Annexes codes sources

TP1

```
import numpy as np
import cv2

#question 1
def soldes(prix):
    a = prix * 0.8
    b = prix * 0.6
    c = prix * 0.5
    return a,b,c

retour = soldes(130)
print(f"les prix soldés sont de {retour}")


#Q2
tableau = np.array(np.random.randint(0,10,size=(3,4)))

print(tableau)
print("Collone:")
print(np.sum(tableau,axis=0)) #0 est en collone, 1 ou -1 est en ligne
print("Ligne:")
print(np.sum(tableau,axis=-1)) #0 est en collone, 1 ou -1 est en ligne

#Q3
#Wait key est une fonction bloquante qui attend le frappe d'une touche de clavier

img = cv2.imread('GE141002.bmp',cv2.IMREAD_GRAYSCALE) # img est un tableau Numpy

cv2.imshow("Beton", img)
# Mise en pause du programme jusqu'à ce que l'utilisateur appuie sur une touche
x,y = img.shape[:2]
print(f" taille de l'image x:{x} y:{y}")



#Q4
print(img)
#l'image est un tableau de 2 dimension, avec les valeurs de niveau de gris dans
les pixels respective au coordonées x y
#le cas où l'image est stockée dans un tableau a 3 dimensions, c'est le cas ou
l'image traitée est en couleur.

#q5
#1: Opération de
#2: Opération de
#3: Opération de
```

```
#Q3.1

rest, img2 = cv2.threshold(img,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
print(f"seuil:{rest}")

cv2.imshow("Traitee", img2)
#cv2.imwrite("Q3.1.png", img2) sauvegardage Image



#Q3.2

el_struct = cv2.getStructuringElement(cv2.MORPH_RECT,(6,6))


img3 = cv2.morphologyEx(img2,cv2.MORPH_OPEN,el_struct)

cv2.imshow("Apres Ouverture", img3)
#cv2.imwrite("Q3.2.png", img3) sauvegardage Image


img4 = cv2.morphologyEx(img3,cv2.MORPH_CLOSE,el_struct)

cv2.imshow("Apres Fermeture", img4)
#cv2.imwrite("Q3.2.bis.png", img4) sauvegardage Image

#Q3.4

Granulats, _ = cv2.findContours(img4, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
nb_granulats = len(Granulats)
print(f"Nombre de Granulats :{nb_granulats}")

#Q3.5
img5 =cv2.drawContours(img4, Granulats, -1, 125, 3)

cv2.imshow("Contours",img5)
# cv2.imwrite("Q3.5.png", img5) sauvegardage Image

#Q3.6 et Q.7
img6 = cv2.cvtColor(img5,cv2.COLOR_GRAY2BGR)
air_total = 0
for i, contour in enumerate(Granulats):
    air = cv2.contourArea(contour)
    air_total += air
    print(f"Granulat {i+1}: Air en Pixel = {air}")
    #rectangle englobant chaque Granulats
    x,y,w,h = cv2.boundingRect(contour)

    #position du texte
    text_x = x+w // 2
```

```
    text_y = y+h // 2
    #ecriture du numéro
    cv2.putText(img6,str(i+1), (text_x,text_y),cv2.FONT_HERSHEY_DUPLEX, 0.5,
(0,0,255))


air_moyen = air_total / len(Granulats)
print(f"Air moyen des granulats:{air_moyen} pixels")

cv2.imshow("Contours",img6)
#cv2.imwrite("Q3.7.png", img6) sauvegardage Image

cv2.waitKey(0)
# Fermeture de toutes les fenêtres ouvertes
cv2.destroyAllWindows()
```

TP2

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


#entrainement partie 2
# ecole = 'unilasalle'
# print(ecole[3:7]) # s'affiche: lasa
# print(ecole[5:]) # s'affiche: salle
# print(ecole[:3]) # s'affiche: uni

# Déclaration en dur d'un tableau T à 2 lignes et 3 colonnes

# T = np.array([[1, 2, 3], [4, 5, 6]])
# T2 = T[:, 0:2]
# print(T)
# print(T2)

#-> extraction de toutes les lignes et des 2 premières colonnes de T

#lire l'image
img = cv2.imread('LAIT_KO.bmp',cv2.IMREAD_GRAYSCALE)

#Prendre la somme en ligne des valeurs de pixels
projection_ligne = np.sum(img,axis = 0)

#valeur max de cette ligne
valeur_max_ligne = np.max(projection_ligne)

#procédée idem
projection_colonne = np.sum(img,axis = 1)
valeur_max_colonne = np.max(projection_colonne)
```

```
#à decommenter si on veut voir les tracées
# plt.plot(projection_ligne)
# plt.plot(projection_colonne)
# plt.show()

print(f"Valeur Max ligne {valeur_max_ligne}")
print(f"Valeur Max colonne {valeur_max_colonne}")

valeur_seuil_front_colonne = valeur_max_colonne/2
valeur_seuil_front_ligne = valeur_max_ligne/2

Bordure_Superieur = np.where(projection_ligne> valeur_seuil_front_ligne)[0][0] #Tu
n'a que la premiere valeur du tableau
Bordure_Inferieur = np.where(projection_ligne> valeur_seuil_front_ligne)[0][-1]
#Tu a la derniere valeur du tableau
Bordure_Gauche = np.where(projection_colonne > valeur_seuil_front_colonne)[0][0]
Bordure_Droite = np.where(projection_colonne > valeur_seuil_front_colonne)[0][-1]
#En ayant ces 4 valeurs, on peux retrouver les 4 coins du rectangle
#Peut etre faire mieux plus tard quand l'image n'est pas réelement droit

# print(Bordure_Droite)
# print(Bordure_Gauche)
# print(Bordure_Superieur)
# print(Bordure_Inferieur)

plt.show()

#cv2.rectangle(img,(Bordure_Inferieur,Bordure_Droite),
(Bordure_Superieur,Bordure_Gauche),125,5)
#cv2.imshow('image', img)


#3.2 Découpage de la brique de lait

Decoupage = img[Bordure_Gauche:Bordure_Droite,
Bordure_Superieur:Bordure_Inferieur]

cv2.imshow('Decoup', Decoupage)


#decouper la date
Decoupage_date = Decoupage[303:303+95 , 57:57+41]

# Question 4   6

#A l’étape précédente, était-il nécessaire de détecter les quatre cotés de la
brique de lait pour pouvoir extraire la date
#Tout depend de comment la photo est prise et si on peut avoir la photo toujour
dans le même angle, même position dans l'image, cela implique de caller les brique
de lait dans le systeme
#dans un endroit a faible tolérance, et de les arreter, prendre la photo, puis de
les relancer, dans un contexte de production, cela est très inéfficace, donc on
```

```
peut prefere de prendre le
#photo dans l'elan du brique de lait, risquant une décalage, c'est pourquoi on
doit reprendre la.
#
#Mais en plus, la methode otsu ne fonctionnera pas correctement, car il y aura
beacoup de noir, qui nous rendra le filtre otsu non voulu
#

#binarisation


_, Decoupage_binarized =
cv2.threshold(Decoupage_date,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

cv2.imshow('binarized', Decoupage_binarized)

Decoupe_Copy= np.copy(Decoupage_binarized)


# 5 Extraction du premier Caractère de la date

Contours, _ = cv2.findContours(Decoupage_binarized, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
nb_Contours = len(Contours)
print(f"Nombre de Caratéres :{nb_Contours}")



Decoupage_binarized_Contours =cv2.drawContours(Decoupage_binarized, Contours, -1,
125, 3)
cv2.imshow("Contours",Decoupage_binarized_Contours)


img_Extraction = cv2.cvtColor(Decoupage_binarized_Contours,cv2.COLOR_GRAY2BGR)
air_total = 0
for i, contour in enumerate(Contours):
    air = cv2.contourArea(contour)
    air_total += air
    #rectangle englobant chaque Granulats
    x,y,w,h = cv2.boundingRect(contour)
    #position du texte
    text_x = x+w // 2
    text_y = y+h // 2
    #ecriture du numéro
    cv2.putText(img_Extraction,str(i+1), (text_x,text_y),cv2.FONT_HERSHEY_DUPLEX,
0.5, (0,0,255))


cv2.imshow("Extrait",img_Extraction)


#Initialisation de l'etalon
Etalon = cv2.imread('1.BMP',cv2.IMREAD_GRAYSCALE)
_,Etalon_Binarisee =
```

```
cv2.threshold(Etalon,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

height, width = np.shape(Etalon)

print(height)
print(width)

x,y,w,h = cv2.boundingRect(Contours[0])
img_cara_Extraction = Decoupe_Copy[y:y+height,x:x+width] #attention, j'ai rajoutée
une ligne de comparaison pour que la dimension correspond

cv2.imshow("Caractere",img_cara_Extraction)
#6
#Q7



# print(np.size(Etalon_Binarisee,axis = 0))
# print(np.size(img_cara_Extraction,axis = 0))

cv2.imshow('Bonjou',Etalon_Binarisee)

difference = cv2.absdiff(img_cara_Extraction,Etalon_Binarisee)
Nb_Pixel_diff= np.count_nonzero(difference)

Pourcentage = Nb_Pixel_diff/np.size(Etalon)*100


print(f"Nombres de pixels different {Nb_Pixel_diff}")
print(f"Nombre de pixel dans l'image etalon {np.size(Etalon)}")
print(f"Pourcentage de difference {Pourcentage} %")


if Pourcentage <20 :
    print("caractére correct")
else:
    print("caractére incorrect")



cv2.waitKey(0)
# Fermeture de toutes les fenêtres ouvertes
cv2.destroyAllWindows()
```

TP3 image fixe

```
import cv2
import numpy as np
```

```
Image_Bille = cv2.imread("boules.png")


cv2.imshow("Bonjour",Image_Bille)

taille_x, taille_y, nbr_canneaux = np.shape(Image_Bille)

print(f'Taille de x {taille_x}')
print(f'Taille de y {taille_y}')
print(f'Taille de canneaux {nbr_canneaux}')

Image_Bille_bleu, Image_Bille_vert, Image_Bille_rouge = cv2.split(Image_Bille)

cv2.imshow("Rouge",Image_Bille_rouge)

#En faisant la methode des seuils, nos n'aurons jamais seulement le rouge,
parceque on aura toujour des composant.
#Il faudrai faire un tri sur les trois variable RGB et afficher la abnde passante
de la couleur rouge sur une image en couleur et non juste la variable R qui
retourne une image en noir et blanc
#ce qui fait que sur le retour on aura toutes les couleurs qui ont une composante
rouge, ce qui ne fait pas de tri.

##HSV

Image_Bille_HSV = cv2.cvtColor(Image_Bille,cv2.COLOR_BGR2HSV)
Image_Bille_Hue, Image_Bille_Saturation, Image_Bille_lightness =
cv2.split(Image_Bille_HSV)

I = 10
Masque_1 = cv2.inRange(Image_Bille_Hue,0,I)
Masque_2 = cv2.inRange(Image_Bille_Hue,179-I,179)
Masque_final = cv2.bitwise_or(Masque_1,Masque_2)
cv2.imshow('Rouge',Masque_final)
#ici le problème c'est qu'on a toujours la bille blanche qui est affiché
##Question 13
rouge_inferieur = np.array([0,50,50])
rouge_superieur = np.array([I,255,255])

rouge_inferieur_2 = np.array([180-I,50,50])
rouge_superieur_2 = np.array([180,255,255])


Masque_1 = cv2.inRange(Image_Bille_HSV,rouge_inferieur,rouge_superieur)
Masque_2 = cv2.inRange(Image_Bille_HSV,rouge_inferieur_2,rouge_superieur_2)

Masque_totale = cv2.bitwise_or(Masque_1,Masque_2)

cv2.imshow('Rouge nouveau',Masque_totale)

##le résultat n'est pas satisfaisant car on a toujours les contours des billes
bleues qui s'affichent mais on a plus la bille blanche

el_struct = cv2.getStructuringElement(cv2.MORPH_RECT,(6,6))
```

```
Ouverture = cv2.morphologyEx(Masque_totale,cv2.MORPH_OPEN,el_struct)

cv2.imshow('Ouverture',Ouverture)

Fermeture = cv2.morphologyEx(Ouverture,cv2.MORPH_CLOSE, el_struct)

cv2.imshow('Fermeture',Fermeture)

Countours, _ = cv2.findContours(Fermeture, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)

Img_Contours =cv2.drawContours(Fermeture, Countours, -1, 125, 2)

cv2.imshow('Countours',Img_Contours)

for contour in Countours:
    (x,y), radius = cv2.minEnclosingCircle(contour)
    center = (int(x),int(y))
    radius = int(radius)
#print(center)
    cv2.circle(Image_Bille, center, radius, (0,255,0),2)
    cv2.imshow('image avec cercle', Image_Bille)


cv2.waitKey(0)

cv2.destroyAllWindows()
```

TP3 Video

```
import cv2
import numpy as np


def findCercleRouge(Image):
    Image_HSV  = cv2.cvtColor(Image,cv2.COLOR_BGR2HSV)
    I = 8
    rouge_inferieur = np.array([0,50,50])
    rouge_superieur = np.array([I,255,255])
    rouge_inferieur_2 = np.array([180-I,50,50])
    rouge_superieur_2 = np.array([180,255,255])
    Masque_1 = cv2.inRange(Image_HSV,rouge_inferieur,rouge_superieur)
    Masque_2 = cv2.inRange(Image_HSV,rouge_inferieur_2,rouge_superieur_2)
    Masque_totale = cv2.bitwise_or(Masque_1,Masque_2)
    el_struct = cv2.getStructuringElement(cv2.MORPH_RECT,(6,6))
    Ouverture = cv2.morphologyEx(Masque_totale,cv2.MORPH_OPEN,el_struct)
    Fermeture = cv2.morphologyEx(Ouverture,cv2.MORPH_CLOSE, el_struct)
    Countours, _ = cv2.findContours(Fermeture, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
```

```
    for contour in Countours:
        (x,y), radius = cv2.minEnclosingCircle(contour)
        center = (int(x),int(y))
        radius = int(radius)
        #print(center)
        cv2.circle(Image, center, radius, (0,255,0),2)
    cv2.imshow('TP3', Image)


Image_Bille = cv2.imread("boules.png",)

findCercleRouge(Image_Bille)

cap = cv2.VideoCapture('billes.mp4')
    # Vérifier si succès
if not cap.isOpened():
    print("Erreur : impossible d'ouvrir la vidéo.")
    exit()

while True:
    # Lire une frame de la vidéo
    ret, frame = cap.read()
    # Vérifier si la fin de la vidéo n'est pas atteinte
    if not ret:
        break

    # Afficher la frame
    findCercleRouge(frame)

    # Attendre 100 millisecondes avant de passer à la frame suivante
    cv2.waitKey(100)
    # Libérer les ressources de la vidéo et fermer la fenêtre d'affichage

cv2.waitKey(0)
cap.release()
cv2.destroyAllWindows()
```