École Centrale de Pékin

TP 2 - Prise en main d'OpenCV

OpenCV est une bibliothèque open source de traitement d'image et de vision par ordinateur. Elles est écrite en C++ mais possède de nombreux wrappers qui permettent son utilisation avec d'autres langages de programmation. Ce n'est pas, loin s'en faut, la seule bibliothèque existante, mais c'est une des plus courantes et des plus adaptées à la vision par ordinateur en temps réel.
Pour chaque question, les difficultés rencontrées et résultats obtenus devront être illustrés dans votre rapport. Vous devez aussi rendre vos codes (inutile de les copier dans le rapport de TP). Les TPs peuvent être réalisés soit en C++ soit en Python.

1 Présentation de la structure de données cv : :Mat

On se propose dans un premier temps de faire un redimensionnement "à la main" d'une image en OpenCV. Le format utilisé par OpenCV pour gérer les images est `cv::Mat`, n'hésitez pas à aller lire la documentation en ligne `https://docs.opencv.org/4.5.5/d3/d63/classcv_1_1Mat.html`.

Chaque matrice possède un nombre de lignes (`rows`) et de colonnes (`cols`). Le nombre de lignes représente la hauteur de l'image alors que le nombre de colonnes représente la largeur et le pixel $(0,0)$ est en haut à gauche.)

Chaque `Mat` possède un `type de données` qui correspond à ce qu'il y a de stocké dans la matrice. Les types OpenCV ont une convention de nommage :
`CV_<bit-depth>U|S|FC(<number_of_channels>)` où `U` représente un type de données entier non signé, `S` représente un type entier signé et `F` représente un type flottant.

Notez toutefois une information importante :

- en fonction du type d'image (couleur ou niveau de gris), les pixels des `cv::Mat` ne sont pas du même type !
  - une image en niveau de gris stockera des pixels de type `uchar` (c'est-à-dire `unsigned char`) `sur un seul canal`. Cela correspond au `type` OpenCV `CV_8UC1`
  - une image en couleur stockera des pixels de type `uchar` (c'est-à-dire `unsigned char`) `sur trois canaux` (un canal par couleur). Cela correspond donc au type OpenCV `CV_8UC3`.
- si vous souhaitez un peu plus d'exemples, vous pouvez aller à l'adresse suivante : `https://docs.opencv.org/4.5.5/d6/d6d/tutorial_mat_the_basic_image_container.html`

Voici un exemple d'accès à un pixel pour une image (`m_imG`) en niveau de gris (donc de type `CV_8UC1`) et pour une image en couleurs (`m_imC`) donc de type `CV_8UC3` :

```
int grayVal = m_imG.at<uchar>(0,0);
cv::Vec3b col = m_imC.at<Vec3b>(0,0);
int greenCol = col[1];
```

Le code OpenCV fourni sur la plateforme pédagogique pour le TP2 vous permet d'ouvrir une image en niveau de gris (par défaut ../imagesDeTest/monarch.png ou celle dont le chemin d'accès est passé en paramètre lors de l'exécution) et de l'afficher dans une fenêtre tant que la touche `ESC` n'est pas enfoncée. Commencez par compiler et exécuter ce code.

2 Redimensionnement d'image

On souhaite maintenant redimensionner une image de 256x256 pixels (par exemple `pepper.png`) en 64x64 "à la main" en ne sélectionnant qu'un seul pixel sur 4. Pour ce faire, modifiez le code OpenCV fourni pour pouvoir ouvrir une image en niveau de gris (de préférence ayant une taille sur les 2 dimensions multiples de 64).

Créez une nouvelle image (i.e. un objet de type `cv::Mat`) "vide" en utilisant la méthode `create` (attention au type de l'image) ou le constructeur de la classe `cv::Mat`. Exemple d'appel à `create` :

```
// on suppose une image 'img' de type Mat deja creee
// ou chargee depuis un fichier
Mat monImg;
monImg.create(nbL, nbC, img.type());
// on cree donc une image
// de nbL 'rows', nbC 'cols'
// et du meme type que 'img'
// NB : pour les 'rows' et 'cols' voir plus haut
```

Il est possible d'accéder à chaque pixel d'une image 2D directement en OpenCV en implémentant une double boucle pour le parcours. Il faut toutefois faire attention à la manière dont les images en couleur sont stockées en mémoire pour écrire un parcours qui fonctionnera pour tous les types d'images.
Le parcours le plus efficace et "simple" à écrire est le suivant :

```
for(int i = 0; i < res.rows;i++) {
    for(int j = 0;j < res.cols;j++){
        res.at<uchar>(i, j) = // ce que vous voulez !
    }
}
```

Vous pouvez aller lire le tutoriel d'OpenCV sur l'accès aux pixels d'une image ici :
`http://docs.opencv.org/2.4/doc/tutorials/core/how_to_scan_images/how_to_scan_images.html#howtoscanimagesopencv`
Créez une image de taille 64x64 à partir d'une image plus grande en entrée. Commentez le résultat obtenu.

3 Application de Filtres de Convolution

Appliquez les filtres suivants à l'aide d'OpenCV à l'image `peppers-512.png` et décrivez leurs effets.

Il y a deux manières d'appliquer les filtres avec OpenCV :
- le faire à l'aide de boucles en travaillant directement sur les éléments de la matrice représentant l'image
- utiliser des fonctions adéquates de l'API, à chercher dans la doc : `https://docs.opencv.org/4.5.5/` (indice regardez une fonction `filter2D`)

Liste des filtres à appliquer à l'image d'entrée :

$$\frac{1}{16}\begin{pmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{pmatrix} \tag{3.1}$$

$$\frac{1}{9}\begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{pmatrix} \tag{3.2}$$

$$\begin{pmatrix}1 & -3 & 1\\ -3 & 9 & -3\\ 1 & -3 & 1\end{pmatrix} \tag{3.3}$$

$$\begin{pmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{pmatrix} \tag{3.4}$$

$$\begin{pmatrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{pmatrix} \tag{3.5}$$

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix} \tag{3.6}$$

Pour chaque filtre, précisez à quoi il sert et montrez son résultat.

4 Morphologie Mathématique

Attention : l'objectif principal de cette partie est la manipulation d'image pixel par pixel, il ne s'agit donc en aucun cas d'utiliser les fonctions d'OpenCV qui effectuent les mêmes opérations. Il faut faudra parcourir une image comme nous l'avons vu plus haut.

Pour vous aider à vérifier vos résultats, nous fournissons sur le serveur pédagogique un programme permettant d'utiliser les fonctions d'OpenCV de morphologie mathématique.

Transformez l'image `peppers-512.png` en une image binaire et écrivez des filtres d'`érosion` et de `dilatation`. Quel est l'élément structurant utilisé ?

Que donne l'`ouverture` et la `fermeture` de l'image `peppers` ? Que pouvez-vous déduire d'une image de différence entre une image dilatée et érodée ?
Bonus : testez les mêmes opérations en changeant d'élément structurant.

5 Transformée de Fourier

La transformée de Fourier permet de représenter une image dans son domaine fréquentiel à l'aide d'un spectre. Le spectre est une fonction complexe 2D qui peut être exprimée sous la forme d'un spectre d'amplitude (*magnitude* en anglais) et d'un spectre de phase, la première représentation étant la plus utilisée. Pour quelques informations sur la Transformée de Fourier, voir ici :
https://cs184.eecs.berkeley.edu/public/sp21/lectures/lec-3-sampling-aliasing-and-antialias
lec-3-sampling-aliasing-and-antialiasi.pdf

L'implémentation de la transformée de Fourier en OpenCV n'est pas forcément triviale. Pour ce TP, vous devez utiliser le fichier disponible sur le serveur pédagogique pour vous aider. Si vous souhaitez, vous pouvez lire des informations sur la DFT et OpenCV sur les liens suivants :

https://docs.opencv.org/4.5.5/d8/d01/tutorial_discrete_fourier_transform.html

https://docs.opencv.org/4.5.5/d8/d01/tutorial_discrete_fourier_transform.html

Effectuez les transformées de Fourier des images D1r.pgm, D11r.pgm et D46r.pgm et commentez les structures qui apparaissent sur les transformées.

Ouvrez peppers-512.png et effectuez sa transformée de Fourier. Effectuez une rotation de l'image de départ et effectuez sa transformée de Fourier. Commentez le résultat obtenu.