

Jafar : Introduction au module Image

Le module Image

Thèmes abordés

- ➔ Problématique
- ➔ OpenCV et la structure *IPLImage*
- ➔ La classe *Image*
- ➔ Le lien avec OpenCV
- ➔ ImagePreprocessor / DataReader

Problématique

- ➔ Pourquoi un module image ?
- ➔ Besoins :
 - Unification des structures images pour tous les modules de vision à venir
 - Structure de donnée moderne/extensible
 - Fonctions I/O, utilitaires et de traitements d'image bas niveau
- ➔ Solution ?

Problématique

➔ Le challenger retenu :

- OpenCV
 - Structure de données plus complète (en C)
 - Fonctions utilitaires
 - Fonctions de traitement d'image orientée vision
 - Fonctions orientée robotique pratiquement inexistantes

Thèmes abordés

- ➔ Problématique
- ➔ **OpenCV et la structure *IPLImage***
- ➔ La classe *Image*
- ➔ Le lien avec OpenCV
- ➔ ImagePreprocessor / DataReader

OpenCV et la structure IplImage

➞ OpenCV

- Librairie de fonctions relatives a la vision développée initialement chez Intel
- OpenSource
- Très grande communauté (active)
- Codes optimisés
- Ne fonctionnent que sur des machines x86 (Windows/Linux) et PowerPC (MacOS X)

OpenCV et la structure IplImage

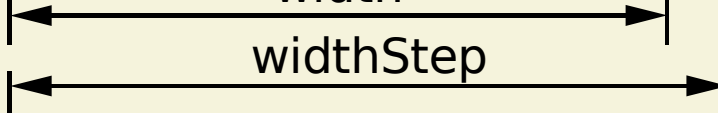
➔ *IplImage*

- Structure employée par toutes les fonctions de traitement d'image d'OpenCV
- Paramètres importants :
 - Taille de l'image : *width, height*
 - Nombre de plans chromatiques (*nChannels*)
 - Profondeur des pixels (*depth*) :
 - Le tableau de pixels : *imageData*
 - Rembourrage sur la largeur : *widthStep*

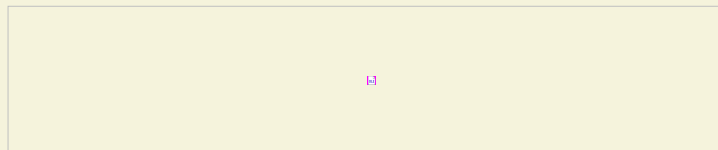
OpenCV et la structure *IplImage*

➔ Précisions sur les paramètres de *IplImage*

- *nChannels* : 1,2,3 ou 4. Mais les fonctions d'OpenCV ne travaillent en général qu'avec 1 ou 3 plans.
- Profondeur possibles (*depth*) :
 - unsigned 1-bits/8-bits/16-bits
 - signed 8-bits/16-bits/32bits
 - floating point 32-bits/64-bits

- *widthStep* : 

The diagram illustrates the relationship between *width* and *widthStep*. It shows two horizontal double-headed arrows. The top arrow is labeled *width* and spans a certain distance. The bottom arrow is labeled *widthStep* and spans a longer distance, indicating that *widthStep* is greater than or equal to *width*.

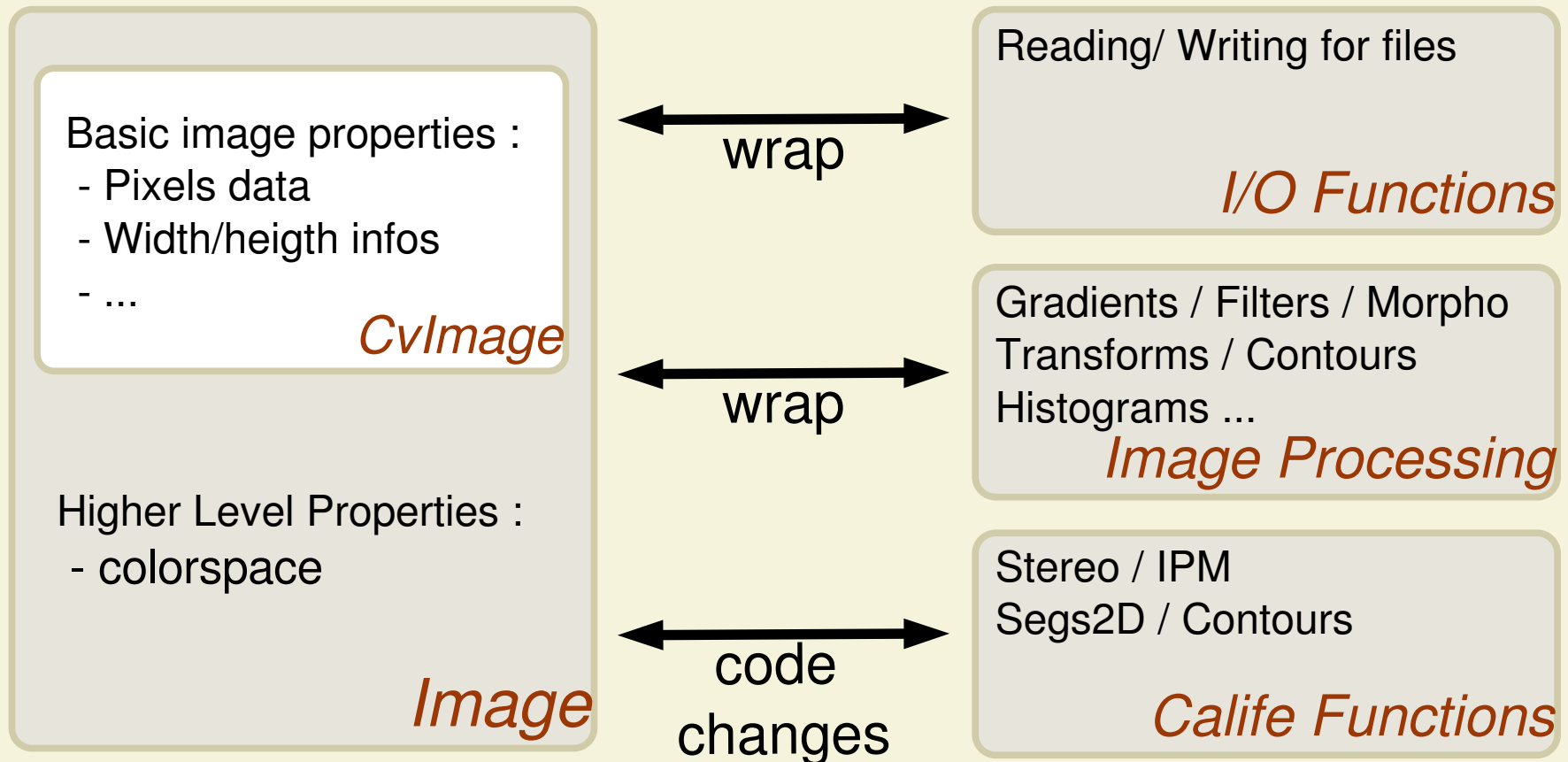


Thèmes abordés

- ➔ Problématique
- ➔ OpenCV et la structure *IPLImage*
- ➔ **La classe *Image***
- ➔ Le lien avec OpenCV
- ➔ ImagePreprocessor / DataReader

La classe Image

➡ « Un gros schéma ... »



La classe Image

- ➔ ... vaut mieux qu'un petit discours »
- Création d'une classe *Image* (C++)
extension de *CvImage*
- *CvImage* est une encapsulation (C++) de
la structure *IplImage* utilisée par OpenCV
(C)
- Réemploi au plus simple des fonctions
OpenCV « bas-niveau »
- Réinjection du code Calife « haut-
niveau » en le modifiant

La classe Image

➔ Les paramètres d'une *Image* :

- la taille
- bit depth (de 1bit à 64bits en flottant)
- l'espace de couleur
 - `JfrImage_TypeColorSpace { JfrImage_CS_GRAY, JfrImage_CS_BGR, JfrImage_CS_RGB, JfrImage_CS_HSV, JfrImage_CS_HSL, JfrImage_CS_CMY, JfrImage_CS_CMYK, JfrImage_CS_LUV, JfrImage_CS_LAB, JfrImage_CS_XYZ, JfrImage_CS_YXY, JfrImage_CS_BayerBG, JfrImage_CS_BayerRG, JfrImage_CS_YCbCr, JfrImage_CS_UNKNOWN }`

La classe JfrImage

⇒ I/O fonctions :

- Support du *depth* u8 uniquement
- Support de 1 ou 3 plans chromatiques
- Formats de fichiers :
 - Windows bitmaps : BMP, DIB
 - JPEG files : JPEG, JPG, JPE
 - Portable Network Graphics : PNG
 - Portable image format : PBM, PGM, PPM
 - Sun rasters : SR, RAS
 - TIFF files - TIFF, TIF

La classe Image

- ➞ Fonctions utilitaires :
 - Cloner une image sans recopie
 - Copie du header *JfrImage*
 - Conversion de *depth*
 - Changement d'échelle
 - Conversion d'espace de couleur

La classe Image

➞ Accéder aux pixels :

- Les pixels sont stockés dans le *char* imageData*.
- C'est donc toujours un *char** quelque soit le *depth* choisi : *u8* (unsigned 8-bits) ou *f32* (floating-point 32-bits)
- Lors d'un accès aux pixels un *cast* de *imageData* est donc nécessaire.
- Deux méthodes sont possibles ...

La classe Image

➡ La mauvaise :

```
#include "image/JfriImage.hpp"  
using namespace jafar::image;
```

```
JfriImage img(800, 600, s32, 1);  
int* pxPtr;
```

```
pxPtr = static_cast<int*>(img.getImageData());
```

- Sur des machines x86 sous *Linux* le type *int* est associé à *signed 32-bits* => ça marche
- Mais quand est-il sur MacOS X ou de futurs plateformes ou avec d'autres compilateurs ?

La classe Image

➡ La bonne :

```
#include "image/Image.hpp"  
#include <stdint.h>  
using namespace jafar::image;
```

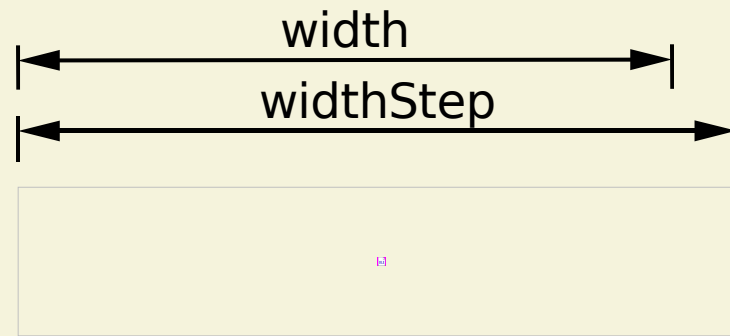
```
JfrImage img(800, 600, s16, 1);
```

```
pxPtr = reinterpret_cast<uint16_t*>img.data();
```

- Cette méthode garantit l'utilisation du bon type pour parcourir les tableaux de pixels et fonctionne sur tout types de plateformes

La classe Image

- ➔ WidthStep :
- Alignement des lignes sur 8 octets
 - WidthStep est la vrai longueur des lignes exprimee en nombre de sizeof(char)
 - Conversion penible



La classe Image

➞ Un vrai bout de code :

```
#include "image/Image.hpp"  
using namespace jafar::image;
```

```
Image img(800, 600, JfrlImage::s16, 1);
```

```
for(int i=0;i<height;i++) {  
    uint16_t *pxPtr = reinterpret_cast<uint16_t*>src_.data(i);  
    for(int j=0;j<width;j++)  
        *(pxPtr++) = 1;  
}
```

Thèmes abordés

- ➔ Problématique
- ➔ OpenCV et la structure *IPLImage*
- ➔ La classe *Image*
- ➔ **Le lien avec OpenCV**
- ➔ ImagePreprocessor / DataReader

Utilisation des fonctions d'OpenCV

➞ Principe :

- Encapsulation d'une *IplImage* dans la classe *CvImage* => permet l'utilisation des fonctions d'*OpenCV*
- Comment appeler les fonctions ?
 - Directement :
`cvLaplace(source,dst);`
 - A travers un *wrap* de la fonction *OpenCV* :
`source.resize(dst);`

Thèmes abordés

- ➔ Problématique
- ➔ OpenCV et la structure *IPLImage*
- ➔ La classe *JfrlImage*
- ➔ Le lien avec OpenCV
- ➔ **ImagePreprocessor / DataReader**

Data Reader

- ➔ Le problème : utilisation de beaucoup de jeux de données différents
- ➔ La solution :
 - des fichiers de configuration
 - une interface C++ pour lire ces données

DataReader

➡ Exemple:

```
DataReader dR("exterieur2007", 2);
```

```
StereoReader* sR = dR.getStereoReader(dR);
```

```
sR->calibrationFileName(); // fichier de la calibration  
stéréo
```

```
Image* img = sR->left()->loadImage(0); // Charge la  
première image gauche de la série
```

```
sR->right()->calibrationFileName(); // fichier de  
calibration de la caméra droite
```


ImagePreprocessor

- ➔ Preprocessing: correction de distortion, debayerisation, ...
- ➔ La chaîne de traitements dépend de :
 - la source de donnée
 - de l'algorithme final
- ➔ Eviter de multiples dépendences
- ➔ Evolution futur: créer automatiquement la chaîne

ImagePreprocessor

➡ Example d'un noeud:

```
class ImagePreprocessorNode {  
    public:  
        virtual void preprocessImage(const jafar::image::Image&  
src, jafar::image::Image& dst) =0;  
};
```

```
class ImagePreprocessorBlur {  
    public:  
        virtual void preprocessImage(const jafar::image::Image&  
src, jafar::image::Image& dst)  
        {  
            cvSmooth(src, dst)  
        }  
};
```

ImagePreprocessor et DataReader

- ➔ Utiliser un préprocesseur avec un objet DataReader

```
DataReader dR("exterieur2007", 2);  
StereoReader* sR = dR.getStereoReader(dR);
```

```
Jafar::Preprocessing::Preprocessing* leftrectif = new  
    Jafar::Preprocessing::Preprocessing();  
leftrectif->load(sR.left().calibrationFileName());
```

```
leftPreProc = new ImagePreprocessor.new(width, height, depth,  
    colorSpace);
```

```
leftpPreProc.appendNode(leftrectif);  
sR.left().setImagePreprocessor( leftProc )
```