

# TP VISION Master SII USTHB – 2024/2025

<https://www.eroaya.com/vision/>

abada.lyes@gmail.com

lyes\_abada@yahoo.fr

# Fonctions de base OpenCV

**TP1 : Fonctions de base OpenCV**

**TP2 : Histogrammes,**

**TP3 : Lissage ( filtres moyen et médian ),**

**TP4 : Binarisation & Seuillage,**

**TP5 : Filtres de convolution,**

**TP6 : Filtres Morphologiques,**

**TP7 : La couleur et La conversion,**

**TP8 : Lire et écrire des flux vidéo,**

**TP9 : ...**

**TP10 : ...**

**TP11 : ...**

# OpenCV

- OpenCV (pour Open Computer Vision) est une bibliothèque graphique libre, initialement développée par Intel, spécialisée dans le traitement d'images en temps réel.
- Importation du package python,

**Import cv2**

# OpenCV

## OpenCV 1.x

- Une image peut être mémorisée à l'intérieur d'une structure d'OpenCV

## Depuis OpenCV 2.x

- la structure de base est la matrice en C et Array en Python.
- Une image peut être considérée comme une matrice de pixel.
- Toutes les opérations de bases des matrices (array en python) sont disponibles

# Fonctions de base OpenCV

## [cv2.imread](#) [cv2.imshow](#)

Dans les applications de vision par ordinateur, les images font partie intégrante du processus de développement.

Souvent, il est nécessaire de lire les images et de les afficher.

Pour lire et afficher une image à l'aide d'OpenCV Python, On peut utiliser :

**cv2.imread()** : pour lire l'image dans une variable.

**cv2.imshow()**: pour afficher l'image dans une fenêtre séparée.

# Fonctions de base OpenCV

## cv2.imread : `imread(filename, flags)`

**Filename** : est le chemin complet de l'image

**Flags** : est un argument optionnel, il indique la représentation de l'image après chargement.

❑ **if `img` is `None` :** (python 3)

OpenCV propose plusieurs valeur pour l'argument flags par exemple :

❑ `cv2.IMREAD_UNCHANGED` = -1

❑ `cv2.IMREAD_GRAYSCALE` = 0

❑ `cv2.IMREAD_COLOR` = 1

La valeur par défaut est 1 c-à-d image en couleurs.

Ex: `img_gray = cv2.imread('test.jpg', cv2.IMREAD_GRAYSCALE)`

# Fonctions de base OpenCV

[cv2.imshow](#) : `imshow(window_name, image)`

**Window\_name** : est le nom de la fenêtre qui affiche l'image

**Image** : l'image à afficher (nom de la variable)

**Ex:**

```
img_gray = cv2.imread('test.jpg', cv2.IMREAD_GRAYSCALE)
```

```
cv2.imshow('grayscale image', img_gray)
```

```
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

Les fonctions **Waitkey** et **destroyAllWindows** sont liées à la fonction **imshow** afin d'attendre une touche de clavier ensuite libérer toutes les fenêtres.

# Fonctions de base OpenCV

## Creation d'une image vide

`np.zeros((height,width,channels), dtype="uint8")`

### Ex:

```
img = np.zeros((h, w,c), np.float32)
```

```
h,w,c = vis.shape
```

# Fonctions de base OpenCV

54	58	255	8	0
45	24	25	214	23
85	124	85	23	55
22	78	25	21	0
52	52	36	127	47

Single Channel Array

54	58	255	8	0					
	45	0	78	51	100	74			
		85	47	34	185	207	21	36	
			22	20	148	52	24	147	123
				52	36	250	74	214	278
					158	0	78	51	247
						72	74	136	251
								74	

3 Channel Arrays

# Fonctions de base OpenCV

## Les types d'images (numpy array)

numpy.uint8

numpy.uint16

numpy.uint32

numpy.uint64

numpy.int8

numpy.int16

numpy.int32

numpy.int64

numpy.float16

numpy.float32

numpy.float64

# Fonctions de base OpenCV

## Dimensions d'une image :

L'attribut shape permet de récupérer la dimension de l'image ainsi le nombre de canaux pour les images couleur

**`h,w = img.shape`**

**`h,w,c = img.shape`** (plusieurs canaux)

## Accès aux pixels :

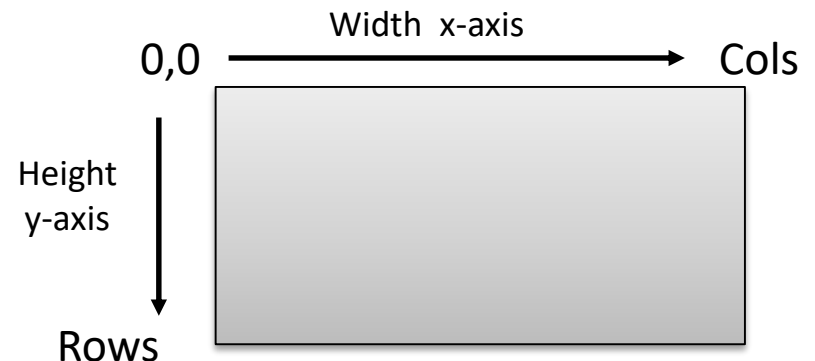
L'accès aux valeurs des pixels se font simplement avec les crochés:

**`ng = img[y,x]`**

**`b,g,r = img[i,j]`**

**`img.item(y,x,c)`**

**`img.itemset((y,x,c),val)`**



# Tp1 exercice

Sous Visual studio:

Créer un nouveau projet :

1. Charger une image quelconque à l'aide de la fonction ***imread*** en niveau de gris dans ***img***.
2. Vérifier si le chargement est fait correctement : **if *img* is None :**
3. Créer une nouvelle image ***imgResultat*** du même taille que ***img***
4. Changer les valeurs des pixels de ***imgResultat*** par les valeurs inverse de ***img*** (255 – niveau de gris de *img*)
5. Afficher ***img*** et ***imgResultat***
6. **Changer le type de l'image (uint32, float64...)**
7. **Debug Avec VSCode**

➔ Même exercice avec image couleur

# TP1 Exemple : *imread*, *imshow*, *imwrite*

```
import cv2
import numpy as np
img = cv2.imread('img/img.png', cv2.IMREAD_GRAYSCALE)
if img is None :
    print('image vide')
    exit(0)
else:
    h,w = img.shape
    imgRes = np.zeros((h,w), np.uint8)
    for y in range(h):
        for x in range (w):
            imgRes[y,x]= 255- img[y,x]
    cv2.imwrite('test.png',imgRes)

    cv2.imshow('image gray',img)
    cv2.imshow('image grayRes',imgRes)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

# TP1 Exemple : *imread, imshow, imwrite*

```
1  import cv2
2  import numpy as np
3  img = cv2.imread("img2.jpg",cv2.IMREAD_COLOR)
4  h,w,c = img.shape
5  #imgRes = np.zeros((h,w),np.uint8)
6  imgRes = np.zeros(img.shape,img.dtype)
7  cv2.resize()
8  imgRes2 = img[0:150,0:200,:]
9
10 if img is None :
11     print('image vide')
12 else:
13     print('image chargée')
14
15 for y in range(h):
16     for x in range (w):
17         imgRes[y,x] = 255 - img[y,x]
18
19 cv2.imwrite("img2.png",imgRes)
20
21 cv2.imshow("image1",img)
22 cv2.imshow("image2",imgRes2)
23 cv2.waitKey(0)
24 cv2.destroyAllWindows()
```

## TP2 : Histogrammes

-L'histogramme associe à chaque niveau de gris, le nombre de pixels ayant ce niveau de gris.

-L'histogramme cumulé : il associe à chaque niveau de gris le pourcentage de pixels qui sont au moins aussi sombre que lui ; donc on aura un petit pourcentage pour la valeur 0, et 100% pour 255 ou avant (ça dépend du niveau de gris maximum dans l'image).

## TP2 : Histogrammes

-L'histogramme normalisé : aussi appelé étirement d'histogramme ça consiste à normaliser le niveau de gris entre 0 et 255, ceci est intéressant notamment pour des images sombres mais l'inconvénient est le bruit qui entachera d'avantage le résultat.

$$> P_{norm} = [(P - P_{min}) * 255] / (P_{max} - P_{min})$$

# TP2 : Histogrammes

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

from numpy.lib.shape_base import vsplit

img = cv2.imread('usthbg.png',cv2.IMREAD_GRAYSCALE)

imgNorm = np.zeros((img.shape),np.uint8)
h,w = img.shape
min = 255
max = 0
for y in range(h):
    for x in range(w):
        if img[y,x] > max:
            max = img[y,x]
        if img[y,x] < min:
            min = img[y,x]

for y in range(h):
    for x in range(w):
        imgNorm[y,x] = (img[y,x]-min)*255/(max-min)

cv2.imshow('imgae source',img)
cv2.imshow('image normal',imgNorm)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

# TP2 : Histogrammes

```
hist1 = np.zeros((256,1),np.uint16)
for y in range(h):
    for x in range(w):
        hist1[img[y,x],0] += 1
hist2 = cv2.calcHist([imgNorm],[0],None,[256],[0,255])
plt.figure()
plt.title('Image normalisee')
plt.xlabel('GrayScale')
plt.ylabel('nbPixels')
plt.plot(hist2)
plt.plot(hist1)
plt.xlim([0,255])
plt.show()
```

# TP2 - Objectif

- Création d'une image utilisant le constructeur `Mat()`;
- Modification des pixels tel que la valeur d'un pixel (x,y) égale à :  
$$(((x + y) * 255 / (rows + cols)) / 2. + 64);$$
- Utilisation de la fonction **imwrite**  
`bool imwrite(const string& filename, InputArray image, const vector<int>&params=vector<int>() )`

# TP2 - Objectif

- Étirement de l'image créée dans l'étape précédente
- Affichage de l'image avant et après l'étirement
- Affichage de l'histogramme de l'image avant et après l'étirement

**Exo 3**

**Examen 2021-2022**

## Exo 3 :Examen 2021-2022

Compléter le programme Python ci-dessous en répondant aux questions suivantes :

1-Ecrire une fonction qui retourne la position du pixel noir dans l'image.

```
1  import cv2
2  import numpy as np
3  from random import randrange
4
5  def createImgWithPointRand(h,w):
6      img = np.ones((heightImg,widthImg),np.float32)
7      #randrange(x) return une valeur aléatoire entre 0 et x
8      randPointY,RandPointX = randrange(heightImg),randrange(widthImg)
9      img[randPointY,RandPointX] = 0
10     return img
11
12
13     heightImg=200
14     widthImg =400
15
16     img = createImgWithPointRand(heightImg,widthImg)
17
18     cv2.imshow('image gray',img)
19
20     '''la fonction waitKey return le code ASCII d'un caractère dans
21     la variable q. Le code ASCII de '0'=48, '1'=49, '2'=50....etc. '''
22     q = cv2.waitKey(0) & 0xFF
23     cv2.destroyAllWindows()
```

## Exo 3 :Examen 2021-2022

2-Ajouter les instructions nécessaires afin de déplacer le pixel noir avec un pas (en nombre de pixels) dans les quatre directions en tapant sur les touches 2, 4, 6 et 8. ('4' : Gauche, '6' Droite, '8' : Haut, '2' : Bas) et '0' pour quitter le programme.

3-Afficher le pixel noir dans l'image avec un carré (5x5 pixels) sans l'utilisation des boucles

4-Sauvegarder l'image après le dernier déplacement avec un type de compression qui garde l'image sans aucun changement.

5-Nous voulons afficher une forme (cercle, ellipse, croix ....) à la place du carré. Quel est le filtre qui permet de réaliser cette opération et comment ? (donner des explications)

> USTHB > cours vision ABADA 2021-2022 > examen vision > examen2022.py > ...

## Exo 3 : Examen 2021-2022

```
import cv2
import numpy as np
from random import randrange

def createImgWithPointRand(h,w):
    img = np.ones((h,w),np.float32)
    randY,randX = randrange(h),randrange(w)
    img[randY,randX] = 0
    return(img)

def findBlackPixel(img):
    h,w = img.shape
    for y in range (h):
        for x in range (w):
            if(img[y,x]==0):
                return (y,x)

heightImg, widthImg = 200,400
img = createImgWithPointRand(heightImg,widthImg)
q= 'a'
pas = 3
(py,px) = findBlackPixel(img)
while(True):
    if 50 == q and py+pas < heightImg :
        img[py,px] = 1
```

## Exo 3 : Examen 2021-2022

```
img[py,px] = 1
img[py+pas,px] = 0
py = py+pas
if 56 == q and py-pas >= 0 :
    img[py,px] = 1
    img[py-pas,px] = 0
    py = py-pas
if 52 == q and px-pas >= 0 :
    img[py,px] = 1
    img[py,px-pas] = 0
    px = px-pas
if 54 == q and px+pas < widthImg :
    img[py,px] = 1
    img[py,px+pas] = 0
    px = px+pas
imgRes = img.copy()
imgRes[py-2:py+2,px-2:px+2] = 0
cv2.imshow("Image vide",imgRes)
q = cv2.waitKey(0) & 0xFF
if ord('0')==q :
    break
```

```
cv2.destroyAllWindows()
```

# TP2\_2 : Histogrammes (vertical et horizontal) code (1/2)

```
1  import cv2
2  import numpy as np
3
4  img = cv2.imread("testb.png", cv2.IMREAD_GRAYSCALE)
5  cv2.threshold(img, 130, 255, cv2.THRESH_BINARY, img)
6
7  h, w = img.shape
8  lignes = np.zeros((h), np.uint16)
9  cols = np.zeros((w), np.uint16)
10
11 for i in range(h):
12     for j in range(w):
13         if(img[i, j] == 0):
14             lignes[i] += 1
15             cols[j] += 1
16 imgLignes = np.zeros(img.shape, np.uint8)
17 imgCols = np.zeros(img.shape, np.uint8)
18 imgLignes[:, :] = 255
19 imgCols[:, :] = 255
20
```

# TP2\_2 : Histogrammes (vertical et horizontal) code (2/2)

```
for i in range(h):
    for j in range(lignes[i]):
        imgLignes[i,j]=0

for j in range(w):
    for i in range(cols[j]):
        imgCols[i,j]=0

#cv2.imshow("image source",img)
cv2.imshow("image Lignes",cv2.hconcat([img,imgLignes]))
cv2.imshow("image Cols",cv2.vconcat([img, imgCols]))
cv2.waitKey(0)
cv2.destroyAllWindows()
```

# TP2\_2 :Histogrammes (vertical et horizontal)

## Histogramme vertical

image Lignes

### الأساسيات

الطريقة المتبعة في عرض المعلومات هي طريقة الإحصار وفيه نلف وندور حول فكرة واحدة مرات عديدة وهي طريقتي في تعلم أي شئ ولذلك لا تتضايق إن قلت شيئا غير واضح لأنني سأعود إليه ثانيا إن شاء الله.  
بالنسبة للكمبيوتر، ماهي مكوناته؟ أو لا نتفق على الآتي ... الكمبيوتر مثل الإنسان (مع الفارق الشاسع في التشبيه) فكما أن للإنسان جسد (شيء مادي ملموس) ونفس بشرية (شيء معنوي غير محسوس) تعطيه شخصيته المميزة له، فكل كمبيوتر له جسد وشيء مثل النفس التي تعطيه خصائص تميزه عن الأجهزة الأخرى .. وباختصار جسم الجهاز يسمى المكونات المادية أو العتاد Hardware ونفسه هي البرامج software (ما سمعش عن سوفت وير الموبايل؟ بيقلوا السوفت وير ضرب وكلام زي كده؟ الفكرة واحدة وهتشف).

## Histogramme horizontal

image Cols

الطريقة المتبعة في عرض المعلومات هي طريقة الإحصار وفيه نلف وندور حول فكرة واحدة مرات عديدة وهي

**Exo 3**

**Examen 2021-2022**

Compléter le programme Python ci-dessous en répondant aux questions suivantes :

1-Ecrire une fonction qui retourne la position du pixel noir dans l'image.

```
1  import cv2
2  import numpy as np
3  from random import randrange
4
5  def createImgWithPointRand(h,w):
6      img = np.ones((heigthImg,widthImg),np.float32)
7      #randrange(x) return une valeur aléatoire entre 0 et x
8      randPointY,RandPointX = randrange(heigthImg),randrange(widthImg)
9      img[randPointY,RandPointX] = 0
10     return img
11
12
13     heigthImg=200
14     widthImg =400
15
16     img = createImgWithPointRand(heigthImg,widthImg)
17
18     cv2.imshow('image gray',img)
19
20     '''la fonction waitKey return le code ASCII d'un caractère dans
21     la variable q. Le code ASCII de '0'=48, '1'=49, '2'=50....etc. '''
22     q = cv2.waitKey(0) & 0xFF
23     cv2.destroyAllWindows()
```

# TP3 :Lissage : filtres moyen et médian

2-Ajouter les instructions nécessaires afin de déplacer le pixel noir avec un pas (en nombre de pixels) dans les quatre directions en tapant sur les touches 2, 4, 6 et 8. ('4' : Gauche, '6' Droite, '8' : Haut, '2' : Bas) et '0' pour quitter le programme.

3-Afficher le pixel noir dans l'image avec un carré (5x5 pixels) sans l'utilisation des boucles

4-Sauvegarder l'image après le dernier déplacement avec un type de compression qui garde l'image sans aucun changement.

5-Nous voulons afficher une forme (cercle, ellipse, croix ....) à la place du carré. Quel est le filtre qui permet de réaliser cette opération et comment ? (donner des explications)

# TP3 :Lissage : filtres moyen et médian

**Filtre Moyen** : Il consiste à remplacer un pixel par la moyenne des niveaux de gris de son voisinage.

**Filtre Médian** : On remplace le pixel par la médiane de son voisinage après le classement par ordre croissant.

Par exemple pour un voisinage  $3 \times 3$ , le rang de la médiane sera  $(9-1)/2 + 1$ . En général il est préférable au filtre moyen.

# TP3 :Lissage : filtres moyen et médian

```
import cv2
import numpy as np

voisinage = 3
def filtreMoyenNMG(img):
    h,w = img.shape
    imgMoy = np.zeros(img.shape,np.uint8)

    for y in range(h):
        for x in range(w):
            if x<voisinage/2 or x>w-voisinage/2 or y < voisinage/2\
                or y>h-voisinage/2:
                imgMoy[y,x]= img[y,x]
            else:
                imgV = img[int(y-voisinage/2):int(y+voisinage/2)+1,\
                    int(x-voisinage/2):int(x+voisinage/2)+1]
                #moy = 0
                #for yv in range(voisinage):
                #    for xv in range(voisinage):
                #        moy += imgV[yv,xv]
                #moy /= voisinage*voisinage
                imgMoy[y,x]= np.mean(imgV)
    return imgMoy
```

# TP3 :Lissage : filtres moyen et médian

```
> USTHB > cours vision ABADA 2021-2022 > TPs VISION > 2021-2022-py > TP4py > TP4
def filtreMedianNNG(img):
    h,w = img.shape
    imgMed = np.zeros(img.shape,np.uint8)
    for y in range(h):
        for x in range(w):
            if x<voisinage/2 or x>w-voisinage/2 or y < voisinage/2\
                or y>h-voisinage/2:
                imgMed[y,x]= img[y,x]
            else:
                imgV = img[int(y-voisinage/2):int(y+voisinage/2)+1,\
                    int(x-voisinage/2):int(x+voisinage/2)+1]
                #t = np.zeros((voisinage*voisinage),np.uint8)
                #for yv in range(voisinage):
                #    for xv in range(voisinage):
                #        t[yv*voisinage+xv] = imgV[yv,xv]
                #t.sort()
                #imgMed[y,x]= t[(voisinage*voisinage-1)/2]
                imgMed[y,x]= np.median(imgV)
    return imgMed

img = cv2.imread('usthb.jpg',cv2.IMREAD_GRAYSCALE)
imgMoy = filtreMoyenNNG(img)
imgMed = filtreMoyenNNG(img)
cv2.imshow('image source', img)
cv2.imshow('image moy',imgMoy)
cv2.imshow('image med',imgMed)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

# TP4 : Binarisation & Seuillage

La binarisation est la transformation d'une image en image binaire utilisant un seuil, c'ad qui ne contient que deux couleurs : le noir (0) et le blanc (255).

Il existe de très nombreux algorithmes entre autre le seuillage et le seuillage inverse.

Seuillage => si le pixel est plus clair que le seuil il devient blanc sinon il devient noir

C'est le contraire pour le seuillage inverse.

## TP4 : Binarisation & Seuillage

```
cv2.threshold(src, Threshold, max_value ,  
threshold_type, dst);
```

**src**: l'image initiale en niveaux de gris.

**Threshold**: le seuil de binarisation

**max\_value** : la couleur claire (le blanc) = 255 pour une vraie binarisation.

**threshold\_type**: l'algorithme de seuillage utilisé.

**dst** : l'image en niveau de gris dont laquelle on récupère le résultat du seuillage.

# TP4 : Binarisation & Seuillage

**threshold\_type**: l'algorithme de seuillage utilisé.

**THRESH\_BINARY = 0** (seuil binaire)

- Si  $\text{src}(x,y) > \text{seuil}$
- $\text{dst}(x,y) = \text{maxValue}$
- sinon
- $\text{dst}(x,y) = 0$

# TP4 : Binarisation & Seuillage

**threshold\_type**: l'algorithme de seuillage utilisé.

**THRESH\_BINARY\_INV = 1** (seuil binaire inverse)

- Si  $\text{src}(x,y) > \text{seuil}$
- $\text{dst}(x,y) = 0$
- sinon
- $\text{dst}(x,y) = \text{maxValue}$

# TP4 : Binarisation & Seuillage

**threshold\_type**: l'algorithme de seuillage utilisé.

**THRESH\_TRUNC = 2** (Troncature du seuillage )

- Si  $\text{src}(x,y) > \text{seuil}$
- $\text{dst}(x,y) = \text{seuil}$
- sinon
- $\text{dst}(x,y) = \text{src}(x,y)$

# TP4 : Binarisation & Seuillage

**threshold\_type**: l'algorithme de seuillage utilisé.

**THRESH\_TOZERO = 3** (Seuil à zéro)

- Si  $\text{src}(x,y) > \text{seuil}$
- $\text{dst}(x,y) = \text{src}(x,y)$
- sinon
- $\text{dst}(x,y) = 0$

# TP4 : Binarisation & Seuillage

**threshold\_type**: l'algorithme de seuillage utilisé.

**THRESH\_TOZERO\_INV = 4** (Seuil à zéro inverse)

- Si  $\text{src}(x,y) > \text{seuil}$
- $\text{dst}(x,y) = 0$
- sinon
- $\text{dst}(x,y) = \text{src}(x,y)$

## TP4 : Binarisation & Seuillage

```
cv2.createTrackbar( (trackbar_name, window_name,  
value, count, on_change );
```

**trackbar\_name**: le nom de la trackbar

**window\_name**: le nom de la fenêtre

**value** : un pointeur sur le paramètre entier que la trackbar va piloter.

**count** : la valeur maximale du paramètre (minimum est toujours 0)

**on\_change** : la fonction de callback .

## TP4 : Binarisation & Seuillage

une trackbar est la barre qui nous permet de choisir la valeur d'un paramètre entier en faisant glisser le curseur avec la souris.

A chaque modification une fonction est déclenchée

void **on\_change** (int):

Pour créer une trackbar et l'associer à une fenêtre on utilise la fonction : **cv2.createTrackbar**

# TP4 : Binarisation & Seuillage

```
D: > D > USTHB > cours vision ABADA 2021-2022 > TPs VISION > 2021-2022-py > TP4py > TP4.py >
1  import cv2
2  import numpy as np
3  img = cv2.imread('usthb.jpg',cv2.IMREAD_GRAYSCALE)
4  imgRes = np.zeros(img.shape,np.uint8)
5  th = 0
6  type = 0
7  #0: Binary # 1: Binary Inverted # 2: Threshold Truncated-
8  # 3: Threshold to Zero # 4: Threshold to Zero Inverted
9  def afficher():
10     cv2.threshold(img,th,255,type,imgRes)
11     cv2.imshow('result',imgRes)
12  def changeTh(x):
13     global th
14     th = x
15     afficher()
16  def changeType(x):
17     global type
18     type = x
19     afficher()
20  cv2.namedWindow('result')
21  cv2.createTrackbar('threshold','result',0,255,changeTh)
22  cv2.createTrackbar('type','result',0,4,changeType)
23  afficher()
24  cv2.waitKey()
25  cv2.destroyAllWindows()
```

# TP5 : Filtrés de convolution :

Les filtres le plus connus sont :

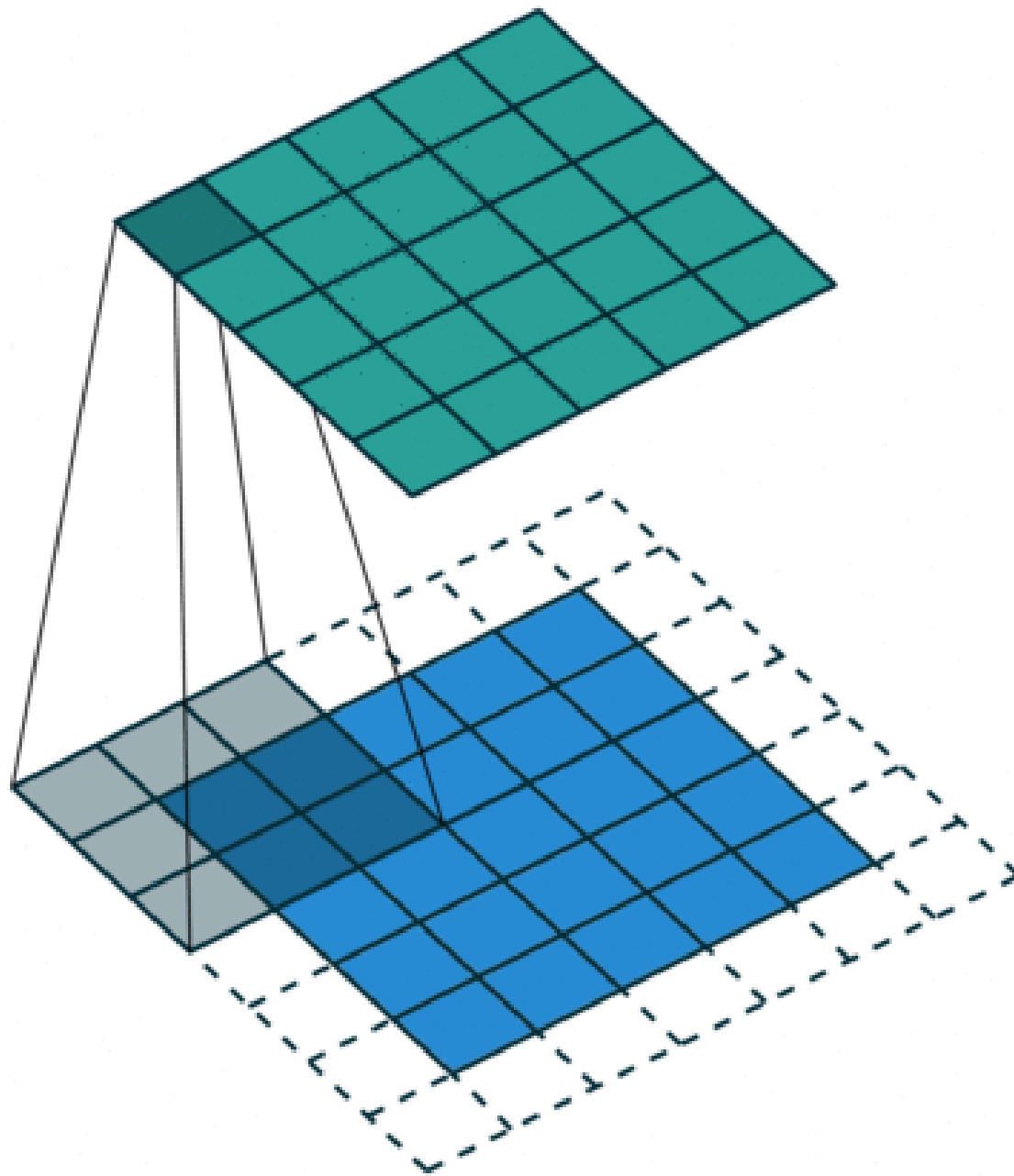
100	100	100	100	100
100	100	100	100	100
100	100	150	100	100
100	100	100	100	100
100	100	100	100	100

\*

0	-1	0
-1	5	-1
0	-1	0

=

100	100	100	100	100
100	100	50	100	100
100	50	350	50	100
100	100	50	100	100
100	100	100	100	100



# TP5 : Filtres de convolution :

## Le filtre Gaussien :

Le filtre Gaussien rend les images floues mais il altère moins possible les détails et les contours.

On attribue au pixel un poids d'autant plus grand qu'il est Proche du centre du masque.

$1/81$

$$\begin{bmatrix} 2 & 4 & 6 & 4 & 2 \\ 3 & 6 & 9 & 6 & 3 \\ 2 & 4 & 6 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix}$$

# TP5 : Filtres de convolution :

## **Le filtre Gaussien :**

Il faut diviser les coefficients du masque de convolution  
Par leur somme pour que l'image du résultat soit bien  
normalisée ;

les niveaux de gris dans l'intervalle  $[0,255]$ .

Un masque de convolution conserve la dynamique de  
l'image càd la somme des coefficients =1 et les valeurs  
sont toutes  $\geq 0$ .

## TP5 : Filtres de convolution :

**Le filtre Laplacien :**

Il ne préserve pas la dynamique de l'image.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

En appliquant le filtre Laplacien, on trouve des valeurs négatives et d'autres supérieurs à 255.

# TP5 : Filtres de convolution :

## Le filtre Laplacien :

Leur approximation par 0 et 255 fera perdre détails de l'image et obtient des contours de blanc sur noir.

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Il est nécessaire de faire une normalisation appelée compression d'histogramme (même principe que la normalisation d'histogramme).

# TP5 : Filtres de convolution :

**filter2D(**

InputArray src,

int ddepth, // le nombre de bits de l'image résultat(8,16...)

InputArray kernel, // la matrice de convolution

Pointanchor = (-1, -1), // centre de la mat conv

//(-1,-1) centre par défaut

double delta = 0, //valeur ajouté au pixel apres Conv

int borderType=4 // BORDER\_REPLICATE, BORDER\_CONSTANT,  
BORDER\_REFLECT\_101, BORDER\_WARP, BORDER\_TRANSPARENT,  
BORDER\_DEFAULT, BORDER\_ISOLATED

)

## TP5 : Filtres de convolution :

```
1 import numpy as np
2 import cv2
3
4 img = cv2.imread("test.png", cv2.IMREAD_GRAYSCALE)
5 cv2.threshold(img, 125, 255, 1, img)
6
7 #kernel = np.array([[1, 2, 1],[2, 4, 2],[1, 2, 1]])
8 #kernel = kernel / 16
9
10 kernel = np.array([[0, -1, 0],[-1, 4, -1],[0, -1, 0]])
11
12 #resulting_image = cv2.filter2D(img, -1, kernel, anchor=(-1, -1), \
13 # delta=0, borderType=cv2.BORDER_REPLICATE)
14
15 resulting_image = cv2.filter2D(img, -1, kernel)
16
17 cv2.imshow("original image", img)
18 cv2.imshow("filter2d image", resulting_image)
19 cv2.imwrite("Filter2d Sharpened Image.jpg", resulting_image)
20 cv2.waitKey()
21 cv2.destroyAllWindows()
```

## TP6 : Filtres Morphologiques :

En traitement d'images, on utilise des opérateurs morphologiques pour effectuer certains traitements sur l'image binaire.

Ces opérateurs utilisent un élément structurant (un carré, un cercle, un croix, etc..)  $3 \times 3$  ou  $5 \times 5$

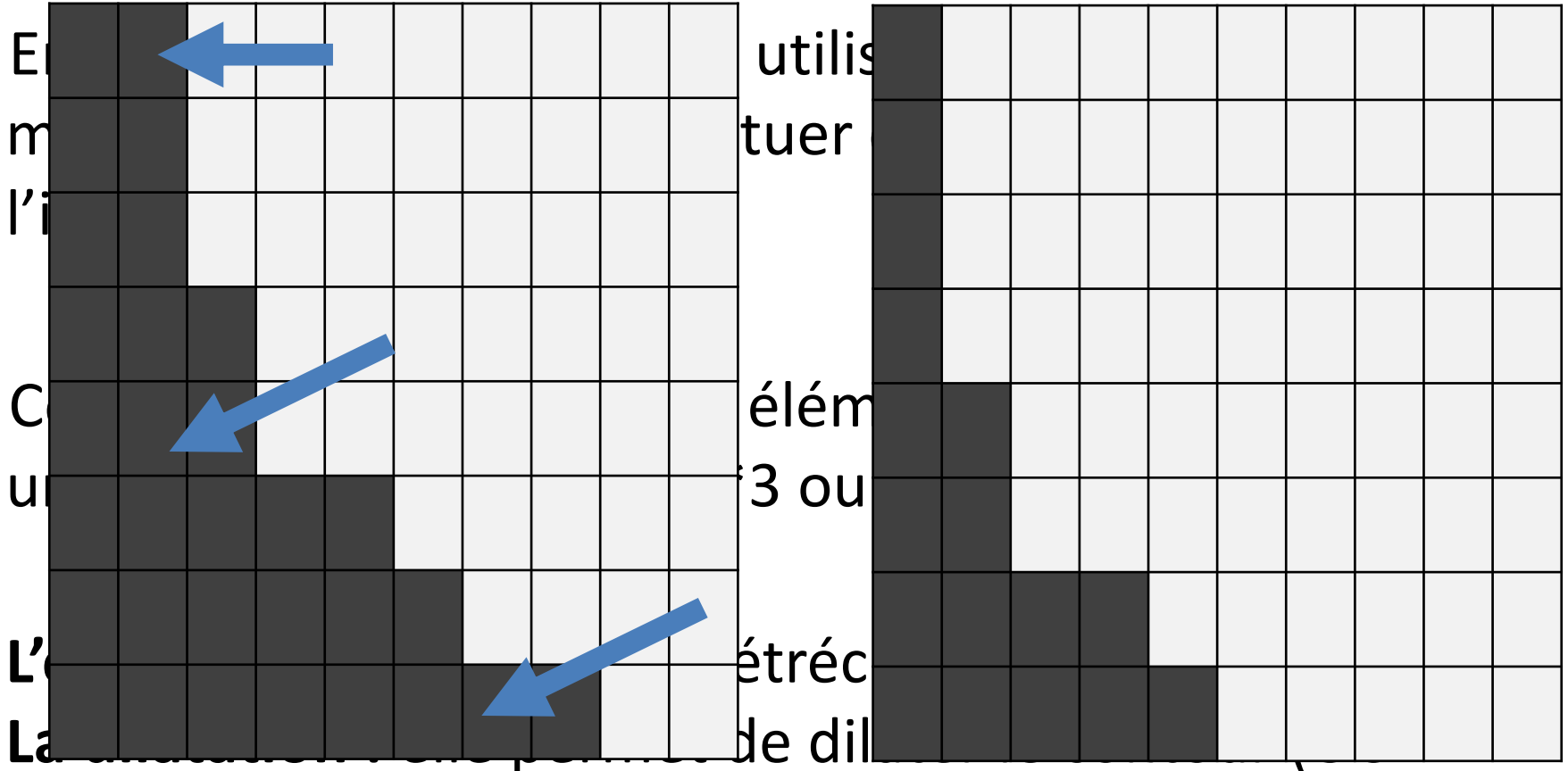
**L'érosion** : elle permet de rétrécir le contour (ET logique)

**La dilatation** : elle permet de dilater le contour (OU logique)

TP

# L'érosion

S :

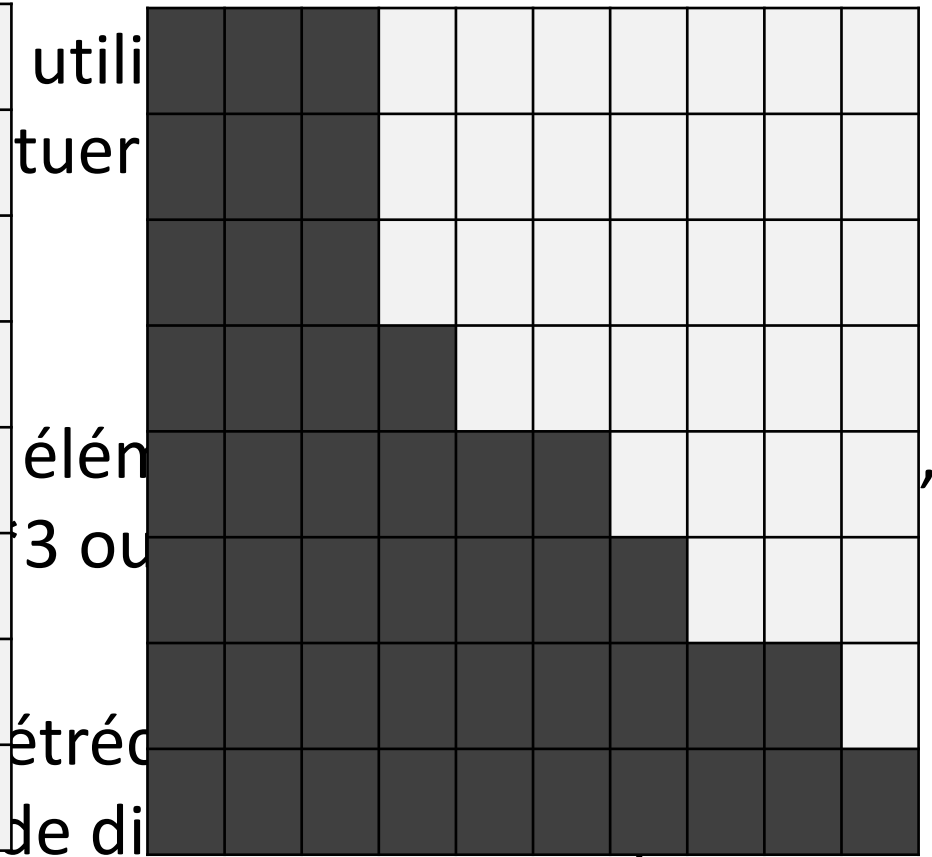
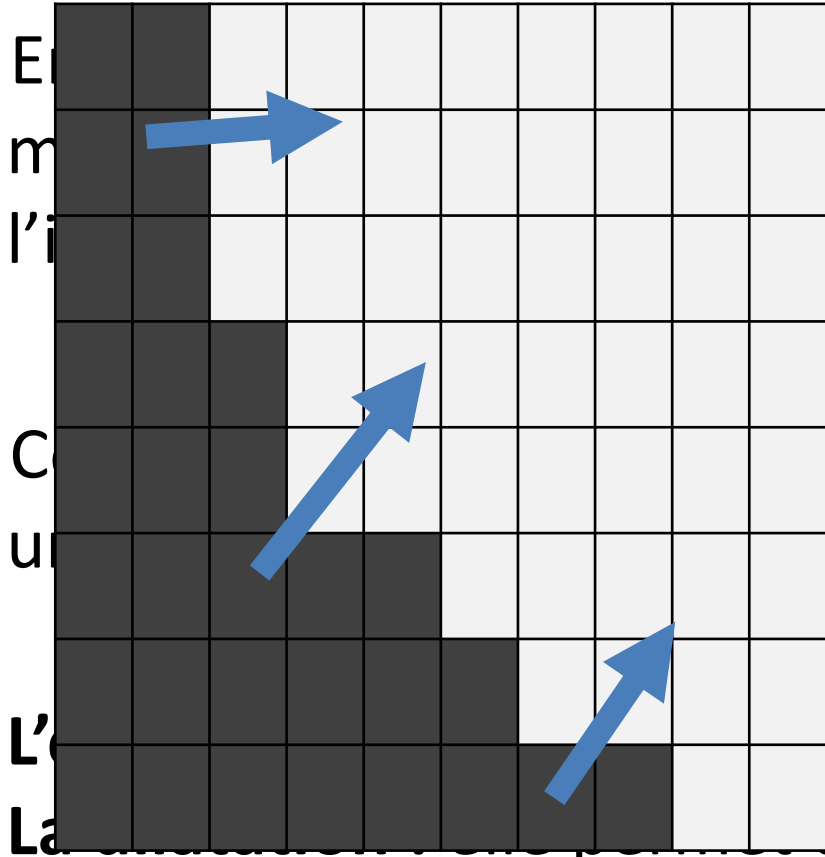


logique)

TP

# La dilatation

S :



logique)

## TP6 : Filtres Morphologiques :

**L'ouverture** : c'est une érosion suivie d'une dilatation.

- L'ouverture permet de supprimer les pixels du bruit et d'adoucir les bords.

**La fermeture** : c'est le contraire de l'ouverture, donc une dilatation puis une érosion ;

- elle permet de combler les vides.

**Le gradient** : il permet de dégager le contour des objets ;

## TP6 : Filtres Morphologiques :

Pour créer un élément structurant, on utilisera la fonction:  
getStructuringElement(shape, Size, Anchor);

MORPH\_RECT=0

size(3,3):

```
[[1 1 1]
 [1 1 1]
 [1 1 1]]
```

size(5,5):

```
[[1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]]
```

MORPH\_CROSS=1

size(3,3):

```
[[0 1 0]
 [1 1 1]
 [0 1 0]]
```

size(5,5):

```
[[0 0 1 0 0]
 [0 0 1 0 0]
 [1 1 1 1 1]
 [0 0 1 0 0]
 [0 0 1 0 0]]
```

MORPH\_ELLIPSE=2

size(3,3):

```
[[0 1 0]
 [1 1 1]
 [0 1 0]]
```

size(5,5):

```
[[0 0 1 0 0]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [0 0 1 0 0]]
```

# TP6 : Filtres Morphologiques :

Pour l'érosion et la dilatation on utilise les fonctions :

**erode**(src, dst, Kernel,...);      **dilate**(src, dst, Kernel, ...);

Pour les autres opérateurs, on utilise la fonction :

**morphologyEx**(src,dst,op,kernel,...)

- **Op** :

MORPH\_OPEN = 2;

MORPH\_CLOSE = 3;

MORPH\_GRADIENT = 4;

MORPH\_TOPHAT = 5 (image – ouverture)

MORPH\_BLACKHAT = 6 (fermeture - image)

- **Kernel** : shape récupéré par la fonction : **getStructuringElement**

# TP6 : Filtres Morphologiques : (code 1/2)

```
1 import cv2
2 import numpy as np
3
4 sizeDelate = 1
5 sizeErode = 1
6 cv2.namedWindow("Erosion")
7 cv2.namedWindow("Dilatation")
8 img = cv2.imread("testb.png",cv2.IMREAD_GRAYSCALE)
9 cv2.threshold(img,130,255,0,img)
10
11 def delate_func():
12     #kernel = np.ones((sizeDelate,sizeDelate),np.uint8)
13     kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(sizeDelate*2+1,sizeDelate*2+1))
14     img_delate = cv2.dilate(img,kernel,iterations=1)
15     #img_delate = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
16     cv2.imshow("Dilatation",img_delate)
17 def erode_func():
18     #kernel = np.ones((sizeErode,sizeErode),np.uint8)
19     kernel = cv2.getStructuringElement(cv2.MORPH_CROSS,(sizeErode*2+1,sizeErode*2+1))
20     img_erode = cv2.erode(img,kernel,iterations=1)
21     cv2.imshow("Erosion",img_erode)
```

# TP6 : Filtres Morphologiques : (code 2/2)

```
def changeEsize(x):
    global sizeErode
    sizeErode = x
    erode_func()

def changeDsize(x):
    global sizeDelate
    sizeDelate = x
    delate_func()

cv2.createTrackbar("Size Erode", "Erosion", 0, 21, changeEsize)
cv2.createTrackbar("Size Delate", "Dilatation", 0, 21, changeDsize)
erode_func()
delate_func()

cv2.imshow("image source", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

## TP7 : La couleur et La conversion :

Un espace de couleur est un modèle mathématique abstrait représenté par un tuple de couleurs (3 ou 4 valeurs) appelé composantes.

Deux sont utilisés de manière majoritaires : RVB et TSV (RGB et HSV :Hue (teinte), saturation, value).

**Le RVB** est énormément utilisé en informatique, mais en vision il possède un grand défaut en décrivant simultanément la luminance.

# TP7 : La couleur et La conversion :

**HSV** les décrit séparément ce qui permet de reconnaître les objets par leur couleur indépendamment de la luminance.

## 1- La Teinte (Hue) :

La teinte est codée suivant l'angle qui lui correspond sur le cercle des couleurs :

0° ou 360° : rouge ;

60° : jaune ;

120° : vert ;

180° : cyan ;

240° : bleu ;

300° : magenta.



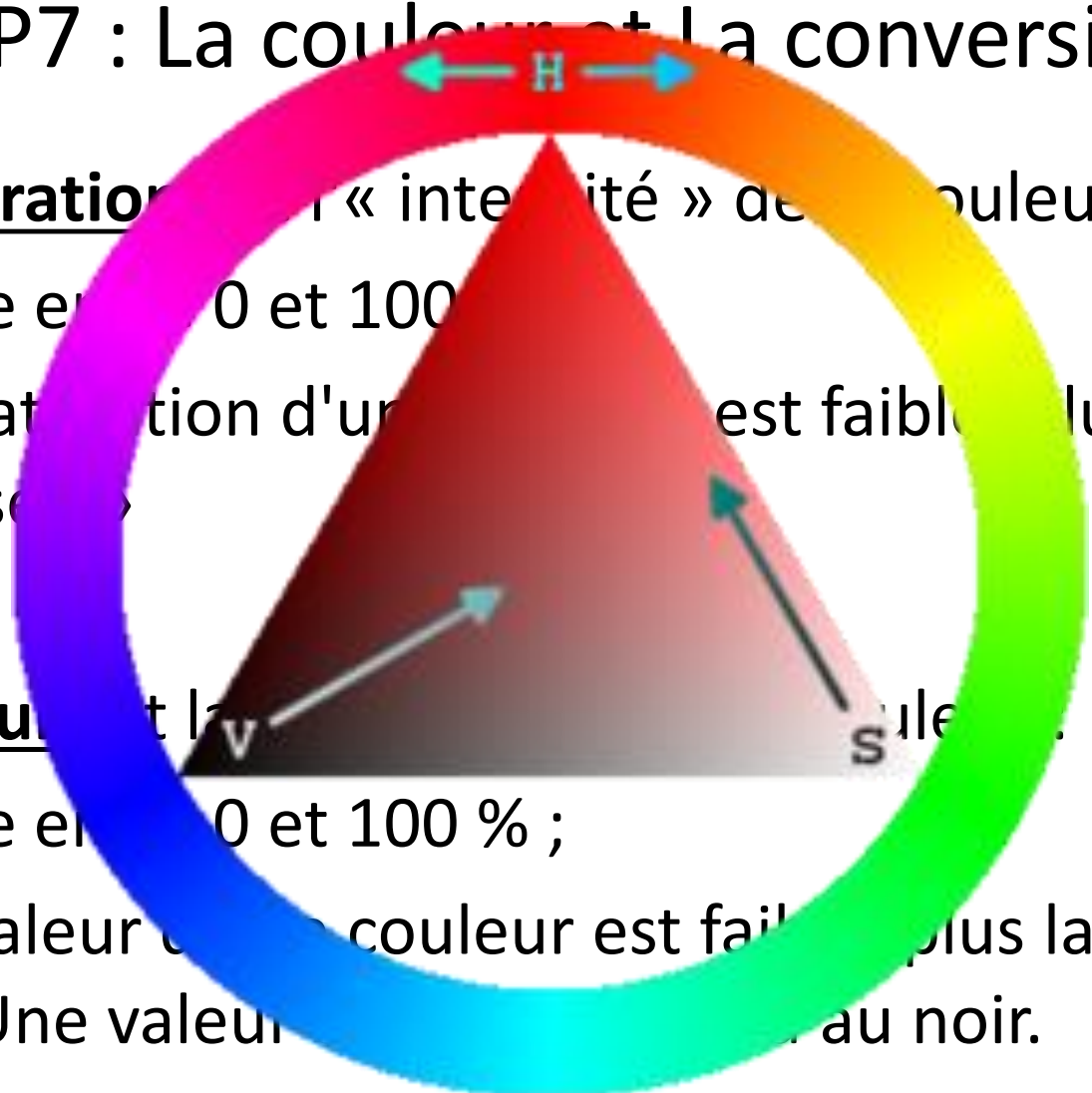
# TP7 : La couleur et La conversion :

## 2- La saturation est la « intensité » de la couleur :

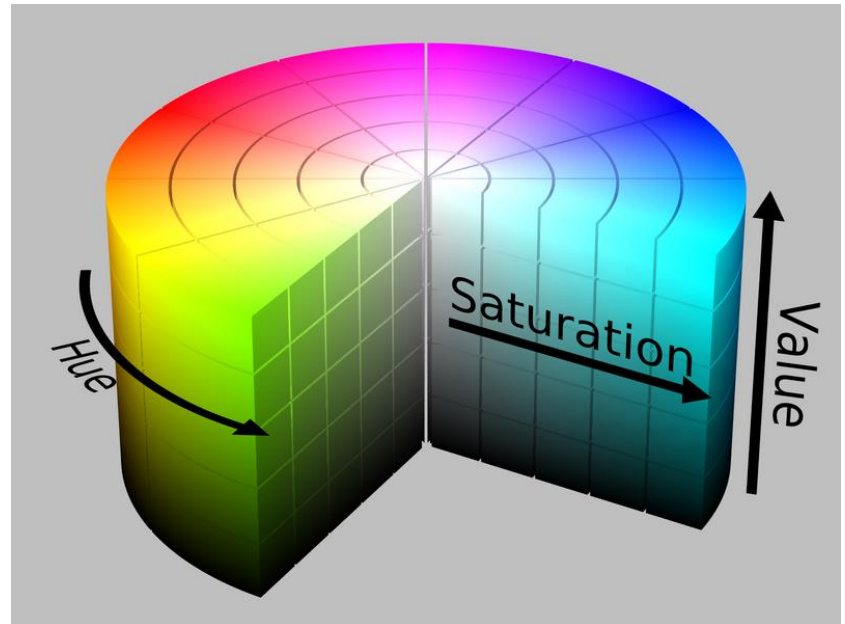
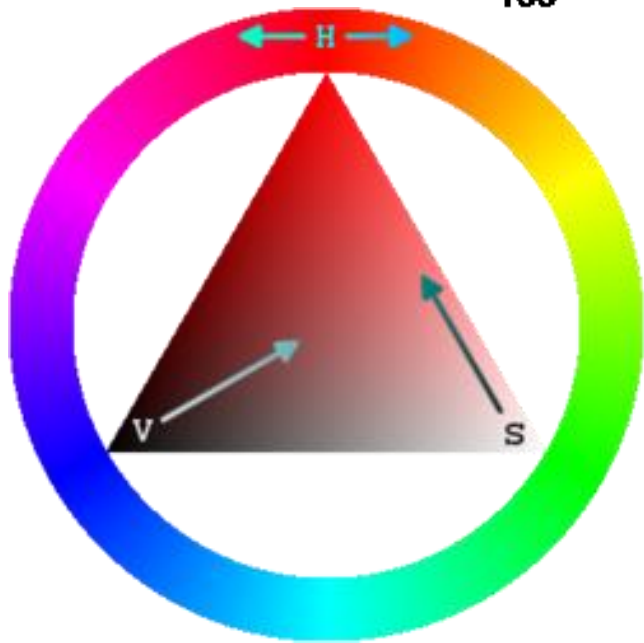
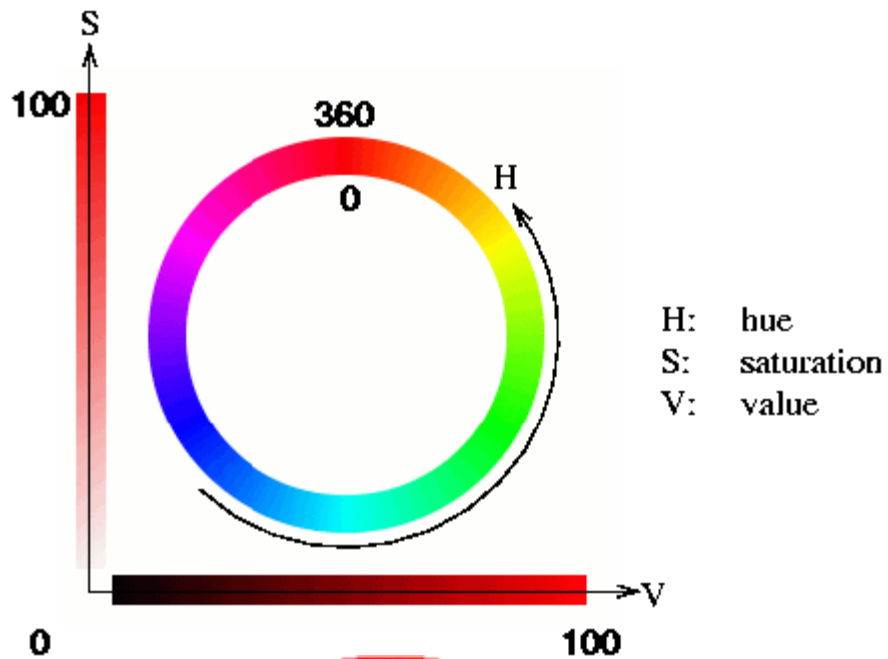
- elle varie entre 0 et 100 % ;
- plus la saturation d'une couleur est faible, plus l'image sera « grise » ;

## 3- La valeur est la « luminosité » de la couleur :

- elle varie entre 0 et 100 % ;
- plus la valeur d'une couleur est faible, plus la couleur est sombre. Une valeur de 0 correspond au noir.



# TP7 : La couleur et La conversion :



# TP7 : La couleur et La conversion :

Dans OpenCV, les couleurs sont par défaut en BGR (les canaux bleu et rouge sont inversés).

Pour afficher les images de manière naturelle, il faut les convertir vers le BGR.

**HSV Sous OpenCV :**

**H : 0–180,**

**S : 0–255,**

**V : 0–255**

# TP7 : La couleur et La conversion :

[La conversion des couleurs cvtColor\(src, code, dst\);](#)

Code:      **COLOR\_<type src>2<type dst>**

Ex:              COLOR\_BGR2GRAY      COLOR\_HSV2BGR  
                  COLOR\_BGR2RGB      COLOR\_HSV2RGB  
                  COLOR\_BGR2HSV      COLOR\_RGB2HSV

[Conversion de type :](#)

uint8(img)      0 →  $2^8$

uint16(img)     0 →  $2^{16}$

float32(img)    0 → 1

float64(img)    0 → 1

# TP7 : La c

## La conversion des coul

Code: **COLOR\_<typ**

Ex: COLOR\_

COLOR\_

COLOR\_

## Conversion de type :

uint8(img)  $0 \rightarrow 2^8$

uint16(img)  $0 \rightarrow 2^{16}$

float32(img)  $0 \rightarrow 1$

float64(img)  $0 \rightarrow 1$

D: > D > USTHB > cours vision ABADA 2021-2022 > TPs VISION > 2022-202

```
1 import cv2
2 import numpy as np
3
4 img = cv2.imread('usthb.jpg',cv2.IMREAD_COLOR)
5 if img is None :
6     print("erreur de chargement")
7     exit(0)
8 img_b = np.zeros(img.shape,img.dtype)
9 img_g = np.zeros(img.shape,img.dtype)
10 img_r = np.zeros(img.shape,img.dtype)
11 img_hsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
12
13 img_b[:, :, 0] = img[:, :, 0]
14 img_g[:, :, 1] = img[:, :, 1]
15 img_r[:, :, 2] = img[:, :, 2]
16
17 img_float32 = np.float32(img)/255
18
19 cv2.imshow("img",img)
20 cv2.imshow("img_32",img_float32)
21 #cv2.imshow("img_b",img_b)
22 #cv2.imshow("img_g",img_g)
23 #cv2.imshow("img_r",img_r)
24
25 cv2.waitKey(0)
26 cv2.destroyAllWindows()
27
```

## TP8 : Lire et écrire des flux vidéo:

Un flux vidéo est une succession d'images qui ont été prises à des intervalles de temps réguliers, il peut venir d'un fichier vidéo ou d'une caméra.

les images sont appelées des trames.

La période de temps  $T$  qui sépare deux trames est la période d'échantillonnage.

La fréquence ( $f$ ) est appelée frame rate :  $f=1/T$

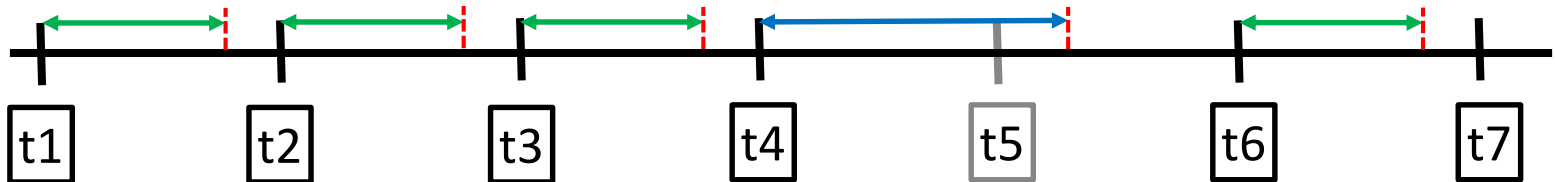
# TP8 : Lire et écrire des flux vidéo:

Les modes de lecture :

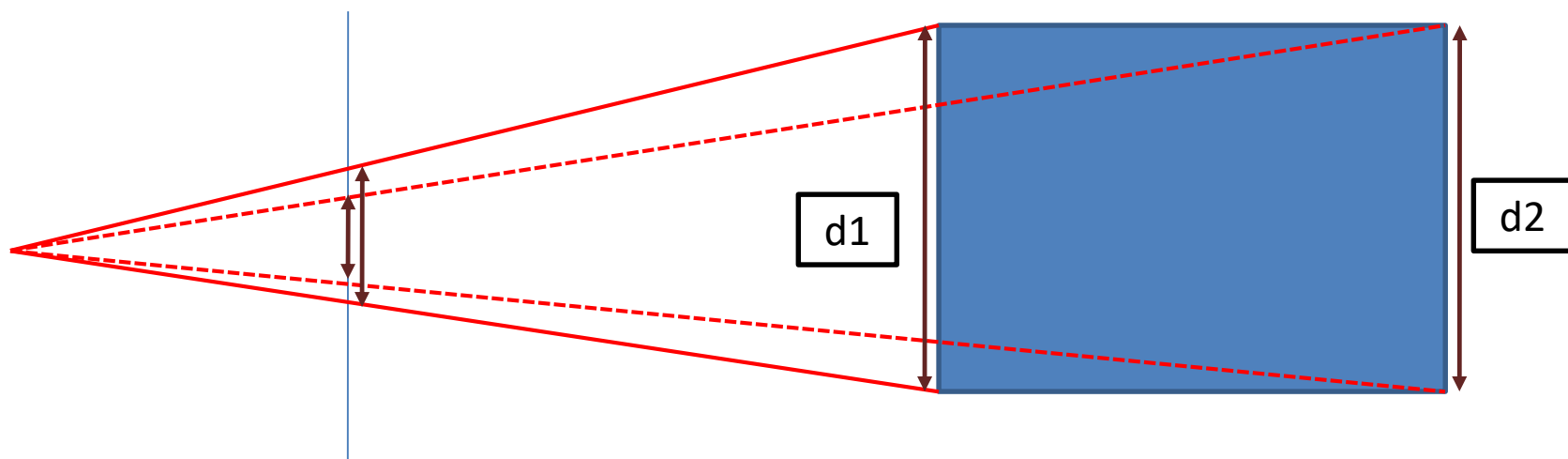
- La lecture exhaustive est la plus facile à mettre en œuvre ; on traite toutes les frames sans se préoccuper du temps du traitement (utile pour modifier un fichier dans son intégralité).



- La lecture temps réel consiste à lire les frames au moment de leur capture. Si le traitement est long on doit sauter quelques frames, mais s'il est plus rapide, on doit attendre un moment pour lire la frame suivante.



# TP 9: Calibrage d'une camera :



# TP 9: Calibrage d'une camera :

Calibrer une caméra consiste à :

- Estimer les paramètres intrinsèques de la caméra,
- Sa position et orientation par rapport au repère du monde qui a été choisi (paramètres extrinsèques).

## TP 9: Calibrage d'une camera :

- Conversion d'unités:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

# TP 9: Calibrage d'une camera : Partie: 1

```
TP9.py
D: > D > USTHB > cours vision ABADA 2021-2022 > TPs VISION > 2022-2023-py > TP9 > TP9.py
1  from ctypes import sizeof
2  import cv2
3  import numpy as np
4  import os
5  import glob
6
7
8  CHECKERBOARD = (6,9)
9  criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
10 objpoints = []
11 imgpoints = []
12 objp = np.zeros((1, CHECKERBOARD[0] * CHECKERBOARD[1], 3), np.float32)
13 objp[0,:,:2] = np.mgrid[0:CHECKERBOARD[0], 0:CHECKERBOARD[1]].T.reshape(-1, 2)
14 #prev_img_shape = None
15
16 cap = cv2.VideoCapture(0)
17 if not cap.isOpened :
18     print("erreur ")
19     exit(0)
20
21 while True:
22
23     ret,img = cap.read()
24     gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
25     ret, corners = cv2.findChessboardCorners(gray, CHECKERBOARD, cv2.CALIB_CB_ADAPTIVE_THRESH +
26     cv2.CALIB_CB_FAST_CHECK + cv2.CALIB_CB_NORMALIZE_IMAGE)
27
28     if ret == True:
```

# TP 9: Calibrage d'une camera : Partie: 2

```
TP9.py
D: > D > USTHB > cours vision ABADA 2021-2022 > TPs VISION > 2022-2023-py > TP9 > TP9.py
26     cv2.CALIB_CB_FAST_CHECK + cv2.CALIB_CB_NORMALIZE_IMAGE)
27
28     if ret == True:
29         objpoints.append(objp)
30         corners2 = cv2.cornerSubPix(gray, corners, (11,11),(-1,-1), criteria)
31         imgpoints.append(corners2)
32         img = cv2.drawChessboardCorners(img, CHECKERBOARD, corners2, ret)
33     cv2.imshow('img',img)
34     if cv2.waitKey(100)&0xFF == ord('0') :
35         break
36
37     cv2.destroyAllWindows()
38
39     #h,w = img.shape[:2]
40
41     ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)
42
43     print("Camera matrix: \n")
44     print(mtx)
45     print("Distortion coefficient: \n")
46     print(dist)
47     print("Rotation Vectors: \n")
48     print(rvecs)
49     print("Translation Vectors: \n")
50     print(tvecs)
51
```

# TP 9: Calibrage d'une camera : Partie: 3

TP9.py

D: > D > USTHB > cours vision ABADA 2021-2022 > TPs VISION > 2022-2023-py > TP9 > TP9.py > ...

```
52
53
54 #-----
55 #-----
56 axis = np.float32([[3,0,0], [0,3,0], [0,0,-3]]).reshape(-1,3)
57 def draw(img, corners, imgpts):
58     corner = tuple(np.uint16(corners[0]).ravel())
59     img = cv2.line(img, corner, tuple(np.uint16(imgpts[0]).ravel()), (255,0,0), 5)
60     img = cv2.line(img, corner, tuple(np.uint16(imgpts[1]).ravel()), (0,255,0), 5)
61     img = cv2.line(img, corner, tuple(np.uint16(imgpts[2]).ravel()), (0,0,255), 5)
62     return img
63 axis2 = np.float32([[0,0,0], [0,3,0], [3,3,0], [3,0,0],[0,0,-3],
64 [0,3,-3],[3,3,-3],[3,0,-3] ])
65 def draw2(img, corners, imgpts):
66     imgpts = np.int32(imgpts).reshape(-1,2)
67     img = cv2.drawContours(img, [imgpts[:4]],-1,(0,255,0),-3)
68     for i,j in zip(range(4),range(4,8)):
69         img = cv2.line(img, tuple(np.uint16(imgpts[i])), tuple(np.uint16(imgpts[j])),(255),3)
70     img = cv2.drawContours(img, [imgpts[4:]],-1,(0,0,255),3)
71     return img
72 while True:
73     ret,img = cap.read()
74     gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
75     ret, corners = cv2.findChessboardCorners(gray, CHECKERBOARD,
76     cv2.CALIB_CB_ADAPTIVE_THRESH + cv2.CALIB_CB_FAST_CHECK +
77     cv2.CALIB_CB_NORMALIZE_IMAGE)
78     if ret == True:
```

# TP 9: Calibrage d'une camera : Partie: 4

```
74 gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
75 ret, corners = cv2.findChessboardCorners(gray, CHECKERBOARD,
76 cv2.CALIB_CB_ADAPTIVE_THRESH + cv2.CALIB_CB_FAST_CHECK +
77 cv2.CALIB_CB_NORMALIZE_IMAGE)
78 if ret == True:
79     corners2 = cv2.cornerSubPix(gray, corners, (11,11),(-1,-1), criteria)
80     ret,rvecs, tvecs = cv2.solvePnP(objp, corners2, mtx, dist)
81     imgpts, jac = cv2.projectPoints(axis, rvecs, tvecs, mtx, dist)
82     img = draw(img,corners2,imgpts)
83     cv2.imshow('img',img)
84     if cv2.waitKey(100)&0xFF == ord('0') :
85         break
86 cv2.destroyAllWindows()
87 cap.release()
```

# TP 9: Calibrage d'une camera :

Calibrer une caméra consiste à :

- Estimer les paramètres intrinsèques de la caméra,
- Sa position et orientation par rapport au repère du monde qui a été choisi (paramètres extrinsèques).

# TP 10: Détection d'un objet par couleur

```
TP9.py ● TP10_G3.py ●
D: > D > USTHB > cours vision ABADA 2021-2022 > TPs VISION > 2022-2023-py > TP10 > TP10_G3.py > ...
1
2 import cv2
3 import numpy as np
4 lo = np.array([95,100,30])
5 hi = np.array([125,255,255])
6 def detect_inrange(image,surfaceMin,surfaceMax):
7     points=[]
8     image = cv2.cvtColor(image,cv2.COLOR_BGR2HSV)
9     image = cv2.blur(image,(5,5))
10    mask = cv2.inRange(image,lo,hi)
11    mask = cv2.erode(mask,None,iterations=2)
12    mask = cv2.dilate(mask,None,iterations=2)
13    elements = cv2.findContours(mask,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)[-2]
14    elements=sorted(elements, key=lambda x:cv2.contourArea(x), reverse=True)
15    for element in elements:
16        print('Surface :',cv2.contourArea(element))
17        if cv2.contourArea(element)>surfaceMin and cv2.contourArea(element)<surfaceMax:
18            ((x, y), rayon)=cv2.minEnclosingCircle(element)
19            points.append(np.array([int(x), int(y)]))
20            break
21    return image,mask,points
```

# TP 10: Détection d'un objet par couleur

```
20         break
21     return image,mask,points
22 VideoCap=cv2.VideoCapture(0)
23 while(True):
24     ret, frame=VideoCap.read()
25     cv2.flip(frame,1,frame)
26     image,mask,points = detect_inrange(frame,3000,7000)
27     if (len(points)>0):
28         cv2.circle(frame, (points[0][0], points[0][1]), 10, (0, 0, 255), 2)
29     if mask is not None :
30         cv2.imshow('mask', mask)
31         cv2.imshow('frame', frame)
32         if cv2.waitKey(10)&0xFF==ord('q'):
33             break
34 VideoCap.release()
35 cv2.destroyAllWindows()
```