

TP 1 : Détection d'une balle Colorée

Dans ce TP, nous allons mettre en pratique les notions vues durant le cours pour détecter un objet de couleur. Ce sera l'occasion de vous familiariser avec le Wrapper EmguCv ainsi qu'avec la documentation de EmguCv et OpenCv.

Le TP est conçu pour une détection sur une seule Image (pour des raisons de visio et de facilité conception) mais nous verrons aussi comment récupérer une image à partir d'une webcam.

Les consignes numérotées en gras sont les points que vous avez à coder vous-mêmes.

1 Créer un nouveau projet C# avec Visual Studio

Se reporter à la fiche « Installation ».

2 Afficher une Image avec EmguCV

Afin d'utiliser EmguCv, ajouter les includes correspondants :

```
using Emgu.CV;  
using Emgu.CV.Structure;  
using Emgu.CV.CvEnum;  
using Emgu.CV.Util;
```

1) Ouvrir une image avec EmguCV.

Classes :

Emgu.CV.Mat

2) Alternative : Récupérer une image à partir de la webcam.

Classes :

Emgu.CV.Mat

Emgu.CV.VideoCapture

Fonctions :

VideoCapture.QueryFrame()

L'image que nous allons utiliser dans ce TP s'appelle crochet.jpg dans le dossier Image.

3) Afficher une Image

Fonctions :

CvInvoke.Imshow()

CvInvoke.WaitKey()

Nous verrons dans le TP2 comment découpler l'acquisition des images de la caméra et la boucle de jeu.

3 Accès aux pixels

Bien que EmguCV 3.0 recommande d'utiliser la structure Mat, celle-ci ne possède pas de méthodes d'accès aux pixels. Pour cela, il faut convertir notre Matrice de type Mat en objet de type Image.

Classes :

EmguCv.Mat

EmguCv.Image<Type_de_l_image, type_du_pixel>

Fonctions :

EmguCv.Mat.ToImage< Type_de_l_image, type_du_pixel>()

Les types d'images sont définis par EmguCV, les plus courants sont : Bgr, Hsv, Gray.

Les types de pixels sont les types de bases du C#, les plus courants utilisés en image sont : byte (char en C++), int, float.

La donnée membre Data est un tableau contenant les pixels de l'image. EmguCv.Image.Data[x,y,0] permet d'accéder au premier canal du pixel à la position (x,y).

- 4) **Ajouter une fonction CopyToImage qui permet de copier les pixels d'une image couleur vers une autre. Ajouter deux paramètres optionnels offsetX et offsetY afin de permettre de positionner l'image copiée dans l'image de destination.**
- 5) **Surcharger la fonction afin de permettre la copie d'une image en niveau de gris dans une image couleur.**
- 6) **Créer une fonction GrayCopyChannelToImage qui permet d'effectuer la copie d'un canal spécifié par un index dans une image de destination. Ajouter deux paramètres optionnels offsetX et offsetY afin de permettre de positionner l'image copiée dans l'image de destination.**

4 Première manipulation d'images

- 7) **Convertir l'image crochet.jpg en niveau de gris.**

Fonctions :

CvInvoke.CvtColor()

- 8) **Appliquer une transformation «miroir vertical » à l'image crochet.jpg.**

Fonctions :

CvInvoke.Flip()

- 9) **Dans une nouvelle Image, copier l'image originale, l'image en niveau de gris et l'image miroir côte-à-côte et afficher cette image à l'écran.**

Fonctions :

CopyToImage()



Figure 1 - Mirroir vertical

5 Seuillage d'une image HSV

Pour détecter notre balle, nous allons utiliser l'information de couleur et en particulier les différentes nuances de vert, le seuillage étant plus complexe à paramétrer dans l'espace BVR, nous allons convertir l'image en HSV.

H = teinte

S = intensité (saturation)

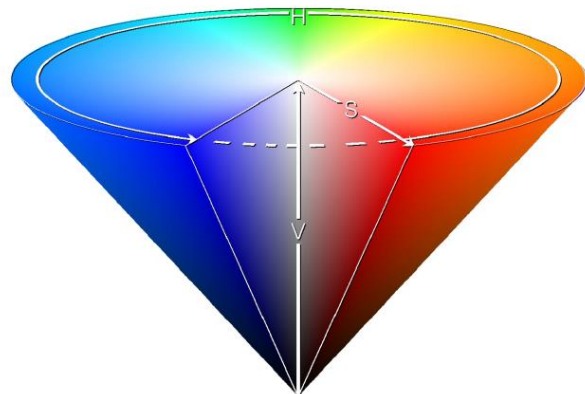
V = brillance = plus la *valeur* d'une couleur est faible, plus la couleur est sombre

H = 0...179 / S = 0...255 / V = 0...255

(HSV = TSV en français)

A diviser par 2 dans OpenCV

- 0° ou 360° : **rouge** ;
- 60° : **jaune** ;
- 120° : **vert** ;
- 180° : **cyan** ;
- 240° : **bleu** ;
- 300° : **magenta**.



10) Convertir l'image au format HSV

11) Afficher chacun des canaux séparément sur une nouvelle image. Observer chacun des canaux, quel canal semble le plus simple à seuiller ?

Fonctions :

CVInvoke.CvtColor()

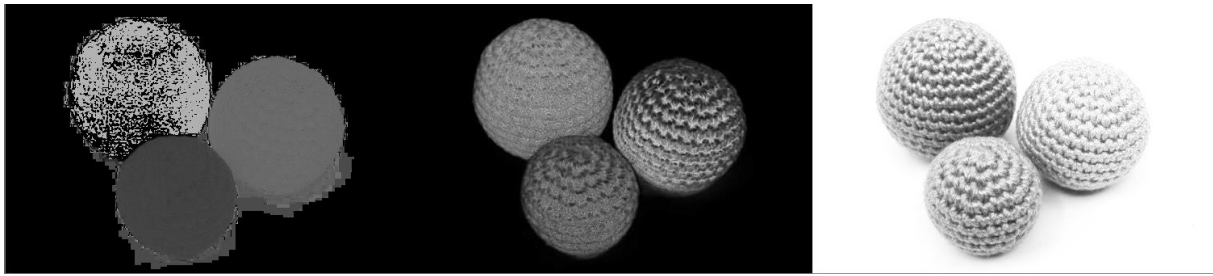


Figure 2 - de gauche à droite Hue, Saturation, Values

12) Seuiller l'image en utilisant un intervalle défini par une borne inférieure et une borne supérieure. Vous pouvez régler les seuils en utilisant l'image hsv.png.

Astuce : Les bornes sont à définir en utilisant le type des pixels. Hsv borneInf = new Hsv(teinteMin, saturationMin, ValeurMin).

L'image obtenue après un seuillage n'est pas toujours propre. Des artefacts (petits morceaux parasites) sont souvent présents.

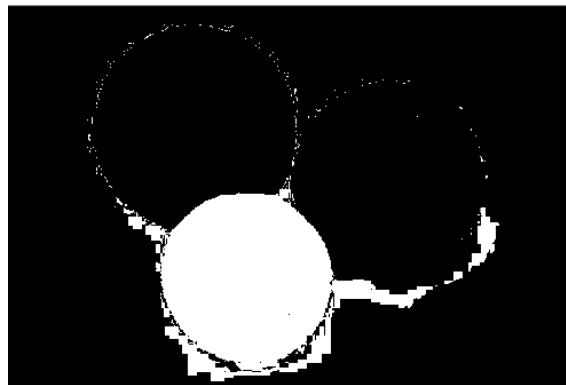


Figure 3 - Artefacts de seuillage dus au format JPEG

6 Filtrage des artefacts

13) Filtrer les artefacts de l'image seuillée. Quel filtre morphologique utiliser dans ce cas ?

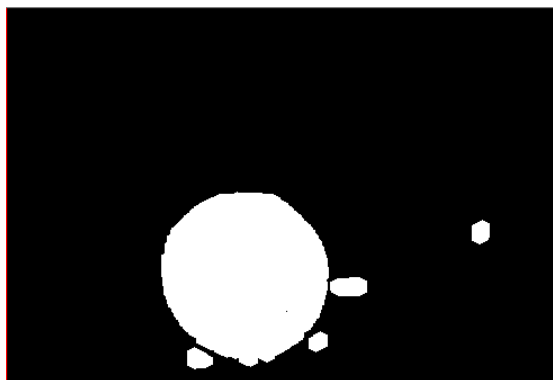


Figure 4 - Filtrage des artefacts

Note : Il peut rester des morceaux après le filtrage des artefacts, nous les prendrons en compte lors de la prochaine étape.

7 Contours et aires

14) Calculer les contours des éléments de l'image et récupérer le contour de l'objet ayant l'aire la plus élevée. Afficher ce contour.

Classes :

Emgu.CV.Util.VectorOfVectorOfPoint

Emgu.CV.VectorOfPoint

Fonctions :

CvInvoke.FindContours()

CvInvoke.ContourArea()

CvInvoke.DrawContours()

Les moments d'une image sont des sommes pondérées d'informations contenues dans celle-ci. Ceux-ci permettent de caractériser une image. Certains moments ont de plus la propriété d'être invariant aux transformations géométriques de mise à l'échelle, translation ou rotation ce qui peut être pratique pour identifier un même objet dans plusieurs scènes. Par exemple les moments de Hu.

$$M_{ij} = \sum_x \sum_y x^i y^j I(x, y)$$







id	Image	H[0]	H[1]	H[2]	H[3]	H[4]	H[5]	H[6]
K0		2.78871	6.50638	9.44249	9.84018	-19.593	-13.1205	19.6797
S0		2.67431	5.77446	9.90311	11.0016	-21.4722	-14.1102	22.0012
S1		2.67431	5.77446	9.90311	11.0016	-21.4722	-14.1102	22.0012
S2		2.65884	5.7358	9.66822	10.7427	-20.9914	-13.8694	21.3202
S3		2.66083	5.745	9.80616	10.8859	-21.2468	-13.9653	21.8214
S4		2.66083	5.745	9.80616	10.8859	-21.2468	-13.9653	-21.8214

Figure 5 - Valeur des moments de Hu pour différentes transformations

15) Calculer les moments du plus gros contour.

Fonctions :

`CvInvoke.Moments()`

Astuce :

La fonction peut aussi prendre en paramètre une liste de points.

16) Calculer le centroïde du contour à partir des moments M10, M00, M01, M00. Afficher le centroïde de la balle.

Fonctions :

`CvInvoke.Circle()`

8 Boîte englobante

17) Calculer la boîte englobante de la balle et afficher celle-ci.

Fonctions :

`CvInvoke.MinAreaRect()`

`CvInvoke.BoundingRectangle()`

`CvInvoke.BoxPoints()`

`CvInvoke.Line()`

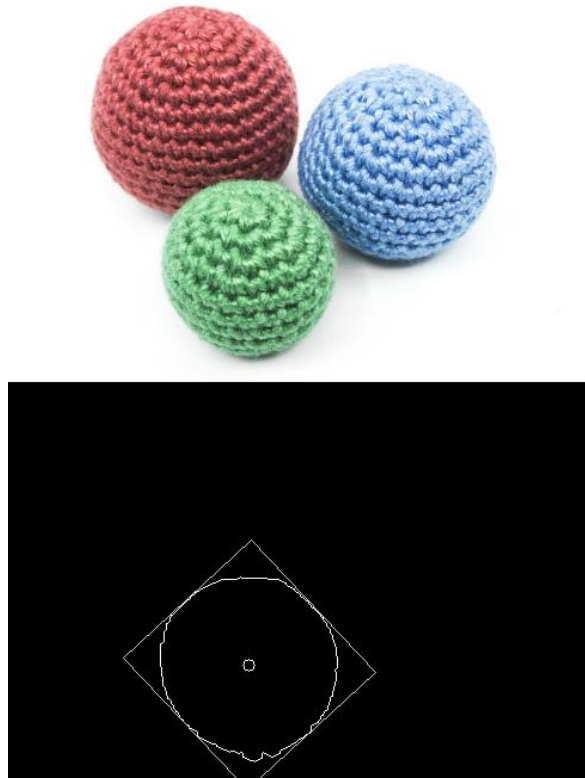


Figure 6 - boîte englobante et centroïd

9 Flou

18) Peut-on réduire la présence des artefacts de la question 8 ? Tester les 3 types de flous vus en cours.

Fonctions :

`CvInvoke.Blur()`

`CvInvoke.MedianBlur()`

`CvInvoke.GaussianBlur()`

10 Bonus

19) Tester votre détecteur sur d'autres images. Fonctionne t'il ? quel sont les modifications à apporter. Quel sont les difficultés observées ?

L'espace HSV n'est pas le seul espace couleur. Certains espaces ont été créé pour être invariants à certains types de lumières. On peut citer en particulier les espaces l1l2l3 invariant aux reflets spéculaires et C1C2C3 invariant aux ombres sur les objets mat.

$$C1 = \arctan (R / \text{Max}(B, G))$$

$$C2 = \arctan (G / \text{Max}(B, R))$$

$$C3 = \arctan (B / \text{Max}(R, G))$$

20) Implémenter cet espace. Tester avec différentes images. Cet espace semble-t-il efficace ?
Convertir l'image hsv.png dans l'espace C1C2C3. Que remarquez-vous ?