3A IE

TP Image

CHAPUS Louka, CONTANT Maxence

Table des matières

1. TP n°2

1. Fourrier 2D

a. Algorithme

Une solution pour calculer la transformée de Fourrier 2D est de calculer une première fois la transformée 1D sur les lignes puis de calculer la transformée 1D sur les colonnes de l'image précédente. Et on obtient comme cela la transformé 2D d'une image.

b. Harmoniques pures

Pour commencer, on créait une image de 128 x 128 avec une fréquence d'oscillation de 0.1 selon n, 0 selon m et une fréquence fe à 1.

```
f_1 = 0.1
f_2 = 0
f_e = 1

img=atom(128,128,f_1,f_2);
affiche_image(img)
fourier2d(img,f_e)
```

Figure [1] code utilisé

On appelle simplement les fonctions qui sont donné dans le sujet avec les bonnes valeurs et on obtient l'image suivante :

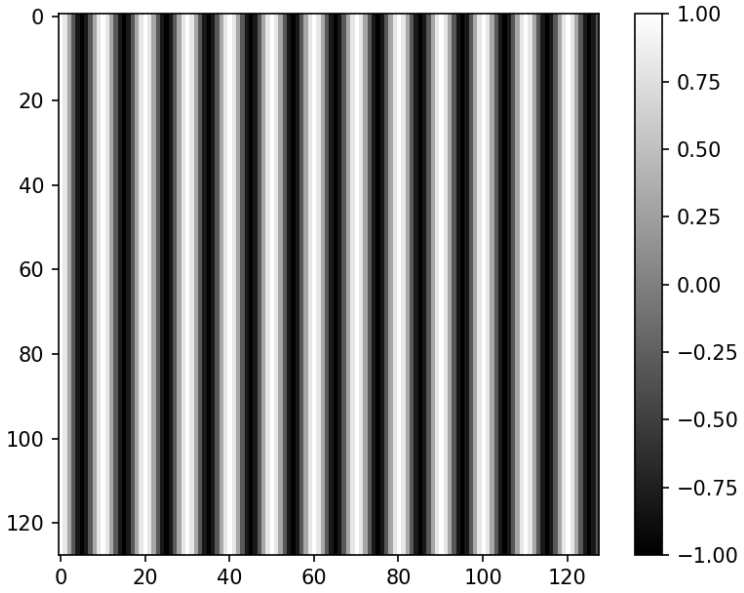

Figure [2], image avec $f_1 = 0{,}1$ et $f_2 = 0$

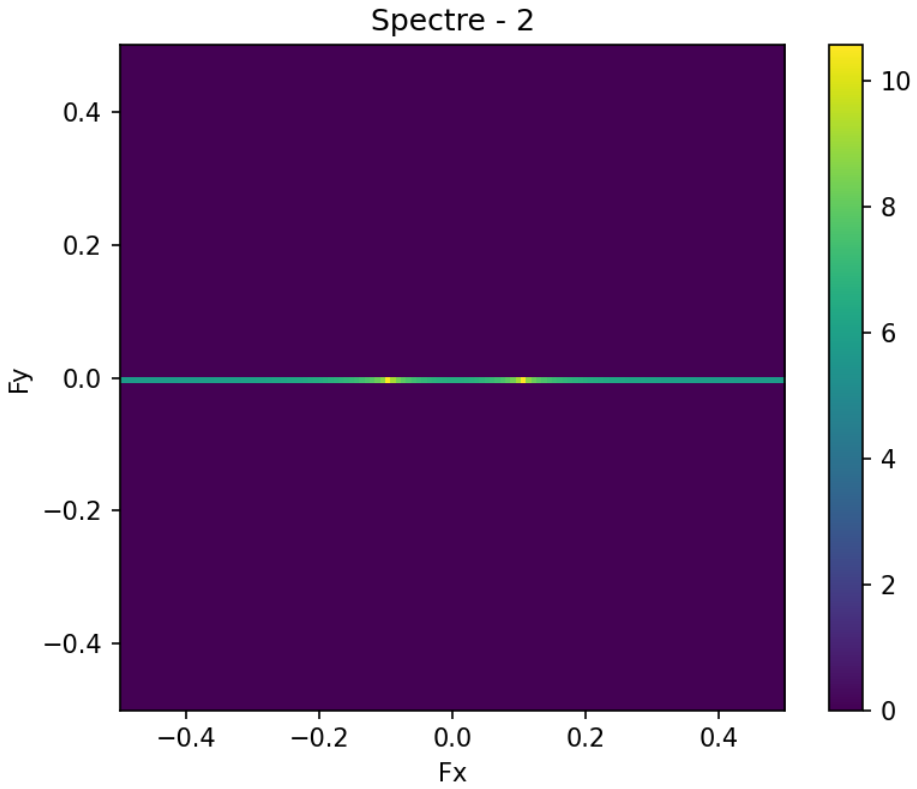

Figure [3], Spectre de la figure [2]

On remarque que lorsqu'on a une image avec des lignes verticales alors on obtient un spectre qui ne contient uniquement une ligne horizontale avec deux pics d'intensités à -0,1 et 0,1. On a donc ensuite essayé différentes fréquences d'oscillation mais sans changer la fréquence d'échantillonnage.

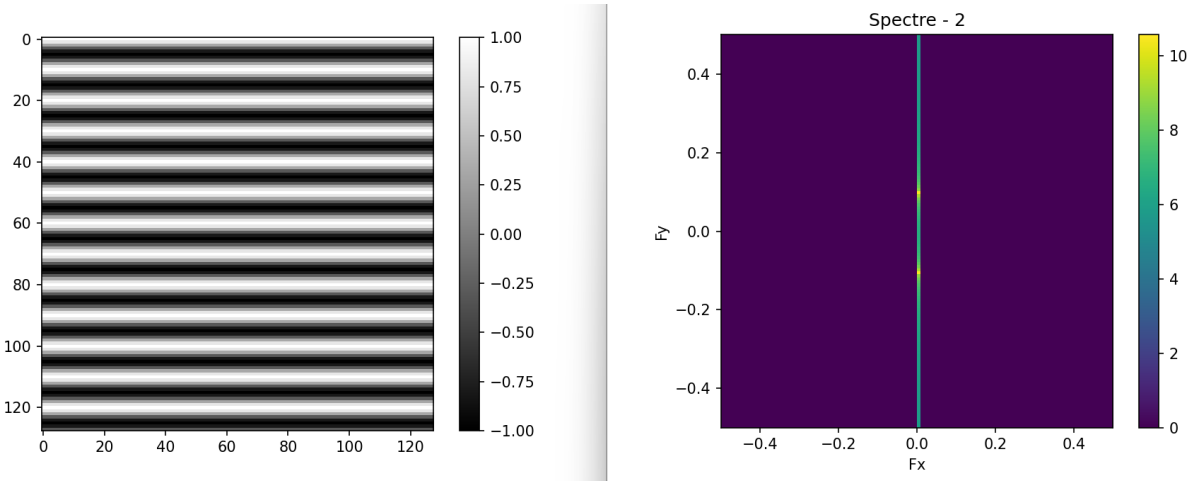

Figure [4], image avec $f_1 = 0$ *et* $f_2 = 0{,}1$ ainsi que son spectre

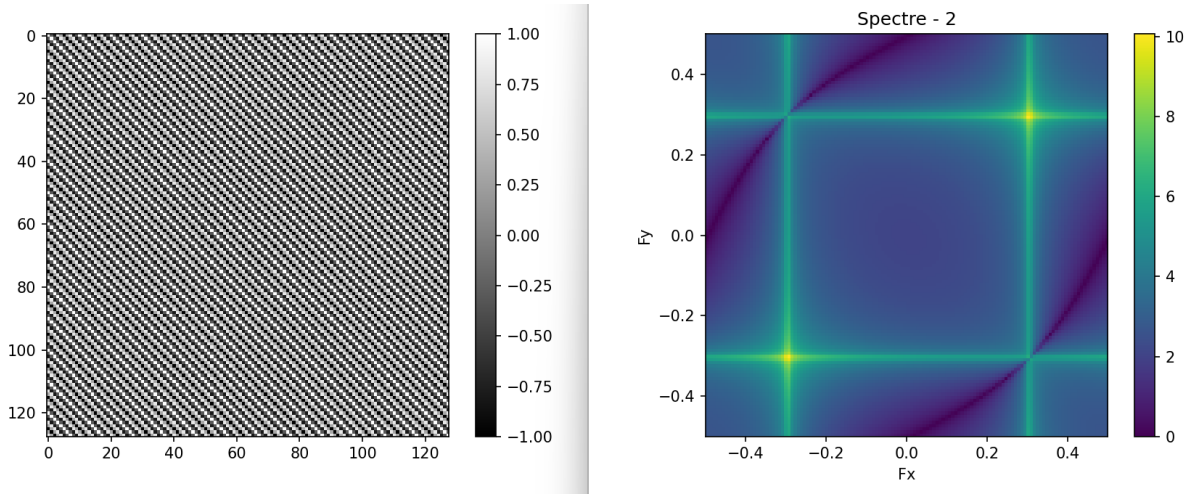

Figure [5], image avec $f_1 = 0{,}3$ et $f_2 = 0{,}3$ ainsi que son spectre

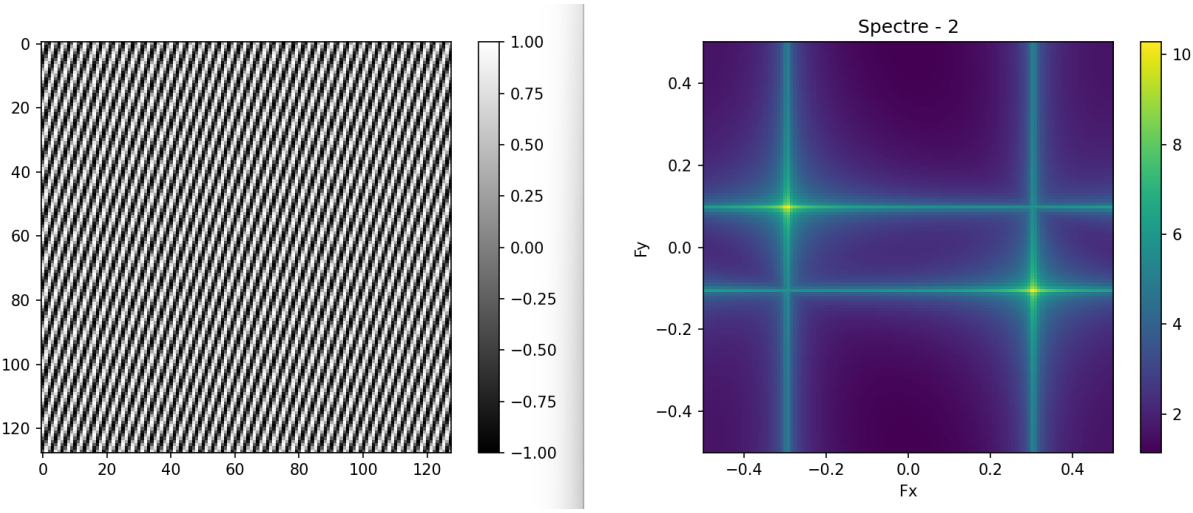

Figure [6], image avec $f_1 = -0{,}3$ et $f_2 = 0{,}1$ ainsi que son spectre

On constate bien avec les figures [2] à [6] que lorsque l'image a des lignes verticales on obtient un spectre avec une ligne horizontale et inversement. De plus pour les images qui ont des lignes en diagonales on voit sur le spectre qu'il y a une ligne verticale et une ligne horizontale ce qui correspond bien à une diagonale.

c. Contour

On se propose par la suite d'analyser les contours d'une image à l'aide de 3 images créés avec le code de la figure [7]. On va faire une image avec des lignes verticales, une avec des lignes horizontales et une avec des lignes en diagonales pour bien voir la différence dans les spectres des trois images.

```
f_e = 1

rupt0=atom(128,128,0.05,0);
rupt1=atom(128,128,0,0.05);
rupt2=atom(128,128,0.05,0.05);

affiche_image(rupt0)
affiche_image(rupt1)
affiche_image(rupt2)

fourier2d(rupt0,f_e)
fourier2d(rupt1,f_e)
fourier2d(rupt2,f_e)
```

Figure [7] code utilisé pour générer les images et les spectres

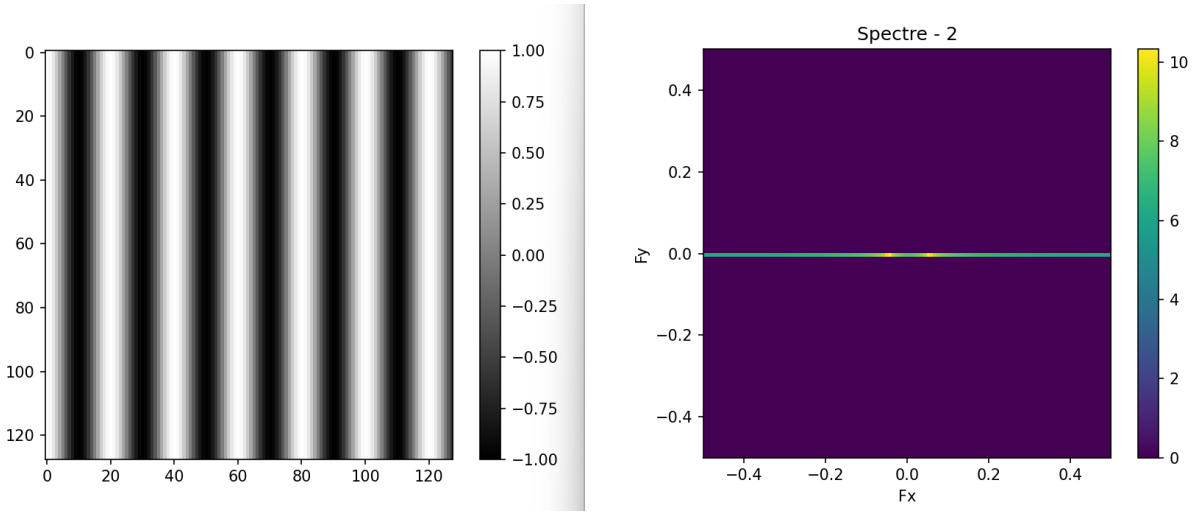

Figure [8] Image avec le contour vertical

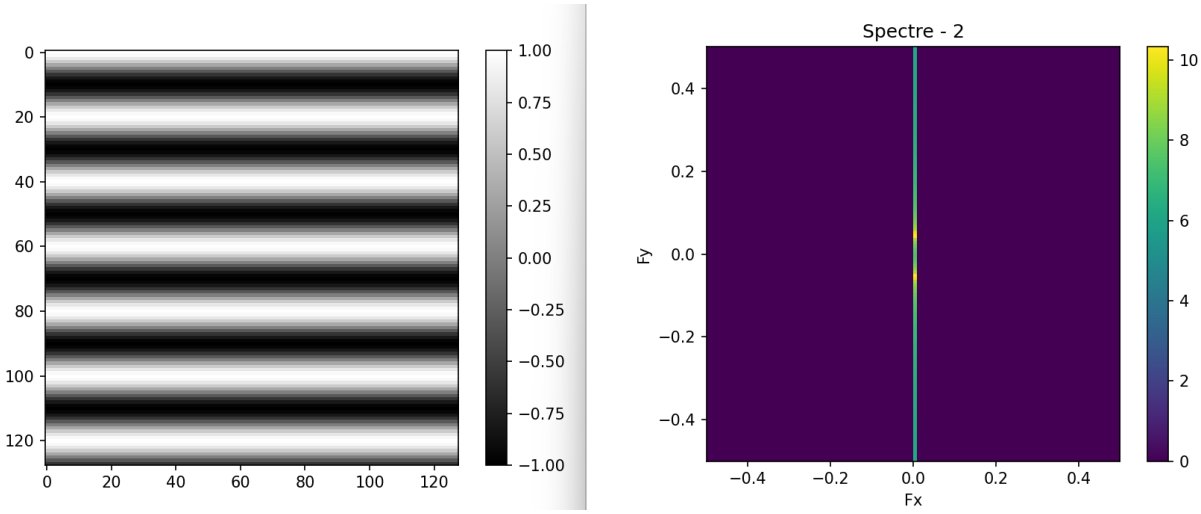

Figure [9] Image avec le contour horizontal

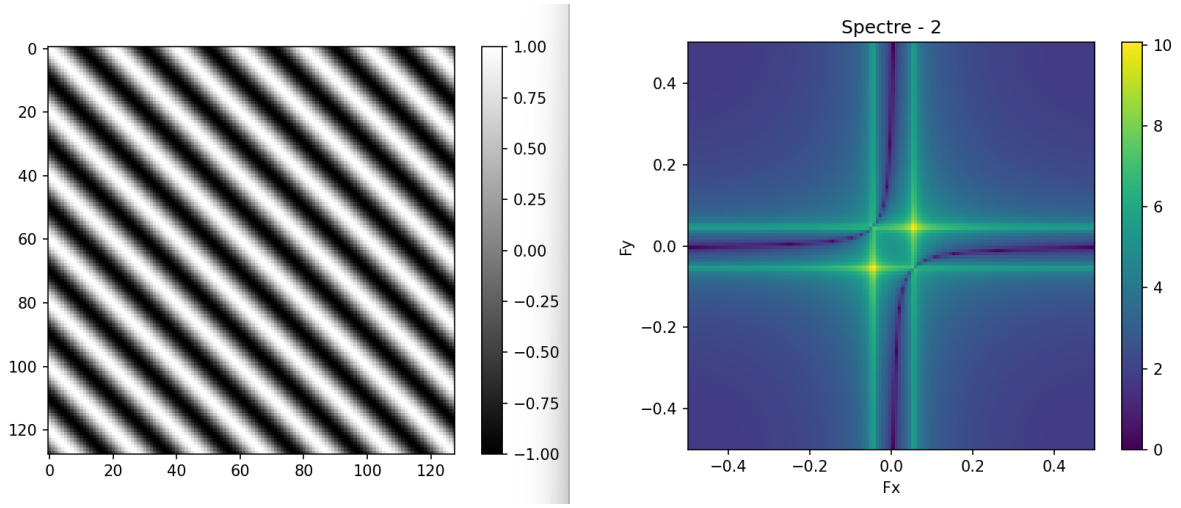

Figure [10] Image avec le contour oblique

On voit qu'a chaque fois pour le spectre il y a deux pics d'intensité qui correspondent à la fréquence des oscillations. De plus tous les spectres contiennent des lignes verticales, horizontales ou les deux, ce qui correspond à la rupture des contours sur les images de bases.

d. Texture

On va maintenant analyser les transformées de Fourrier 2D de différentes textures et en déduire les paramètres les plus importants. On va donc utiliser le code de la Figure [11] juste en changeant le nom de l'image à chaque fois pour avoir l'image souhaiter.

```
#Ouvre une image
img = cv.imread("C:/Users/louka/Desktop/Cours/TP_Image/imagesTP/Leaves0012GP.png")
img = cv.cvtColor(img,cv.COLOR_RGB2GRAY)

affiche_image(img)
fourier2d(img,f_e)
```

Figure [11] code pour les textures

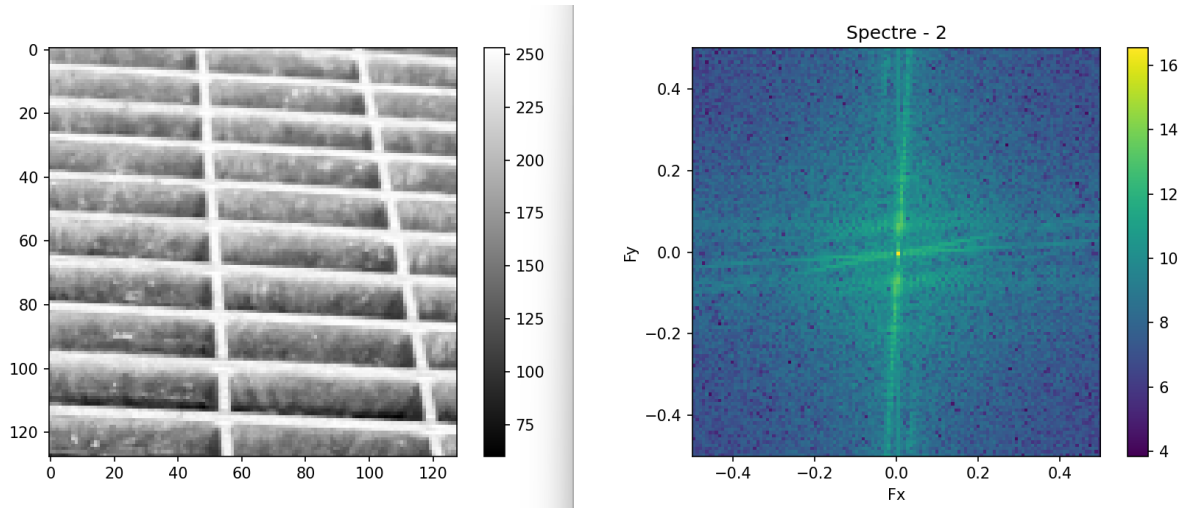

Figure [12] image Metal0007G

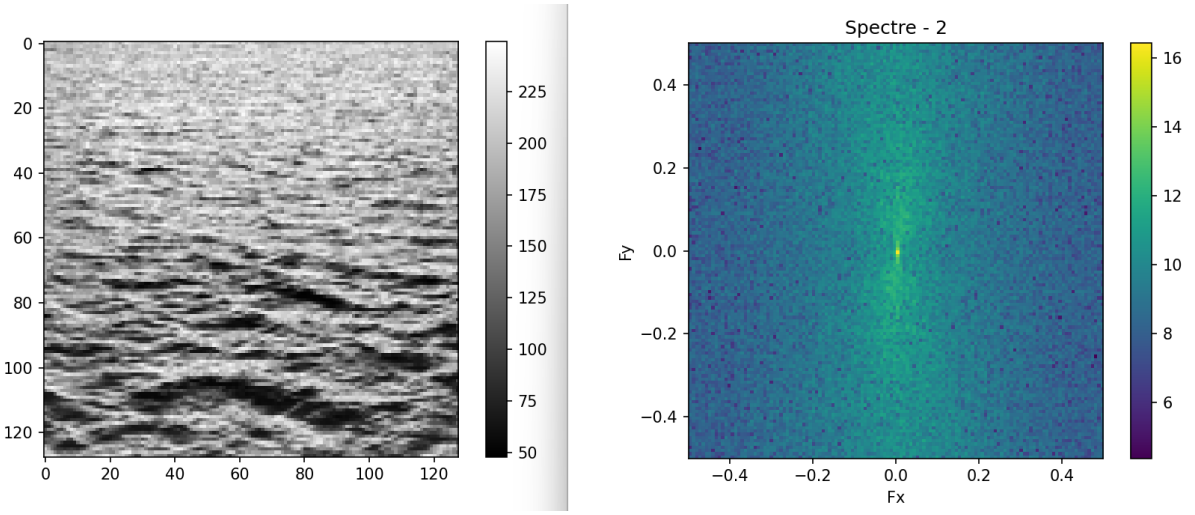

Figure [13] image Water0000G

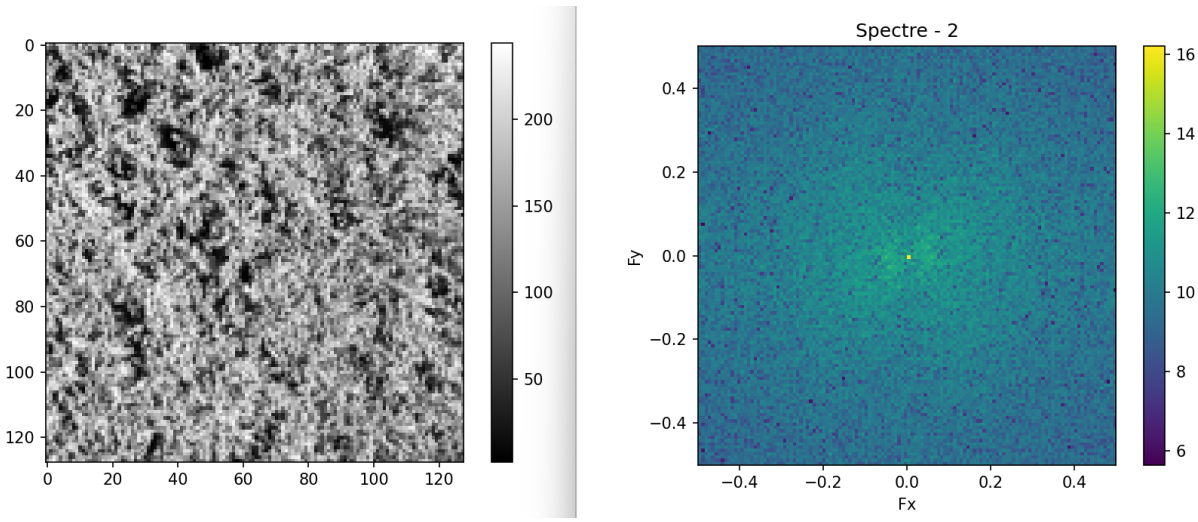

Figure [14] image Leaves0012G

On a testé différentes textures. On remarque bien que le spectre est totalement différent pour chaque image. Pour la Figure [12] on voit assez bien qu'il y a des lignes presque verticales et presque horizontales sur le spectre, ce qui indique que dans l'image de base il y a des lignes quasiment verticales et horizontales. Pour la Figure [13], on voit vaguement une ligne verticale se dessiner donc cela nous indique qu'il y a quelques lignes horizontales dans l'image de base mais vraiment pas beaucoup ou alors elles ne sont pas nettement dessinées. Enfin pour Figure [14], il n'y a aucune ligne ou forme vraiment claire donc pour cette image on ne peut pas conclure sur sa texture, on sait juste qu'il n'y a pas de ligne droite dedans.

Donc un paramètre important pour analyser les textures et de voir si le spectre contient des pics d'intensité et si le spectre contient des lignes assez nettes pour pouvoir conclure d'une certaine forme dans l'image de base.

2. Le phénomène de repliement

Pour cette dernière partie on se propose d'observer le phénomène de repliement spectrale dans le cadre d'une image. Pour ce faire on commence par créer une image comme indiquer dans le sujet à l'aide du code de la Figure [15].

```
img = atom(128,128,0.15,0.37);
affiche_image(img)
fourier2d(img,f_e)

#Utilise la fonction de binarisation du précédent TP
img_binariser = binarisation_manuelle(0.5,img)
affiche_image(img_binariser)
fourier2d(img_binariser,f_e)

image_redimensionner = img[::2, ::2]
affiche_image(image_redimensionner)
fourier2d(image_redimensionner,f_e)
```

Figure [15] code utilisé pour générer les images

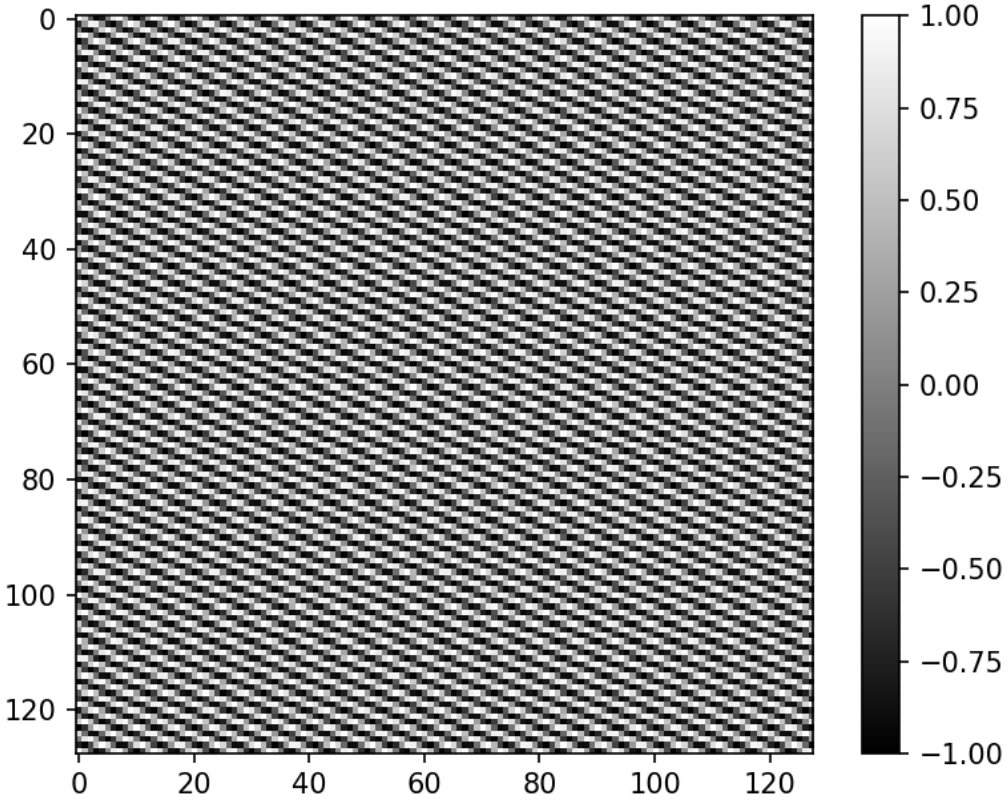

Figure [16] Image non binarisé

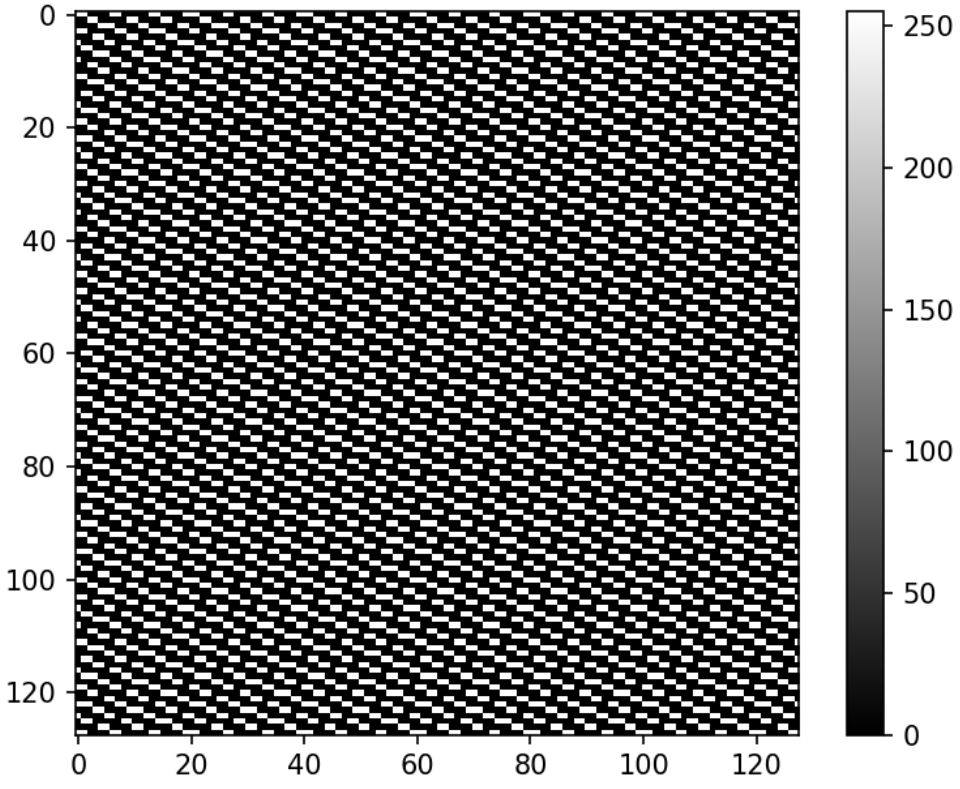

Figure [17] Image binarisé

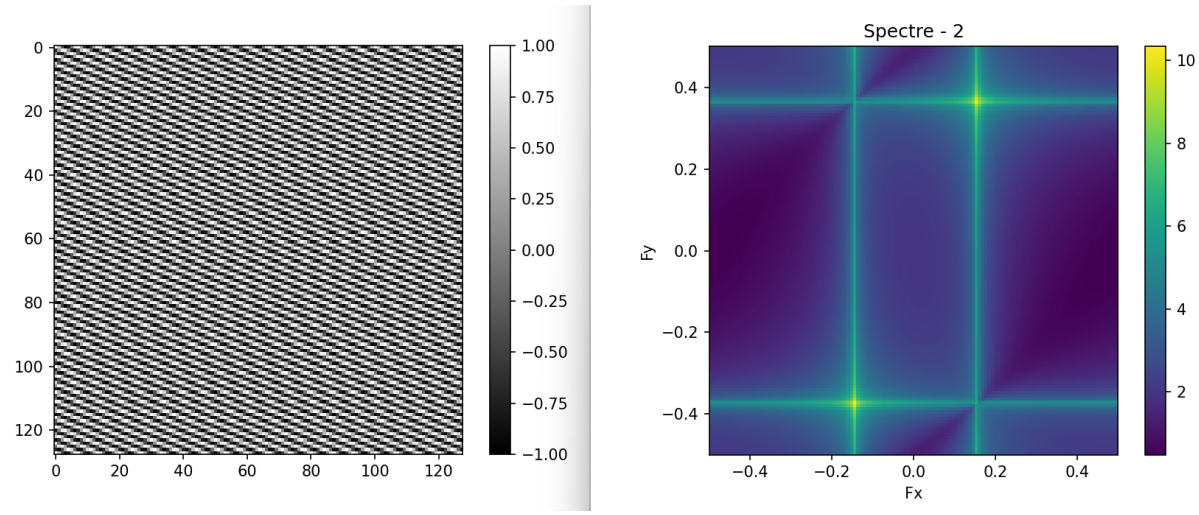

Figure [18] Image non binarisé avec son spectre

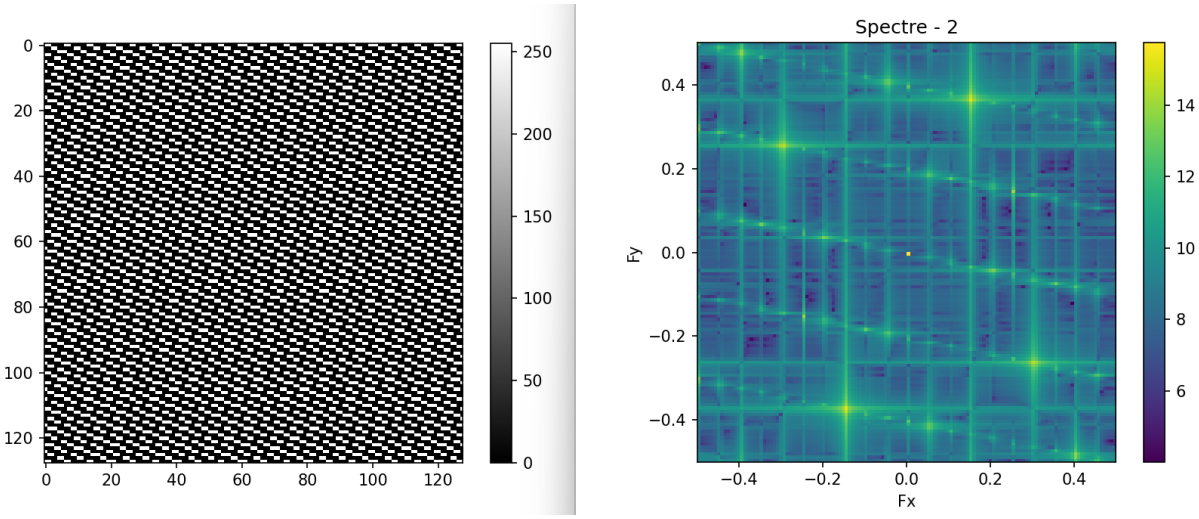

Figure [19] Image binarisé avec son spectre

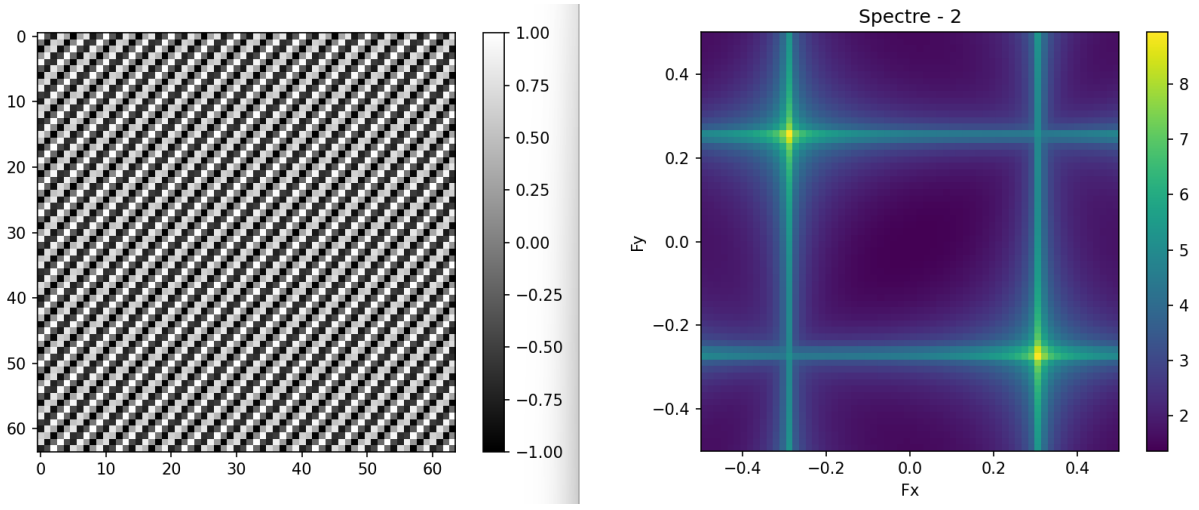

Figure [20] Image réduite avec son spectre

2. TP n°3

1. Changement d'espaces colorimétriques (comparaison HSV/IHLS)

1.

```
#Ouvre une image
img = cv.imread("C:/Users/louka/Desktop/Cours/TP_Image/imagesTP/confiserie-smarties-lentilles_121-50838.jpg")
img = cv.cvtColor(img,cv.COLOR_RGB2HSV)
plt.figure()
plt.imshow(img,cmap="hsv")
plt.colorbar()
```

Figure [21] code

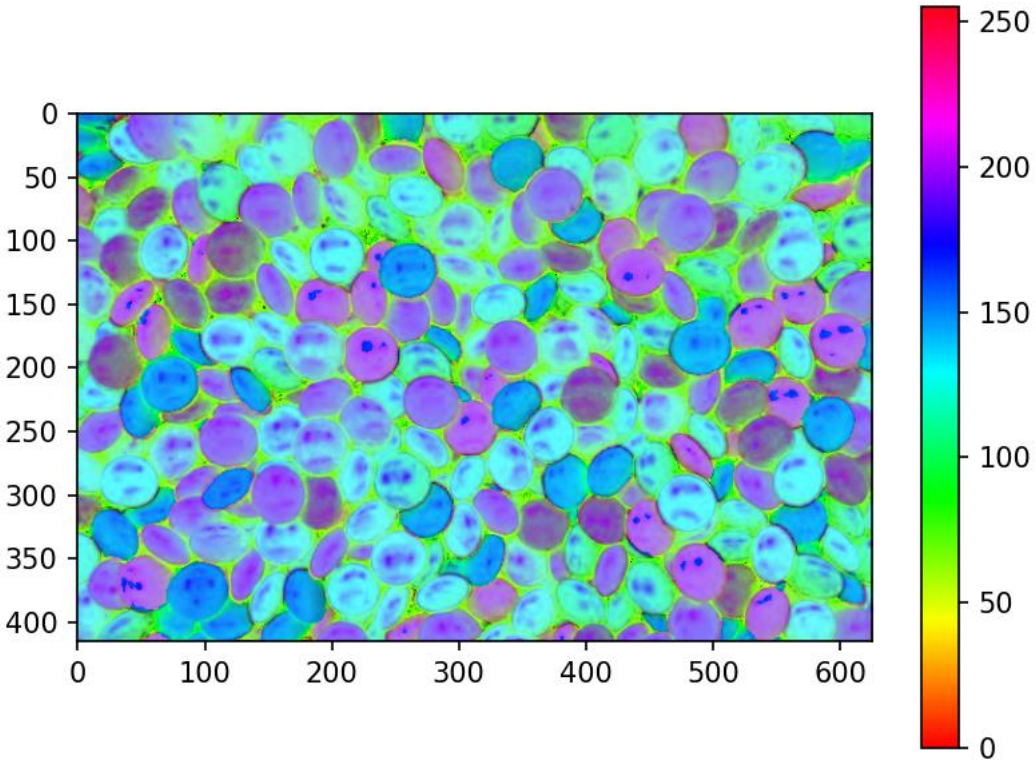

Figure [22] résultat après exécution

On remarque que l'image obtenue est nettement différente de l'originale, c'est normal car se sont deux espace colorimétrique différents.

2.

```
#Affichage des images

# Extraction des composantes HSL
hue = img[:, :, 0] # Composante teinte
lightness = img[:, :, 1] # Composante luminosité/clarté
saturation = img[:, :, 2] # Composante saturation


cv.imshow('Lightness/Clarity Image', lightness)
cv.imshow('Saturation Image', saturation)
cv.waitKey(0)
cv.destroyAllWindows()

plt.figure()
plt.imshow(hue, cmap="hsv")
```

Figure [23] code modifié pour afficher les trois composantes

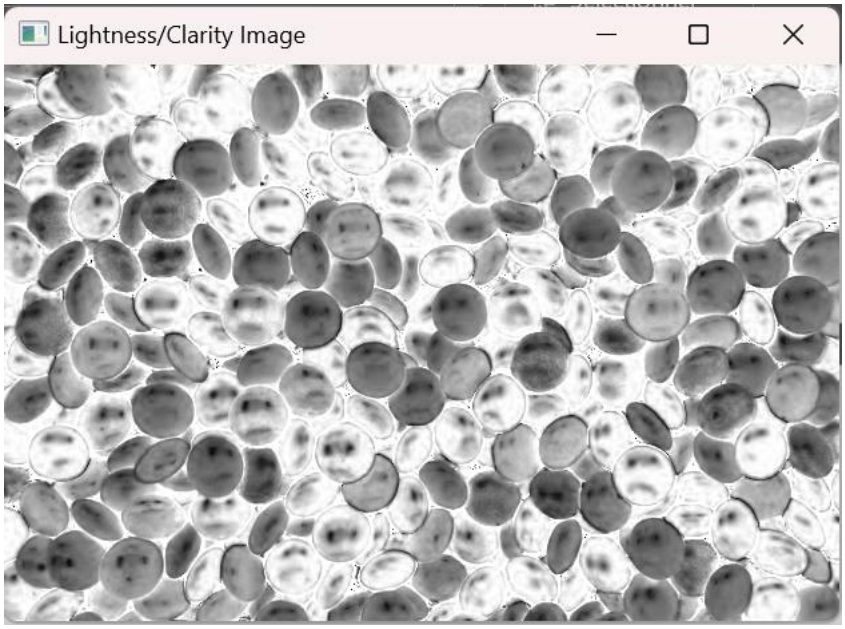

Figure [24] intensité

Figure [25] Saturation

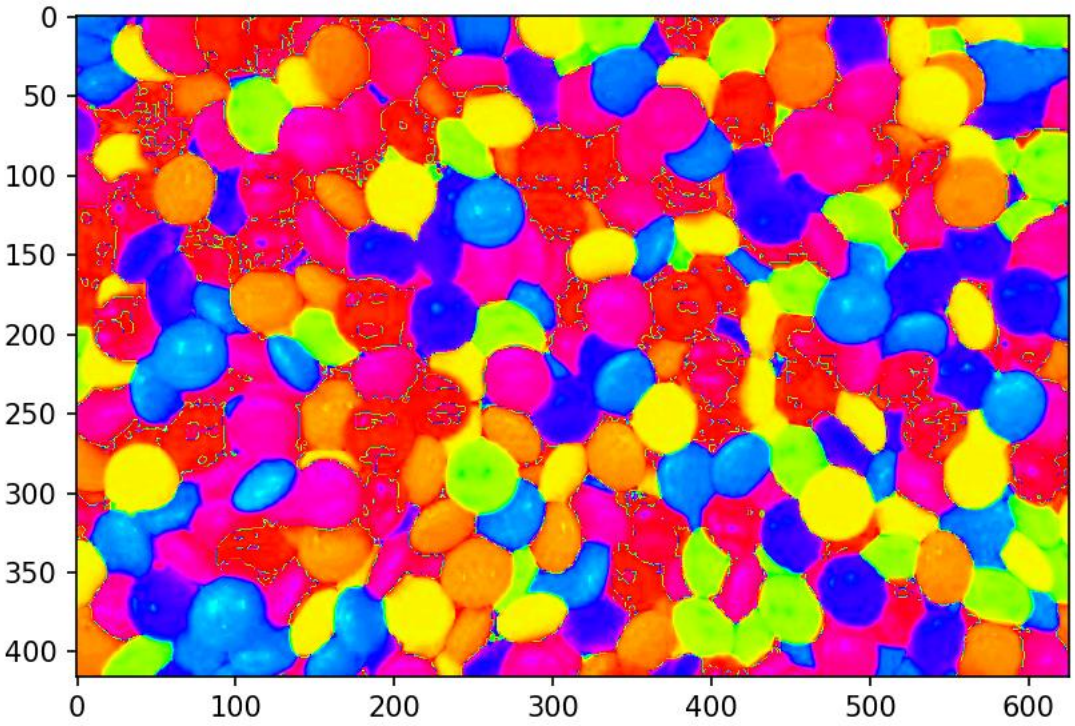

Figure [26] Teinte

On voit sur la Figure [24] pour l'intensité que l'image est en noir et blanc, ce qui est logique car le noir correspond à une intensité très faible et le blanc correspond à une forte intensité. On a aussi vérifié avec l'image des coquelicots et on obtient bien la même chose que dans le cours.

3.

```
def convert_to_IHLS(image):
R = image[:,:,0]
G = image[:,:,1]
B = image[:,:,2]

H = np.zeros_like(image, dtype=np.uint8)
L = np.zeros_like(image, dtype=np.uint8)
S = np.zeros_like(image, dtype=np.uint8)

largeur = len(image)
longueur = len(image[1])

for k in range(largeur):
for i in range(longueur):
r = R[k][i]
g = G[k][i]
b = B[k][i]

denominator = m.sqrt(r**2 + g**2 + b**2 - r*g - r*b - b*g)

if denominator == 0:
H[k][i] = 0 # Valeur par défaut lorsque le dénominateur est égal à zéro
else:
H_prime = m.acos((r - g/2 - b/2) / denominator)
H_prime = H_prime * 180 / m.pi

if b > g:
H[k][i] = 360 - H_prime
else:
H[k][i] = H_prime

L[k][i] = 0.2126*r +0.7152*g+0.0722*b
S[k][i] = max(r,g,b) - min(r,g,b)

return H,L,S


H,L,S = convert_to_IHLS(img)

cv.imshow('Lightness/Clarity Image', L)
cv.imshow('Saturation Image', S)
cv.imshow('Hue', H)
cv.waitKey(0)
cv.destroyAllWindows()

plt.show()
```

Figure [27] code pour convertir l'image en IHSL

Figure [28] intensité en IHSL

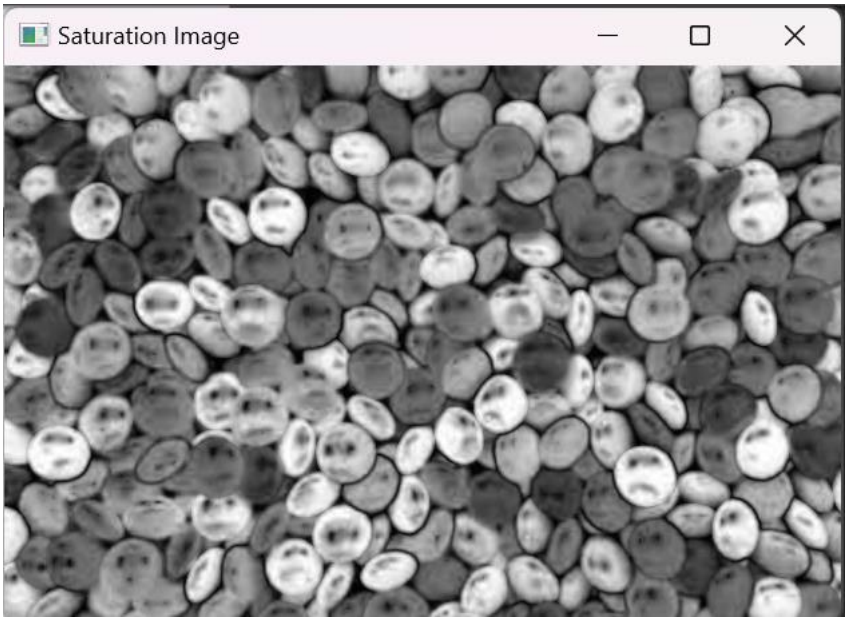

Figure [29] Saturation en IHSL

Pour la teinte on a essayé de l'implémenter mais lorsqu'on essaye d'exécuter le code on obtient une teinte très étrange, presque que du vert. Donc on a décidé de passer à la suite.

4.

Cependant pour la saturation et la luminosité on voit bien que les images sont mieux, on voit plus clairement les détails des deux paramètres.

2. Segmentation d'images

a. Par seuillage

1.

```
image = cv.imread("C:/Users/louka/Desktop/Github/esirem/3A/TP_Image/imagesTP/confiserie-smarties-lentilles_121-50838.jpg")

#Conversion de l'image en espace de couleur BGR
image_bgr = image.copy()

#extraction canal bleu
blue_channel = image_bgr[:, :, 0]
red_channel = image_bgr[:, :, 2]


blue_mask = np.logical_and(blue_channel > 220, red_channel < 130)

masked_image = np.zeros_like(image_bgr)
masked_image[blue_mask] = image_bgr[blue_mask]

# Extraction des canaux rouge et vert de l'image BGR
red_channel = image_bgr[:, :, 2]
green_channel = image_bgr[:, :, 1]

#Création d'un masque pour les pixels jaunes
yellow_mask = np.logical_and(red_channel > 220, green_channel > 220)

#Création d'une image masquée avec seulement les pixels jaunes
masked_image2 = np.zeros_like(image_bgr)
masked_image2[yellow_mask] = image_bgr[yellow_mask]

#Affichage de l'image originale et de l'image masquée
cv.imshow('Original Image', image)
cv.imshow('Masked Image (Blue)', masked_image)
cv.imshow('Masked Image (yellow)', masked_image2)
cv.waitKey(0)
cv.destroyAllWindows()
```

Figure [30] code utilisé pour le seuillage

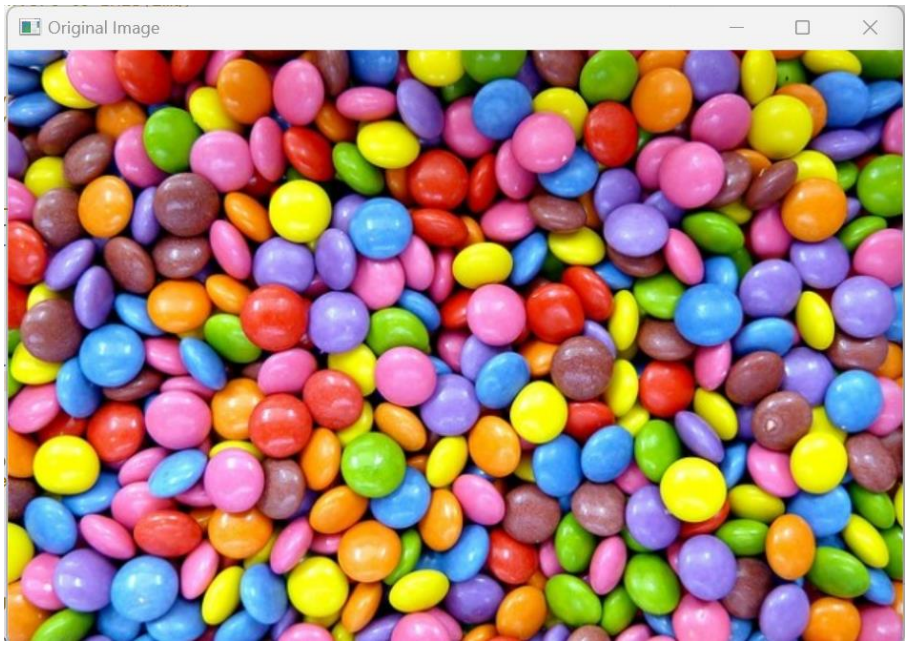

Figure [31] Image originale

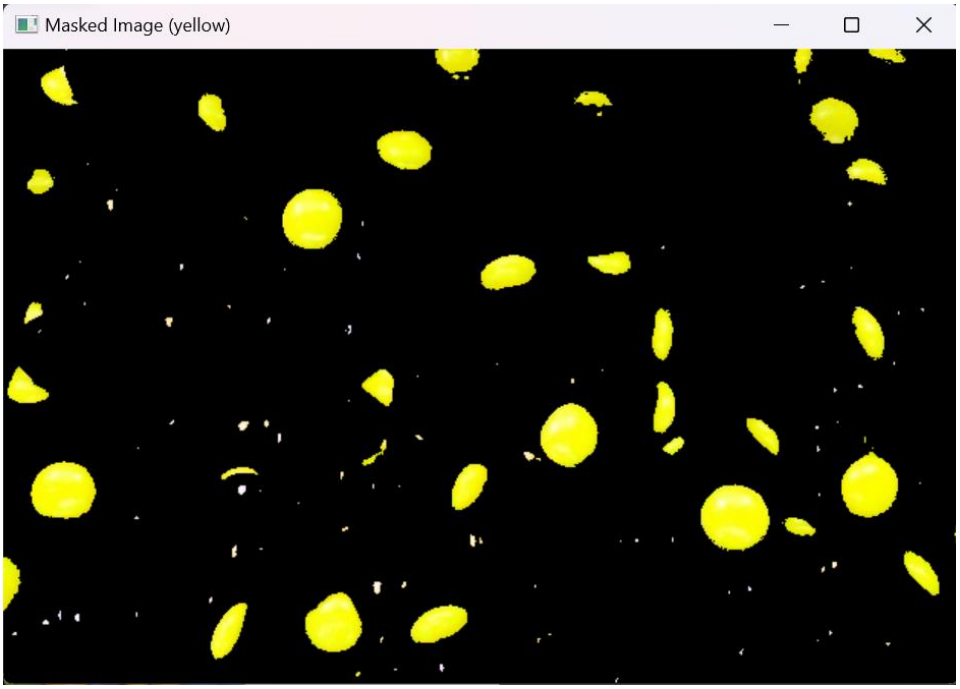

Figure [32] Segmentation des smarties jaune

2.

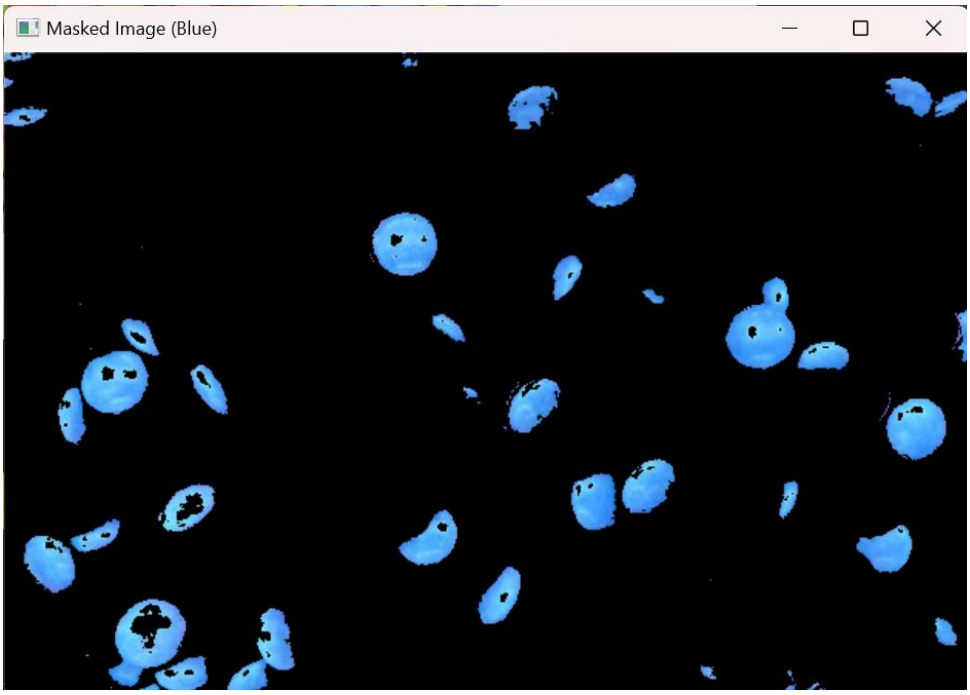

Figure [33] Segmentation des smarties bleu

On définit le seuil manuellement après avoir fais plusieurs tests, on remarque qu'il y a des trous dans les smarties, c'est pourquoi par la suite on va essayer d'améliorer l'image en utilisant des outils de morphologie pour reboucher les trous.

3.

```
#Opération d'ouverture pour éliminer les petits bruits
kernel_open = np.ones((5, 5), np.uint8)
opened_mask = cv.morphologyEx(initial_mask.astype(np.uint8), cv.MORPH_OPEN, kernel_open)

#Opération de fermeture pour fermer les petits et grands trous
kernel_close = np.ones((15, 15), np.uint8)
closed_mask = cv.morphologyEx(opened_mask, cv.MORPH_CLOSE, kernel_close)

#Création d'une image masquée avec seulement les pixels bleus améliorés
masked_image = cv.bitwise_and(image_bgr, image_bgr, mask=closed_mask.astype(np.uint8))

#Affichage de l'image originale, du masque initial et de l'image masquée améliorée
cv.imshow('Masked Image (Improved)', masked_image)
cv.waitKey(0)
cv.destroyAllWindows()
```

Figure [34] code utiliser pour améliorer la segmentation

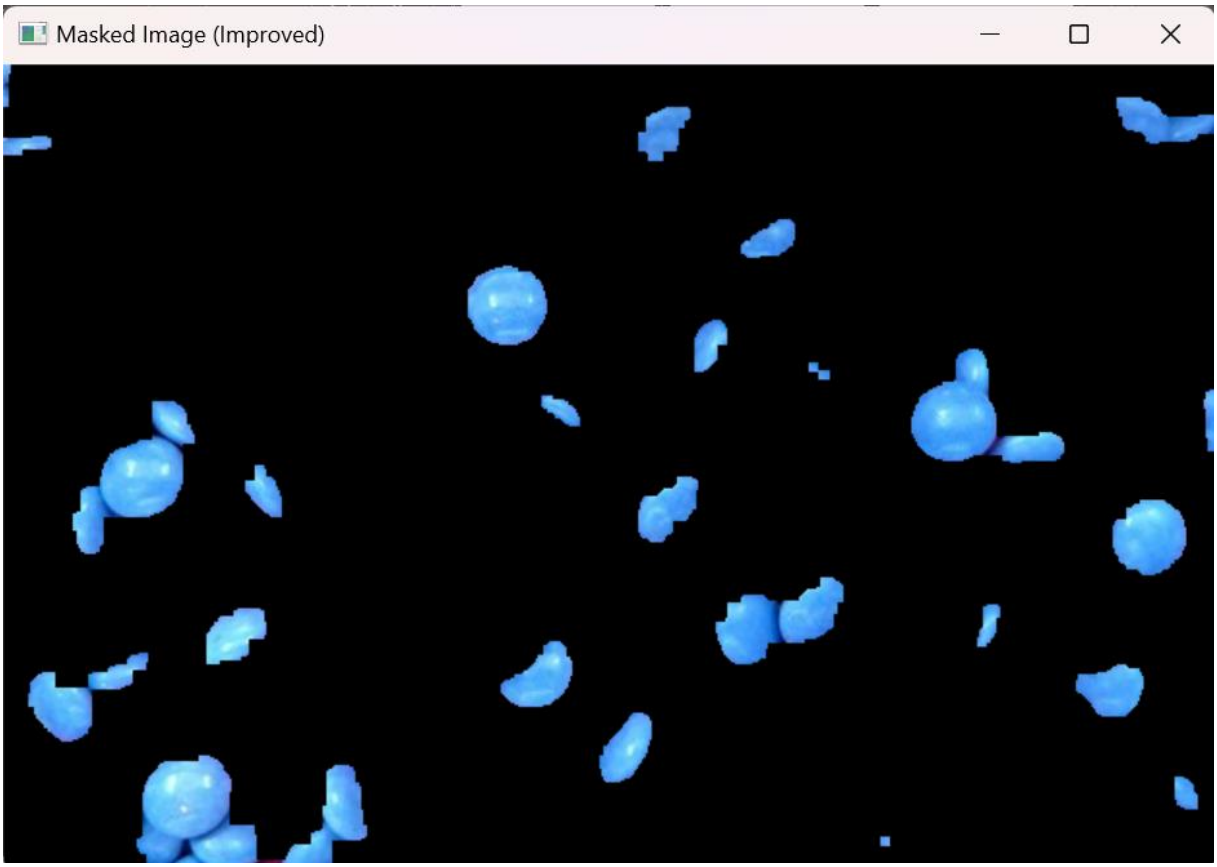

Figure [35] Image obtenu après segmentation

On voit bien qu'après la segmentation on arrive a rebouché les trous sans trop déformer l'image de base, il y a quand même quelques parties qui ont été lisser mais dans l'ensemble tous les trous ont été rebouché.

4.

```
image_bgr =
cv.imread(r"C:/Users/louka/Desktop/Github/esirem/3A/TP_Image/imagesTP/CerisierP.jpg")

#Conversion de l'image en espace HSV
image_hsv = cv.cvtColor(image_bgr, cv.COLOR_BGR2HSV)

plt.figure()
plt.imshow(image_hsv[:, :, 0], cmap="hsv")
plt.colorbar()
plt.show()

#Définition de la plage de teinte pour le bleu
lower_blue = np.array([100, 50, 50])
upper_blue = np.array([130, 255, 255])

#Création du masque pour les pixels bleus
blue_mask = cv.inRange(image_hsv, lower_blue, upper_blue)

inverse_mask = cv.bitwise_not(blue_mask)

#Application du masque à l'image originale
mask_sky = cv.bitwise_and(image_bgr, image_bgr, mask=inverse_mask)

#Affichage de l'image originale et de l'image avec le ciel bleu filtré
cv.imshow("Original Image", image_bgr)
cv.imshow("image ciel filtré", mask_sky)

cv.waitKey(0)
cv.destroyAllWindows()
```

Figure [36] Code pour filtrer l’image

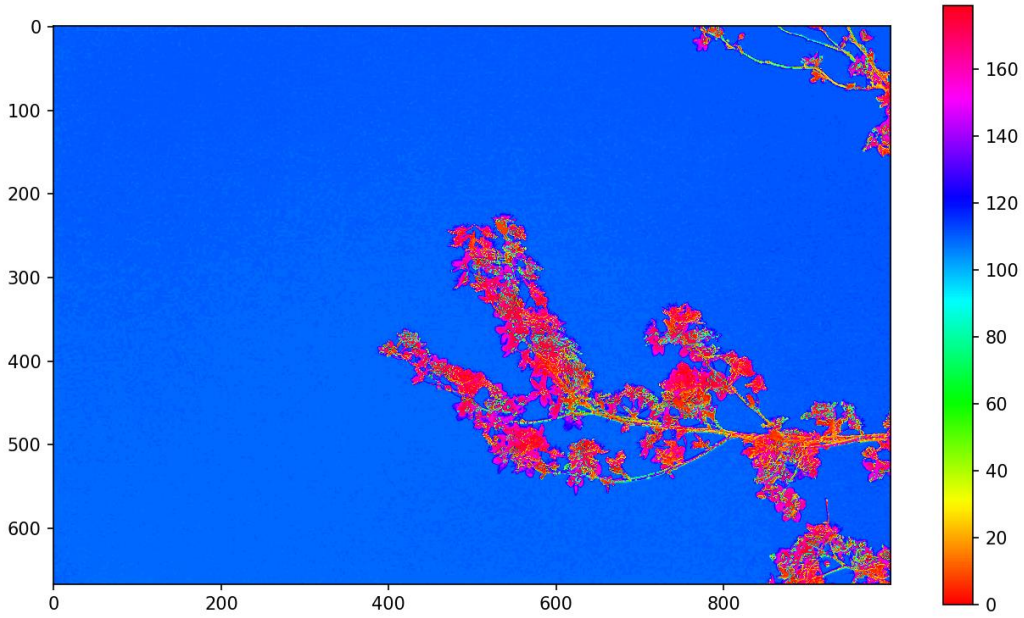

Figure [37] Teinte de l'image

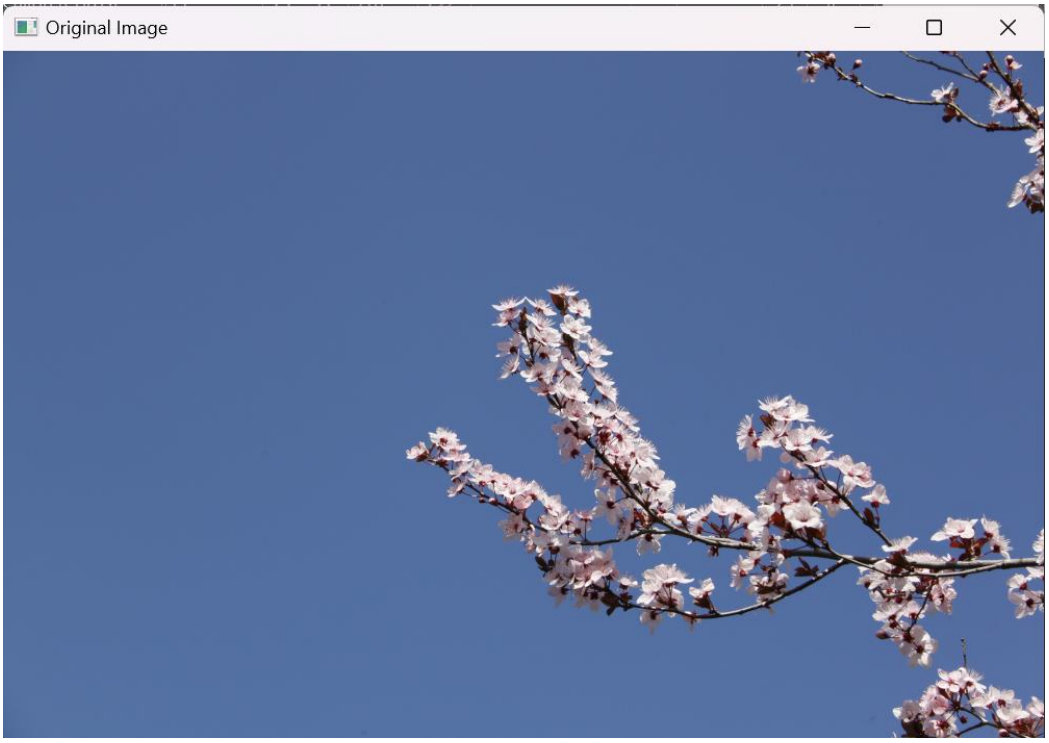

Figure [38] Image originale

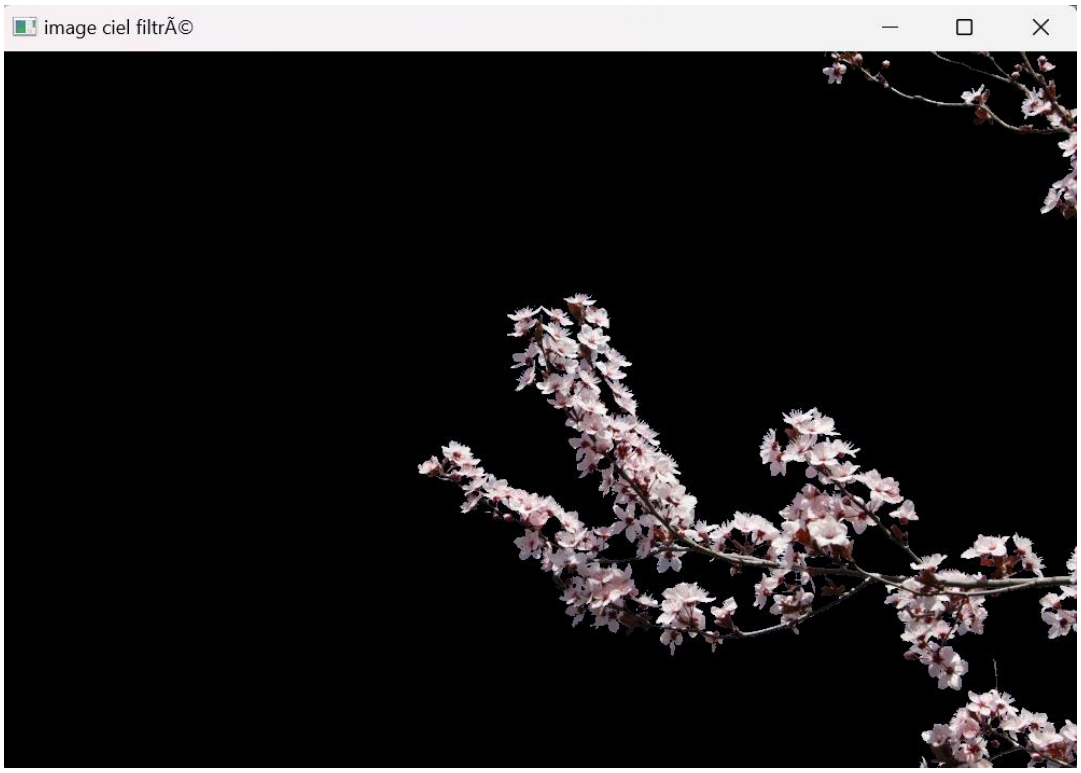

Figure [39] Image filtrée

On commence par afficher la teinte de l'image en hsv pour pouvoir déterminer le seuil haut et le seuil bas que l'on va utiliser pour filtrer l'image, ici entre 100 et 130 (ce qui correspond au bleu du ciel). Puis on crée un masque que l'on inverse pour avoir du noir dans les parties que l'on ne veut pas et du blanc ou on veut garder. On fait ensuite un ET logique entre l'image de base et le masque et on obtient l'image Figure [39].

5.

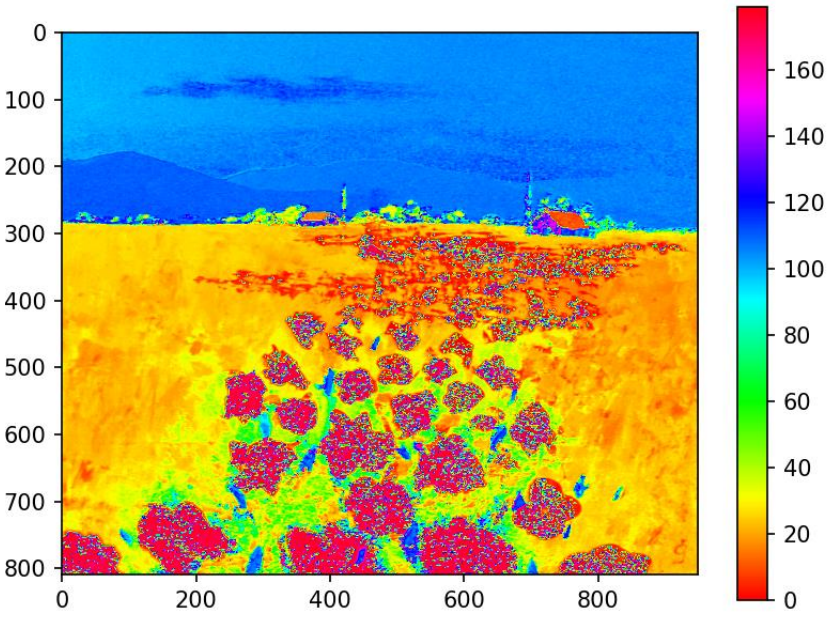

Figure [40] Teinte des coquelicots

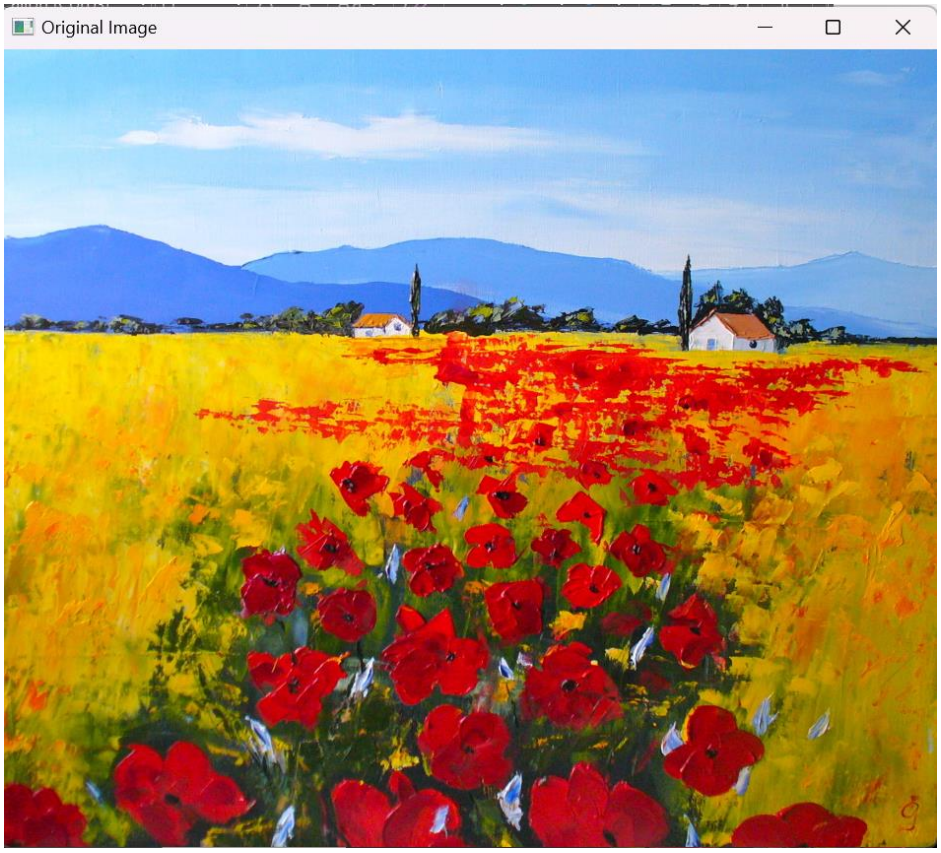

Figure [41] Image de base

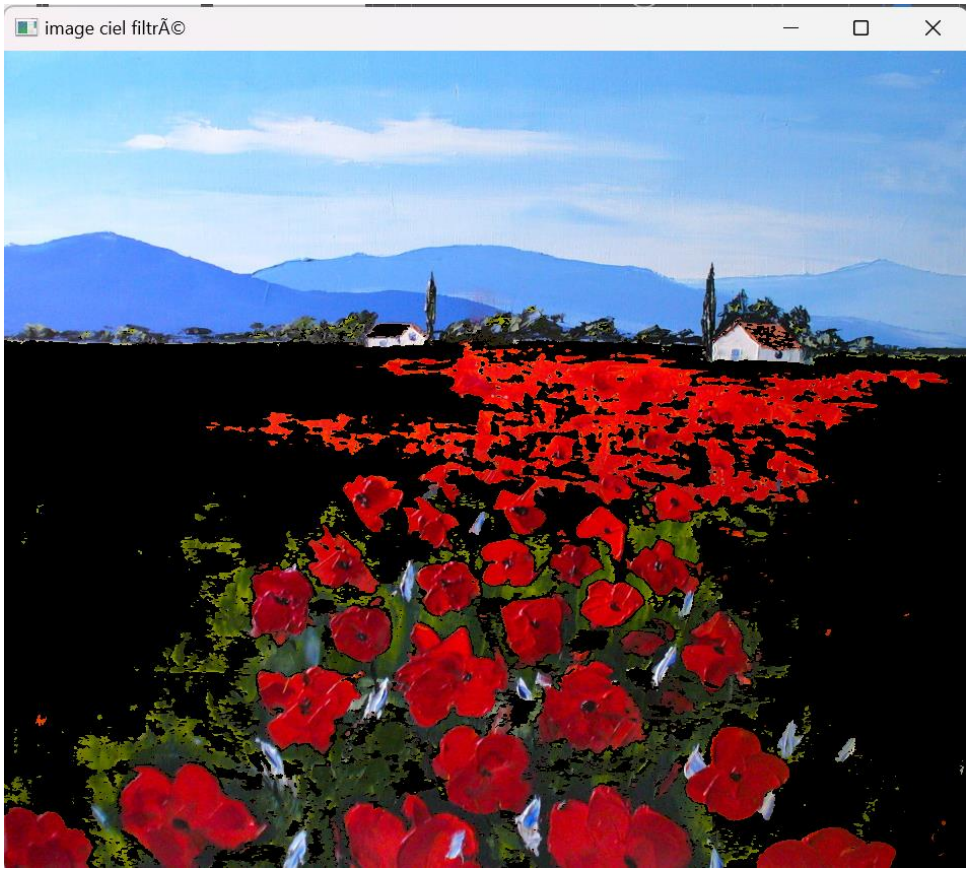

Figure [42] Image filtré

Ici on teste avec l'image des coquelicots, on veut cette fois-ci enlever le jaune de l'image. On utilise la Figure [40] pour déterminer les seuils ici 10 et 30. Et on utilise le code de la Figure [36] en adaptant les valeurs pour obtenir le Figure [42].

6.

```
# Chargement de l'image
image_bgr =
cv.imread(r"C:/Users/louka/Desktop/Github/esirem/3A/TP_Image/imagesTP/CerisierP.jpg")

# Conversion de l'image en espace HSV
image_hsv = cv.cvtColor(image_bgr, cv.COLOR_BGR2HSV)

plt.figure()
plt.imshow(image_hsv[:, :, 0], cmap="hsv")
plt.colorbar()
plt.show()

# Définition de la plage de teinte pour le bleu
lower_blue = np.array([100, 50, 50])
upper_blue = np.array([130, 255, 255])

# Création du masque pour les pixels bleus
blue_mask = cv.inRange(image_hsv, lower_blue, upper_blue)

# Application du masque au canal rouge et au canal vert pour les modifier en rouge
image_bgr[:, :, 0] = np.where(blue_mask > 0, 0, image_bgr[:, :, 0]) # Canal bleu
image_bgr[:, :, 1] = np.where(blue_mask > 0, 0, image_bgr[:, :, 1]) # Canal vert

# Affichage de l'image originale et de l'image avec le ciel bleu filtré
cv.imshow("Original Image", image_bgr)

cv.waitKey(0)
cv.destroyAllWindows()
```

Figure [43] code pour changer la couleur de fond

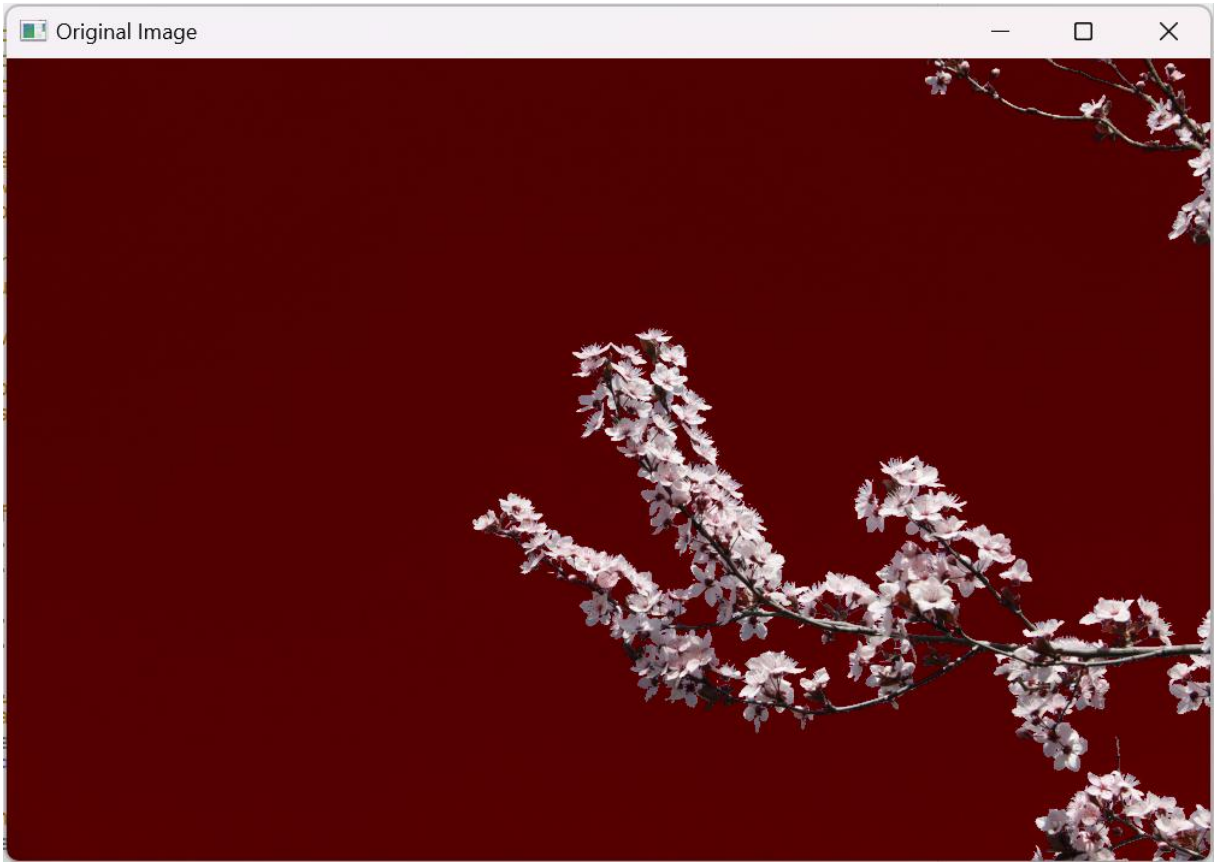

Figure [44] Image obtenu

Cette fois-ci on veut changer la couleur du fond de l'image en mettant la couleur de notre choix. Pour afficher un fond rouge on filtre simplement les composantes vertes et bleus de notre image sans toucher à la composante rouge. Et obtient l'image sur la Figure [44].

b. Pour aller plus loin

1.

On a repris le code donné sur github avec le lien dans le cours. Lorsqu'on essaie d'utiliser le code avec les images fournies sur github tout se passe bien. En revanche lorsqu'on essaie avec d'autre image on obtient des messages d'erreurs.