Aide-mémoire OpenCV : fonctions de base

Ce petit formulaire résume les classes et fonctions les plus importantes pour manipuler des images avec la librairie OpenCV.

1. Matrices (classe Mat)

1.1 Stockage en mémoire

Les matrices OpenCV sont stockée **lignes par lignes**.

- $M(i)$ = ième ligne
- $M(i,j)$ = ième ligne, jième colonne
- En général : $M(i,...,j)$: i = indice externe, j = indice interne (le plus local)

1.2 Types de données

- Types mono-canal : `CV_8U, CV_8S, CV_16U, CV_16S, CV_32S, CV_32F, CV_64F`
- Types multi-canaux : `CV_XXC(ncanaux)` (nombre de canaux : 1 à 512)
- Exemples :
  - `CV_8U` ou `CV_8UC1` = image mono-canal 8-bit (image en niveaux de gris)
  - `CV_8UC(3)` ou `CV_8UC3` = image à trois canaux couleur (par exemple RGB)

1.3 Initialisation d'une matrice

- `Mat(dimy,dimx,type[,valeur par défaut (Scalar) ou pointeur vers tampon déjà initialisée (void *)])`
- Forme alternative : `Mat(Size(dimx,dimy),...)`
  - Exemple : Image BGR 320 par 200, initialisée en blanc : `Mat(200,320,CV_8UC3,Scalar(255,255,255))`
  - Exemple : Image BGR 320 par 200, à partir d'un tampon déjà initialisé : `Mat(Size(320,200),CV_8UC3,buffer)`
- `Mat m = Mat::zeros(dimy,dimx,type)` : Matrice initialisée à zéro
- `Mat m = Mat::ones(dimy,dimx,type)` : Matrice initialisée à un
- `Mat m2 = m.clone()` : copie complète (les deux matrices sont indépendantes)
- `Mat m2 = m` : copie d'en-tête seulement (les deux matrices pointent vers les mêmes données)

1.4 Propriétés d'une matrice

- `int Mat::channels()` Nombre de canaux (1, 2, 3, ...)
- `int Mat::depth()` Résolution (8, 16, 32, ...)
- `int Mat::type()` Type complet (`CV_8UC3, CV_32F, ...`)
- `Size Mat::size()` Dimensions (`largeur = m.size().width, hauteur = m.size().height`)
- `int Mat::cols, int Mat::rows` Idem

1.5 Accès aux pixels / éléments d'une matrice

- Accès individuel à un pixel : `m.at<type>(y,x)` avec type = `char, unsigned char, float, ...`
- Forme alternative : `m.at<type>(Point(x,y))`
- Pointeur vers les données brutes (premier pixel) : `m.ptr<type>()`
- Pointeur vers les données brutes (ième ligne) : `m.ptr<type>(i)`
- Affectation d'un ensemble de pixels : `m.setTo(Scalar[,masque])`

1.6 Extraire une zone d'intérêt

```
Mat roi = Mat(I, Rect(x,y,largeur,hauteur));
```

Les données ne sont pas copiées (la matrice `roi` pointe vers les pixels de la matrice source).

1.7 Masques

Un masque dans OpenCV est une matrice monocanal 8 bits (type : `CV_8U`), où chaque élément vaut 0 ou 255.

- Comparaisons simples (renvoi un masque) : $A >= B, A < B$, etc.
- `inRange(I, borne_inf, borne_sup, masque)`

Beaucoup de fonctions OpenCV acceptent un argument optionnel de masquage (l'opération est effectuée seulement pour les pixels non nuls du masque).

2. Classes utiles

- Rectangle : `Rect(x,y,width,height)`
- Dimensions : `Size(width,height)`
- Coordonnées 2d (entières) : `Point(x,y)`
- Spécification de la valeur d'un pixel (1 - 4 canaux) : `Scalar(v0[,v1[,v2[,v3]]])`
- Pixel 3 canaux 8 bits : `Vec3b` (opérateur `[]` pour accéder aux valeurs).

3. Interface homme / machine

3.1 Interface utilisateur

`imshow("titre fenêtre",I)` Affiche l'image I dans une fenêtre graphique

`int waitKey()` Attends que l'utilisateur appuie sur une touche du clavier (retourne le code ASCII).

`int waitKey(delais_ms)` Retourne le code ASCII ou -1 si le délais expire en premier.

3.2 Entrées / sorties

`Mat I = imread("nom fichier.ext")` Lecture d'une image. Supporte de nombreux formats (ext = jpg, png, bmp, etc.)

`imwrite("nom fichier.ext", I)` Écriture d'une image. Les pixels doivent être normalisés entre 0 et 255 (même en flottant).

4. Opérations de base sur les images

`cvtColor(I, O, code)` Changement d'espace de couleur (conversion vers / à partir de niveaux BGR / RGB). code = `COLOR_XXX2YYY, COLOR_YYY2XXX`, avec `XXX = RGB` ou `BGR`, et `YYY = GRAY, XYZ, YCrCb, HSV, HLS` ou `Bayer`. Par exemple, conversion BGR vers niveaux de gris : `cvtColor(I, O, COLOR_BGR2GRAY)`

`I.convertTo(O,otype[,scale])` Conversion de type. Par exemple, conversion 8 bits non signés (entre 0 et 255) vers flottant entre 0 et 1.0 : `I.convertTo(O,CV_32F,1.0/255)`

`resize(I, O, Size(width,height))` Redimensionnement d'une image.

`resize(I, O, Size(0,0), fx, fy)` Idem, sans préciser la taille de sortie mais en donnant le facteur de réduction (si < 1) ou agrandissement (si > 1) suivant chaque dimension.

`normalize(I, O, min, max, NORM_MINMAX)` Normalisation d'une image. Par exemple, normalisation entre 0 et 255 : `normalize(I, O, 0, 255, NORM_MINMAX)`

`threshold(I, O, seuil, 255, THRESH_BINARY)` Seuillage et calcul d'une matrice binaire en sortie (valeur = 0 si $<$ seuil, 255 si supérieur)

`split(const Mat &I, Mat *O)` Séparation des différents canaux de `I` dans `O[0], O[1],`, ...(démultiplexage)

`merge(const Mat *I, int nchn, Mat &O)` Regroupement de plusieurs canaux séparés `I[0], I[1],` ...dans la même image `O` (multiplexage)

5. Fonctions de dessin

`circle(Mat &img, Point center, int radius, Scalar color, int thickness, int linetype)` avec linetype = 4, 8 ou `LINE_AA` (anti-repliement)

`line(Mat &img, Point p1, Point p2, ...)`

`rectangle(Mat &img, Point p1, Point p2, ...)` ou `rectangle(Mat &img, Rect r, ...)`

`putText(Mat &img, texte, Point pos, fontFace, fontScale, ...` avec fontFace = `FONT_HERSHEY_SIMPLEX`, ...