Relatório Laboratorio 7

Thiago Ayres Kimura – RA: 22.221.045-2

Explicação Código (Contorno):

contorno.py

```python
#pip install opencv-python

import math

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

#Importa e converta para RGB
img = cv2.imread('./AVIAO_01.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

```

Nesta primeira parte do Código importamos as bibliotecas necessárias para o funcionamento do código assim como a imagem, também convertermos a imagem para RGB na linha 11.

```python

#Convertendo para preto e branco (RGB -> Gray Scale -> BW)
img_gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
a = img_gray.max()
_, thresh = cv2.threshold(img_gray, a/2*1.7, a,cv2.THRESH_BINARY_INV)


tamanhoKernel = 5
kernel = np.ones((tamanhoKernel,tamanhoKernel), np.uint8)
thresh_open = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)

#Filtro de ruído (bluring)
img_blur = cv2.blur(img_gray, ksize=(tamanhoKernel,tamanhoKernel))

# Detecção borda com Canny (sem blurry)
edges_gray = cv2.Canny(image=img_gray, threshold1=a/2, threshold2=a/2)
# Detecção borda com Canny (com blurry)
edges_blur = cv2.Canny(image=img_blur, threshold1=a/2, threshold2=a/2)

```

Nesta parte convertemos para preto e branco, assim como definimos o tamanho do kernel. Aplicamos o filtro de ruido na linha 26, e nas linhas 29 e 31 detectamos a borda com Canny sem blurry e com blurry, respectivamente

```
# contorno
contours, hierarchy = cv2.findContours(
                                    image = thresh,
                                    mode = cv2.RETR_TREE,
                                    method = cv2.CHAIN_APPROX_SIMPLE)
contours = sorted(contours, key = cv2.contourArea, reverse = True)
img_copy = img.copy()
final = cv2.drawContours(img_copy, contours, contourIdx = -1,
                         color = (255, 0, 0), thickness = 2)


#plot imagens
imagens = [img,img_blur,img_gray,edges_gray,edges_blur,thresh,thresh_open,final]
formatoX = math.ceil(len(imagens)**.5)
if (formatoX**2-len(imagens))>formatoX:
    formatoY = formatoX-1
else:
    formatoY = formatoX
for i in range(len(imagens)):
    plt.subplot(formatoY, formatoX, i + 1)
    plt.imshow(imagens[i],'gray')
    plt.xticks([]),plt.yticks([])
plt.show()

```

Aqui aplicamos o contorno nas imagens e plotamos a imagem com os efeitos aplicados a ela.

Imagens(Contorno):

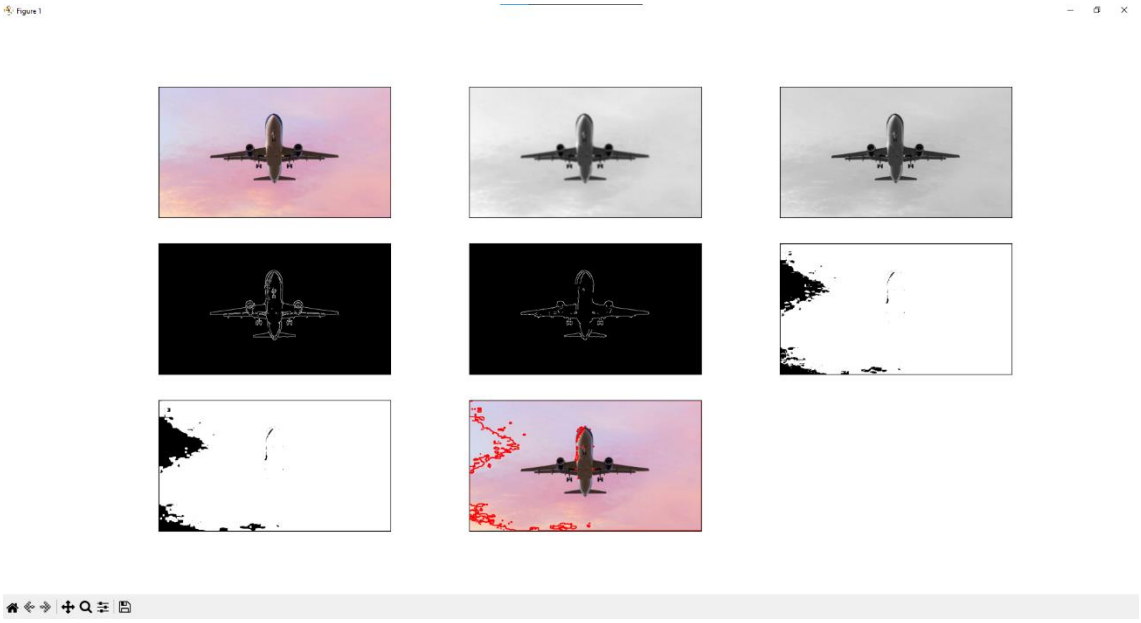


Figure 1

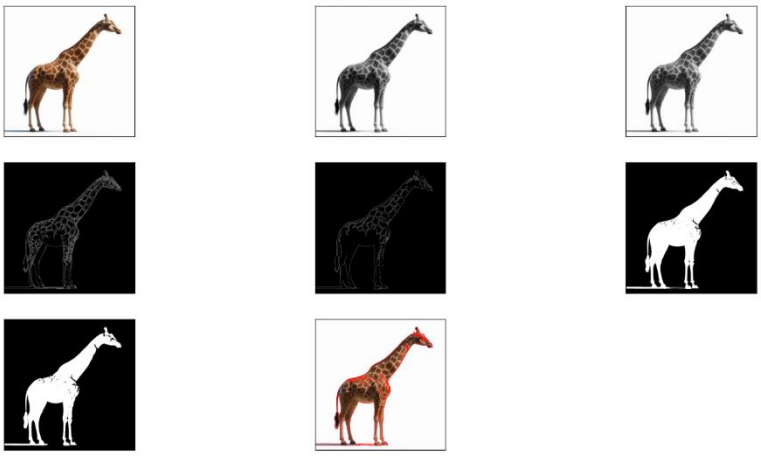


Figure 1

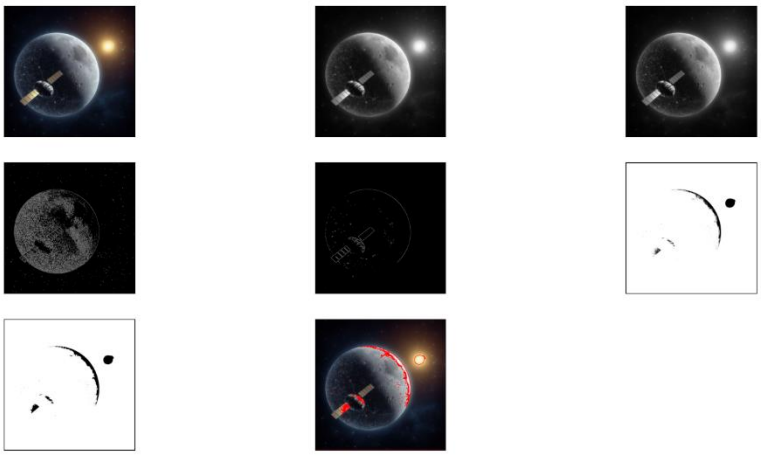
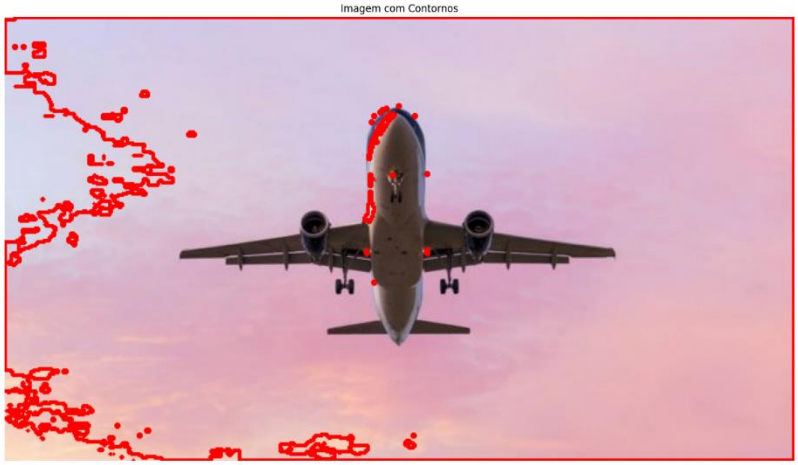


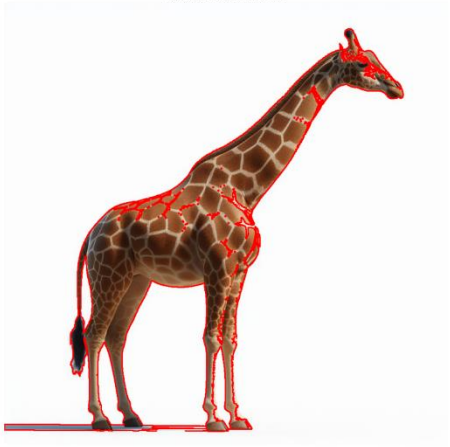
Figure 2



(x, y) = (51.4, 94.4)
[216, 213, 234]

Figure 2

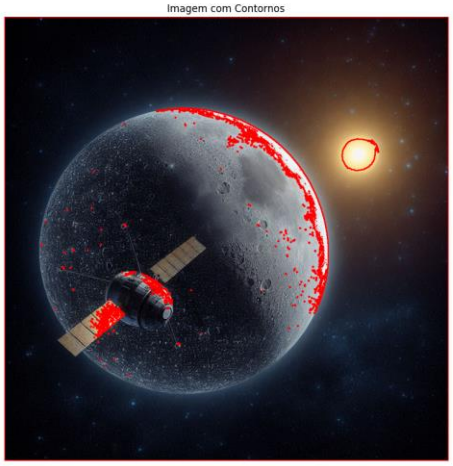
Imagem com Contornos



(x, y) = (633., 115.)
[253, 253, 253]

Figure 2

Imagem com Contornos



(x, y) = (731., 171.)
[73, 46, 35]

Explicação Código (Espaço de cores):

espacoCores.py

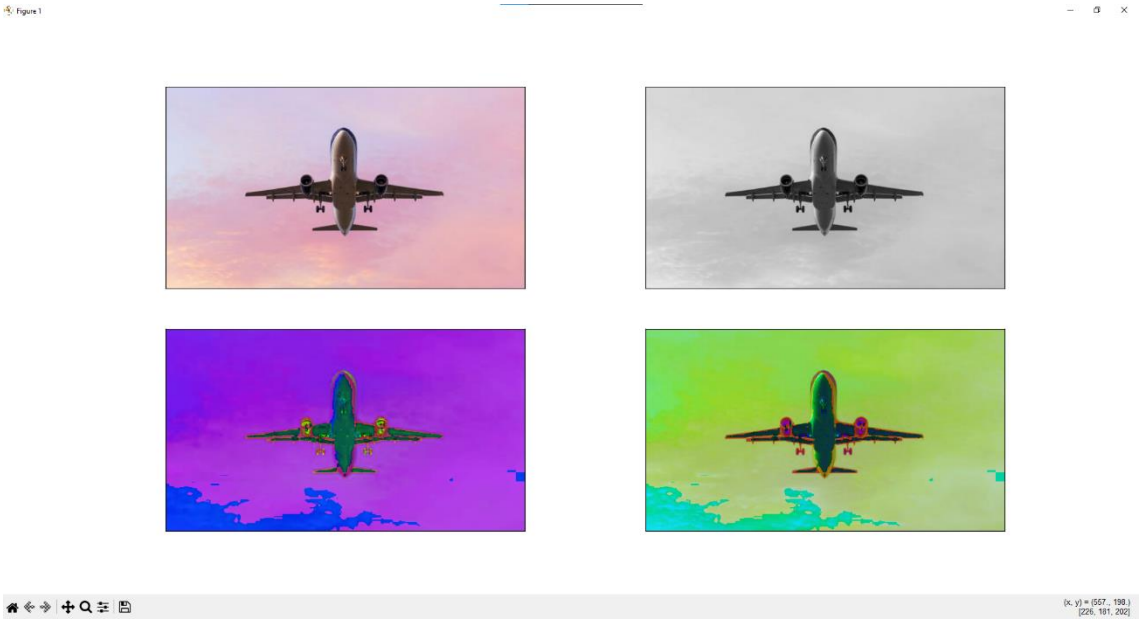
```python
#pip install opencv-python

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

# Carregans imagem
img = cv2.imread('./AVIAO_01.jpg')

```

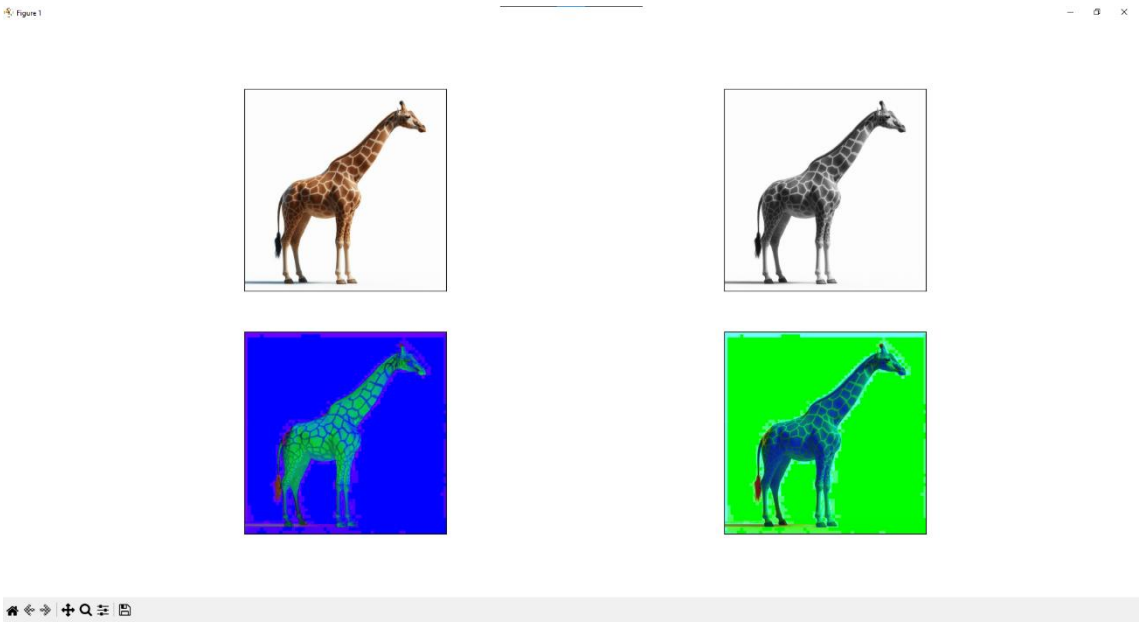
Na primeira parte do código importamos as bibliotecas necessárias para o código funcionar, também carregamos a imagem que iremos aplicar os filtros.

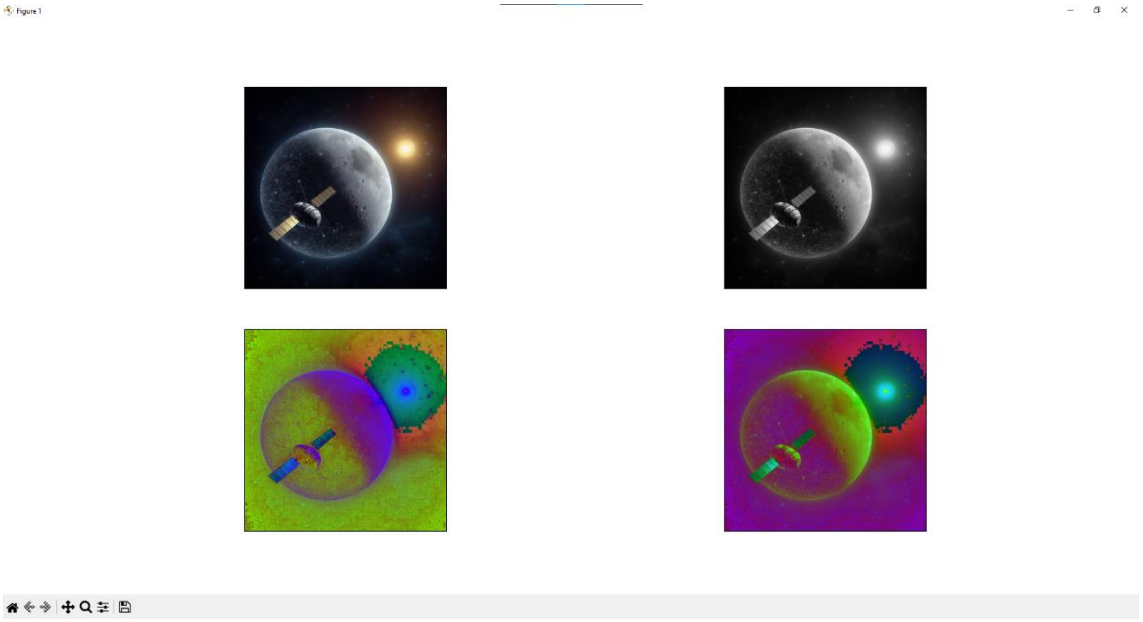
```python

# Convertendo espaço de cores
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
img_hls = cv2.cvtColor(img, cv2.COLOR_BGR2HLS)

#plot imagens
imagens = [img_rgb,img_gray,img_hsv,img_hls]

for i in range(4):
    plt.subplot(2,2,i+1)
    plt.imshow(imagens[i],'gray')
    plt.xticks([]),plt.yticks([])
plt.show()




plt.show()

```

Aqui convertemos as imagens para RGB, Cinza e HSV e HLS. Logo depois, plotamos a imagem para visualizarmos o espaço de cores.

Imagens(Espaço de Cores):







Explicação Código (Operador Morfológico):

operadorMorfologico.py

```python
#pip install opencv-python
import numpy as np
import math
import cv2
import matplotlib.pyplot as plt

#Importa e converta para RGB
img = cv2.imread('./AVIAO_01.jpg')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

#Filtro de ruído (bluring)
img_blur = cv2.blur(img,(5,5))
```

No começo do código importamos as bibliotecas necessárias para o código funcionar, importamos também as imagens e convertemos para RGB e por fim, aplicamos o filtro de ruido na imagem.

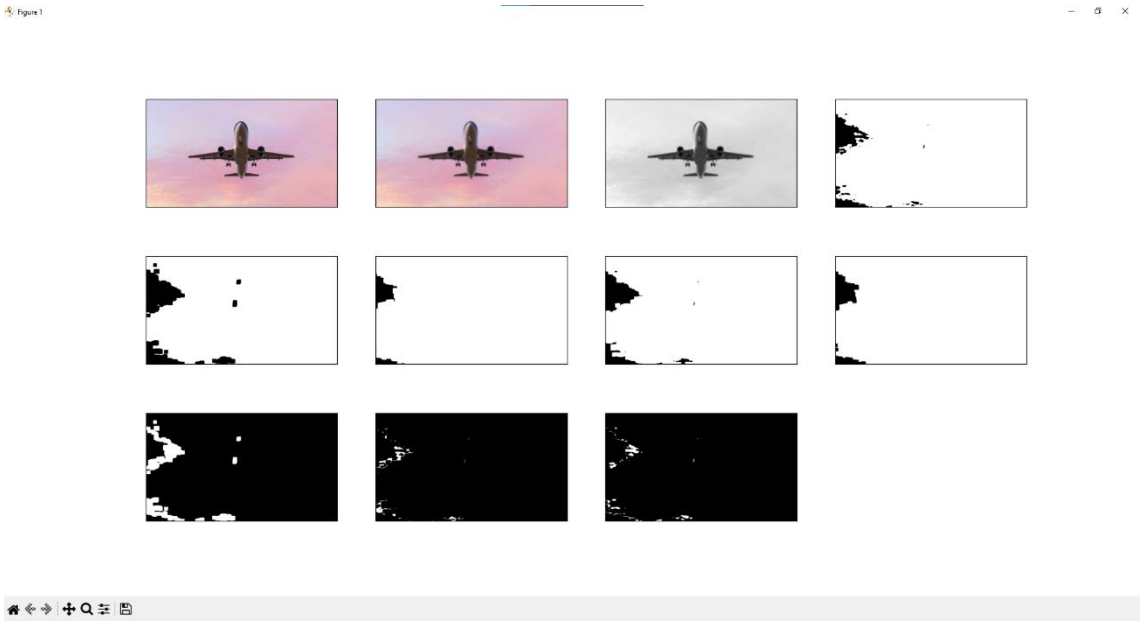
```python
#Convertendo para preto e branco (RGB -> Gray Scale -> BW)
img_gray = cv2.cvtColor(img_blur, cv2.COLOR_RGB2GRAY)
a = img_gray.max()
_, thresh = cv2.threshold(img_gray, a/2+100, a,cv2.THRESH_BINARY_INV)

#preparando o "kernel"
kernel = np.ones((12,12), np.uint8)
```

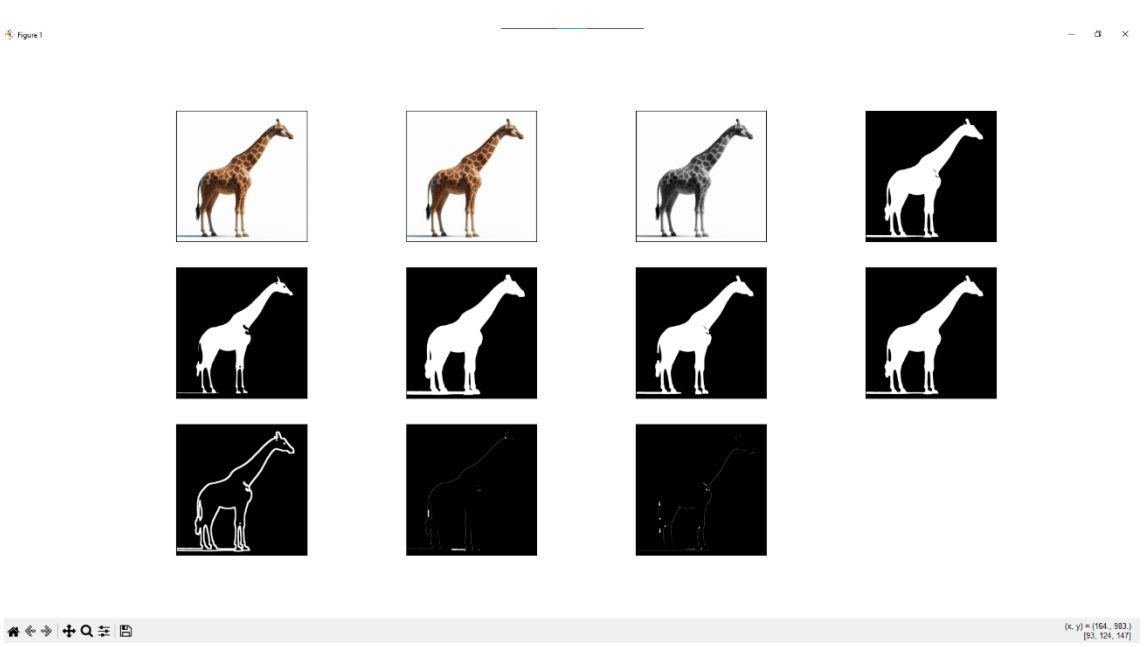
Convertemos a imagem para preto e branco e preparamos o Kernel para tratarmos a imagem.

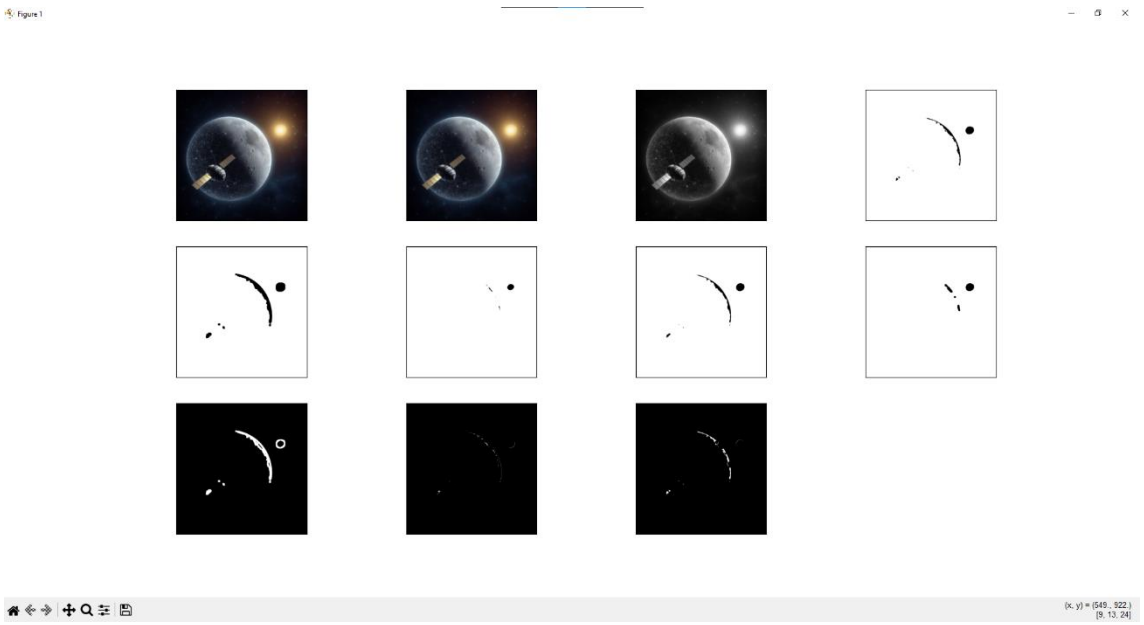
```python
#operadores Morfologicos
img_dilate = cv2.dilate(thresh,kernel,iterations = 1)
img_erode = cv2.erode(thresh,kernel,iterations = 1)
img_open = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)
img_close = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel)
img_grad = cv2.morphologyEx(thresh, cv2.MORPH_GRADIENT, kernel)
img_tophat = cv2.morphologyEx(thresh, cv2.MORPH_TOPHAT, kernel)
img_blackhat = cv2.morphologyEx(thresh, cv2.MORPH_BLACKHAT, kernel)

# Plot the images
imagens = [img, img_blur,  img_gray,thresh,img_erode,img_dilate, img_open, img_close, img_grad,
           img_tophat, img_blackhat]

formatoX = math.ceil(len(imagens)**.5)
if (formatoX**2-len(imagens))>formatoX:
    formatoY = formatoX-1
else:
    formatoY = formatoX

for i in range(len(imagens)):
    plt.subplot(formatoY, formatoX, i + 1)
    plt.imshow(imagens[i],'gray')
    plt.xticks([]),plt.yticks([])
plt.show()

```

Nesta parte, aplicamos os operadores morfológicos e plotamos as imagens.

Imagens (Operador Morfológico):







https://github.com/ThiagoAKimura/Atividades_Intelegencia_Artificial.git