Imagerie numérique - Introduction à OpenCV
Feuille de TD N^{o}1
ESIEA

L. Beaudoin

1 Introduction à OpenCV

1.1 Une vision globale

1.1.1 Ressources disponibles

OpenCV est une bibliothèque dédiée au traitement d'images (CV pour Computer Vision). Cette bibliothèque est sous licence BSD. Les sites de références sont :

- `http://opencv.willowgarage.com/wiki/`
  Le site officiel, qui héberge notamment la dernière version stable téléchargeable, le wiki officiel, la doc...
- `http://opencv.willowgarage.com/documentation/c/index.html`
  Le lien de la documentation en ligne : à référencer tout de suite, vous allez beaucoup vous en servir !

OpenCV existe en plusieurs langages et sous plusieurs OS. Dans le cadre de ce cours, nous programmerons en C et sous linux. OpenCV a aussi son ouvrage de référence : **"Learning Opencv", G. Bradsky and A. Kaehler, O'Reilly**. Plusieurs exemplaires sont disponibles au centre de documentation.

1.1.2 OpenCV ? Pourquoi faire ?

OpenCV est né en 1999 de la volonté de la société d'Intel de développer des applications avancée en traitement d'image. Un des porteurs du projet s'est aussi apperçu que les universités développaient chacune dans leur coin des bibliothèques à pérénité et efficacité variables. Une idée s'est imposé alors : créer une bibliothèque suffisamment complète, optimisée et compatible avec les contraintes du monde industriel et universitaire (d'où le choix de la licence BSD) à vocation universelle. Les objectifs d'OpenCV sont en résumé de :

- promouvoir le traitement d'image,
- aider au développement d'algorithmes de vision en utilisant du code ouvert, optimisé et supporté par une large communauté.

La liste des applications potentielles est très large et aujourd'hui de très nombreux projets utilisent cette librairie. Parmis ceux-ci, il y a la reconnaissance d'objet, le tracking d'objets ou de personnes en 2D ou 3D, la reconnaissance de visage, la vidéo-surveillance, la robotique (lauréat du Grand Darpa Challenge en 2006)...

1.1.3 L'architecture de la librairie

Les briques principales d'OpenCV sont :

- `cxcore` qui contient les structures et algorithmes basiques. Toutes les autres briques s'appuient au moins sur celle-ci,
- `cv` dédiée au algorithmes de traitement d'image et de vision,
- `HighGUI` dédiée à la réalisation des interfaces Homme-Machine et la gestion des flux entrée-sortie image et vidéo,
- `MLL` (Machine Learning Library) dédiée aux applications d'apprentissage.

Comme Intel est depuis le départ très fortement dans le projet, elle propose d'augmenter les performances de base d'OpenCV en y intégrant sa propre bibliothèque, Intel Performance Primitive (IPP), dédiée aux processeurs de la famille Intel.

1.2 Premiers pas et exemples

Dans cette section, on vous présente des exemples de codes pour charger et afficher une image, une vidéo, un flux vidéo d'une webcam. Tous ces codes se compilent par la commande :
`gcc -Wall toto.c ‘pkg-config --cflags --libs opencv‘`
(‘ est la combinaison alt-gr 7 et `toto.c` le fichier a compiler).

1.2.1 Chargement et affichage d'une image

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

int main(int argc, char** argv)
{
   IplImage* img=cvLoadImage(argv[1], CV_LOAD_IMAGE_COLOR);
   if (img==NULL){
      printf("Caramba, pas vu pas pris!\n");
      return(1);
      }
   cvNamedWindow("Exemple image", CV_WINDOW_AUTOSIZE);
   cvShowImage("Exemple image",img);
   cvWaitKey(0);
   cvReleaseImage(&img);
   cvDestroyWindow("Exemple image");
   return(0);
}
```

1.2.2 Chargement et affichage d'une vidéo

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

int main (int argc, char **argv)
{
  CvCapture* capture=cvCaptureFromFile(argv[1]);
  IplImage * frame;
  char c;
  cvNamedWindow("Exemple video", CV_WINDOW_AUTOSIZE);
  while(1){
    frame = cvQueryFrame(capture);
    if(frame==NULL){
      break;
      printf("Caramba, pas vu pas pris!\n");
    }
    cvShowImage("Exemple video", frame);
    c=cvWaitKey(33);/*attente de 33ms*/
    if(c==27) break;/* touche esc*/
  }
  cvReleaseCapture(&capture);
  cvDestroyWindow("Exemple video");
  return(0);
}
```

1.2.3 Affichage d'un flux vidéo d'une webcam

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

int main()
{
  char c;
  IplImage *frame;
  CvCapture* capture = cvCaptureFromCAM(0);
  cvNamedWindow("Exemple webcam", CV_WINDOW_AUTOSIZE);
  while(1){
    frame = cvQueryFrame(capture);
    if (frame == NULL){
      printf("Caramba, pas vu pas pris!\n");
      break;
    }
    cvShowImage("Exemple webcam",frame);
    c= cvWaitKey(33);
    if (c == 27) break;
  }
  cvReleaseCapture(&capture);
  cvDestroyWindow("Exemple webcam");
  return(0);
}
```

2 HighGUI - Les fonctions de base

2.1 Gestion des images

2.1.1 IplImage

Bien qu'écrit en `C`, les objets OpenCV sont souvent liés entre eux par des notions hiérarchiques. Par exemple, le type `IplImage`, qui correspond à une image (1 (gris) à 4 (couleurs + transparence) plans ;chaque plan codé de 8 à 32 bits etc...) hérite du type `CvMat` (pour matrice) qui lui-même hérite du type `CvArr` (array). En pratique, cela signifie qu'une fonction dont un paramètre est de type `CvArr` par exemple tolèrera aussi un paramètre de type `CvMat` ou `IplImage`. Les autres pimitives (i.e. les types fondamentaux) d'OpenCv peuvent être trouvés dans le fichier `cxtypes.h` du répertoire `OpenCV/cxcore/include`. Pour trouver où sur l'ordinateur est installé le répertoire `OpenCV`, il faut utiliser la commande `pkg-config --cflags opencv`. Le tableau suivant résume quelques primitives.

structure	contient	représente
`CvPoint`	`int x, y`	un point dans une image
`CvSize`	`int width, height`	la taille d'une image
`CvRect`	`int x, y, width, height`	un extrait rectangulaire d'une image
`CvScalar`	`double val[4]`	un tableau des valeurs colorimétriques d'un point

Les principaux composants de la structure `IplImage` sont :

```
typedef struct _IplImage
{
    ...
    int  nChannels;
    ...
    int  depth;
    ...
    int  width;
    int  height;
    ...
```

```
    char *imageData;
    int  widthStep;
    ...
}
IplImage;
```

où :

- `nChannels` est le nombre de cannaux (1 pour le gris, 3 pour une image couleurs, 4 si on y ajoute la transparence) de l'image,
- `depth` est la profondeur en bits pour chaque canal. Les valeurs possibles sont :

macro	type du pixel
IPL_DEPTH_8U	unsigned char
IPL_DEPTH_8S	char
IPL_DEPTH_16S	short int
IPL_DEPTH_32S	long int
IPL_DEPTH_32F	float
IPL_DEPTH_64F	double

- `width` est la largeur de l'image en pixels,
- `height` est la hauteur de l'image en pixels,
- `imageData` est le pointeur sur le début du tableau des données de l'image. ATTENTION : pensez à retyper les données si celles-ci ne sont pas de type `char`!,
- `widthStep` est le nombre d'octets en mémoire rééellement pris par une ligne. Pour des raisons d'optimisation, cette taille peut être différente de `width * depth * nChannels`.

Les autres paramètres de la structure `IplImage` sont décrits dans la documentation en ligne.

2.1.2 Les principales fonctions

- `IplImage* cvLoadImage(const char* filename, CV_LOAD_IMAGE_COLOR)`
  charge le fichier image `filename` dans une structure `IplImage*`. Les formats standarts de fichiers images (BMP, JPEG etc...) sont supportées nativement. D'autres options sont possibles autres que `CV_LOAD_IMAGE_COLOR` qui force l'image a être chargée en mémoire en 24 bits (3 canaux couleur-voir la documentation en ligne). ATTENTION : par défaut, les images sont chargées dans le format BGR (Blue Green Red),
- `void cvReleaseImage(IplImage** image)`
  permet de désallouer la place mémoire occupée par l'`IplImage*` nommée `image`. Attention, c'est bien le pointeur sur cette structure qu'il faut passer. Comme précédemment, un bon réflexe est d'écrire le `cvReleaseImage` dès que vous écrivez le `cvLoadImage`,
- `int cvSaveImage(const char* filename, const CvArr* image)`
  permet de sauver l'image `image` dans le fichier `filename` dont l'extension définiera le format de sortie souhaité (BMP, JG etc...).
- `IplImage* cvCreateImage(CvSize size, int depth, int channels)`
  permet la création d'une `IplImage` et d'allouer la place mémoire afférente.
  Par exemple, `cvCreateImage(cvSize(200,300),IPL_DEPTH_8U,3)` retourne un pointeur sur une `IplImage` de 200 colonnes et 300 lignes et composée de 3 plan images chacun codé en `unsigned char`.
- `IplImage* cvCloneImage(const IplImage* image)`
  fait une copie complète (et indépendante) de `image`, i.e. crée une nouvelle `IplImage`, alloue la place mémoire identique à celle de `image` et copie les valeurs des pixels de `image`.
- `void cvCopy(const CvArr* src, CvArr* dst, const CvArr* mask=NULL)`
  permet de copier les pixels de l'image `src` dans l'image `dst` (qui doit donc être déjà allouée). si `mask` est non `NULL`, permet de limiter la copie aux pixels non nuls de `mask`.

2.2 Gestion des fenêtres

- `int cvNamedWindow(const char* name, CV_WINDOW_AUTOSIZE)`
  permet de créer une fenêtre nommée `name` et dont le `flag` `CV_WINDOW_AUTOSIZE` taille l'image par défaut sur la taille de l'image qui sera affichée dans la fenêtre,

- `void cvDestroyWindow(const char* name)`
  permet de libérer la place mémoire allouée par la fenêtre `name`. Un bon réflexe est de mettre un `cvDestroyWindow` dès que vous utilisez `cvNamedWindow`,
- `void cvShowImage(const char* name, const CvArr* image)`
  permet d'afficher l'image `image` préalablement chargée dans la fenêtre `name`.

2.3 Gestion de la vidéo

- `CvCapture`
  est la primitive permettant de travailler sur un flux vidéo, que celui-ci provienne d'une caméra ou d'un fichier. La structure est privée, vous n'avez donc pas à vous soucier de ce quelle contient,
- `CvCapture* cvCaptureFromFile(const char* filename)`
  permet de capturer un flux vidéo à partir du fichier `filename`,
- `CvCapture* cvCaptureFromCAM(int index)`
  permet de capturer un flux vidéo à partir d'une webcam. Si plusieurs caméras sont branchées sur l'ordinateur, `index` permet de choisir parmis elles,
- `IplImage* cvQueryFrame(CvCapture* capture)`
  permet de capturer l'image en cours du flux `capture`. ATTENTION : vous ne devez ni désallouer ni modifier l'image retournée. Si vous avez un tel besoin, il faut alors copier l'image renvoyée pour la modifier par exemple,
- `void cvReleaseCapture(CvCapture** capture)`
  permet de désallouer proprement la mémoire allouée par `CvCapture`

2.4 Exercices

2.4.1

Afficher simultanément une image couleur et sa conversion en niveau de gris. Pour cela, on utilisera la fonction `cvCvtColor` et l'aide en ligne bien entendu.

2.4.2

Afficher une vidéo couleur et simultanément sa conversion en niveau de gris en utilisant ce que vous avez fait à l'exercice précédent.

3 Un peu de dessin

3.1 Primitives de dessins et de texte

Les principales primitives de dessin sont :

- `void cvLine(CvArr* img, CvPoint pt1, CvPoint pt2, CvScalar color, int thickness=1, int lineType=8, int shift=0)`
  qui permet de tracer un segment dans l'image `img` entre les points `pt1` et `pt2`. `color` correspond à la couleur du tracé. Pour limiter les risques d'erreur, il est fortement recommandé d'utiliser la macro `CV_RGB(r,g,b)` où `r`, `g` et `b` sont les valeurs respectivement du canal rouge, vert et bleu. Par exemple, `CV_RGB(255,0,0)` donne la couleur rouge vif. `thickness` est la largeur en pixels du trait et `lineType` est le nombre de plus proches voisins autorisés (8 a un rendu plus lisse que 4) ?.
- `void cvRectangle(CvArr* img, CvPoint pt1, CvPoint pt2, CvScalar color, int thickness=1, int lineType=8, int shift=0)`
  permet de tracer un rectangle dans l'image `img` de sommets opposés `pt1` et `pt2`.
- `void cvCircle(CvArr* img, CvPoint center, int radius, CvScalar color, int thickness=1, int lineType=8, int shift=0)`
  permet de tracer un cercle de centre `center` et de rayon en pixels `radius`.

Pour afficher du texte, les principales primitives sont :

- `void cvInitFont(CvFont* font, int fontFace, double hscale, double vscale, double shear=0, int thickness=1, int lineType=8)`

qui va initialiser de manière adéquate la fonte `font` du texte que vous allez afficher dans l'image. `fontFace` représente la police à utiliser. La documentation donne la liste des possibilités. `CV_FONT_HERSHEY_COMPLEX` ou `CV_FONT_HERSHEY_COMPLEX_SMALL` sont une bonne option. `hscale` et `vscale` sont les facteurs d'échelle (1 ou 0.5 sont seulement permis) horizontal et verticale respectivement. `shear` représente la pente des caractères par rapport à la verticale. Si `shear` vaut 0, les caractère seront verticaux. Pour 1, ils seront en italique.
- `void cvPutText(CvArr* img, const char* text, CvPoint org, const CvFont* font, CvScalar color)` permet d'afficher dans l'image `img` la chaîne de caractères `text` à la position `org` en utilisant la fonte `font` précédemment initialisée.

3.2 Gestion des évènements

Intéressons nous maintenant aux évènements que l'on peut déclencher par l'intéraction avec la souris. Pour cela, il faut pouvoir identifier quand un évènement particulier a lieu et déclencher alors une suite d'actions. Sous OpenCV, cette suite d'actions est définie dans une fonction (souvent nommée `Callback` dans la littérature) et dont le prototype est toujours :
- `void Foo(int event, int x, int y, int flags, void* param)`

où `event` (par exemple de valeur `CV_EVENT_LBUTTONDOWN` - cf la documentation pour les autres options) est l'évènement déclencheur de l'exécution de la fonction `Foo` (vous êtes bien sûr libre de choisir un autre nom ;)). `x` et `y` sont les coordonnées de la souris. `flags` (par exemple de valeur `CV_EVENT_FLAG_LBUTTON` permet de spécifier d'autres évènements complémentaire (comme la pression simultannée par exemple d'un bouton gauche avec la touche MAJ par exemple). `param` tout autre type de données qui serait nécessaire à l'exécution de votre fonction `Foo`.

Pour rendre active la fonction précédente, il ne reste plus qu'à spécifier de quelle fenêtre la fonction `Foo` est dépendante. C'est le rôle de la fonction :
- `void cvSetMouseCallback(const char* windowName, CvMouseCallback Foo, void* param=NULL)`

où `windowName` est le nom de la fenêtre contenant l'image, `Foo` la fonction d'action et `param` les éventuelles données nécessaires à `Foo`.

Une autre fonction utile est :
- `int cvWaitKey(int delay=0)`
  qui attend que l'on presse une touche au clavier pendant `delay` millisecondes (attente infinie si `delay`$\leq$0). La fonction retourne le code ascii de la touche pressée.

3.3 Gestion d'une trackbar

- `int cvCreateTrackbar(const char* trackbarName, const char* windowName, int* value, int count, CvTrackbarCallback onChange)`
  permet de créer la trackbar `trackbarName` de la fenêtre `windowName`. `value` est la valeur reflétant la position actuelle du slider et `count` la position maximale du slider (la position minimale est toujours 0). `onChange` est la fonction `Callback` appelée lorsque la position du slider change. Son prototype est nécessairement :
  - `void Foo(int position)`
- `int cvGetTrackbarPos(const char* trackbarName, const char* windowName)`
  permet de récupérer la position actuelle du slider dans la trackbar.
- `void cvSetTrackbarPos(const char* trackbarName, const char* windowName, int pos)`
  permet de fixer la position `pos` du slider dans la trackbar.

3.4 Quelques exemples

3.4.1 Dessin à la souris

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

#define TRUE 1
#define FALSE 0
```

```
#define NBCOLS 500
typedef struct ParamClick
{
  IplImage * Img;
  IplImage * Clone;
  CvRect * box;
  char draw_box;
  char * WindowName;
}
  ParamClick;
/*****************************/
void Trace(int event, int x, int y, int flags, void* param)
{
  ParamClick *Param;
  Param = (ParamClick *)param;
  switch (event){
  case CV_EVENT_MOUSEMOVE:{
    if(Param->draw_box == TRUE){
      Param->box->width = x-Param->box->x;
      Param->box->height = y-Param->box->y;

    }
    break;
  }
  case CV_EVENT_LBUTTONDOWN:{
    Param->draw_box = TRUE;
    Param->box->x = x;
    Param->box->y = y;
    Param->box->width = 0;
    Param->box->height = 0;
    break;
  }
  case CV_EVENT_LBUTTONUP:{
    Param->draw_box = FALSE;
    /* on s'assure que (X,Y) sont bien en haut a gauche */
    if (Param->box->width < 0){
      Param->box->width = -1 * Param->box->width;
      Param->box->x = Param->box->x - Param->box->width;
    }
    if (Param->box->height < 0){
      Param->box->height = -1 * Param->box->height;
      Param->box->y = Param->box->y - Param->box->height;
    }
    cvRectangle(Param->Img,
cvPoint(Param->box->x,Param->box->y),
cvPoint(Param->box->x+Param->box->width,
Param->box->y+Param->box->height),
cvScalar(255,0,0,0),
CV_FILLED,8,0);
    cvCopy( Param->Img, Param->Clone, NULL );
    break;
  }
  case CV_EVENT_RBUTTONDOWN:{
    cvZero(Param->Img);
    cvCopy( Param->Img, Param->Clone, NULL );
    break;
```

```
  }
  }/* fin switch*/
}
/*****************************/
int main()
{
  IplImage * Img, *Clone;
  char ExitKey;
  ParamClick Param;
  CvRect Box;

  cvNamedWindow("Image",CV_WINDOW_AUTOSIZE);
  Img = cvCreateImage( cvSize(NBCOLS,NBCOLS), IPL_DEPTH_8U, 3 );
  cvZero(Img);
  Param.Img = Img;
  Param.draw_box = FALSE;
  Box = cvRect(0,0,0,0);
  Param.box = &Box;
  Clone = cvCloneImage(Param.Img);
  Param.Clone = Clone;
  cvSetMouseCallback("Image", Trace, (void*) &Param );
  while(1){
    if ( Param.draw_box ){
      cvCopy(Param.Img,Param.Clone,NULL);
      cvRectangle(Param.Clone,
  cvPoint(Param.box->x, Param.box->y),
  cvPoint(Param.box->x+Param.box->width,
  Param.box->y+Param.box->height),
  cvScalar(255,0,0,0),
  1,8,0);
    }
    cvShowImage("Image",Param.Clone);
    ExitKey = cvWaitKey(15);
    if ( ExitKey == 27)
      break;
  }
  cvDestroyWindow("Image");
  cvReleaseImage(&Img);
  return(0);
}
```

3.4.2 Viewer vidéo

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

#define TRUE 1
#define FALSE 0

CvCapture * Video;
/*********************************/
void MovePos(int Pos)
{
  int NbFrames;
  NbFrames = (int)cvGetCaptureProperty(Video,CV_CAP_PROP_FRAME_COUNT);
  cvSetCaptureProperty(Video,
```

```
       CV_CAP_PROP_POS_FRAMES,
       (int)(Pos*NbFrames/100.));
}
/*********************************/
int main(int argc, char ** argv)
{
  IplImage * Frame;
  char StopKey;
  int Position;
  cvNamedWindow("Video",CV_WINDOW_AUTOSIZE);
  Video = cvCreateFileCapture(argv[1]);
  if (Video == NULL){
    printf("Caramba, pas de fichier video!\n");
    return(0);
  }
  cvCreateTrackbar( "Position","Video",&Position,
    100,
    MovePos );
  cvSetTrackbarPos("Position", "Video", 0);
  while(1){
      Frame = cvQueryFrame(Video);
      if ( Frame == NULL)
break;

      cvShowImage("Video",Frame);
      StopKey = (char) cvWaitKey(33);
      /* sortie par la touche esc */
      if (StopKey == 27)
        break;
  }
  cvDestroyWindow("Video");
  cvReleaseCapture(&Video);
  return(0);
}
```

3.5 Exercices

3.5.1

Modifier l'exemple 3.4.1 pour que le slider se déplace lorque la vidéo défile.

3.5.2

Afficher les valeurs rouge, vert et bleu du pixel d'une image pointé par la souris.On pourra utiliser la fonction `cvGet2D` pour récupérer le `CvScalar` représentant les couleurs `BGR` d'un pixel.

3.5.3

Afficher une image couleur et convertir en NB les pixels contenus dans le rectangle dessiné à la souris. Un click droit doit permettre de revenir à l'image originale. Pour cela, on utilisera les fonctions relatives au ROI (Region Of Interest) et la documentation en ligne.