|  |  |
| --- | --- |
| Ministère de l’Enseignement Supérieur  Et de la Recherche Scientifique  \*\*\* \* \*\*\*  Université de Carthage  \*\*\* \* \*\*\*  **Institut National des Sciences**  **Appliquées et de Technologie** |  |

Mini-Projet : Traitement d’images

Réalisé par : RIM BESBES

Groupe : IIA\_4\_1

Année universitaire : 2022/2023

1. **Débruitage de l’image liftingbodybruite.png**

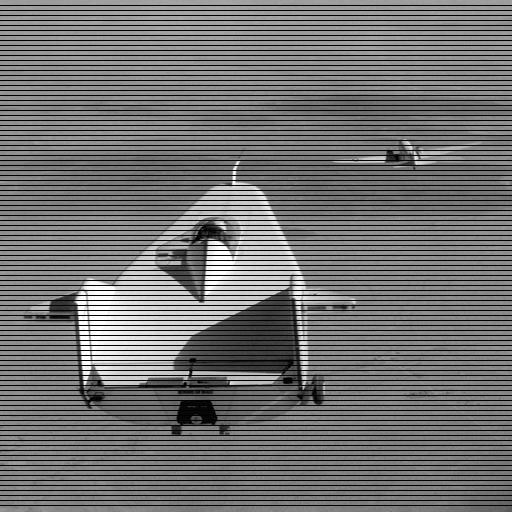


Figure : image bruitée

Le bruit en question est constitué de raies noires horizontales. On remarque que ces raies se superposent sur l’image de façon périodique, à une fréquence spatiale précise. Ce bruit aurait été généré peut-être par le procédé d’acquisition. Or on sait que la transformation de Fourier d’un sinus correspond à deux composantes harmoniques symétriques.

🡺 L’idée est donc d’utiliser un filtre coupe-bande pour supprimer les deux composantes harmoniques.

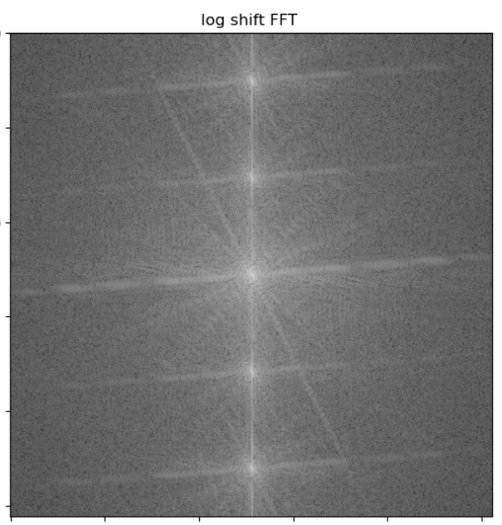
On trace donc la fft de l’image bruité (fig.2) :

Figure : Transformée de Fourier de l'image bruitée

On constate la présence 4 points où on a une densité d’informations. Le point central correspond à la fréquence nulle. Donc les points restants devraient correspondre au bruit. Or on s’attendait à voir deux points symétriques correspondant à une seule harmonique. Le bruit n’est donc pas totalement périodique mais presque. En traçant les axes, on localise les composantes parasites sur la fft.

Le filtre à appliquer est un filtre coupe bande, qui va rejeter les fréquences en question.

En faisant quelques recherches sur les différents types de filtre coupe-bande, j’ai trouvé que **le filtre de Notch** est le filtre sélectif le plus efficace. La fonction de transfert du filtre est construite comme produit de filtres passe-haut dont le centre a été translaté vers le centre des fréquences d’intérêt. Pour les filtres passe-haut on utilise des filtres gaussiens.

Les filtres passe-haut sont construits avec des filtres passe-bas avec la formule suivante :

Avec correspondant à la distance d’un point de la fft par rapport au centre, et , la fréquence de coupure.

Le script de la fonction qui crée le filtre est le suivant :

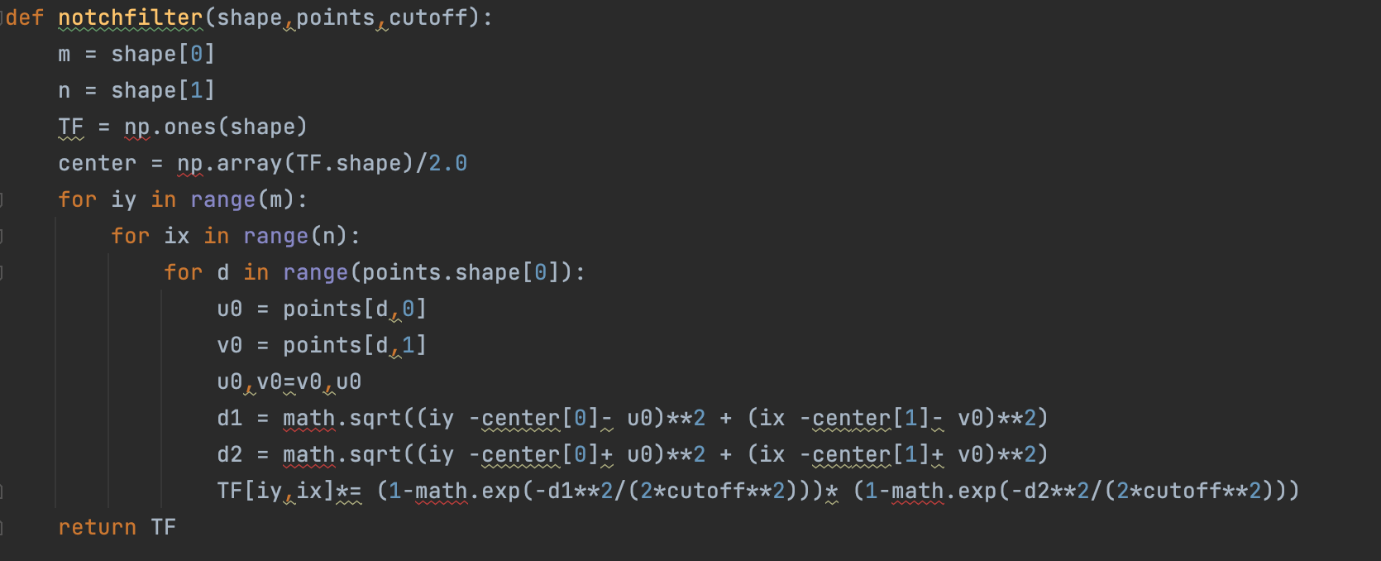


Figure : Code de création du filtre

Pour appliquer le filtre, on multiplie la fft de l’image par le filtre fréquentiel obtenu. Pour la reconstruction de l’image, on utilise la transformée de Fourier inverse et on prend le réel. La fréquence de coupure et les points d’intérêt ont été déterminés visuellement à l’aide de la (fig.2)

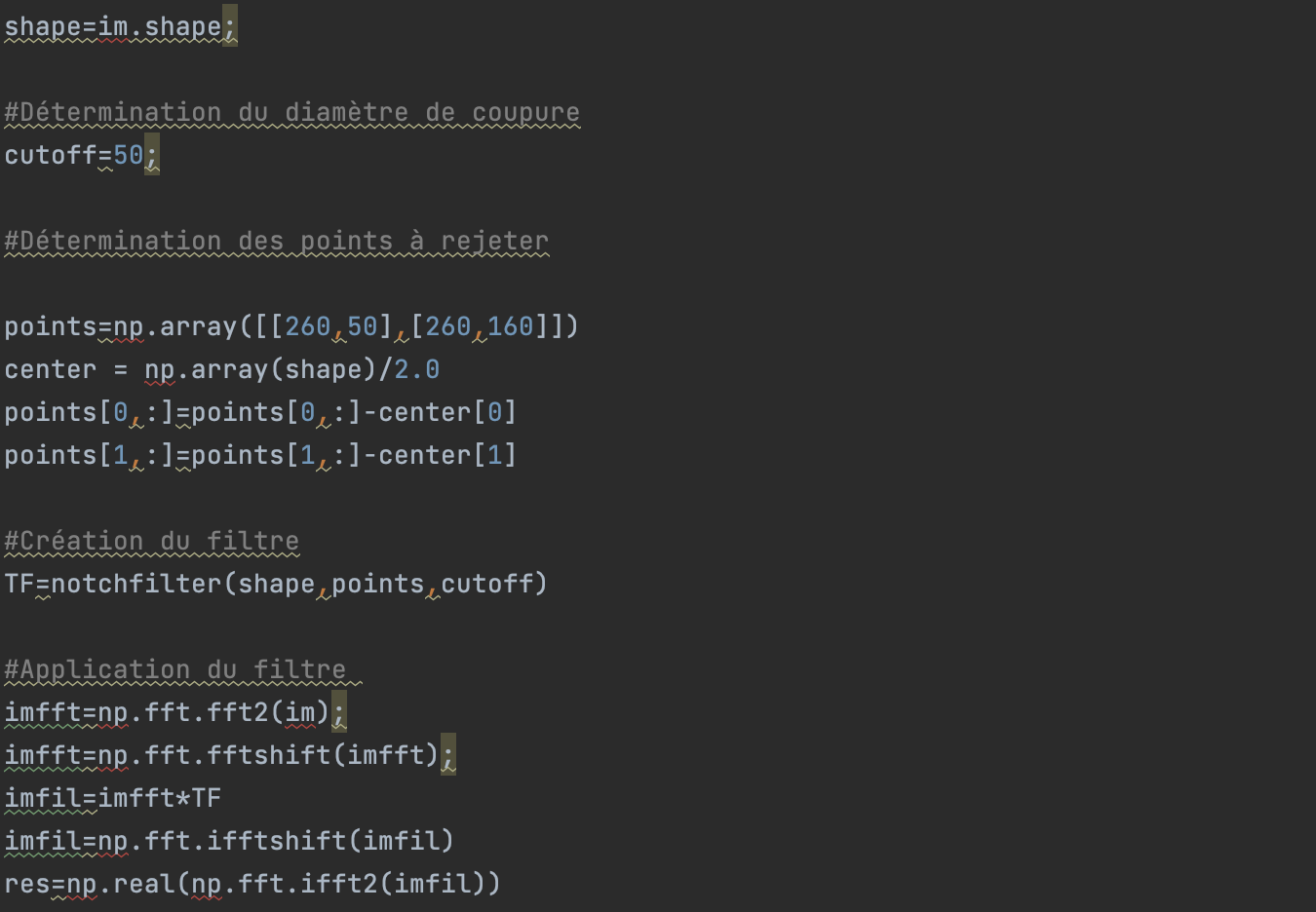


Figure : Application du filtre

Les figures (fig.5) et (fig.6) montrent respectivement le filtre créé et l’image résultante du filtre :

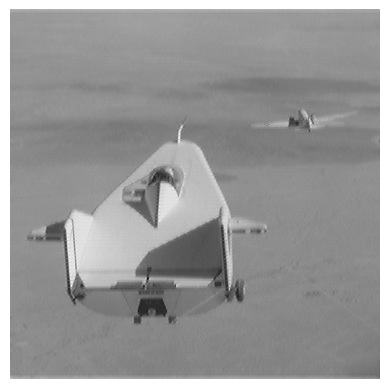


Figure : Image débruitée

Figure : Filtre de Notch

🡺 On remarque que les raies noires horizontales et périodiques ont disparues même si quand on regarde de très proche on voit quelques traces de raies mais qui ne sont pas perceptibles à première vue. L’image a été bien débruité.

1. **Filtrage de l’image cartebruitee.png**

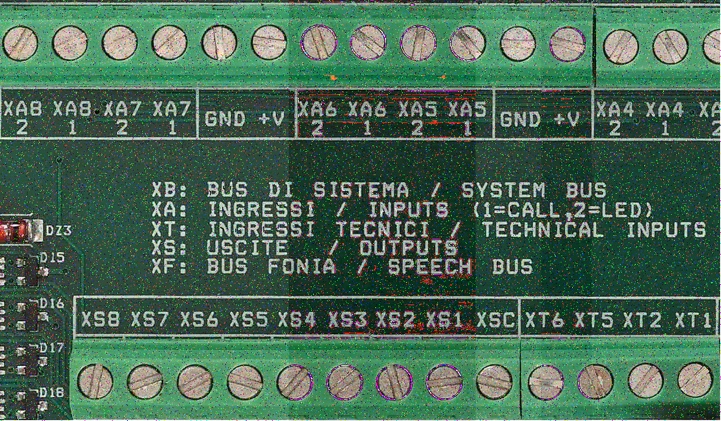
La première étape demandée est d’appliquer le filtrage adéquat de façon à éliminer toute sorte de bruit.

Figure : Carte bruitée

On remarque qu’il s’agit d’un bruit de type poivre et sel. Or on a vu en cours que le filtre adéquat pour éliminer ce type de bruit est le filtre passe-bas médian. On applique donc ce filtre :

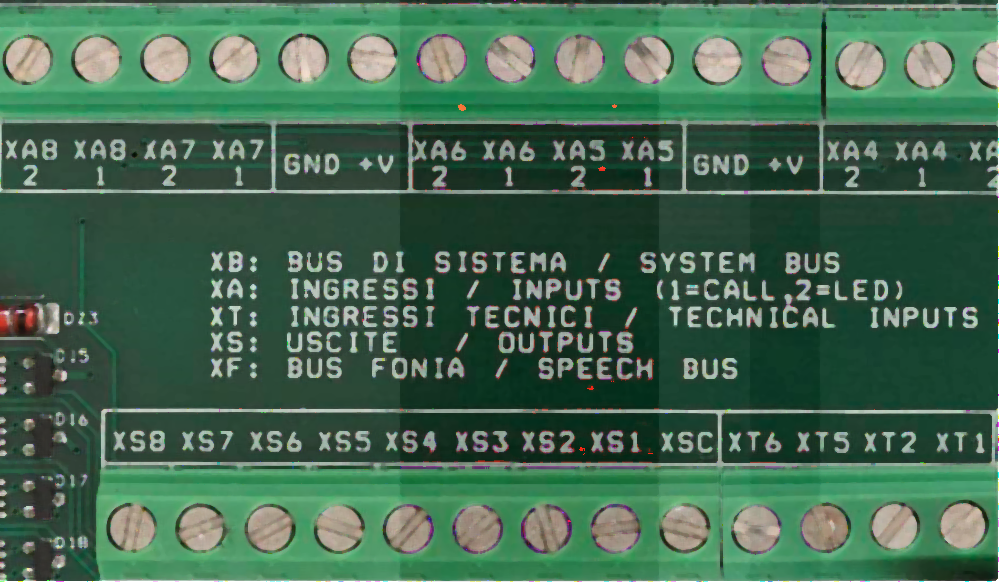
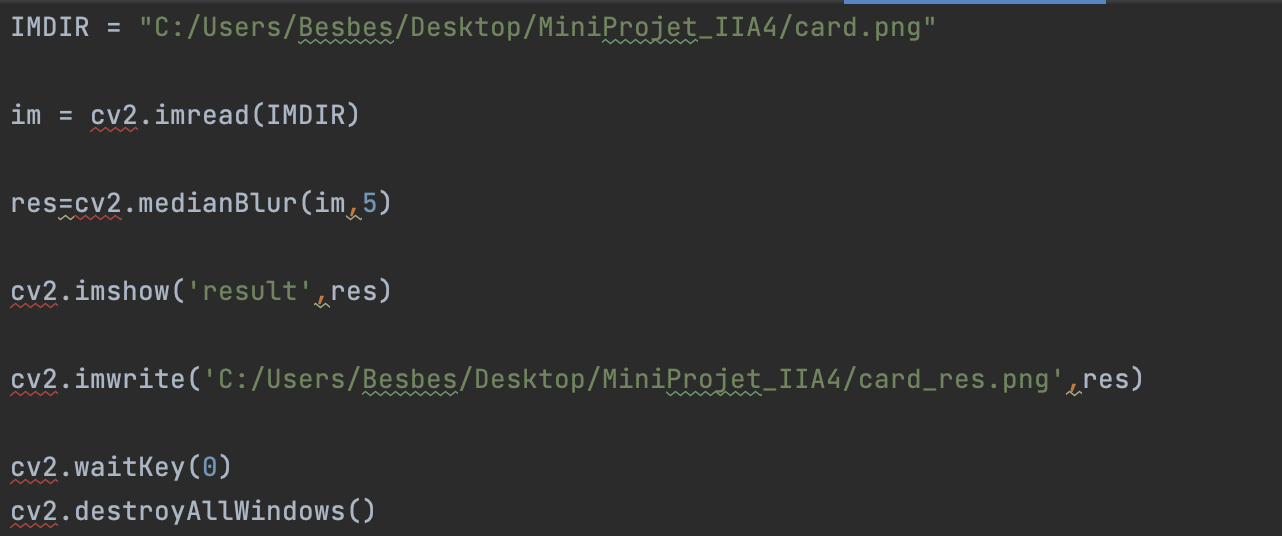


Figure 8 : Code du filtre médian

Figure : Code du filtre médian

Figure : Carte débruité

La figure (fig.8) montre la carte débruitée. Le filtrage a été donc efficace.

La deuxième étape consiste à faire le traitement nécessaire sur la bande centrale pour obtenir les cinq lignes sous forme de caractères blanc sur fond noir, en veillant à éliminer toute sorte d’altération qui touchent les lettres qui la composent.

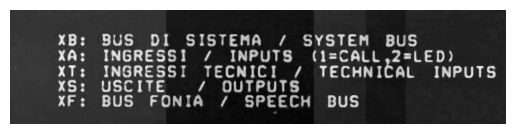
On commence par couper la région d’intérêt puis on transforme l’image en niveaux de gris :

Figure : Région d'intérêt en grayscale

On remarque que certaines lettres sont altérées comme pour le mot ‘bus’ dans la première ligne ou encore la lettre ‘C’ de ‘CALL’ dans la deuxième ligne. Pour réparer ces cassures dans les lettres, on utilise une transformation morphologique. Mais d’abord on veut transformer cette image en image binaire, et donc en segmentant le fond gris noir. Pour cela, on trace l’histogramme de l’image :

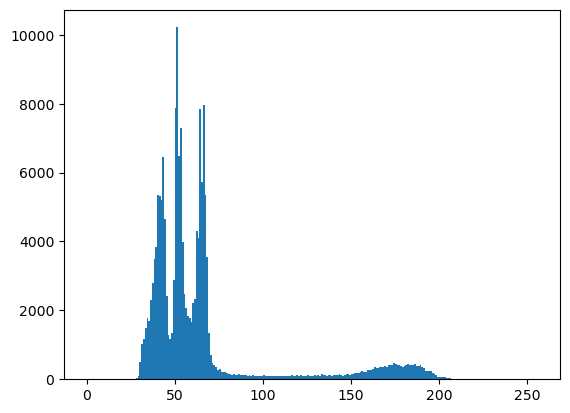


Figure : Histogramme de la carte débruitée

On retrouve un histogramme bi-modale, les intensités les plus basses correspondent donc au fond et les plus hautes à la région d’intérêt (les cinq lignes). On effectue un seuillage pour séparer la région d’intérêt :

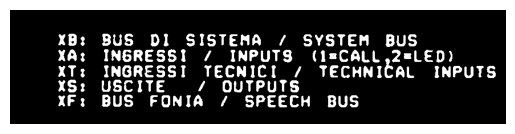


Figure : Image binarisée

Deux idées sont possibles pour combler les vides dans les lettres : soit une dilatation, soit une dilatation suivie d’une érosion donc une fermeture. On compare donc les deux méthodes pour décider laquelle est la meilleure :

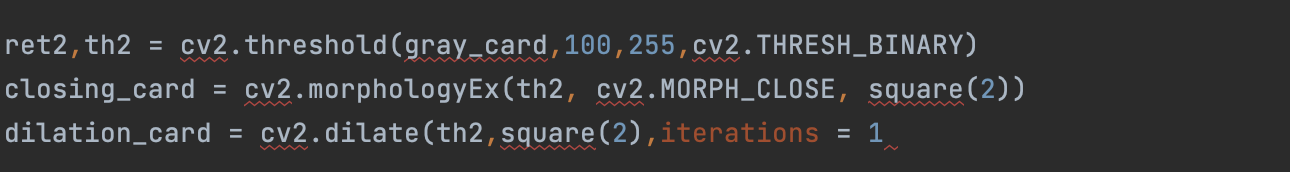


Figure : Code pour la transformation morphologique

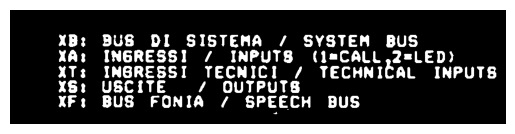


Figure : Résultat de la fermeture

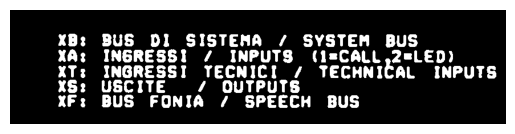


Figure : Résultat de la dilatation

On constate que la dilatation donne de meilleurs résultats. En effet, on retrouve le ‘U’ dans ‘BUS’ à la première ligne qui est clair et bien dessiné dans la dilatation alors qu’il est encore altéré dans la fermeture. C’est le même cas pour les ‘X’ qui sont mieux dessinés dans la dilatation. 🡺 On retient donc la dilatation comme transformation morphologique adéquate.

Enfin, on souhaite détecter les contours de ces caractères. On applique le filtre de Canny, vu en TP 3, pour la détection de contours :

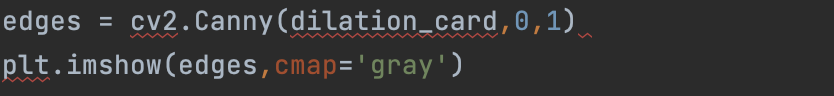


Figure : Code pour le filtre de Canny

L’image résultante est la suivante :

Figure : Résultat du filtre de Canny

🡺 On arrive à bien détecter les contours avec le filtre de Canny, le résultat est satisfaisant et toutes les lettres sont facilement distinguables.

1. **Transformation de couleurs**

L’objectif dans cette partie est de transformer les couleurs du drapeau de l’Allemagne de façon à obtenir le drapeau de la Lituanie :



Figure : Drapeau de la Lituanie

Figure : Drapeau de l'Allemagne

L’idée est de détecter les différentes bandes de couleurs, en construisant des masques pour chaque bande. On a donc besoin de déterminer les plages des codes couleur des différentes couleurs noir, rouge et jaune. Suite à quelques recherches, j’ai trouvé qu’il est préférable de travailler dans l’espace colorimétrique HSV du fait qu’il est plus facile à utiliser car la composante hue correspond à la teinte de l’image, alors qu’en RGB, 3 composantes définissent la couleur. De ce fait, il est plus facile de définir un intervalle pour une teinte donnée avec le code HSV qu’avec RGB.

Le code RGB des nouvelles couleurs est déterminé simplement à l’aide de l’outil Paint.

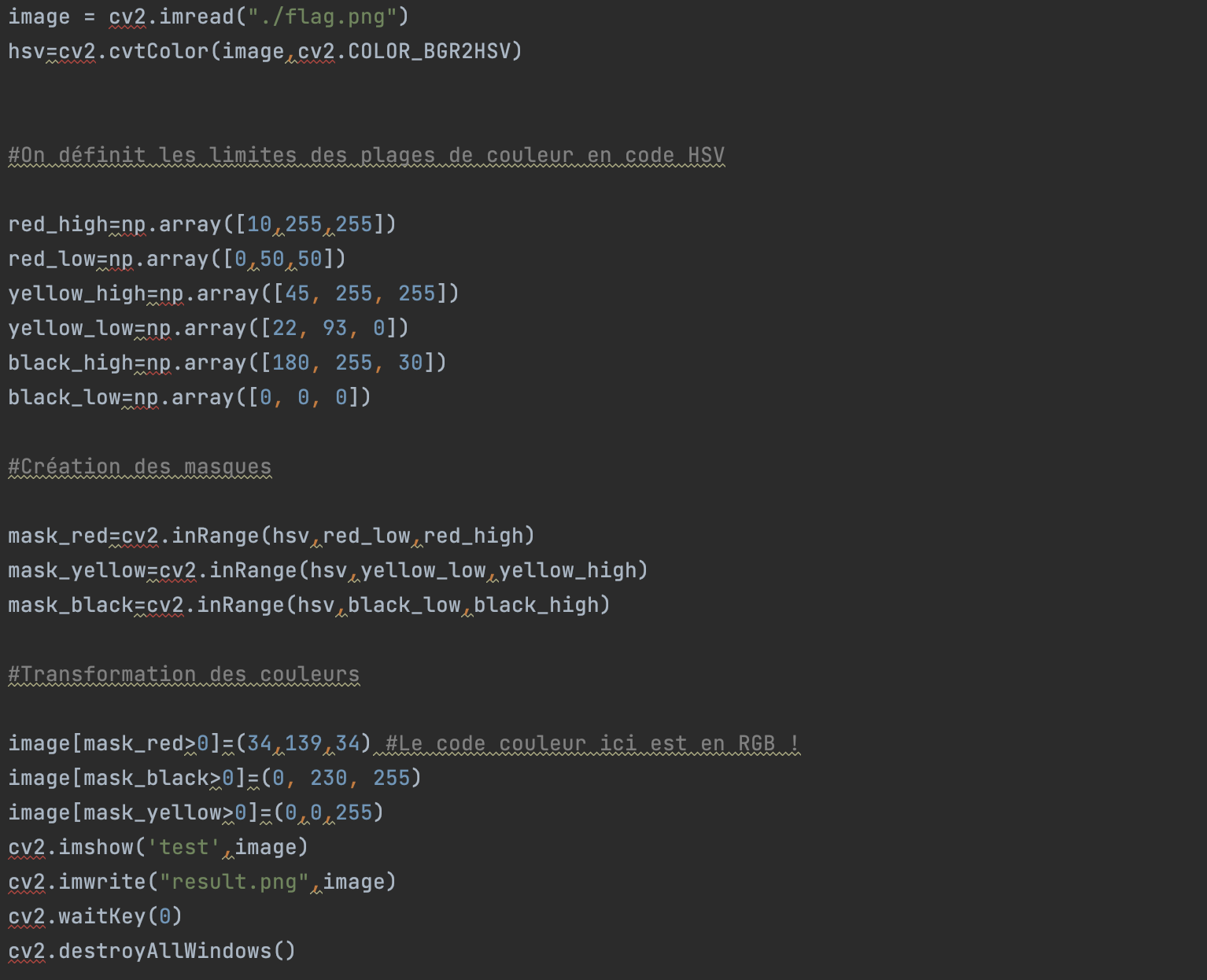


Figure : Script de la transformation de couleur

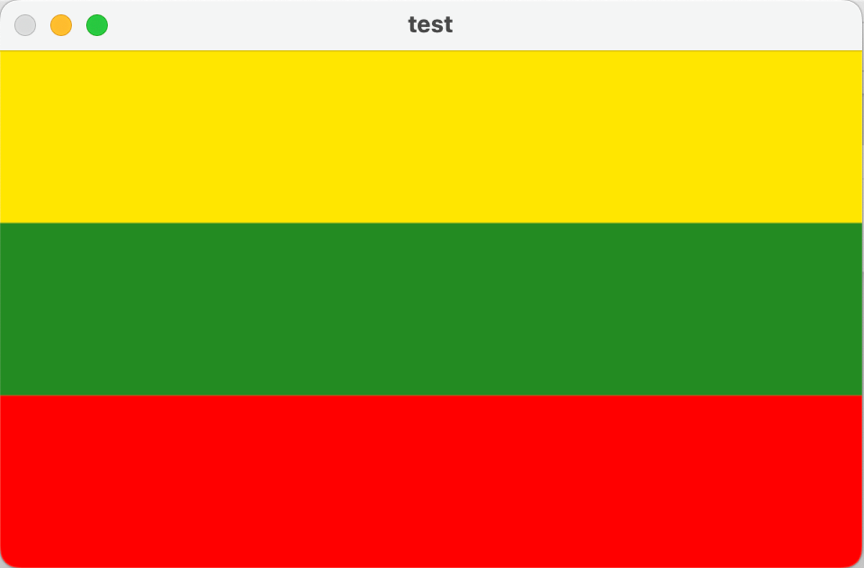
Le résultat obtenu est le suivant :

Figure : Résultat de la transformation

**ANNEXE : SCRIPT PYTHON DES 3 PARTIES**

Le code ci-dessous est en texte et n’est pas une capture d’écran contrairement au rapport.

1ère PARTIE :

import matplotlib.pyplot as plt  
import numpy as np  
import os  
import skimage.io as io  
import cmath  
import math  
import cv2  
from skimage.morphology import square  
  
  
def notchfilter(shape,points,cutoff):  
 m = shape[0]  
 n = shape[1]  
 TF = np.ones(shape)  
 center = np.array(TF.shape)/2.0  
 for iy in range(m):  
 for ix in range(n):  
 for d in range(points.shape[0]):  
 u0 = points[d,0]  
 v0 = points[d,1]  
 u0,v0=v0,u0  
 d1 = math.sqrt((iy -center[0]- u0)\*\*2 + (ix -center[1]- v0)\*\*2)  
 d2 = math.sqrt((iy -center[0]+ u0)\*\*2 + (ix -center[1]+ v0)\*\*2)  
 TF[iy,ix]\*= (1-math.exp(-d1\*\*2/(2\*cutoff\*\*2)))\* (1-math.exp(-d2\*\*2/(2\*cutoff\*\*2)))  
 return TF  
  
IMDIR = "C:/Users/Besbes/Desktop/MiniProjet\_IIA4/plane.png/"  
  
im = io.imread(IMDIR,as\_gray=True)  
shape=im.shape;  
  
#Détermination du diamètre de coupure  
cutoff=50;  
  
#Détermination des points à rejeter  
  
points=np.array([[260,50],[260,160]])  
center = np.array(shape)/2.0  
points[0,:]=points[0,:]-center[0]  
points[1,:]=points[1,:]-center[1]  
  
#Création du filtre  
TF=notchfilter(shape,points,cutoff)  
  
#Application du filtre  
imfft=np.fft.fft2(im);  
imfft=np.fft.fftshift(imfft);  
imfil=imfft\*TF  
imfil=np.fft.ifftshift(imfil)  
res=np.real(np.fft.ifft2(imfil))  
  
plt.imshow(res, cmap='gray')  
plt.axis('off')

2ème PARTIE :

IMDIR = "C:/Users/Besbes/Desktop/MiniProjet\_IIA4/card.png"  
  
im = cv2.imread(IMDIR)  
  
res=cv2.medianBlur(im,5)  
  
cv2.imshow('result',res)  
  
cv2.imwrite('C:/Users/Besbes/Desktop/MiniProjet\_IIA4/card\_res.png',res)  
  
cv2.waitKey(0)  
cv2.destroyAllWindows()  
  
ret2,th2 = cv2.threshold(gray\_card,100,255,cv2.THRESH\_BINARY)  
closing\_card = cv2.morphologyEx(th2, cv2.MORPH\_CLOSE, square(2))  
dilation\_card = cv2.dilate(th2,square(2),iterations = 1  
  
edges = cv2.Canny(dilation\_card,0,1)  
plt.imshow(edges,cmap='gray')

3ème PARTIE

image = cv2.imread("./flag.png")  
hsv=cv2.cvtColor(image,cv2.COLOR\_BGR2HSV)  
  
  
#On définit les limites des plages de couleur en code HSV  
  
red\_high=np.array([10,255,255])  
red\_low=np.array([0,50,50])  
yellow\_high=np.array([45, 255, 255])  
yellow\_low=np.array([22, 93, 0])  
black\_high=np.array([180, 255, 30])  
black\_low=np.array([0, 0, 0])  
  
#Création des masques  
  
mask\_red=cv2.inRange(hsv,red\_low,red\_high)  
mask\_yellow=cv2.inRange(hsv,yellow\_low,yellow\_high)  
mask\_black=cv2.inRange(hsv,black\_low,black\_high)  
  
#Transformation des couleurs  
  
image[mask\_red>0]=(34,139,34) #Le code couleur ici est en RGB !  
image[mask\_black>0]=(0, 230, 255)  
image[mask\_yellow>0]=(0,0,255)  
cv2.imshow('test',image)  
cv2.imwrite("result.png",image)  
cv2.waitKey(0)  
cv2.destroyAllWindows()