Vision et traitement d'images embarqué
Lire et traiter un flux vidéo avec OpenCV
Annexe_ OpenCV
IGM_M2-SIS

OpenCV est une librairie de **traitement d'image** qui offre un bon éventail de fonctions intégrées et puissantes : http://opencv.org
Une description détaillée des fonctions OpenCV est disponible en ligne à l'adresse suivante : http://opencv.org/downloads.html

Ce tutorial décrit la récupération du flux vidéo d'une webcam tout en analysant chaque commande pour bien comprendre leur utilité. Il est également disponible en ligne à l'adresse suivante : http://www.geckogeek.fr/lire-le-flux-dune-webcam-camera-video-avec-opencv.html

1. Ouvrir le flux d'une vidéo / Webcam / caméra

Avant de pouvoir lire un quelconque **flux vidéo** il faut déjà l'ouvrir. Définissez premièrement la variable qui va stocker ce flux :

```
CvCapture*capture;
```

1.1. Ouvrir une vidéo

Ouvrir une vidéo est chose **relativement aisée**.

```
capture =cvCreateFileCapture("/chemin/de/votre/vidéo/test.avi");
```

1.2. Ouvrir le flux d'une Webcam / Caméra

De manière générale il est facile de lire le flux d'une webcam. Si vous souhaitez utiliser une caméra analogique (non numérique) il vous faudra probablement utiliser un convertisseur pour pouvoir la brancher en USB sur votre ordinateur.

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

Le paramètre dans cvCreateCameraCapture() permet de définir quel flux vidéo récupérer. Généralement le « CV_CAP_ANY » permet de prendre le premier disponible (donc par exemple une webcam intégrée à votre ordinateur). Si vous avez plusieurs flux vidéo, la valeur « 0 » permet de prendre le premier disponible. Les valeurs diffèrent ensuite selon le port « usb » utilisé.

1.3. Vérifier que le flux est bien ouvert

Il est possible qu'ouvrir le flux vidéo **ne se passe pas comme prévu**. Pour pallier à ce problème, il est fortement conseillé de réaliser une petite vérification sur la variable « capture ».

```
if(!capture){
    printf("Ouverture du flux vidéo impossible !\n");
return1;
 }
```

Pour améliorer la vérification vous pourriez même retenter l'ouverture du flux X fois jusqu'à décider d'arrêter (pour éviter une boucle infinie). En effet dans certains cas, la webcam/caméra n'est pas détectée immédiatement.

2. Récupérer une image et l'afficher

Le principe du traitement vidéo est de procéder à **une analyse du flux image par image**. A chaque traitement d'image terminé vous affichez l'image résultante dans une **fenêtre**.

2.1. Créer une fenêtre

Ce qui est bien avec OpenCV c'est que la création d'une fenêtre ne change pas en fonction du système d'exploitation. Par ailleurs vous pouvez **créer plusieurs fenêtres** pour afficher dedans plusieurs résultats (images ou vidéos). Une fenêtre est définie par un nom :

```
cvNamedWindow("Original", CV_WINDOW_AUTOSIZE);
```

Le flag « CV_WINDOW_AUTOSIZE » (qui est en fait un nombre entier : 0 ou 1) permet d'indiquer si la fenêtre doit s'adapter **automatiquement** ou non.

2.2. Récupérer une image

Une image est stockée dans une classe spécifique à OpenCV : **IplImage**. Donc quelque part dans votre code avant de commencer vous devez la définir :

```
IplImage*image;
```

Il existe **deux façons** de récupérer une image du flux vidéo. La plus utilisée est la suivante :

```
image  =cvQueryFrame(capture);
```

Il faut indiquer en paramètre le flux vidéo à utiliser pour récupérer l'image. Cela devient vital dans le cas où vous utilisez plusieurs sources de vidéo !

2.3. Afficher une image

Une fois l'image stockée dans une variable, il suffit de **l'afficher** dans une fenêtre préalablement créée.

```
cvShowImage("Original", image);
```

Notez qu'en général, avant d'afficher l'image, on peut effectuer **des traitements** sur cette dernière. Prenez en compte que la communauté d'OpenCV offre de nombreux codes en exemple.

3. Boucler et temporiser

Pour récupérer les images petit à petit il vous faudra **créer une boucle** et faire une petite pause entre chaque image.

3.1. Faire une pause

Faire une pause dans OpenCV permet en même temps de récupérer une interaction clavier. Cela permet ainsi d'analyser la **touche** qui a subie une interaction et de procéder à certaines actions en conséquence.

```
key=cvWaitKey(10);
```

La commande cvWaitKey() assure une attente de X ms et renvoie l'interaction clavier (s'il y en a eu une). Le temps d'attente **varie en fonction du framerate** que vous souhaitez imposer (ici 10 ms). Par contre vous ne pourrez pas aller au-delà de ce que propose votre caméra. Notez par ailleurs que mettre un temps d'attente trop court peut causer un freeze « plantage informatique » sur certaines versions de Linux. Mettez donc en général au moins 5 ms.

3.2. Faire une boucle

Et enfin il vous faut assembler tous les aspects présentés ci-dessus **pour lire et afficher la vidéo.** Ici nous nous basons sur l'interaction clavier pour arrêter la boucle « while ».

```
// Boucle tant que l'utilisateur n'appuie pas sur la touche q (ou Q)
while(key !='q'&& key !='Q'){
// On récupère une image
    image =cvQueryFrame(capture);
     // On affiche l'image dans une fenêtre
cvShowImage("Original", image);
     // On attend 10ms
key =cvWaitKey(10);
}
```

4. Libérer la mémoire utilisée

Etape très (très) importante, **libérer toute la mémoire** que vous avez utilisé en créant ces divers objets.

4.1. Libérer la fenêtre

Une seule fonction suffit à fermer toutes les **fenêtres** :

```
cvDestroyAllWindows();
```

Toutefois dans le cas où vous ne souhaiteriez que **fermer une seule et unique fenêtre**, vous pouvez utiliser cette dernière :

```
cvDestroyWindow("Original");
```

4.2. Libérer la variable d'image

Lorsque vous récupérez une **image** d'un flux vidéo vous n'avez pas besoin de la « libérer » dans la boucle while, ni même après. OpenCV gère ceci automatiquement et vous n'avez qu'à votre charge de désallouer une image créée par vos soins :

```
uneAutreImage = cvLoadImage("/chemin/de/votre/image/test.png",iscolor);
```

Le flag « iscolor » (qui est en fait un nombre entier : -1, 0 ou 1) permet de spécifier le type de couleur de l'image chargée (iscolor > 0 →l'image chargéeest forcé d'êtreune image couleur3-canal, iscolor = 0 →l'image chargéeest forcé d'êtreune image niveaux de gris et iscolor < 0 →l'image est chargéetelle qu'elle est.

Si vous veniez à le faire, pensez donc à faire ceci :

```
cvReleaseImage(&uneAutreImage);
```

4.3. Libérer la capture vidéo !

Et enfin, il vous faut libérer la **capture vidéo** (entre autre pour que la webcam/caméra ne soit plus utilisée par le programme).

```
cvReleaseCapture(&capture);
```

Tutorial préparé et publié par Vinz, 14/03/2010

Définition de la structure de l'image avec OpenCV

http://poincare.princeton.edu/madtrac/docs/struct_ipl_image.html

```
typedefstruct _IplImage
{
Int nSize;
int ID;
int nChannels;
int alphaChannel;
int depth;
char colorModel[4];
char channelSeq[4];
int dataOrder;
int origin;
int align;
int width;
int height;
struct _IplROI *roi;
struct _IplImage *maskROI;
void *imageId;
struct _IplTileInfo *tileInfo;
int imageSize;
char *imageData;
int widthStep;
int BorderMode[4];
int BorderConst[4];
char *imageDataOrigin;
}
IplImage;
```

ParamnSize: sizeof(IplImage)

param ID: Version, always equals 0

paramnChannels: Number of channels. Most OpenCV functions support 1-4 channels.

paramalphaChannel: Ignored by OpenCV

param depth: Pixel depth in bits. The supported depths are:

IPL_DEPTH_8U - Unsigned 8-bit integer

IPL_DEPTH_8S - Signed 8-bit integer

IPL_DEPTH_16U - Unsigned 16-bit integer

IPL_DEPTH_16S - Signed 16-bit integer

IPL_DEPTH_32S - Signed 32-bit integer

IPL_DEPTH_32F - Single-precision floating point

IPL_DEPTH_64F - Double-precision floating point

paramcolorModel: Ignored by OpenCV. The OpenCV function *CvtColor* requires the source and destination color spaces as parameters.

paramchannelSeq: Ignored by OpenCV

paramdataOrder: 0 = IPL_DATA_ORDER_PIXEL - interleaved color channels, 1 - separate color channels. *CreateImage* only creates images with interleaved channels. For example, the usual layout of a color image is: b00g00r00b10g10r10

param origin: 0 - top-left origin, 1 - bottom-left origin (Windows bitmap style)

param align: Alignment of image rows (4 or 8). OpenCV ignores this and uses widthStep instead.

paramwidth: Image width in pixels

param height: Image height in pixels

paramroi: Region Of Interest (ROI). If not NULL, only this image region will be processed.

ParammaskROI: Must be NULL in OpenCV

ParamimageId: Must be NULL in OpenCV

ParamtileInfo: Must be NULL in OpenCV

paramimageSize: Image data size in bytes.

paramimageData: A pointer to the aligned image data

paramwidthStep: The size of an aligned image row, in bytes

paramBorderMode: Border completion mode, ignored by OpenCV

paramBorderConst: Border completion mode, ignored by OpenCV

paramimageDataOrigin: A pointer to the origin of the image data (not necessarily aligned). This is used for image deallocation.

The *IplImage* structure was inherited from the Intel Image Processing Library, in which the format is native. OpenCV only supports a subset of possible *IplImage* formats, as outlined in the parameter list above. In addition to the above restrictions, OpenCV handles ROIs differently. OpenCV functions require that the image size or ROI size of all source and destination images match exactly. On the other hand, the Intel Image Processing Library processes the area of intersection between the source and destination images (or ROIs), allowing them to vary independently.