```
# bibliothèques
import numpy as np
import matplotlib.pyplot as
plt
import matplotlib.image as
mpimg
import matplotlib.cm as cm
# chargement de l'image
image=mpimg.imread('IMG_2167.p
ng')
# le fichier doit être dans le
bon dossier, sinon, chemin...
####################
# nature des objets
# le résultat est un tableau
multidimensionnel de  #type
np.ndarray
#print(type(image))
##<class 'numpy.ndarray'>
#print(image.ndim)
## 3
#print(image.shape)
## (480, 640, 3) :  hauteur  ,
largeur
```

```
# Si c'est une matrice de
luminances (niveaux de gris),
le tableau est seulement n x p
# si c'est RGB, n x p x 3
# si c'est RGBA  nx px 4: avec
transparence.
######################
# accès au contenu
# pour l'efficacité, utiliser
calcul vectoriel et tranchage:
# composante rouge de tout:
#print(image[:,:,0])
# toutes les composantes du
pixel i,j:
#print(image[3,85,:3])
###########
### exemple de
vectorialisation: niveau de
gris:
# principe de la
vectorialisation
"""print(image[0:16:4,0:24:12,
:3])
imagetest=np.dot(image[0:16:4,
```

```
0:24:12,0:3],
[.3333,.3333,.3333])
print('et puis')
print(imagetest)"""
# chaque case du tbleau avait
un triplet,
# elle n'a plus qu'une
luminance
# formule luminance = 0.299R
+0.587V + 0.114 B
# on utilise aussi 0.2125 R+
0.7154V+ 0.0721B

"""
plt.subplot(3,1,1)
img1=np.dot(image[:,:,0:3],
[.3333,.3333,.3333])
plt.imshow(img1, cmap=
cm.Greys_r)
plt.title("gris 0.3333R+
0.3333V+ 0.3333B")
plt.tight_layout()
plt.subplot(3,1,2)
img2=np.dot(image[:,:,:3],
```

```
[.299,.587,.114])
#print(img.ndim) # réponse : 2
#plt.imshow(img) # résultat
bizarre...
plt.imshow(img2, cmap=
cm.Greys_r)
plt.title("gris 0.299R +0.587V
+ 0.114 B")
plt.tight_layout()
plt.subplot(3,1,3)
img3=np.dot(image[:,:,:3],
[.2125,.7154,.0721])
plt.imshow(img3, cmap=
cm.Greys_r)
plt.title("gris 0.2125 R+
0.7154V+ 0.0721B")
plt.tight_layout()
"""

"""
##################################
## renversement de l'image
#Renversement gauche/droite
(effet miroir)
```

```
shape = image.shape
newimage = np.ndarray(shape)
newimage = np.fliplr(image)
nnewimage = np.ndarray(shape)
# à deviner...
nnewimage = np.flipud(image)
#up down
plt.imshow(nnewimage)
############################
#### échangeons les
couleurs...

GBRimage = np.ndarray(shape)
GBRimage[:,:,0] =image[:,:,1]
GBRimage[:,:,1] =image[:,:,2]
GBRimage[:,:,2] =image[:,:,0]
plt.imshow(GBRimage)
##############
"""
####Détection des contours
# on construit les luminances
# on construit les gradients
en chaque pixel
# si f(i,j) donne la
```

```
luminance, on approche la DP
selon i par DPi= f(i+1,j)-
f(i,j); puis la norme est
sqrt(DPi**2+ DPj**2)
# on définit un seuil: au-
dessus, le pixel est sur un
contour
# on modifie la matrice des
normes des gradients en ne
gardant que les valeurs 1 (>
seuil) ou 0 (< seuil). Il n'y
a plus que deux couleurs:
noir, blanc.
# 1- Luminances
img=np.dot(image[:,:,:3],
[.299,.587,.114])
# 2 - DP
# on fait la différence du
tableau
#img[i+1,j], 0 <= i < n-1, 0,
<= j < p
#= img[i,j], 1 <= i < n, 0, <=
j < p
# et du tableau
```

```
# img[i,j], 0 <= i < n-1, 0,
<= j < p
#La tableau DPi obtenu a une
ligne de moins,
#celui des DPj, une colonne de
moins.
#On rectifie pour avoir des
tableaux compatibles (une
ligne et une colonne de moins
que l'original...)
###
"""
DPi=(img[1:,:]-img[:-1,:])
[:,:-1]#la fin= -1 col
DPj=(img[:,1:]-img[:,:-1])
[-1:,:]# ..=-1 ligne
normes=np.sqrt(DPi**2+DPj**2)#
calcul vectorialisé
#c'est donc un tableau de
taille n-1 x p-1
# normes, seuil, tracé
copie= normes[:]
plt.subplot(2,1,1)
seuil = .25 # à tester
```

```
normes[(normes<=seuil)]=0.
normes[(normes>seuil)]=1.
# on a utilisé un mask
# le tableau dit n'a plus que
des 0 = noir et des 1 = blanc,
blanc ou noir. On le trace en
''niveau de gris, mais on veut
le noir!''
plt.imshow(1-
normes,cmap=cm.Greys_r)
plt.title('seuil 0.25')
plt.tight_layout()
##
plt.subplot(2,1,2)
seuil = .05 # à tester
copie[(copie<=seuil)]=0.
copie[(normes>seuil)]=1.
plt.imshow(1-
copie,cmap=cm.Greys_r)
plt.title('seuil 0.05')
plt.tight_layout()"""
################################
# filtre sobel
# DPi = f[i,j]+
```

```
2*f[i+1,j]+f[i+2,j]- f[i,j+2]
#-2*f[i+1,j+2]-f[i+2,j+2]
Diffij= img[:,:-2]-img[:,2:]
sobi=Diffij[:-2,:]+2*Diffij[1:
-1,:]+Diffij[2:,:]
Diffji= img[:-2,:]-img[2:,:]
sobj=Diffji[:,:-2]+2*Diffji[:,
1:-1]+Diffji[:,2:]
plt.subplot(2,1,1)
normes=np.sqrt(sobi**2+sobj**2
)
seuil = .25 # à tester
normes[(normes<=seuil)]=0.
normes[(normes>seuil)]=1.
# on a utilisé un mask
# le tableau dit n'a plus que
des 0 = noir et des 1 = blanc,
blanc ou noir. On le trace en
''niveau de gris, mais on veut
le noir!''
plt.imshow(1-
normes,cmap=cm.Greys_r)
plt.subplot(2,1,2)
normes1 = abs(sobi)+abs(sobj)
```

```
seuil = .3 # à tester
normes1[(normes1<=seuil)]=0.
normes1[(normes1>seuil)]=1.
plt.imshow(1-
normes1,cmap=cm.Greys_r)
```

```
#######fig =
plt.figure(figsize=(14, 6.5),
dpi=100)"""
```