# Projet
# A la découverte d'OpenCV

*Professeur :* Valentin Gies

# Table des matières



# 1 Installation

$\Longrightarrow$ Téléchargez en installez Python dans la dernière version si ça n'est pas le cas sur votre ordinateur. **ATTENTION, activez l'option add to PATH** durant l'installation : Lien vers la page de téléchargement de Python pour Windows

$\Longrightarrow$ Depuis le terminal command de Windows (accessible en tapant "cmd" dans la barre d'outils), téléchargez et installez OpenCV pour Python à l'aide de la commande :

```
pip3 install opencv-python
```

$\Longrightarrow$ Installez de la même manière le package matplotlib.

$\Longrightarrow$ OpenCV pour Python doit à présent être installé sur votre ordinateur. Pour cela lancer Idle (l'éditeur de texte par défaut de Python), depuis le menu Démarrer de Windows, puis créez un fichier Python nommé "test.py" dans le répertoire de votre choix, de préférence dans votre repo GitHub que vous aurez au préalable importé. Dans ce fichier, insérez le code suivant, puis apuyez sur F5 pour le tester :

```
import cv2
print(cv2.__version__)
```

Normalement, nous devriez voir dans la console Python la version de OpenCv installée s'afficher.

# 2 Lire une image avec OpenCV

OpenCV supporte les type d'images classiques tels que JPG, BMP, PNG, TIFF.

$\Longrightarrow$ Vous allez ouvrir une image située sur un serveur distant. Pour cela ajouter à votre fichier de test le code ci-dessous. Si jamais un proxy vous bloque la connexion internet, téléchargez directement l'image sur le site distant, mettez là dans votre répertoire de travail et ouvrez la avec le code alternatif :

```
import numpy as np
from urllib.request import urlopen
req = urlopen('http://www.vgies.com/downloads/robocup.png')
arr = np.asarray(bytearray(req.read()), dtype=np.uint8)
img = cv2.imdecode(arr, -1)
cv2.imshow('RoboCup image', img)
cv2.waitKey(0)
```

Code alternatif (si problème de proxy)

```
import numpy as np
img = cv2.imread("robocup.png")
cv2.imshow('RoboCup image', img)
cv2.waitKey(0)
```

$\Longrightarrow$ Vous allez à présent splitter l'image RGB en 3 images séparées à l'aide de la commande *split*.

```
B, G, R = cv2.split(img)
cv2.imshow("original", img)
cv2.waitKey(0)

cv2.imshow("blue", B)
cv2.waitKey(0)

cv2.imshow("Green", G)
cv2.waitKey(0)

cv2.imshow("Red", R)
cv2.waitKey(0)
```

$\Longrightarrow$ Passez l'image dans l'espace HSV à l'aide de la commande *cvtColor*, avec l'option *COLOR_BGR2HSV*, en ajoutant le code suivant à votre fichier de test. Commentez les résultats obtenus et leur différence par rapport à la représentation RGB. Il est à noter qu'en OpenCV, Hue est codée entre 0° et 180°, Saturation et Value étant codés entre 0 et 255.

```
#converting the image to HSV color space using cvtColor function
imagehsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

H, S, V = cv2.split(imagehsv)

cv2.imshow("Hue", H)
cv2.waitKey(0)

cv2.imshow("Saturation", S)
cv2.waitKey(0)

cv2.imshow("Value", V)
cv2.waitKey(0)
```

$\Longrightarrow$ Vous allez à présent filtrer l'image HSV de manière à n'en sortir que la partie jaune correspondant à la balle à l'aide du code ci-dessous. Vous pouvez si vous le souhaitez supprimer les affichages de certaines images obtenues précédemment en commentant le code destiné à les afficher.

```
#Definition des limites basses et hautes de la couleur jaune en HSV
#A noter que le jaune se situe vers les 25 degres dans la roue de couleur HSV en H
lower_yellow = np.array([20, 100, 100])
upper_yellow = np.array([30,255,255])
#Masquage de l'image HSV pour ne garder que les zones jaunes
imagemaskyellow = cv2.inRange(imagehsv, lower_yellow, upper_yellow)
cv2.imshow("Image Masque Jaune", imagemaskyellow)
cv2.waitKey(0)
```

$\Longrightarrow$ Expliquez le fonctionnement de ce code et filtrez à présent par vous même les zones vertes du terrain dans l'image (cf. Fig. 1), puis les zones combres coorespondant aux robots sombres dans l'image.

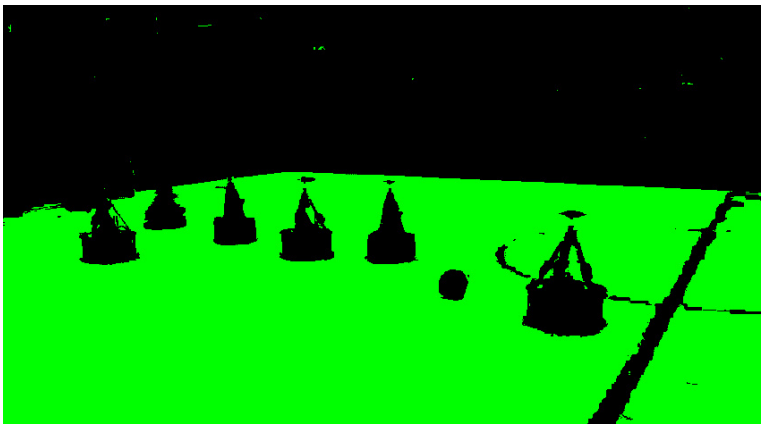

FIGURE 1 – Image uniforme des zones vertes contenus dans l'image d'origine

⟹ Vous allez à présent générer une image regroupant les 3 images à zones uniformes précédentes. Vous devriez obtenir une image ressemblant à celle de la figure 2. Pour cela vous devrez utiliser les fonctions de combinaison booléennes (bitwise_and par exemple) entre les masks obtenus précédemment et des images uniformes de couleur à générer par vous même. Vous pouvez vous aider d'outils en ligne pour cela.

FIGURE 2 – Image combinée

## 3 Transformation manuelle d'une image

⟹ Vous allez à présent apporter des tranformations manuelles à une image à l'aide du code ci-dessous.

```
#TRANSFORMATIONS MANUELLES D'UNE IMAGE
height = img.shape[0]
width = img.shape[1]
channels = img.shape[2]

imgTransform = img

for x in range(0, (int)(width/2)):
        for y in range (0, (int)(height/2)):
                imgTransform[y,x][0] *= 0.5
                imgTransform[y,x][1] *= 0.5
                imgTransform[y,x][2] *= 0.5
```

```
cv2.imshow("Transformation manuelle de l'image", imgTransform)
cv2.waitKey(0)
```

⟹ Exécutez et commentez le code

⟹ Extrayez la composante verte de l'image à la main. Validez avec le professeur

⟹ Appliquez à présent un gradient de niveau de gris sur l'ensemble de l'image de gauche à droite. Validez avec le professeur.

⟹ Plus difficile, colorez un cercle en rouge au centre de l'image en gardant une transparence avec l'image d'origine en fond. Validez avec le professeur.

# 4 Histogrammes avec OpenCV

⟹ A l'aide du code ci-dessous, affichez l'histogramme et l'histogramme cumulé sur un même graphe

```
# importing library for plotting
from matplotlib import pyplot as plt


#Conversion de l'image en niveaux de gris
imageGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
cv2.imshow('Grayscale', imageGray)

#Calcul de l'histogramme
hist,bins = np.histogram(imageGray.flatten(),256,[0,256])
#Calcul de l'histogramme cumule
cdf = hist.cumsum()
cdf_normalized = cdf * float(hist.max()) / cdf.max()
#Affichage de l'histogramme cumule en bleu
plt.plot(cdf_normalized, color = 'b')
#Affichage de l'histogramme en rouge
plt.hist(imageGray.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdf','histogram'), loc = 'upper left')
plt.show()
```

⟹ Vous allez à présent égaliser l'histogramme et l'image d'origine à l'aide du code ci-dessous :

```
#Egalisation de l'histogramme
imgEqu = cv2.equalizeHist(imageGray)
#Calcul de l'histogramme egalise
histEq,binsEq = np.histogram(imgEqu.flatten(),256,[0,256])
#Calcul de l'histogramme écumul éégalis
cdfEq = histEq.cumsum()
cdfEq_normalized = cdfEq * float(histEq.max()) / cdfEq.max()

#Affichage de l'image egalisee
cv2.imshow('Image éégalise', imgEqu)
cv2.waitKey(0)
plt.clf()
```

```
#Affichage de l'histogramme egalise cumule en bleu
plt.plot(cdfEq_normalized, color = 'b')
#Affichage de l'histogramme egalise en rouge
plt.hist(imgEqu.flatten(),256,[0,256], color = 'r')
plt.xlim([0,256])
plt.legend(('cdfEq','histogramEq'), loc = 'upper left')
plt.show()
cv2.waitKey(0)
```

⟹ Commentez les résultats, et ce que l'égalisation fait apparaitre dans l'image.

⟹ Pour aller plus loin, essayez à présent de :
- Décomposer l'image HSV déjà obtenue (vous l'avez déjà fait au début du TP)
- Egaliser les histogrammes de chacune des composantes H, S et V, ce qui vous donnera 3 images en niveau de gris appelées Heq, Seq et Veq.
- Recombiner ces 3 images en une image HSV, et visualiser cette image.

⟹ Que pouvez dire de l'image générée ? Quel intérêt pourrait-elle avoir ?

# 5 Filtrage linéaire

⟹ Testez le floutage de l'image à l'aide de la fonction *GaussianBlur* et du code suivant :

```
img_blur = cv2.GaussianBlur(img,(3,3),0)
```

⟹ Essayez de changer la taille du noyau de bruit gaussien afin de voir l'effet sur l'image.

⟹ Testez à présent l'extraction de contours à l'aide d'un fitre de Sobel à l'aide du code suivant :

```
sobelx = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=0, ksize=5) # Sobel Edge Detection on the
sobely = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=0, dy=1, ksize=5) # Sobel Edge Detection on the
sobelxy = cv2.Sobel(src=img_blur, ddepth=cv2.CV_64F, dx=1, dy=1, ksize=5) # Combined X and Y Sobel Edge

# Display Sobel Edge Detection Images
cv2.imshow('Sobel X', sobelx)
cv2.waitKey(0)

cv2.imshow('Sobel Y', sobely)
cv2.waitKey(0)

cv2.imshow('Sobel X Y using Sobel() function', sobelxy)
cv2.waitKey(0)
```

⟹ Testez l'impact de la taille du noyau de convolution du bruit gaussien sur le filtrage de Sobel. Que pouvez-vous conclure ?

⟹ Vous allez à présent implanter un filtre de Sobel à la main, en définissant votre noyau de convolution (la matrice que l'on balaie sur toute la grille). Implantez un filtre de type gradient horizontal pour la détection des contours supérieurs à l'aide du code suivant :

```
# Filtrage a l'aide de noyaux de convolution definis manuellement
kernel1 = np.array([[-1, -2, -1],
[0, 0, 0],
[1, 2, 1]])
imgSobelHSup = cv2.filter2D(src=img_blur, ddepth=-1, kernel=kernel1)
```

$\Longrightarrow$ A présent, effectuez l'extraction des contours horizontaux inférieurs, ainsi que latéraux gauche et droits, avant de fusionner le tout dans une seule et même image par opération binaire comme effectué précédemment dans le TP.

$\Longrightarrow$ Implantez à la main les autres types de filtres vus en cours : accentuation (sharpen), lissage (blur), outline et validez leur fonctionnement.

# 6 Filtrage non linéaire

## 6.1 Lissage par filtre médian

$\Longrightarrow$ Vous allez à présent tester la fonction filtre median pour la comparer au filtre de lissage linéaire ou gaussien. Pour cela filtrez votre image à l'aide de la fonction *medianBlur(img, ksize)*.

## 6.2 Lissage par filtre bilateral

$\Longrightarrow$ Vous allez à présent tester la fonction filtre median pour la comparer au filtre de lissage linéaire ou gaussien. Pour cela filtrez votre image à l'aide de la fonction *bilateralFilter(img, d, sigmaColor, sigmaSpace)*, où d est le diamètre du noyau de filtrage, sigmaSpace l'étendue spatiale du noyau (type gaussienne), et sigmaColor la tolérance indicative de couleur.
Vous pouvez tester le filtre pour différentes valeurs des coefficients sigma. Commentez.

$\Longrightarrow$ Comparez les différentes méthodes de lissage et leurs performances.

## 6.3 Détection de contours par filtre de Canny

$\Longrightarrow$ Vous allez à présent essayer une autre technique d'extraction de contours, appelée filtre de Canny. Cette méthode repose sur le fait de comparer les magnitudes des gradients obtenus par le filtrage de Sobel afin de dire si ils sont forts (supérieurs à un seuil et donc correspondant à des vrais contours), faibles (inférieurs à un seuil et ne correspondant à rien), ou intermédiaires (devent forts si ils sont rattachés à du fort et supprimés sinon). Implantez le code suivant pour cela :

```
# Detection de contours par filtre de Canny
edges = cv2.Canny(image=img_blur, threshold1=100, threshold2=200)

cv2.imshow('Canny␣Edge␣Detection', edges)
cv2.waitKey(0)
```

$\Longrightarrow$ Faites varier les seuils bas et haut, et commentez les effets. Que pouvez-vous dire du filtre de Canny par rapport au filtre de Sobel ?

# 7 Operations morpho-mathématiques

Dans cette partie, vous allez découvrir des opérations morpho-mathématiques.

⟹ Implantez le code ci-dessous et testez-le :

```
#EROSION - DILATION
# Utilisation d'une matrice de 1 de taille 5x5 comme noyau
kernel = np.ones((5, 5), np.uint8)
img_erosion = cv2.erode(img, kernel, iterations=1)
img_dilation = cv2.dilate(img, kernel, iterations=1)

cv2.imshow('Input', img)
cv2.imshow('Erosion', img_erosion)
cv2.imshow('Dilation', img_dilation)

cv2.waitKey(0)
```

⟹ Commentez les résultats : que fait ce type d'opérations, à quoi peut-il servir ?