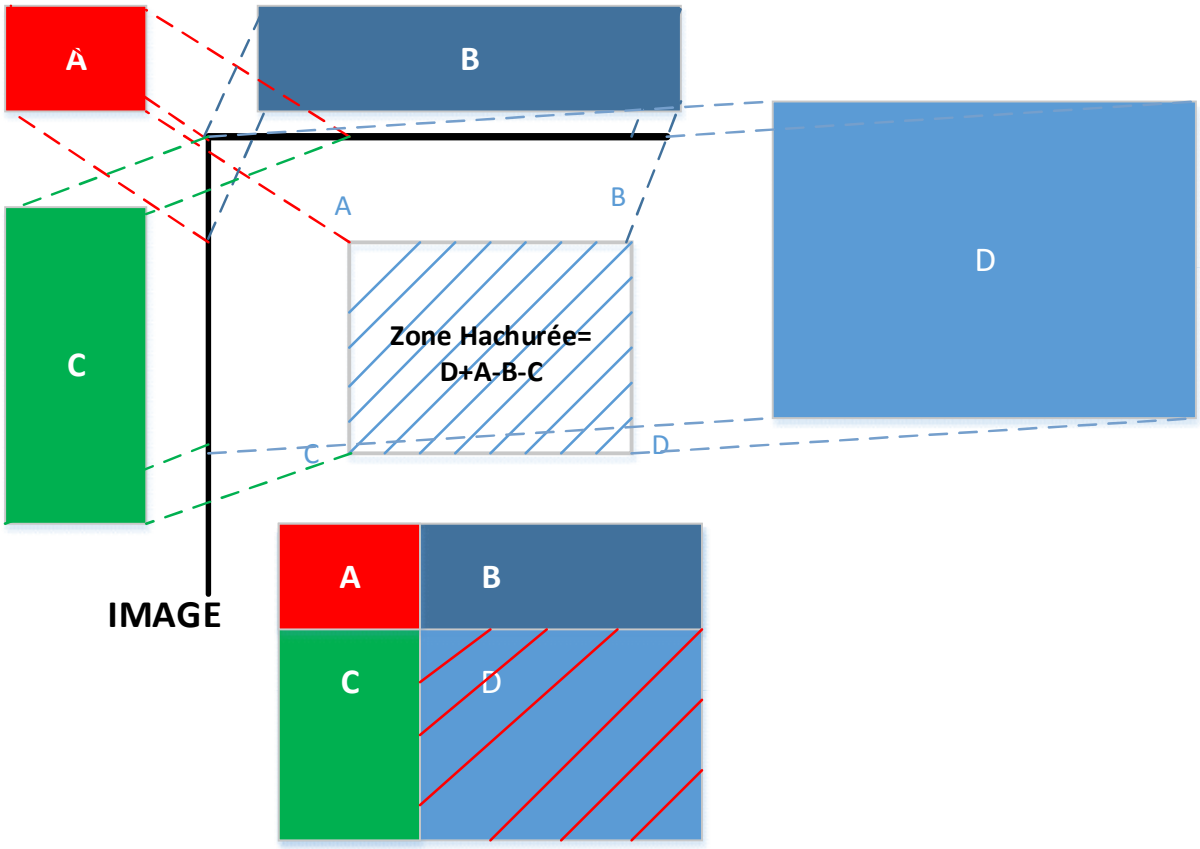
Les caractéristiques sont calculées en soustrayant la somme des pixels noirs à la somme des pixels blancs.

b) image intégrale

Pour calculer rapidement et efficacement ces caractéristiques sur une image, les auteurs proposent également une nouvelle méthode, qu'ils appellent « image intégrale ». C'est une représentation sous la forme d'une image, de même taille que l'image d'origine, qui en chacun de ses points contient la somme des luminances des pixels situés au-dessus de lui et à sa gauche.
Plus formellement, l'image intégrale ii au point (x,y) est définie à partir de l'image i par :



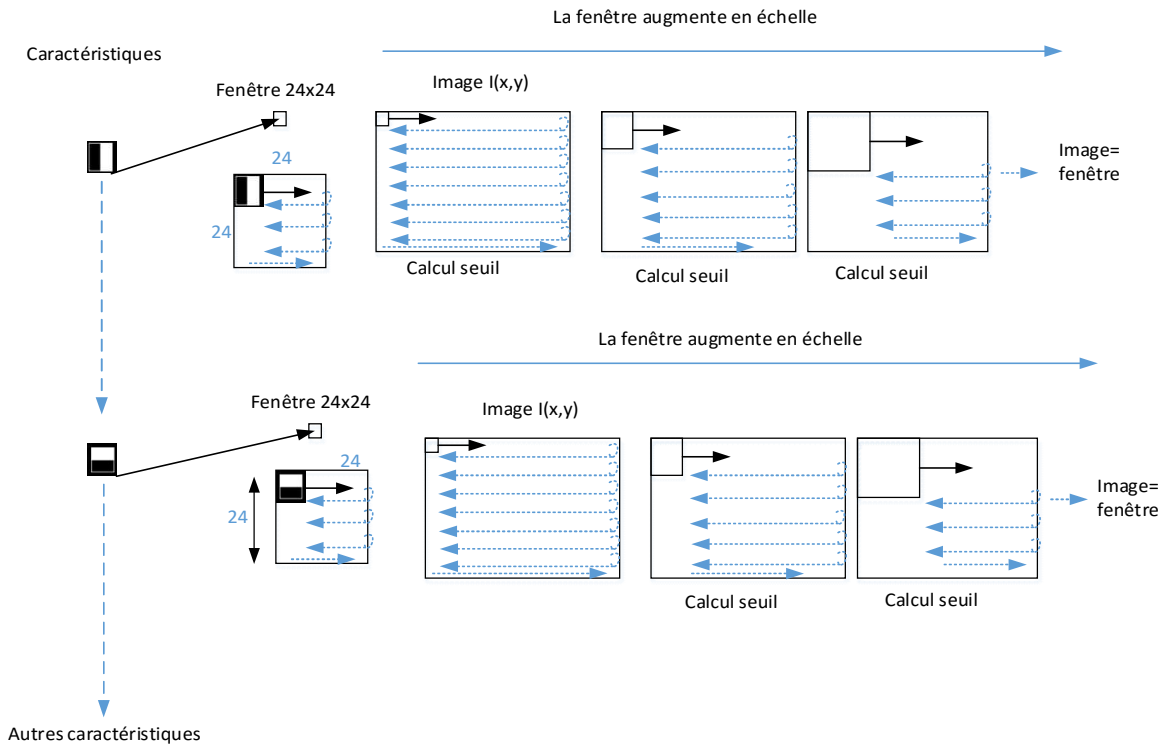
Illustrons le calcul de la somme de la luminance des pixels dans la zone hachurée ABCD ci-dessous



Seulement 4 accès à l'image intégrale nous permet de calculer la somme de la luminance de la zone ABCD (D+A-B-C)
Une caractéristique pseudo-Haar formée de deux zones rectangulaires peut être calculée en seulement 6 accès à l'image intégrale, et donc en un temps constant quelle que soit la taille de la caractéristique.

c) Calcul

Les caractéristiques sont calculées à toutes les positions et à toutes les échelles dans une fenêtre de détection de petite taille, typiquement de 24 × 24 pixels ou de 20 × 15 pixels. Un très grand nombre de caractéristiques par fenêtre est ainsi généré, Viola et Jones donnant l'exemple d'une fenêtre de taille 24 × 24 qui génère environ 160 000 caractéristiques pour une image.



La taille de la caractéristique de pseudo-Haar est variable et permet d'augmenter le nombre de caractéristiques évaluées

En phase de détection, l'ensemble de l'image est parcouru en déplaçant la fenêtre de détection (24x24) d'un certain pas dans le sens horizontal et vertical (ce pas valant 1 pixel dans l'algorithme original). Les changements d'échelles se font en modifiant successivement la taille de la fenêtre de détection. Viola et Jones utilisent un facteur multiplicatif de 1,25, jusqu'à ce que la fenêtre couvre la totalité de l'image. Finalement, et afin d'être plus robuste aux variations d'illumination, les fenêtres sont normalisées par la variance des pixels de la fenêtre.

d) Elaboration du classificateur

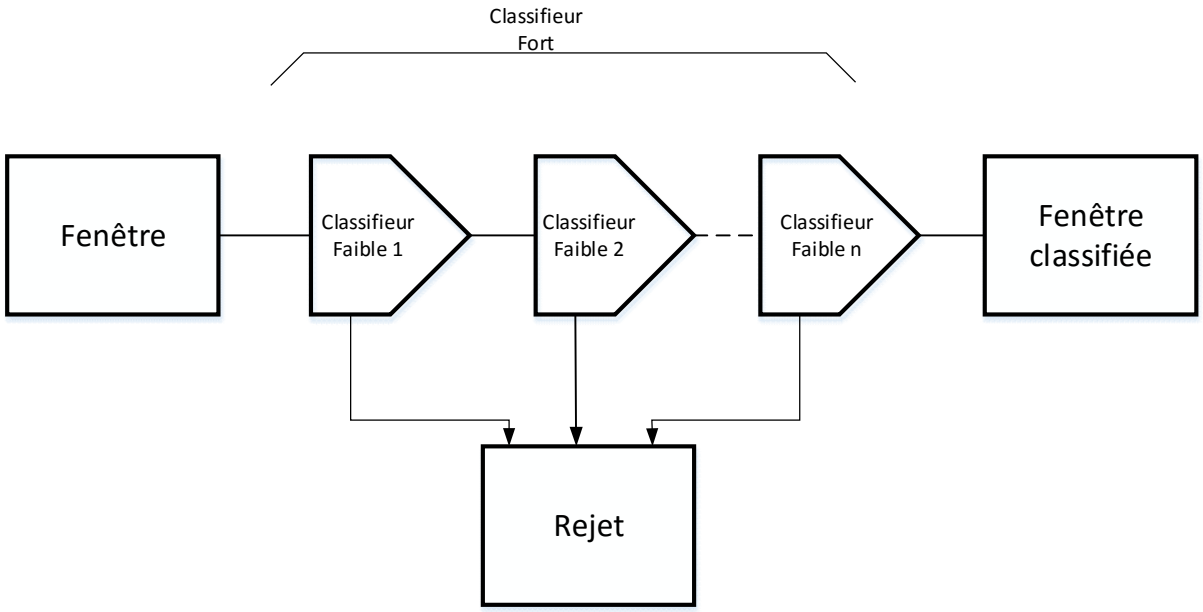
Sélection de caractéristiques par boosting

Le deuxième élément clé de la méthode de Viola et Jones est l'utilisation d'une méthode de boosting afin de sélectionner les meilleures caractéristiques (sur les 14 disponibles). Le boosting est un principe qui consiste à construire un classificateur « fort » à partir d'une combinaison pondérée de classificateur « faibles », c'est-à-dire donnant en moyenne une réponse meilleure qu'un tirage aléatoire. Viola et Jones adaptent ce principe **en assimilant une caractéristique** à un classificateur faible, en construisant un classificateur faible qui n'utilise qu'une seule caractéristique.

Après avoir calculé toutes les caractéristiques provenant de chacune des formes de la fonction de Haar, c'est le temps de définir un seuil selon lequel on décide si l'image en traitement est un visage ou non. Chaque paramètre passe sur toutes les images intégrales calculées à partir des images originales de la base de données adoptée. Ainsi, ces seuils seront enregistrés dans un fichier qui contiendra les poids de tous les classificateurs de Haar.

Une fois le calcul de la fonction Haar a été complété pour les images positives, la moyenne, l'écart-type, le min et le max sont calculés pour toutes les valeurs données par chaque image. Ces paramètres sont calculés afin de catégoriser un classificateur comme bon ou mauvais par une distribution gaussienne : on commence à partir de la valeur moyenne et on élargit selon l'écart type, plus l'écart-type est grand, plus la reconnaissance sera pour les visages.
On doit faire le même calcul pour les images sans objets.
Pour tester les classificateur obtenus et décider quels sont les forts classificateur qu'on va utiliser ensuite dans la chaîne en cascade, chaque classificateur, ayant une distribution gaussienne, doit vérifier 3 conditions :
- Obtenir une reconnaissance d'objets de près de 100% (presque toutes les images d'objets à détecter sont détectées).
- Obtenir une reconnaissance sans objets très faible (la plupart des images sans objets ne sont pas détectées).
- Erreur totale pour le classificateur inférieure à 50%.

Si et seulement si ces trois conditions sont réunies, on continue avec le classificateur en traitement.



L'apprentissage du classificateur faible consiste alors à trouver la valeur du seuil de la caractéristique qui permet de mieux séparer les exemples positifs (image dans l'objet est présent) des exemples négatifs (image dont l'objet est absent).
Le classificateur se réduit alors à un couple (caractéristique, seuil). Au final, un classificateur est une association entre une caractéristiques pseudoHaar et un seuil. C'est ce qu'on appelle un classificateur faible.
Un classifieur h(x), composé d'un descripteur *f*, d'un seuil θ, et d'une parité *p*, donne une prédiction sur la classe à qui appartient *x* (ici, 1 pour visage et 0 pour non visage).

$$h(x, f, p, \theta) = \begin{cases} 1 \; si \; p.f(x) < p.\theta \\ 0 \; dans \; les \; autres \; cas \end{cases}$$

Un classificateur faible ne suffit pas à classifier une image mais c'est l'association de classificateur faible qui va permettre d'obtenir un classificateur fort.
Le nombre de classificateur étant trop élevé pour le calcul lors de la détection, l'algorithme de boosting va permettre de réduire le nombre de classificateur faible.

L'algorithme de boosting utilisé est en pratique une version modifiée d' AdaBoost, qui est utilisée à la fois pour la sélection et pour l'apprentissage d'un classificateur « fort ». Les classificateurs faibles utilisés sont souvent des arbres de décision. Un cas remarquable, fréquemment rencontré, est celui de l'arbre de profondeur 1, qui réduit l'opération de classification à un simple seuillage.
L'algorithme est de type itératif, à nombre d'itérations déterminé. À chaque itération, l'algorithme sélectionne une caractéristique, qui sera ajoutée à la liste des caractéristiques sélectionnées aux itérations précédentes, et le tout va contribuer à la construction du classificateur fort final. Cette sélection se fait en entraînant un classificateur faible pour toutes les caractéristiques et en élisant celle de ces dernières qui génère l'erreur la plus faible sur tout l'ensemble d'apprentissage. L'algorithme tient également à jour une distribution de probabilité sur l'ensemble d'apprentissage, réévaluée à chaque itération en fonction des résultats de classification. En particulier, plus de poids est attribué aux exemples difficiles à classer, c'est à dire ceux dont l'erreur est élevée. Le classificateur « fort » final construit par AdaBoost est composé de la somme pondérée des classificateurs sélectionnés. Les classifieurs de précision plus élevée (taux d'erreur entre 0.1 et 0.3) sont sélectionnés au début de l'apprentissage, et les classifieurs moins précis (taux d'erreur entre 0.4 et 0.5) sont sélectionnés dans les dernières itérations.
L'apprentissage est réalisé sur un très large ensemble d'images positives (c'est-à-dire contenant l'objet) et négatives (ne contenant pas l'objet). Plusieurs milliers d'exemples sont en général nécessaires.
Après la phase d'apprentissage du classificateur, les différents seuils pour chaque classificateur faible sont enregistrés dans un fichier de métadonnées xml. Ce fichier sera exploité lors de la phase de détection.

5.5.1.2) La Détection

La détection s'applique sur une image de test, dans laquelle on souhaite déceler la présence et la localisation d'un objet. En voici les étapes :

- parcours de l'ensemble de l'image à toutes les positions, avec une fenêtre de taille 24 × 24 pixels et variable, application de la cascade à chaque sous-fenêtre, en commençant par le premier étage :

- calcul des caractéristiques pseudo-Haar de l'image utilisée par le classificateur de l'étage courant,
- calcul de la réponse du classificateur,
- passage ensuite à l'étage supérieur si la réponse est positive, à la sous-fenêtre suivante sinon, et enfin l'exemple est déclaré positif si tous les étages répondent positivement ;
- fusion des détections multiples : l'objet peut en effet générer plusieurs détections, à différentes positions et échelles ; cette dernière étape fusionne les détections qui se chevauchent pour ne retourner qu'un seul résultat.

5.5.1.3) La détection sur opencv en utilisant le format Iplimage

Pour commencer, nous allons mettre en place les premières briques du programme. On commence par inclure les fichiers nécessaires au programme.

```
#include <opencv/cv.h>
#include <opencv/highgui.h>
 #include <stdio.h>
```

Ensuite on va déclarer deux de manière globale deux variables, le premier est le classificateur et le second une zone mémoire utilisée comme « buffer » pour la détection des visages. Le fait de les déclarer de manière globale n'est pas forcément idéal, mais le but est ici d'avoir le code le plus simple possible. Pour une application plus importante, il est recommandé de déclarer ces deux variables dans la partie nécessitant la détection de visage.

```
CvHaarClassifierCascade *cascade;
CvMemStorage *storage;
int key;
```

On rentre maintenant dans la fonction main, et nous allons déclarer tous les objets nécessaires par la suite. Nous avons besoin de peu d'élément, le premier sera une image (IplImage), le second sera le périphérique d'entrée de la caméra (CvCapture) et le dernier sera un int nous permettant de sortir de la boucle de traitement. Ce qui nous donne:

```
CvCapture *capture; IplImage *frame; int key;
```

Il faut ensuite initialiser tout ce beau monde. On commence par ouvrir le fichier du classificateur Haar

```
cascade = ( CvHaarClassifierCascade* )cvLoad(
« haarcascade_frontalface_alt.xml », 0, 0, 0 );
```

On ouvre le flux caméra avec ***cvCreateCameraCapture***

```
capture = cvCreateCameraCapture(CV_CAP_ANY)
```

Enfin on initialise l'espace mémoire

```
storage = cvCreateMemStorage( 0 );
```

Afin d'afficher tout cela, nous aurons besoin d'une fenêtre

```
cvNamedWindow( « Window-FT », 1 );
```

Nous sommes en C++, il ne faut donc pas oublier de libérer la mémoire avant la fin du programme, on doit alors effacer tous les objets précédemment créé.

```
cvReleaseCapture( &capture );
cvDestroyWindow( « Window-FT » );
```

```
 cvReleaseHaarClassifierCascade( &cascade );
cvReleaseMemStorage( &storage );
```

Nous avons donc ici la base du programme. Si vous compilez vous ne devriez pas avoir grand-chose qui se passe et c'est normal, il manque l'essentiel, la boucle de traitement d'image ainsi que la fonction permettant l'affichage du tracking facial. C'est ce que nous allons voir dès maintenant.

La boucle de traitement

Cette étape est plutôt simple, mais elle est très importante, le principe est d'effectuer, à chaque image envoyée par la caméra au programme, la détection de visage. On en profite pour rajouter une petite commande permettant de quitter la boucle (et donc le programme) en appuyant sur la touche « q ». Voici la boucle en question

```
while( key != 'q' )
{
img= cvQueryFrame( capture ); detectFaces( img ); key = cvWaitKey( 10 );
}
```

Pas grand-chose à expliquer ici, une boucle while classique, on déclare une frame (image) qui correspondra à chaque instant à l'image envoyée par la caméra. La fonction ***detectFaces*** est la fonction que nous allons étudier juste après qui permet de faire la détection de visage à proprement parler. Si vous voulez tester votre caméra, il vous suffit de remplacer ***detectFaces(img);*** par ***cvShowImage(« Window-FT », img);***si vous compilez, vous devriez voir en temps réel ce que votre caméra est en train de filmer.

La fonction detectFaces

Il ne reste plus qu'à créer la fonction detectFaces afin de permettre le face tracking. Voici la fonction

```
void detectFaces( IplImage *img ){
int i;
CvSeq *faces = cvHaarDetectObjects(img, cascade, storage, 1.1, 3 , 0,  cvSize( 40, 40 )
);
for( i = 0 ; i < ( faces ? faces->total : 0 ) ; i++ )
{
CvRect *r = ( CvRect* )cvGetSeqElem( faces, i );
cvRectangle( img, cvPoint( r->x, r->y ), cvPoint( r->x + r->width, r->y + r->height
), CV_RGB( 255, 0, 0 ), 1 , 8, 0 );
}
cvShowImage( « Window-FT », img );
```

```
}
public static IntPtr cvHaarDetectObjects(
        IntPtr image,
        IntPtr cascade,
        IntPtr storage,
        double scaleFactor,
        int minNeighbors,
        int flags,
        MCvSize minSize
)
```

La fonction récupère en paramètre l'image de la caméra récupérée pendant la boucle de traitement. C'est sur cette image que toutes les opérations vont être effectuées par la suite. C'est là où vous allez apprécier OpenCV, car pour la détection de visage, la méthode de Viola et Jones est déjà implémentée et deviens très facile à utiliser. Tout réside dans la fonction ***cvHaarDetectObjects***.

Le résultat de cette fonction est une série d'objets qui ont passé les critères de sélection définis par votre classificateur. On définit donc une CvSeq qui correspond à une séquence d'objet d'un même type, dans notre cas ce sera nos différents visages détectés.

Au niveau des paramètres de cette fonction ***img*** représente l'image a traité, ***cascade*** est le classificateur choisi pour faire le test, ***storage*** est l'espace mémoire nécessaire pour effectuer l'opération, ***1.1*** représente le « scale factor » l'augmentation de la fenètre (24x24) pour chaque analyse , minNeighbors=***3*** représente le nombre de rectangle voisins qui seront pris en compte pour la détection de l'image , *plus ce nombre sera grand moins le nombre d'objet détecté sera grand mais la détection sera d'une très grande fiabilité*

0 est un paramètre supplémentaire qui permet de rajouter des filtres particuliers par exemple un filtre de canny avec CV_HAAR_DO_CANNY_PRUNING, ***cvSize(40, 40)*** représente la taille minimale d'un objet dans la vidéo.

Nous avons maintenant l'ensemble des visages qui ont été détectés dans l'image dans notre CvSeq, et afin d'afficher le résultat à l'écran nous allons simplement dessiner un carré autour du visage détecté. Ensuite on rentre dans une boucle for qui va passer en revue tous les visages détectés, et pour chacun créer un rectangle autour du visage repéré.

Je ne détaille pas le dessin du rectangle, Enfin la dernière étape est d'afficher à chaque appel de fonction l'image de la caméra avec les carrés permettant de détecter les visages, grâce à la fonction ***cvShowImage(« Window-FT », img);***

Pour cette fonction le « ***Window-FT*** » est le nom de la fenêtre que vous avez créée et le ***img*** représente l'image à afficher dans la fenêtre.

5.5.1.4) Détection de Haar en utilisant l'objet Mat

On peut détecter un objet par la méthode de Viola Jones en utilisant l'objet Mat.
Il nous faut déclarer un classificateur de la classe CascadeClassifier et le charger

CascadeClassifier::CascadeClassifier(const string& filename) // méthode constructeur charge un classificateur

```
CascadeClassifier face_cascade;
face_cascade.load( "C:/OpenCV243/data/Haarcascades/haarcascade_frontalface_alt2.xml" );
```

void CascadeClassifier::**detectMultiScale**(const Mat& image, vector<Rect>& objects, double scaleFactor=1.1, int minNeighbors=3, int flags=0, Size minSize=Size(), Size maxSize=Size())

// Détecte les objets de différentes tailles dans l'image.
// Les objets détectés sont retournés comme une liste de rectangle

Les paramètres de la fonction ::**detectMultiScale**

- **cascade** – Classificateur fichier xml
- **image** – Objet du type Mat format CV_8U contenant l'image avec les objets détectés
- **objects** – Vecteur d'objet rectangle ou chaque rectangle contient l'objet détecté sur l'image.
- **scaleFactor** voir chapitre précédent.
- **minNeighbors** voir chapitre precedent
- **flags** – voir chapitre précédent.
- **minSize** – dimension minimum possible pour l'objet.
- **maxSize** – imension maximum possible pour l'objet

Exemple d'utilisation

```
#include "opencv2/objdetect/objdetect.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>
#include <stdio.h>
```

```
using namespace std;
using namespace cv;

int main( )
{
   Mat image;
   image = imread("lena.jpg", CV_LOAD_IMAGE_COLOR);
   namedWindow( "window1", 1 );   imshow( "window1", image );

   // Load Face cascade (.xml file)
   CascadeClassifier face_cascade;
   face_cascade.load(
"C:/OpenCV243/data/Haarcascades/haarcascade_frontalface_alt2.xml" );

   // Detect faces
   std::vector<Rect> faces;
   face_cascade.detectMultiScale( image, faces, 1.1, 2, 0|CV_HAAR_SCALE_IMAGE,
Size(30, 30) );

   // Draw circles on the detected faces
   for( int i = 0; i < faces.size(); i++ )
   {
      Point center( faces[i].x + faces[i].width*0.5, faces[i].y + faces[i].height*0.5 );
      ellipse( image, center, Size( faces[i].width*0.5, faces[i].height*0.5), 0, 0, 360, Scalar(
255, 0, 255 ), 4, 8, 0 );
   }

   imshow( "Detected Face", image );

   waitKey(0);
   return 0;
}
```

Annexe :

Compilation d'un applicatif sous Opencv et raspberry pi2

Pour lancer une compilation sous opencv deux méthodes :

Méthode 1

Créez un fichier facereco.pro dans votre projet

```
#######################################################################
# Automatically generated by qmake (3.0) jeu. janv. 1 16:50:56 1970
#######################################################################

TEMPLATE = app
TARGET = facereco
INCLUDEPATH += /home/pi/opencv-2.3.1/include/opencv
INCLUDEPATH += /home/pi/opencv-2.3.1/include
INCLUDEPATH += /home/pi/facereco

DEPENDPATH += .
LIBS +=-L/home/pi/opencv-2.3.1/release/lib \
-lopencv_core \
-lopencv_imgproc \
-lopencv_highgui \
-lopencv_ml \
-lopencv_video \
-lopencv_features2d \
-lopencv_calib3d \
-lopencv_objdetect \
-lopencv_contrib \
-lopencv_legacy \
-lopencv_flann \


# Input
SOURCES += facereco.cpp
```

Si vous voulez créer le fichier .pro automatiquement
Lancez qmake –project (comme au dessus il faudra rajouter les librairies)
Lancez qmake
Le fichier Makefile est alors généré puis pour compiler
make

Méthode 2

Créez un fichier CMakeList dans votre projet

```
cmake_minimum_required(VERSION 2.8)
project( facereco)
find_package( OpenCV REQUIRED )
#include_directories(${PNG_INCLUDE_DIR})
add_executable( facereco facereco.cpp )
link_directories( /home/pi/bytefish-libfacerec-e1b143d )
target_link_libraries( facereco /home/pi/bytefish-libfacerec-
e1b143d/libopencv_facerec.a ${OpenCV_LIBS} )
```

Lancez cmake .

make