UNIVERSIDAD NACIONAL DE SAN AGUSTÍN

ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN

COMPUTACION GRAFICA

Practica 12

Alumnos:
Chayña Batallanes Josnick
Perez Rodriguez Angelo Aldo
Pucho Zevallos Kelvin Paul
Vilcapaza Flores Luis Felipe
Sihuinta Perez Luis Armando

Julio 2021

Índice

1

1. Algoritmo de computación gráfica Image Arithmetic (Adición y Sustracción de imágenes). 3

1.1. Implemente la adición de imágenes con los elementos de la Figura 1. Se recomiendo hacer un cast a la imagen antes del procesamiento para evitar el overflow en los píxeles. 3

1.2. Ahora implemente la adición con imágenes a colores (Figura 2). 5

1.3. Implemente la sustracción de imágenes para segmentar letras. Se le está brindando una foto del documento y otra de una hoja en blanco para eliminar el reflejo de la luz (Figura 3). Después de la sustracción aplique thresholding para obtener un resultado similar a la Figura 4. Tiene la liberta de aplicar métodos adicionales para mejorar los resultados, por ejemplo: contrast stretching, histogram equalization, etc. 7

1.4. Implemente la sustracción de imágenes para detectar el cambio o movimiento de objetos en fotogramas. En la Figura 5, se brindan dos fotogramas consecutivos, implemente la sustracción para obtener una imagen donde se visualiza qué objetos han cambiado de posición. 10

2. Algoritmo de computación gráfica Image Arithmetic (Multiplicación y División de imágenes) 12

2.1. Implemente la multiplicación de imagen por una constante con la Figura 1. Evalue con c = 2, c = 5 y c = 7. 12

2.2. Implemente la división de imágenes para segmentar letras. Se le está brindando una foto del documento y otra de una hoja en blanco para eliminar el reflejo de la luz (Figura 2). Después de la división deber a normalizar la imágen a valores entre $[0 - 255]$ (Ecuación 1). Después aplique thresholding para obtener un resultado similar a la Figura 3. Tiene la libertad de aplicar métodos adicionales para mejorar los resultados, por ejemplo: contrast stretching, histogram equalization, etc. 14

2.3. Implemente la división de imágenes para detectar el cambio o movimiento de objetos en fotogramas. En la Figura 4, se brindan dos fotogramas consecutivos, implemente la división para obtener una imagen donde se visualiza que objetos se movieron, quizás sea necesario multiplicar el resultado por una constante para poder visualizar mejor los resultados. Después puede aplicar contrast stretching para mejorar a ´un mas los resultados 17

2.4. Implemente el operador Blending (ecuación 2) y evalúe sus resultados con imágenes de su preferencia, también pruebe diferentes valores de X. 19

3. Algoritmo de computación gráfica Image Aritmética.(Review) 21

4. Algoritmo de computación gráfica Image Logical. 26

4.1. Implemente el operador AND con las imágenes de la Figura 1 para segmentar el objeto que aparece en ambas imágenes (intersección). 27

4.2. Implemente el operador OR con las imágenes de la Figura 1 para unir los objetos que aparecen en las imágenes (fusión). 30
4.3. Implemente el operador XOR con las imágenes de la Figura 1 para detectar los cambios. 34

5. Link de los códigos en github **38**

1. Algoritmo de computación gráfica Image Arithmetic (Adición y Sustracción de imágenes).

1.1. Implemente la adición de imágenes con los elementos de la Figura 1. Se recomiendo hacer un cast a la imagen antes del procesamiento para evitar el overflow en los píxeles.

- **Adiciones de Imágenes**

  - **Imagen Original**:

(a) Imagen 1 (b) Imagen 2

Figura 1: Adición de Imagénes

  - **Imagen Resultado** :

- **Codigo**:

```
import cv2
import numpy as np

imgReal1 = cv2.imread("image_1_1.jpg", cv2.IMREAD_GRAYSCALE) #8 bit
    por escala de grises
imgReal1 = cv2.resize(imgReal1,(400,400))
img1 = imgReal1

imgReal2 = cv2.imread("image_1_2.jpg", cv2.IMREAD_GRAYSCALE) #8 bit
    por escala de grises
imgReal2 = cv2.resize(imgReal2,(400,400))
img2 = imgReal2

cv2.imshow("Image Original 1", imgReal1)
cv2.imshow("Image Original 2", imgReal2)

img1 = img1.astype(int)
img2 = img2.astype(int)
imgResult = img1.copy()

for i in range(len(img1)):
    for j in range(len(img1[0])):
        sum_ = img1[i,j]//2 + img2[i,j]//2
        if sum_ > 255 :
            imgResult[i,j] = 255
        else:
            imgResult[i,j] = sum_
imgResult = imgResult.astype(np.uint8)
cv2.imshow("Resultado", imgResult)
filename = 'Resultado_Image1_2.jpg'
cv2.imwrite(filename, imgResult)
```

```
30
31  cv2.waitKey(0)
32  cv2.destroyAllWindows()
```

1.2. Ahora implemente la adición con imágenes a colores (Figura 2).

- **Adiciones de Imagenes**

  - **Imagen Original**:

(a) Imagen 1 (b) Imagen 2

Figura 2: Adición de Imagenes

  - **Imagen Resultado** :

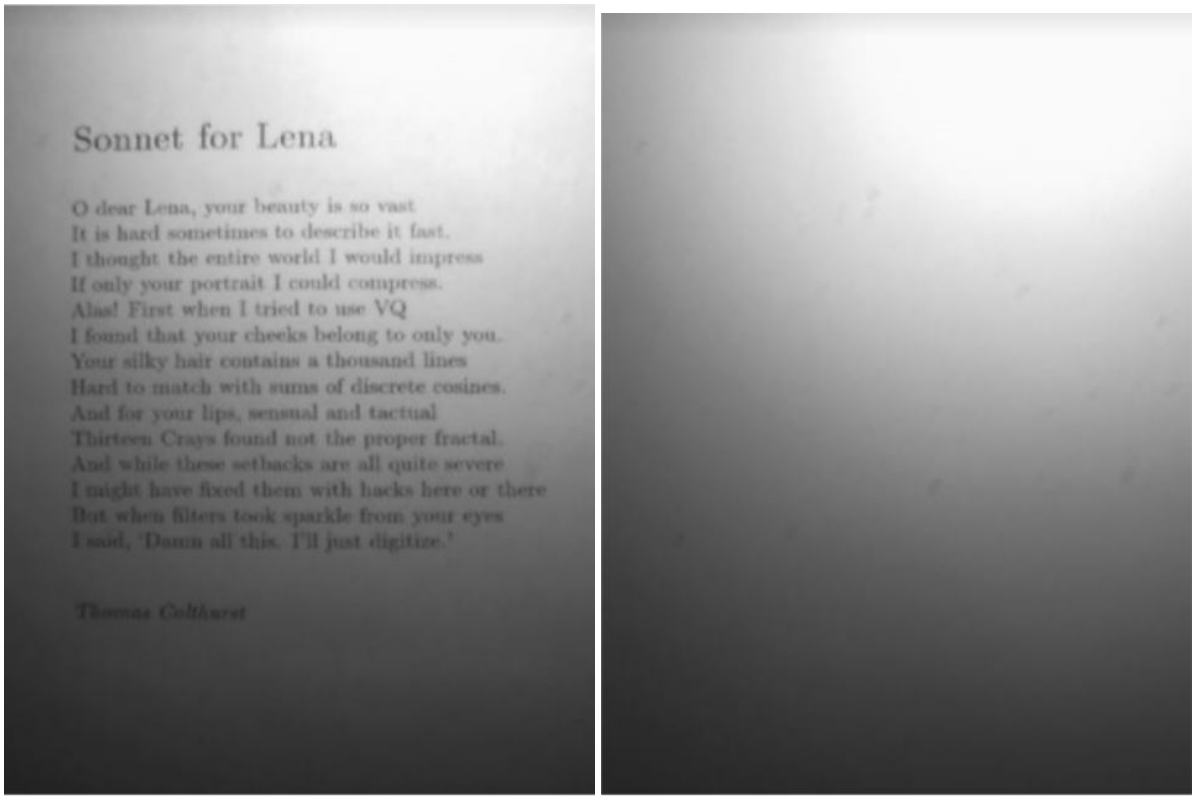
- **Código**:

```
import cv2
import numpy as np

imgReal1 = cv2.imread("ciudad.jpg",1) #8 bit por escala de grises
imgReal1 = cv2.resize(imgReal1,(400,400))
img1 = imgReal1

imgReal2 = cv2.imread("leon.jpg",1) #8 bit por escala de grises
imgReal2 = cv2.resize(imgReal2,(400,400))
img2 = imgReal2


cv2.imshow("Image Original 1", imgReal1)
cv2.imshow("Image Original 2", imgReal2)

B1, G1, R1 = cv2.split(img1)
B2, G2, R2 = cv2.split(img2)
B1, G1, R1 = B1.astype(int), G1.astype(int), R1.astype(int)
B2, G2, R2 = B2.astype(int), G2.astype(int), R2.astype(int)
Bresult,Gresult,Rresult = B1.copy(), G1.copy(), R1.copy()

print(B1.dtype)
# print(img2)
for i in range(len(B1)):
    for j in range(len(B1[0])):
        sum_B = B1[i,j]//2 + B2[i,j]//2
        if sum_B > 255 :
            Bresult[i,j] = 255
        else:
            Bresult[i,j] = sum_B

        sum_G = G1[i,j]//2 + G2[i,j]//2
        if sum_G > 255 :
            Gresult[i,j] = 255
        else:
            Gresult[i,j] = sum_G

        sum_R = R1[i,j]//2 + R2[i,j]//2
        if sum_R > 255 :
            Rresult[i,j] = 255
        else:
            Rresult[i,j] = sum_R

img_result = cv2.merge((Bresult,Gresult,Rresult))
img_result = np.array(img_result, dtype=np.uint8)
cv2.imshow('ResultadoLeonCiudad', img_result)
filename = 'ResultadoLeonCiudad.jpg'
cv2.imwrite(filename, img_result)

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

1.3. Implemente la sustracción de imágenes para segmentar letras. Se le está brindando una foto del documento y otra de una hoja en blanco para eliminar el reflejo de la luz (Figura 3). Después de la sustracción aplique thresholding para obtener un resultado similar a la Figura 4. Tiene la liberta de aplicar métodos adicionales para mejorar los resultados, por ejemplo: contrast stretching, histogram equalization, etc.

- **Sustracción de Imágenes**

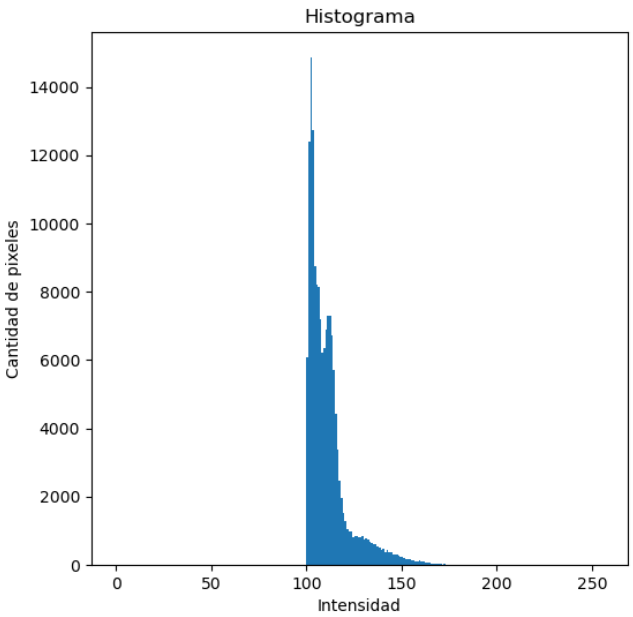
  - **Imagen Original**:



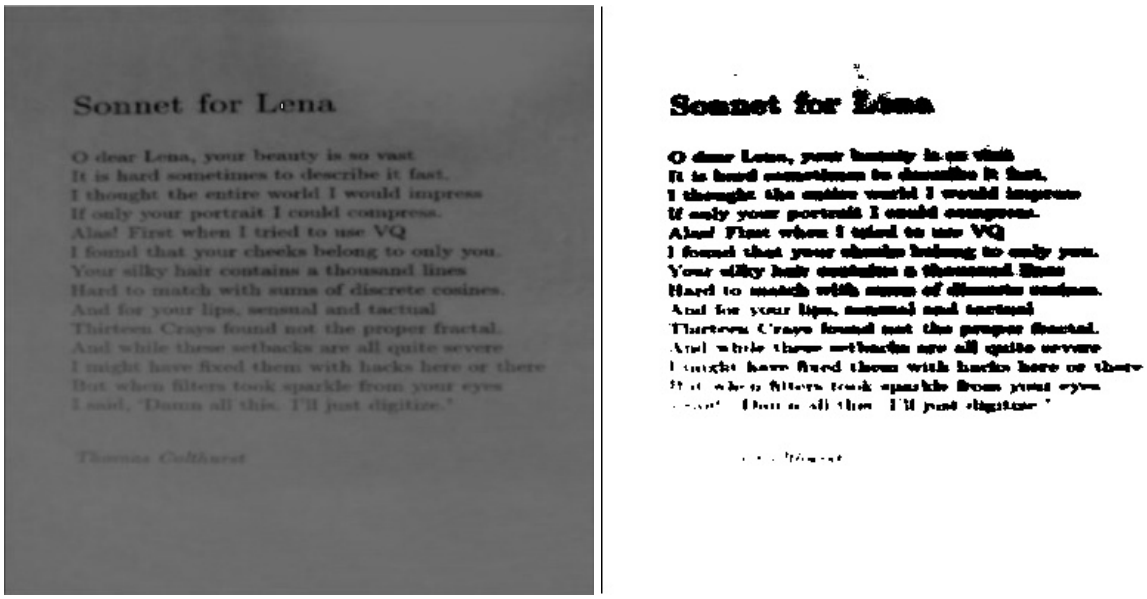
(a) Imagen 1 (b) Imagen 2

Figura 3: Sustracción de Imágenes

  - **Histograma** :
    Se uso un Thresold de 90



- **Resultados** :



(a) Imagen de la Sustracción de imágenes (b) Imagen aplicando Thresolding

Figura 4: Sustracción de Imágenes

- **Código**:

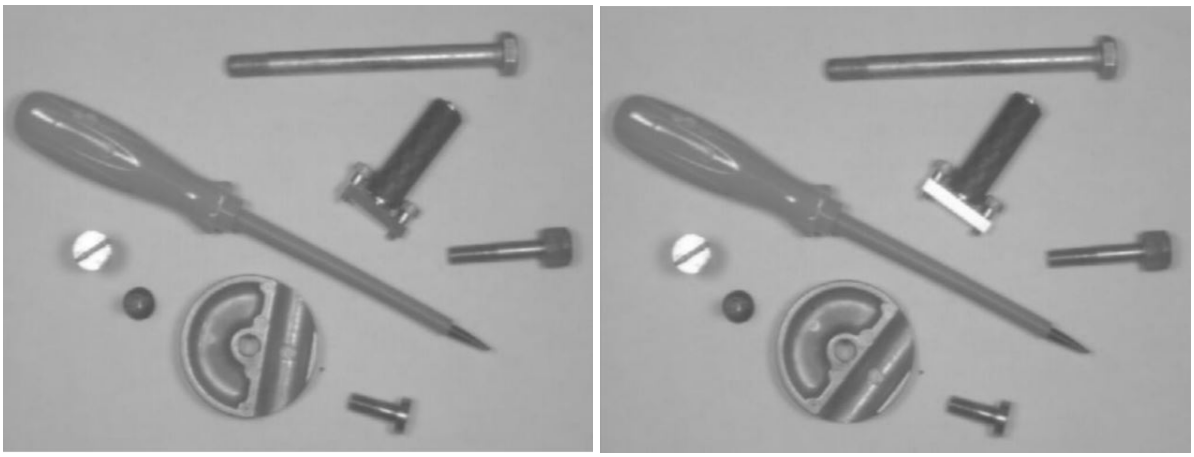
```
import cv2
import numpy as np
from matplotlib import pyplot as plt


def getThresholding(img,threshold):
    modifiedImg = img.copy()
    print(img)
    for i in range(len(img)):
        for j in range(len(img[i])):
            if(img[i][j] < threshold):
                modifiedImg[i][j] = np.uint8(0)
            else:
                modifiedImg[i][j] = np.uint8(255)
    return modifiedImg

imgReal1 = cv2.imread("texto1.jpg", cv2.IMREAD_GRAYSCALE) #8 bit por
   escala de grises
imgReal1 = cv2.resize(imgReal1,(400,400))
img1 = imgReal1

imgReal2 = cv2.imread("textoblanco.jpg", cv2.IMREAD_GRAYSCALE) #8 bit
    por escala de grises
imgReal2 = cv2.resize(imgReal2,(400,400))
img2 = imgReal2

cv2.imshow("Image Original 1", imgReal1)
cv2.imshow("Image Original 2", imgReal2)

img1 = img1.astype(int)
img2 = img2.astype(int)
imgResult = img1.copy()
# print(imgResult.dtype)
print(imgResult)
# imgResult = list(img1.flatten())
# imgResult = np.reshape(imgResult, (img1.shape[0], img1.shape[2]))

c = 110
for i in range(len(img1)):
    for j in range(len(img1[0])):
        rest = img1[i,j] - img2[i,j]
        imgResult[i,j] = rest + c
imgResult = imgResult.astype(np.uint8)
cv2.imshow("Resultado Sustraccion", imgResult)

# filename = 'Resultado_Sustraccion.jpg'
# cv2.imwrite(filename, imgResult)

f, (ax1) = plt.subplots(1, 1,figsize=(6, 6))
ax1.hist(imgResult.ravel(),256,[0,256])
ax1.set_title("Histograma")
ax1.set_xlabel('Intensidad')
ax1.set_ylabel('Cantidad de pixeles')
```

```
52
53  threshold = 90
54  imgThres = getThresholding(imgResult,threshold)
55  cv2.imshow("Resultado Thresolding", imgThres)
56
57  # filename = 'Resultado_Thresolding.jpg'
58  # cv2.imwrite(filename, imgThres)
59
60  # # Filename
61  # plt.savefig('HistogramaThresoldingSus.png')
62
63  plt.show()
64  cv2.waitKey(0)
65  cv2.destroyAllWindows()
```

1.4. Implemente la sustracción de imágenes para detectar el cambio o movimiento de objetos en fotogramas. En la Figura 5, se brindan dos fotogramas consecutivos, implemente la sustracción para obtener una imagen donde se visualiza qué objetos han cambiado de posición.

- **Adiciones de Imágenes**

  - **Imagen Original**:



(a) Imagen 1 (b) Imagen 2

Figura 5: Adición de Imágenes

  - **Imagen Resultado** :

- **Código**:

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

imgReal1 = cv2.imread("tools_1.jpg",  cv2.IMREAD_GRAYSCALE) #8 bit
    por escala de grises
imgReal1 = cv2.resize(imgReal1,(400,400))
img1 = imgReal1

imgReal2 = cv2.imread("tools_2.jpg",  cv2.IMREAD_GRAYSCALE) #8 bit
    por escala de grises
imgReal2 = cv2.resize(imgReal2,(400,400))
img2 = imgReal2

cv2.imshow("Image Original 1", imgReal1)
cv2.imshow("Image Original 2", imgReal2)

img1 = img1.astype(int)
img2 = img2.astype(int)
imgResult = img1.copy()
# print(imgResult.dtype)
print(imgResult)

add = 100
for i in range(len(img1)):
    for j in range(len(img1[0])):
        rest = abs(img1[i,j] - img2[i,j])
        imgResult[i,j] = rest
imgResult = imgResult.astype(np.uint8)
cv2.imshow("Resultado Sustraccion tools", imgResult)

```

```
filename = 'Resultado_Sustraccion_tools.jpg'
cv2.imwrite(filename, imgResult)

arrayImg = imgResult.ravel()
sortArray = np.sort(arrayImg)

# 0% y el 100 %
min_ = int(len(sortArray) * 0.0)
max_ = int(len(sortArray) * 1.0 - 1)
c = sortArray[min_]
d = sortArray[max_]

plt.show()
cv2.waitKey(0)
cv2.destroyAllWindows()
```

2. Algoritmo de computación gráfica Image Arithmetic (Multiplicación y División de imágenes)

2.1. Implemente la multiplicación de imagen por una constante con la Figura 1. Evalue con c = 2, c = 5 y c = 7.

- **Multiplicación de Imágenes**

  - **Imagen Original**:

  - **Imagen Resultado** :

(a) Imagen 1 con constante c=2 (b) Imagen 2 con constante c=5 (c) Imagen 3 con constante c=7

Figura 6: Multiplicación de Imágenes

- **Código**:

```
import cv2
import numpy as np

imgReal1 = cv2.imread("tigre.jpg", 1) #8 bit por escala de grises
imgReal1 = cv2.resize(imgReal1,(400,400))
img1 = imgReal1

cv2.imshow("Image Original 1", imgReal1)

img1 = img1.astype(int)
constantes = [2,5,7]

B, G, R = cv2.split(img1)
Bresult,Gresult,Rresult = B.copy(), G.copy(), R.copy()
for c in constantes:
    for i in range(len(img1)):
        for j in range(len(img1[0])):

            sum_B = B[i,j]*c
            if sum_B > 255 :
                Bresult[i,j] = 255
            else:
                Bresult[i,j] = sum_B

            sum_G = G[i,j]*c
            if sum_G > 255 :
                Gresult[i,j] = 255
            else:
                Gresult[i,j] = sum_G

            sum_R = R[i,j]*c
            if sum_R > 255 :
                Rresult[i,j] = 255
            else:
                Rresult[i,j] = sum_R

```

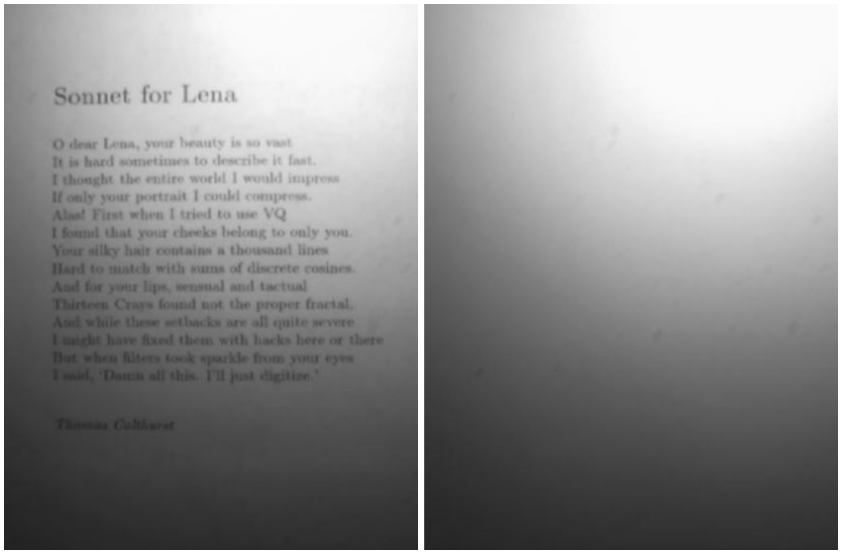
```
37      img_result = cv2.merge((Bresult,Gresult,Rresult))
38      img_result = np.array(img_result, dtype=np.uint8)
39      title_ = "Resultado con constante c = "+ str(c)
40      cv2.imshow(title_, img_result)
41      # filename = "Resultado_C = "+str(c)+".jpg"
42      # cv2.imwrite(filename, img_result)
43
44  cv2.waitKey(0)
45  cv2.destroyAllWindows()
```

2.2. Implemente la división de imágenes para segmentar letras. Se le está brindando una foto del documento y otra de una hoja en blanco para eliminar el reflejo de la luz (Figura 2). Después de la división deber a normalizar la imágen a valores entre [0 − 255] (Ecuación 1). Después aplique thresholding para obtener un resultado similar a la Figura 3. Tiene la libertad de aplicar métodos adicionales para mejorar los resultados, por ejemplo: contrast stretching, histogram equalization, etc.

Después de la división se procedió a normalizar la imagen a valores entre [0 − 255] con la siguiente ecuación

- **División de Imagénes**

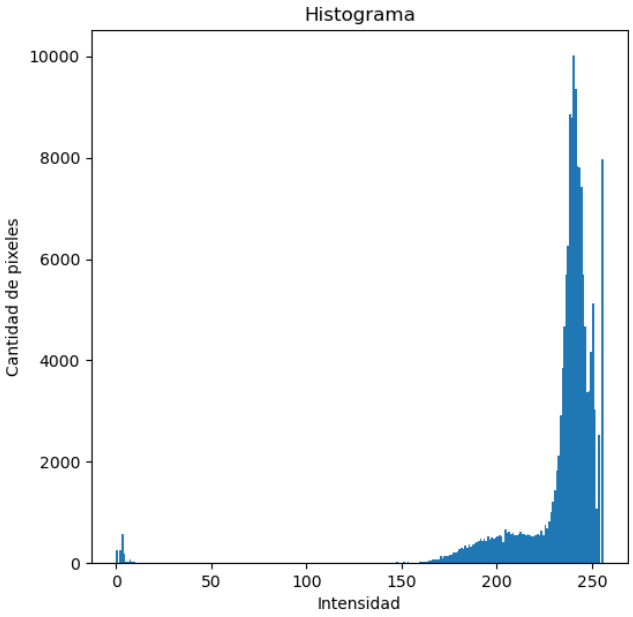
  - **Imagen Original**:



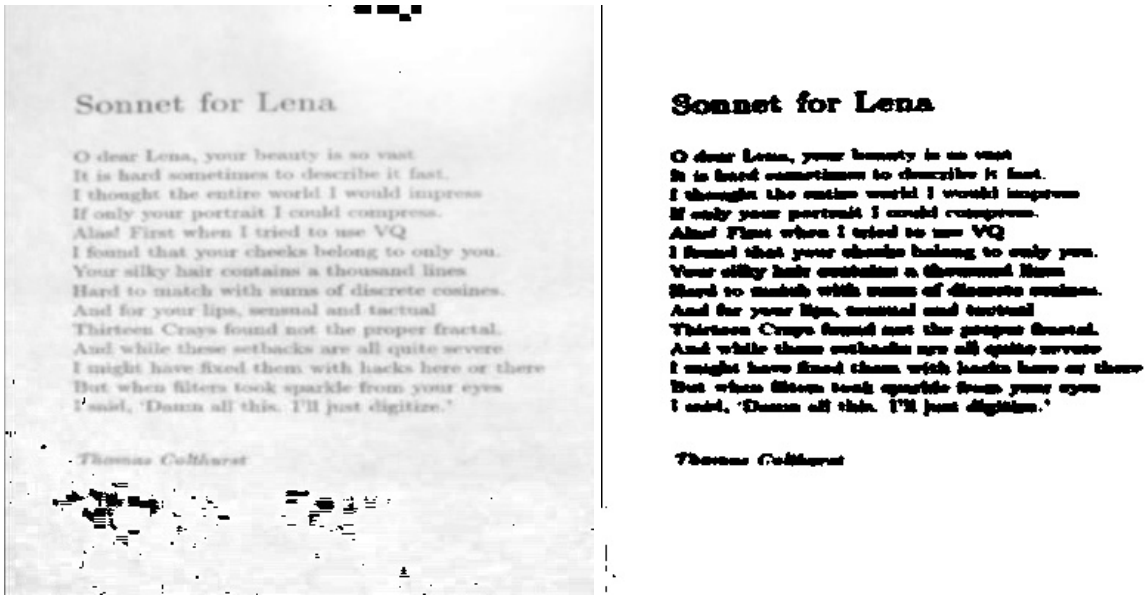
(a) Imagen 1 (b) Imagen 2

Figura 7: División de Imagénes

- **Histograma para aplicar un Thresold**:



- **Imagen Resultado** :



(a) Resultado de la división de las imágenes normalizado (b) Resultado final después de aplicar Thresolding

Figura 8: Resultados Finales

- **Codigo**:

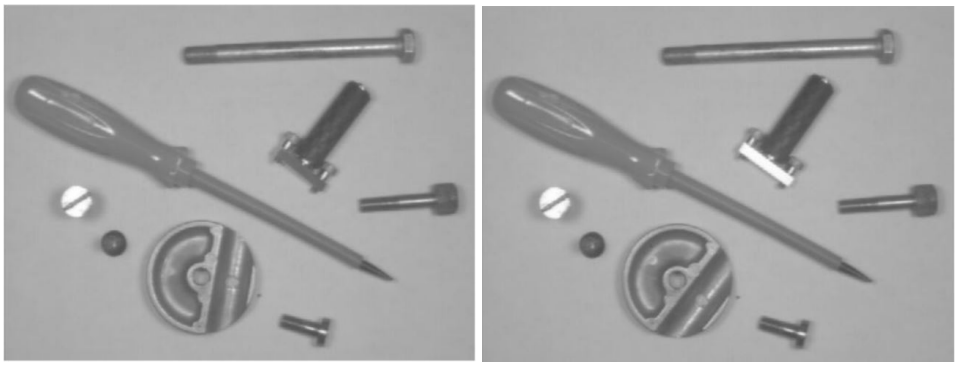
```python
import cv2
import numpy as np
from matplotlib import pyplot as plt
import math

def getThresholding(img,threshold):
    modifiedImg = img.copy()
    # print(img)
    for i in range(len(img)):
        for j in range(len(img[i])):
            if(img[i][j] > 135 and img[i][j] < 222):
                modifiedImg[i][j] = np.uint8(0)
            else:
                modifiedImg[i][j] = np.uint8(255)
    return modifiedImg

imgReal1 = cv2.imread("text1.jpg", cv2.IMREAD_GRAYSCALE) #8 bit por escala de grises
imgReal1 = cv2.resize(imgReal1,(400,400))
img1 = imgReal1

imgReal2 = cv2.imread("text2.jpg", cv2.IMREAD_GRAYSCALE) #8 bit por escala de grises
imgReal2 = cv2.resize(imgReal2,(400,400))
img2 = imgReal2

cv2.imshow("Image Original 1", imgReal1)
cv2.imshow("Image Original 2", imgReal2)

img1 = img1.astype(int)
img2 = img2.astype(int)
imgResult = img1.copy()

for i in range(len(img1)):
    for j in range(len(img1[0])):
        result = img1[i,j]/img2[i,j]
        scala = (result-0)*(255-0)//(1-0) + 0
        imgResult[i,j] = scala
imgResult = imgResult.astype(np.uint8)
cv2.imshow("Resultado Division text", imgResult)

# filename = "Resultado_Division_text.jpg"
# cv2.imwrite(filename, imgResult)


f, (ax1) = plt.subplots(1, 1,figsize=(6, 6))
ax1.hist(imgResult.ravel(),256,[0,256])
ax1.set_title("Histograma")
ax1.set_xlabel('Intensidad')
ax1.set_ylabel('Cantidad de pixeles')

threshold = 120
imgThres = getThresholding(imgResult,threshold)
cv2.imshow("Resultado Thresolding", imgThres)

```

```
54  # filename = "Resultado_Thresolding_text.jpg"
55  # cv2.imwrite(filename, imgThres)
56
57
58  # # Filename
59  # plt.savefig('HistogramaThresoldingtext.png')
60
61  plt.show()
62  cv2.waitKey(0)
63  cv2.destroyAllWindows()
```

2.3. Implemente la división de imágenes para detectar el cambio o movimiento de objetos en fotogramas. En la Figura 4, se brindan dos fotogramas consecutivos, implemente la división para obtener una imagen donde se visualiza que objetos se movieron, quizás sea necesario multiplicar el resultado por una constante para poder visualizar mejor los resultados. Después puede aplicar contrast stretching para mejorar a ´un mas los resultados .

- **División de Imágenes**

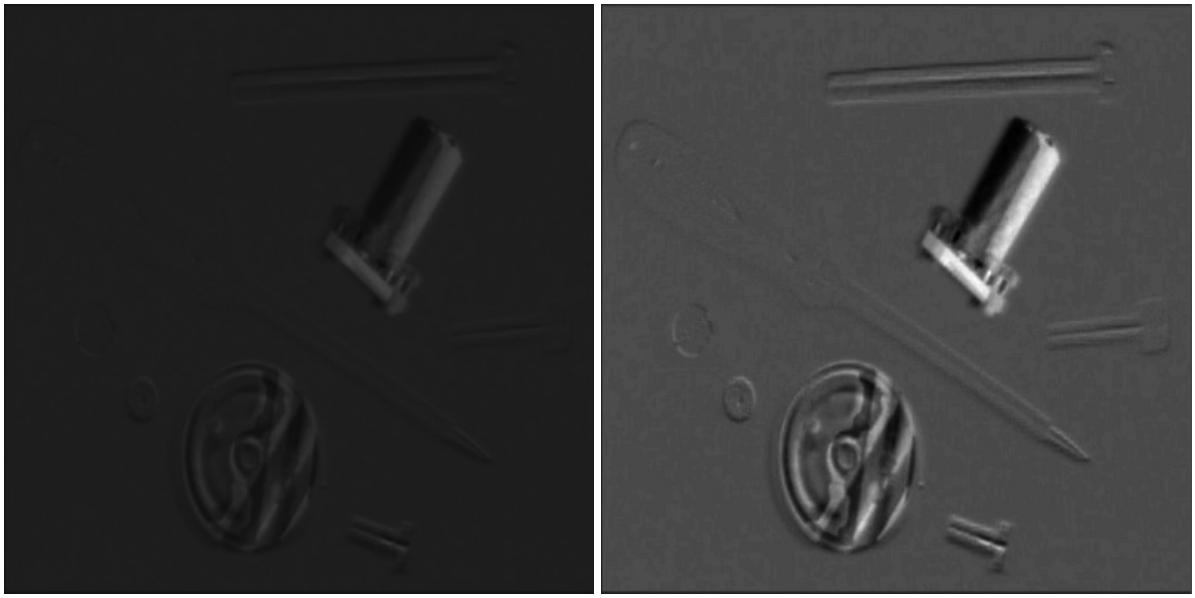
  - **Imagen Original**:



(a) Imagen 1 (b) Imagen 2

Figura 9: División de Imágenes

  - **Imagen Resultado** :



(a) Resultado de la división de las imágenes multiplicado por 30

(b) Resultado final después de aplicar Contrast Stretching

Figura 10: Resultados Finales

- **Código**:

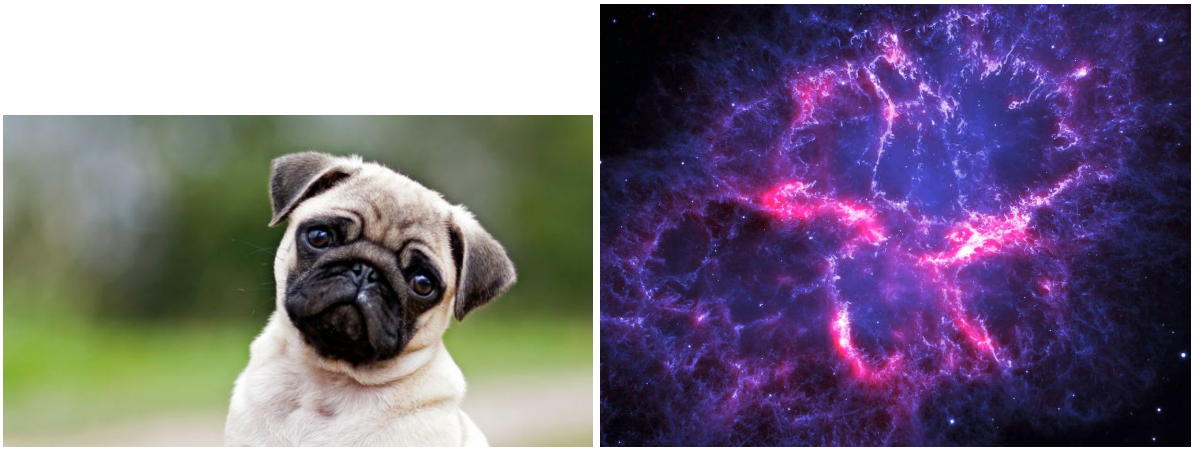
```
import cv2
import numpy as np
import math

def PixelOperations(img,c,d):
    a = 0
    b = 255
    for row in range(len(img)):
        for pixel in range(len(img[row])):
            if(0<=img[row][pixel]  and img[row][pixel]  <= c):
                img[row][pixel] = (a/c)*img[row][pixel]
            elif(c<img[row][pixel] and img[row][pixel] <= d):
                img[row][pixel] = ((img[row][pixel] - c) * ((b-a)/(d-
                    c)))+a
            else:
                 img[row][pixel] = ((img[row][pixel] - d) * ((255-b)
                     /(255-d)))+b
    img = np.array(img, dtype=np.uint8)
    return img

imgReal1 = cv2.imread("tools1.jpg", cv2.IMREAD_GRAYSCALE) #8 bit por
    escala de grises
imgReal1 = cv2.resize(imgReal1,(400,400))
img1 = imgReal1

imgReal2 = cv2.imread("tools2.jpg", cv2.IMREAD_GRAYSCALE) #8 bit por
    escala de grises
```

```
imgReal2 = cv2.resize(imgReal2,(400,400))
img2 = imgReal2

cv2.imshow("Image Original 1", imgReal1)
cv2.imshow("Image Original 2", imgReal2)

img1 = img1.astype(int)
img2 = img2.astype(int)
imgResult = img1.copy()

for i in range(len(img1)):
    for j in range(len(img1[0])):
        result = (img2[i,j]/img1[i,j])*30
        # scala = (result-0)*(255-0)/(1-0) + 0
        imgResult[i,j] = result
imgResult = imgResult.astype(np.uint8)
cv2.imshow("Resultado", imgResult)

filename = "Resultado_Division_Tools.jpg"
cv2.imwrite(filename, imgResult)

arrayImg = imgResult.ravel()
sortArray = np.sort(arrayImg)
# print(sortArray)
# 0% y el 100 %
min_ = int(len(sortArray) * 0.0)
max_ = int(len(sortArray) * 1.0 -1)
c = sortArray[min_]
d = sortArray[max_]
img_Constrast = PixelOperations(imgResult.astype(int),int(c),int(d))
cv2.imshow("Resultado constrast", img_Constrast)

filename = "Resultado_constrast_Tools.jpg"
cv2.imwrite(filename, img_Constrast)

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

2.4. Implemente el operador Blending (ecuación 2) y evalúe sus resultados con imágenes de su preferencia, también pruebe diferentes valores de X.

- **Blending de Imágenes**

  - **Imagen Original**:



(a) Imagen 1 (b) Imagen 2

Figura 11: Blending

- **Imagen Resultado** :

(a) Imagen 1 con constante X=0.25 (b) Imagen 2 con constante X=0.5 (c) Imagen 3 con constante X=0.75

Figura 12: Resultado despues de aplicar Blending con diferentes valores X

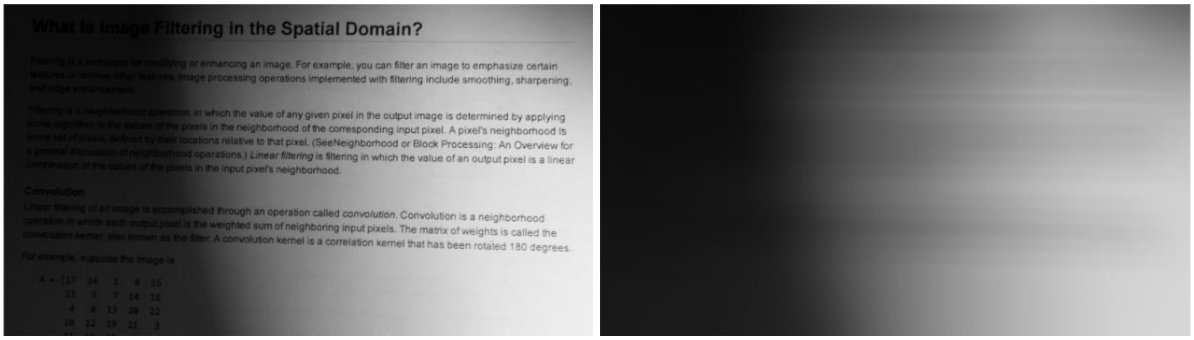
- **Código**:

```
import cv2
import numpy as np

imgReal1 = cv2.imread("pug.jpg",1) #8 bit por escala de grises
imgReal1 = cv2.resize(imgReal1,(400,400))
img1 = imgReal1

imgReal2 = cv2.imread("magico.jpg",1) #8 bit por escala de grises
imgReal2 = cv2.resize(imgReal2,(400,400))
img2 = imgReal2

cv2.imshow("Image Original 1", imgReal1)
cv2.imshow("Image Original 2", imgReal2)

B1, G1, R1 = cv2.split(img1)
B2, G2, R2 = cv2.split(img2)
B1, G1, R1 = B1.astype(int), G1.astype(int), R1.astype(int)
B2, G2, R2 = B2.astype(int), G2.astype(int), R2.astype(int)
Bresult,Gresult,Rresult = B1.copy(), G1.copy(), R1.copy()

X = [0.25,0.5,0.75]
for x in X:
    for i in range(len(B1)):
        for j in range(len(B1[0])):

            Bresult[i,j] =  x*B1[i, j] + (1-x)*B2[i, j]
            Gresult[i,j] =  x*G1[i, j] + (1-x)*G2[i, j]
            Rresult[i,j] =  x*R1[i, j] + (1-x)*R2[i, j]

    img_result = cv2.merge((Bresult,Gresult,Rresult))
    img_result = np.array(img_result, dtype=np.uint8)
    title_ = "Resultado_con_constante_C_=_"+str(x)
    cv2.imshow(title_, img_result)

    # filename = "Resultado_con_constante_C_=_"+str(x)+".jpg"
    # cv2.imwrite(filename, img_result)

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

3. Algoritmo de computación gráfica Image Aritmética.(Review)

- **Imágenes Originales**

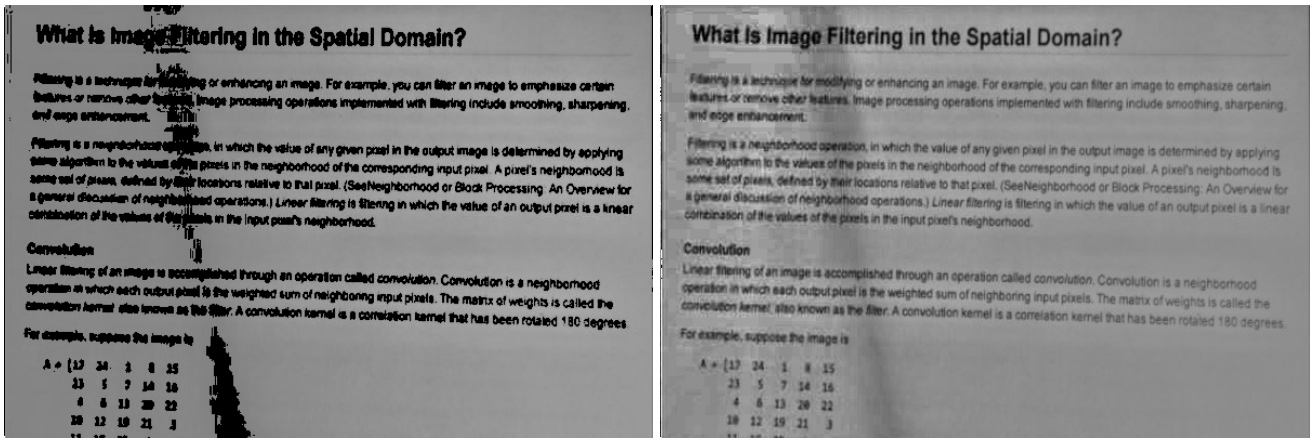
  - **Imagen Original**:



(a) Imagen 1 (b) Imagen 2

Figura 13: Blending

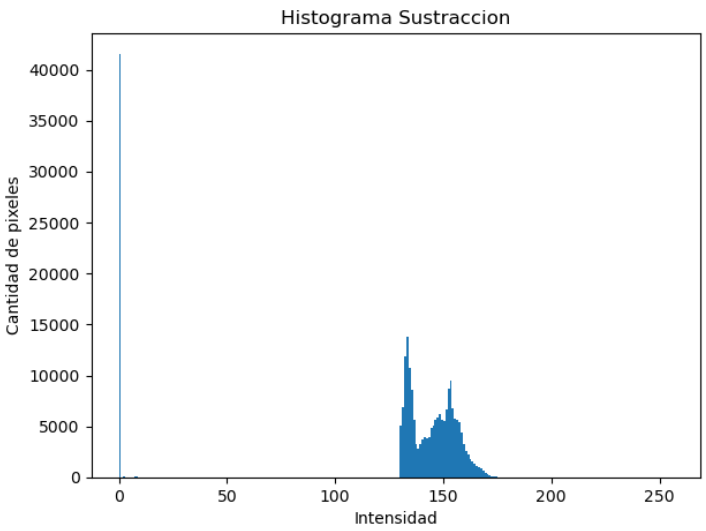
- **Resultado después de aplicar Sustracción y División de imágenes** :

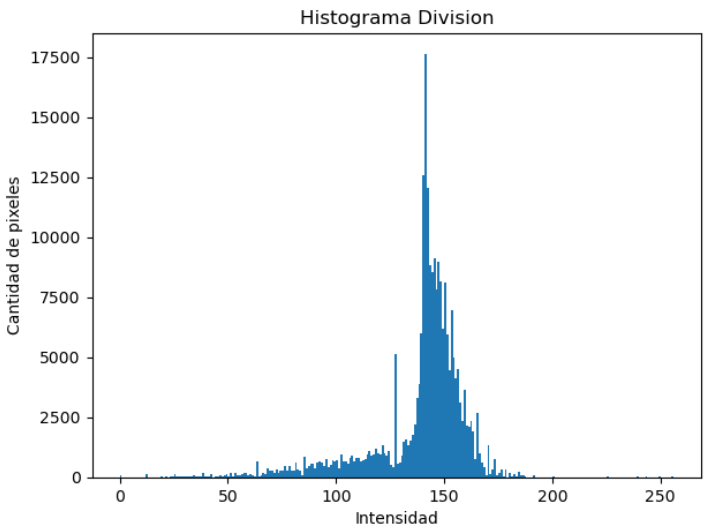


(a) Resultado de la sustracción (b) Resultado de la división

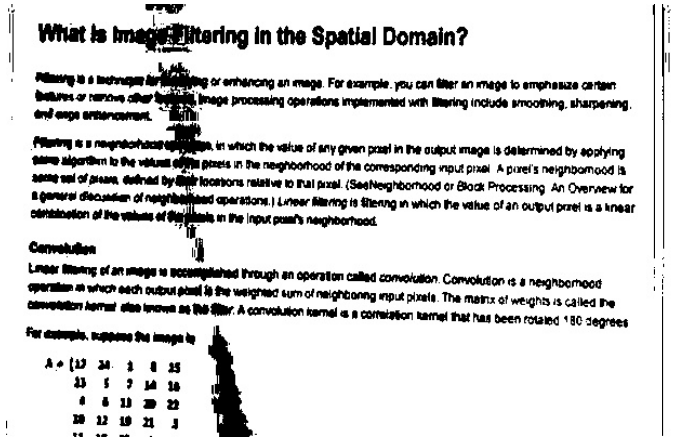
Figura 14: Resultado después de aplicar Sustracción y División

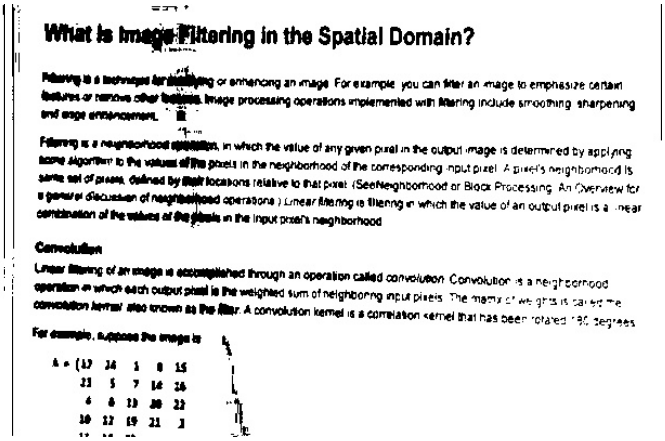
- **Histogramas** :
  Para la sustracción de un Thresold de 1 y para la división se uso un Thresold de 130 pero hubo una condición más para tratar con el problema de la mancha que se ven en ambas imagenes.La condicion tenia un rango de 116 a 129 donde se trato de desaparecer dicha mancha pero tambien se borro algunas palabras por lo que no se pudo borrar del todo. Se presentó una opción donde se trata recuperar las palabras y que la mancha quede un poco gris para no perjudicar palabras.





- **Resultados** :





(a) Resultado de la sustracción después de aplicar Thresolding (b) Resultado de la división después de aplicar Thresolding

Figura 15: Resultado después de aplicar Sustracción y División con un Thresold

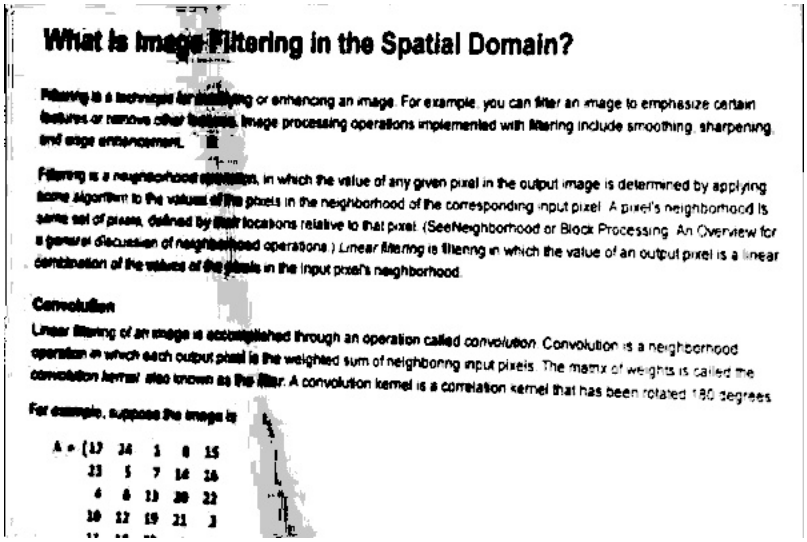


Figura 16: Resultado después de aplicar Sustracción y División con un Thresold

- **Codigo**:

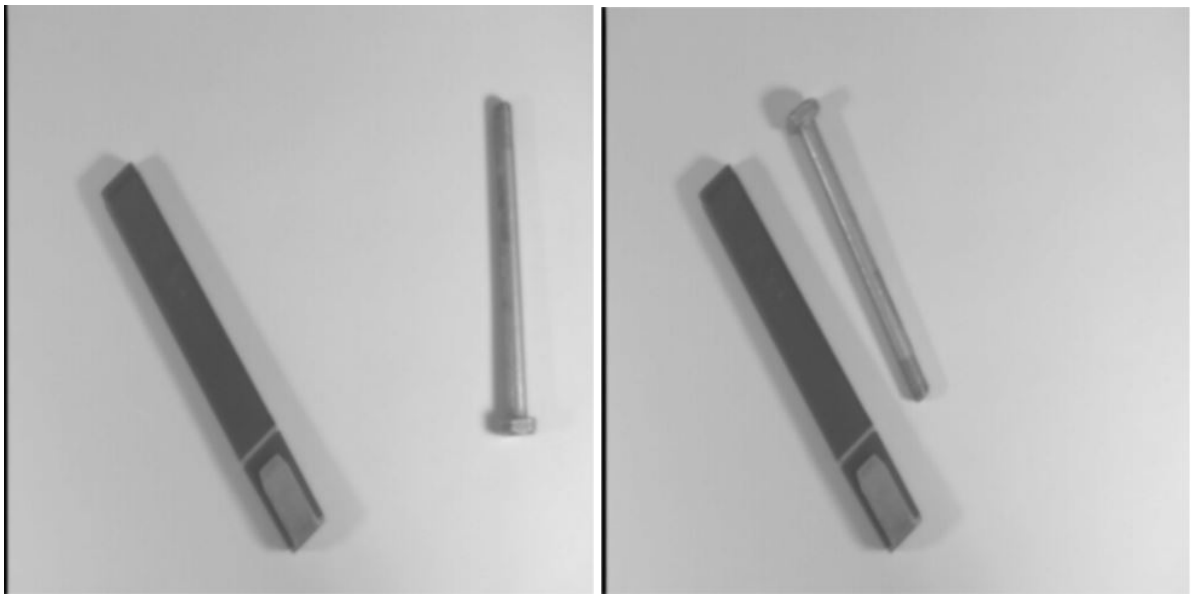
```
import cv2
import numpy as np
from matplotlib import pyplot as plt


def getThresholding(img,threshold):
    modifiedImg = img.copy()
    # print(img)
    for i in range(len(img)):
        for j in range(len(img[i])):
            if(img[i][j] < threshold):
                modifiedImg[i][j] = np.uint8(0)
            else:
                modifiedImg[i][j] = np.uint8(255)
    return modifiedImg

def getThresholding2(img):
    modifiedImg = img.copy()
    # print(img)
    for i in range(len(img)):
        for j in range(len(img[i])):
            if(img[i][j] >= 116 and img[i][j] <= 129):
                modifiedImg[i][j] = np.uint8(255) #200
            elif(img[i][j] < 130):
                modifiedImg[i][j] = np.uint8(0)
            else:
                modifiedImg[i][j] = np.uint8(255)
    return modifiedImg

imgReal1 = cv2.imread("textReview1.jpg", cv2.IMREAD_GRAYSCALE) #8 bit
    por escala de grises
imgReal1 = cv2.resize(imgReal1,(600,400))
```

```
img1 = imgReal1.copy()

imgReal2 = cv2.imread("textReview2.jpg", cv2.IMREAD_GRAYSCALE) #8 bit por escala de grises
imgReal2 = cv2.resize(imgReal2,(600,400))
img2 = imgReal2.copy()

cv2.imshow("Image Original 1", imgReal1)
cv2.imshow("Image Original 2", imgReal2)

img1 = img1.astype(int)
img2 = img2.astype(int)
imgResultSust = img1.copy()
imgResultDiv = img1.copy()

# ====================
# SUSTRACION
# ====================
c = 130
for i in range(len(img1)):
    for j in range(len(img1[0])):
        rest = img1[i,j] - img2[i,j]
        if(rest <0):
            imgResultSust[i,j] = 0
        else:
            scala = rest + c
            if(scala >255):
                imgResultSust[i,j] = 0
            else:
                imgResultSust[i,j] = rest + c

imgResultSust = imgResultSust.astype(np.uint8)
cv2.imshow("Resultado Sustraccion", imgResultSust)
# filename = 'ReviewSustraccion.jpg'
# cv2.imwrite(filename, imgResultSust)

f, (ax1,ax2) = plt.subplots(1, 2,figsize=(15, 5))
ax1.hist(imgResultSust.ravel(),256,[0,256])
ax1.set_title("Histograma Sustraccion")
ax1.set_xlabel('Intensidad')
ax1.set_ylabel('Cantidad de pixeles')

threshold = 1
imgSustThres = getThresholding(imgResultSust,threshold)
cv2.imshow("Thresolding Sustraccion", imgSustThres)
# filename = 'ReviewThresoldingSust.jpg'
# cv2.imwrite(filename, imgSustThres)

# ====================
# DIVISION
# ====================

for i in range(len(img1)):
    for j in range(len(img1[0])):
        result = (img1[i,j]/img2[i,j])
        # print(result)
```

```
            scala = (result-0)*(255-0)/(2-0) + 0
            imgResultDiv[i,j] = int(scala)
    imgResultDiv = imgResultDiv.astype(np.uint8)
    cv2.imshow("Resultado Division", imgResultDiv)
    # filename = 'ReviewDivision.jpg'
    # cv2.imwrite(filename, imgResultDiv)

    ax2.hist(imgResultDiv.ravel(),256,[0,256])
    ax2.set_title("Histograma Division")
    ax2.set_xlabel('Intensidad')
    ax2.set_ylabel('Cantidad de pixeles')

    imgDivThres = getThresholding2(imgResultDiv)
    cv2.imshow("Thresolding Division", imgDivThres)
    # filename = 'ReviewThresoldingDiv.jpg'
    # cv2.imwrite(filename, imgDivThres)

    # Filename
    plt.savefig('HistogramaRview.png')
    plt.show()
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

4. Algoritmo de computación gráfica Image Logical.

- Imágenes de Pruebas

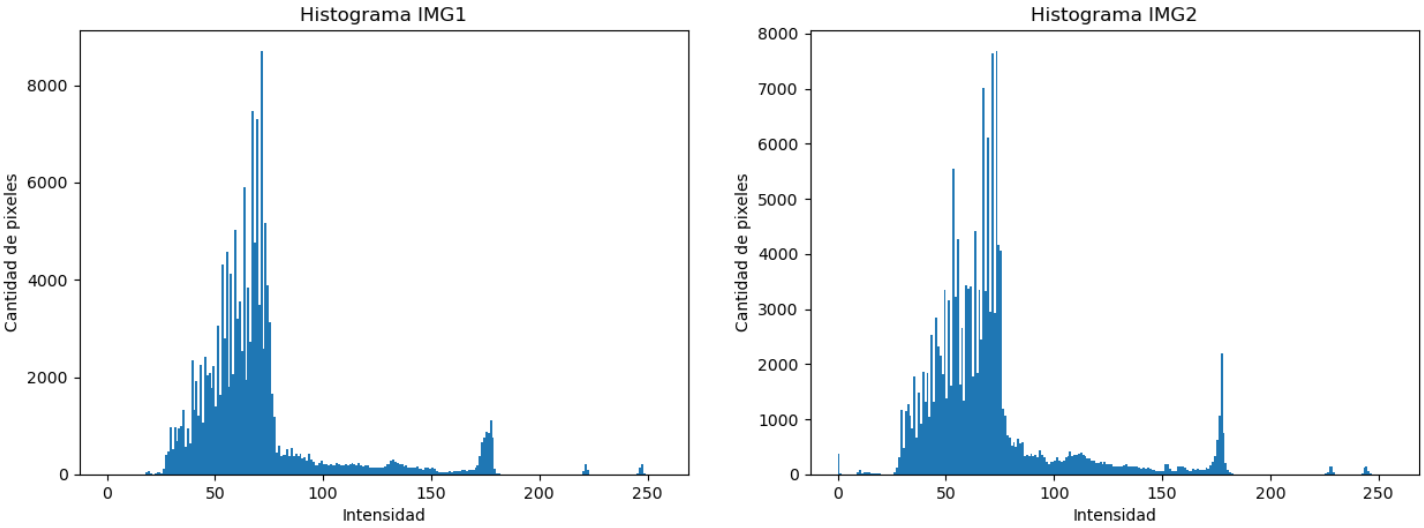


(a) Imagen 1 (b) Imagen 2

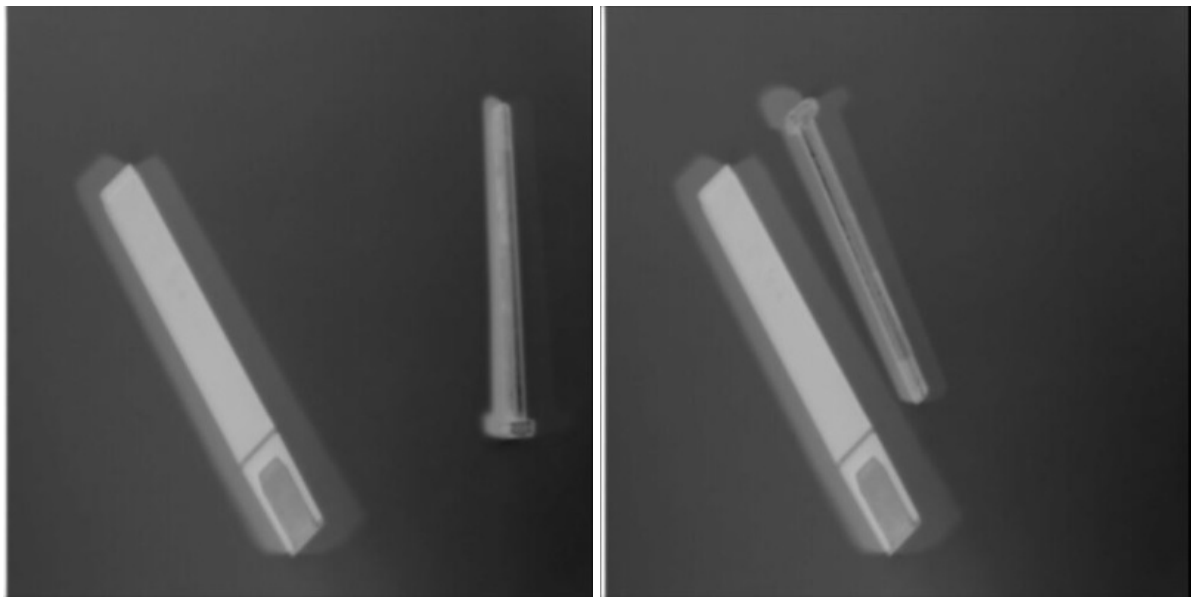
Figura 17: Blending

4.1. Implemente el operador AND con las imágenes de la Figura 1 para segmentar el objeto que aparece en ambas imágenes (intersección).

- Histograma para analizar y definir un Thresold
  Se usó un thresold de 98



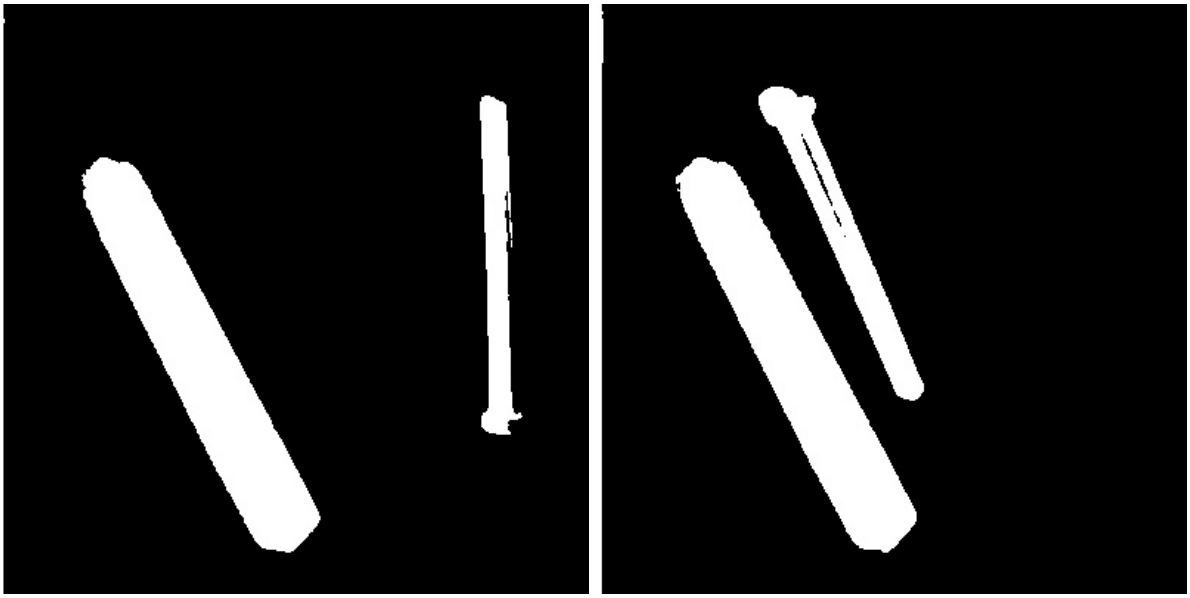
- **Imagenes despues despues de invertir los colores**



(a) Imagen 1 (b) Imagen 2

Figura 18: Inversión de colores en las imágenes

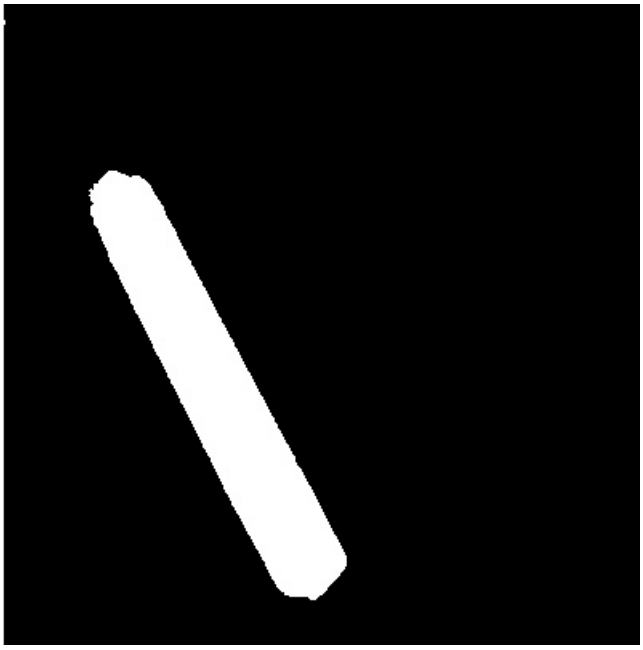
- **Imágenes después de aplicar Thresolding**
- **Resultados después de aplicar el operador AND**



(a) Imagen 1 (b) Imagen 2

Figura 19: Thresolding

Al aplicar el operador AND a cada pixel de las imágenes en su forma binaria se puede llegar a un resultado en la cual se muestra en la figura. La figura resultante muestra que las dos imágenes tienen un objeto en común.



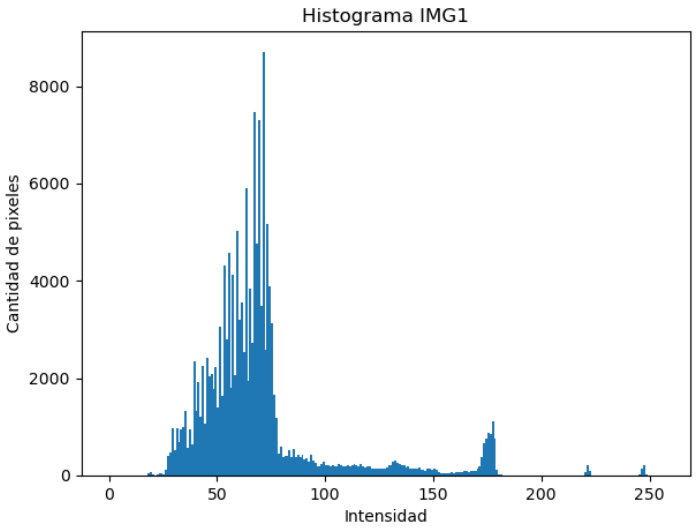
- **Código**:

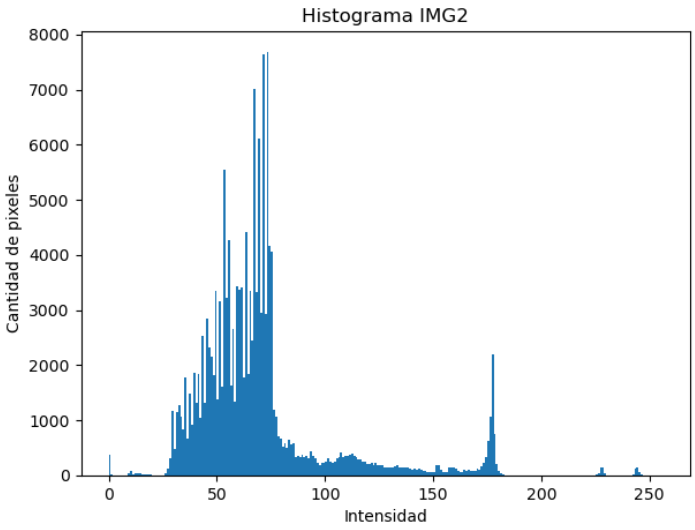
```
import cv2
import numpy as np
from matplotlib import pyplot as plt


def getThresholding(img,threshold):
    modifiedImg = img.copy()
    # print(img)
    for i in range(len(img)):
        for j in range(len(img[i])):
            if(img[i][j] < threshold):
                modifiedImg[i][j] = np.uint8(0)
            else:
                modifiedImg[i][j] = np.uint8(255)
    return modifiedImg

imgReal1 = cv2.imread("pernos1.jpg", cv2.IMREAD_GRAYSCALE) #8 bit por
    escala de grises
imgReal1 = cv2.resize(imgReal1,(400,400))
img1 = imgReal1.copy()

imgReal2 = cv2.imread("pernos2.jpg", cv2.IMREAD_GRAYSCALE) #8 bit por
    escala de grises
imgReal2 = cv2.resize(imgReal2,(400,400))
img2 = imgReal2.copy()

img1 = 255-img1
img2 = 255-img2

cv2.imshow("Image Original 1", img1)
cv2.imshow("Image Original 2", img2)

f, (ax1,ax2) = plt.subplots(1, 2,figsize=(15, 5))
ax1.hist(img1.ravel(),256,[0,256])
ax1.set_title("Histograma IMG1")
ax1.set_xlabel('Intensidad')
ax1.set_ylabel('Cantidad de pixeles')
ax2.hist(img2.ravel(),256,[0,256])
ax2.set_title("Histograma IMG2")
ax2.set_xlabel('Intensidad')
ax2.set_ylabel('Cantidad de pixeles')


threshold = 98
imgThres1 = getThresholding(img1,threshold)
cv2.imshow("Resultado Thresolding de la IMG1", imgThres1)
# filename = 'ResultadoThresoldingPernos1.jpg'
# cv2.imwrite(filename, imgThres1)

imgThres2 = getThresholding(img2,threshold)
cv2.imshow("Resultado Thresolding de la IMG2", imgThres2)
# filename = 'ResultadoThresoldingPernos2.jpg'
# cv2.imwrite(filename, imgThres2)
```

```
52
53  # ============================
54  # OPERADOR AND
55  # ============================
56  imgResult = imgThres1.copy()
57
58  for i in range(len(imgThres1)):
59      for j in range(len(imgThres1[0])):
60          result = imgThres1[i,j] & imgThres2[i,j]
61          imgResult[i,j] = result
62  cv2.imshow("Resultado Operador AND", imgResult)
63
64  # filename = 'ResultadoOperadorAND.jpg'
65  # cv2.imwrite(filename, imgResult)
66
67  # Filename
68  # plt.savefig('HistogramaThresoldPernos.png')
69
70  plt.show()
71  cv2.waitKey(0)
72  cv2.destroyAllWindows()
```

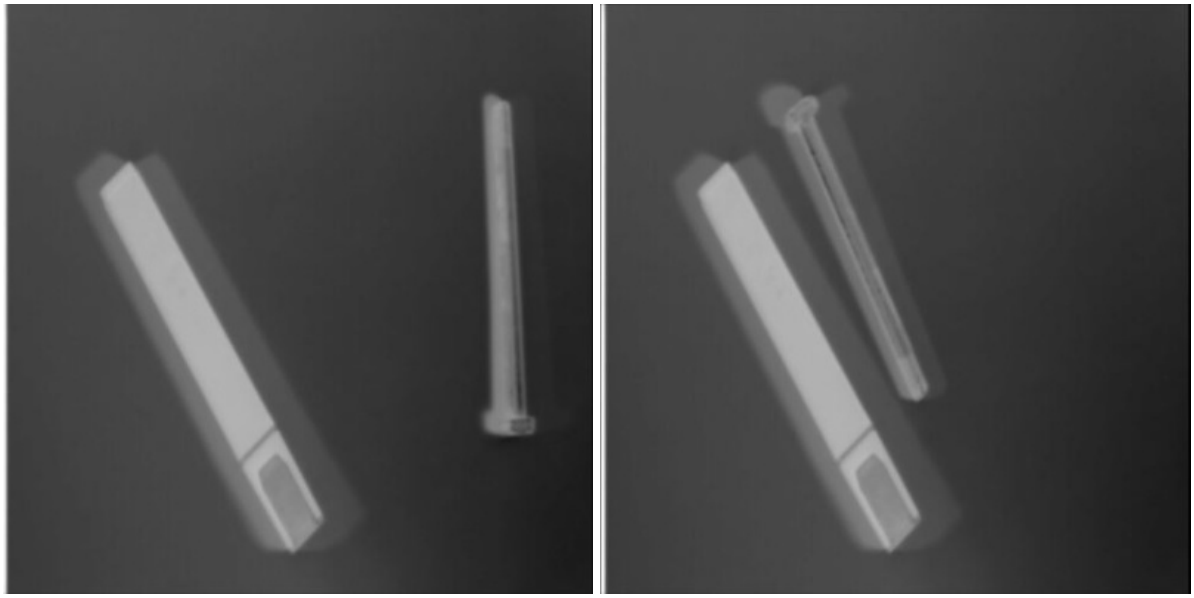
4.2. Implemente el operador OR con las imágenes de la Figura 1 para unir los objetos que aparecen en las imágenes (fusión).

- **Histograma para analizar y definir un Thresold**
  Se uso un thresold de 98





- **Imagenes despues despues de invertir los colores**



(a) Imagen 1 (b) Imagen 2

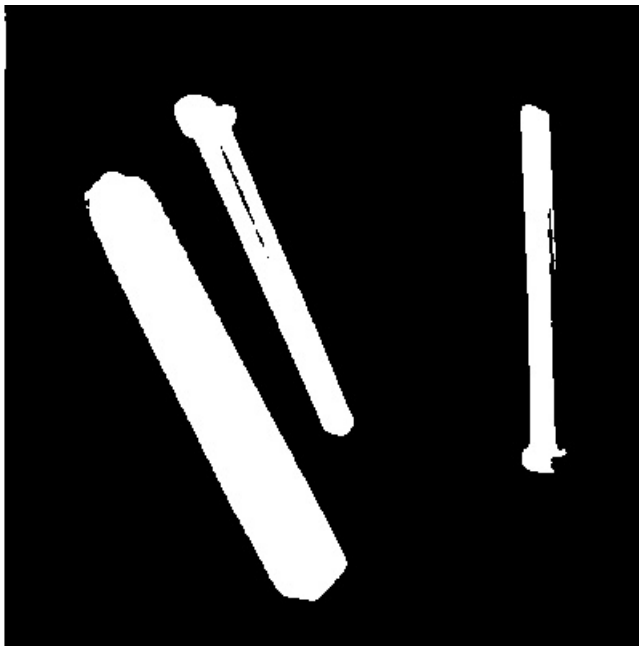
Figura 20: Invercion de colores en las imagenes

- **Imagenes despues de aplicar Thresolding**

(a) Imagen 1 (b) Imagen 2

Figura 21: Thresolding

- Resultados despues de aplicar el operador OR



Al aplicar el operador OR a cada pixel de las imagenes en su forma binaria se puede llegar a un resultado en la cual se muestra en la figura. La figura resultante muestra que las dos imagene pueden llegar a unirse.

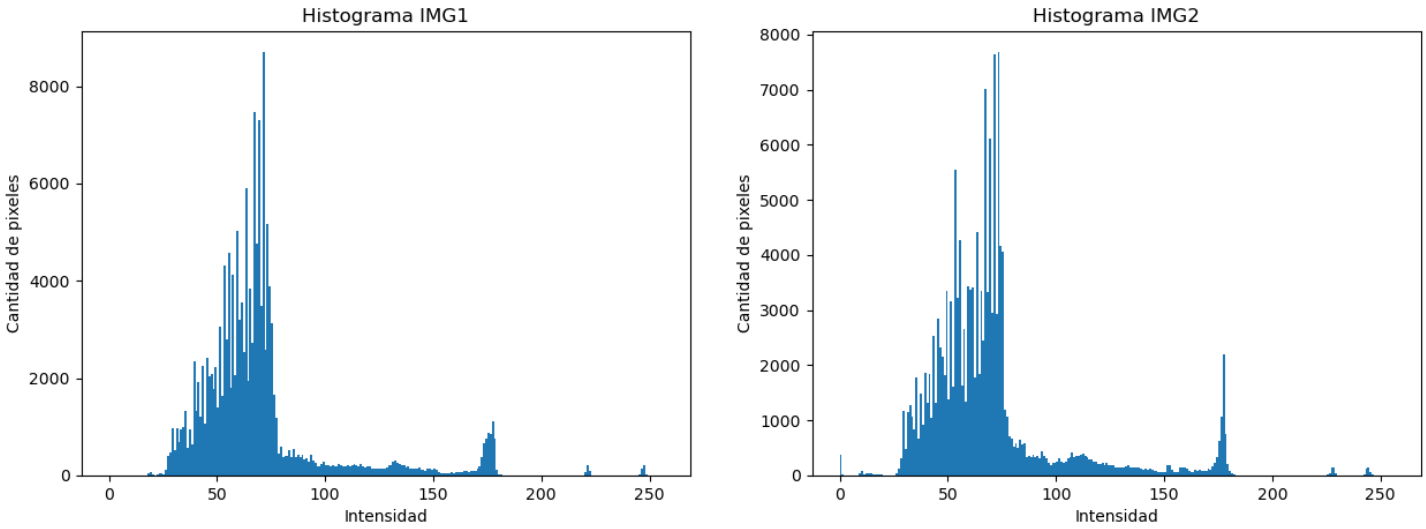
- **Codigo**:

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


def getThresholding(img,threshold):
    modifiedImg = img.copy()
    # print(img)
    for i in range(len(img)):
        for j in range(len(img[i])):
            if(img[i][j] < threshold):
                modifiedImg[i][j] = np.uint8(0)
            else:
                modifiedImg[i][j] = np.uint8(255)
    return modifiedImg

imgReal1 = cv2.imread("pernos1.jpg", cv2.IMREAD_GRAYSCALE) #8 bit por
    escala de grises
imgReal1 = cv2.resize(imgReal1,(400,400))
img1 = imgReal1.copy()

imgReal2 = cv2.imread("pernos2.jpg", cv2.IMREAD_GRAYSCALE) #8 bit por
    escala de grises
imgReal2 = cv2.resize(imgReal2,(400,400))
```

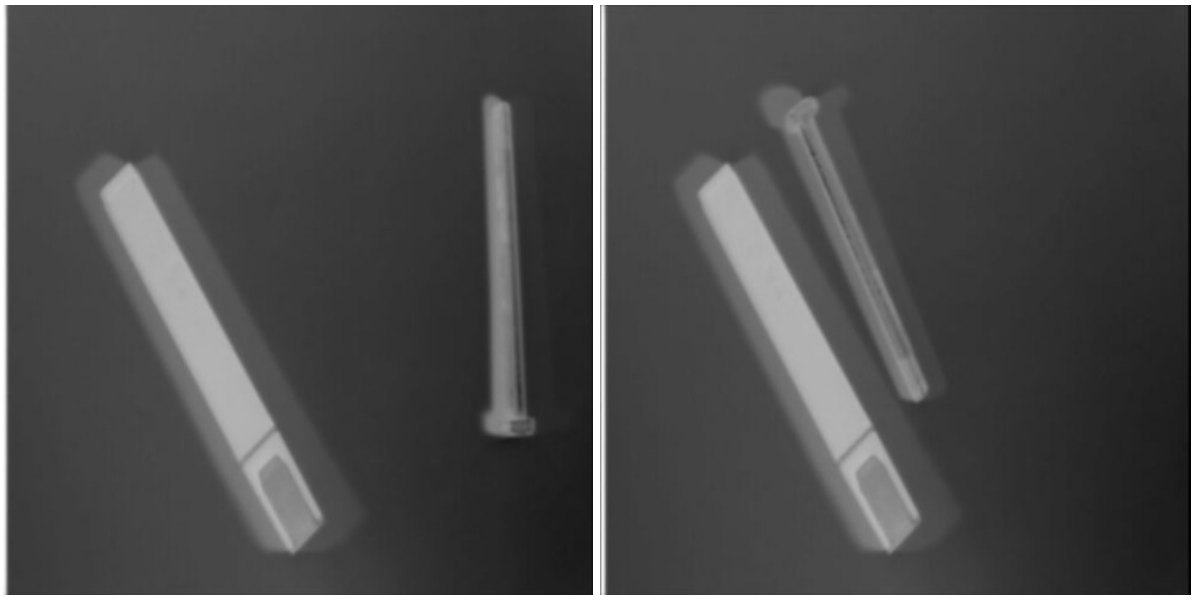
```
img2 = imgReal2.copy()

img1 = 255-img1
img2 = 255-img2


cv2.imshow("Image Original 1", img1)
cv2.imshow("Image Original 2", img2)

f, (ax1,ax2) = plt.subplots(1, 2,figsize=(15, 5))
ax1.hist(img1.ravel(),256,[0,256])
ax1.set_title("Histograma IMG1")
ax1.set_xlabel('Intensidad')
ax1.set_ylabel('Cantidad de pixeles')
ax2.hist(img2.ravel(),256,[0,256])
ax2.set_title("Histograma IMG2")
ax2.set_xlabel('Intensidad')
ax2.set_ylabel('Cantidad de pixeles')


threshold = 98
imgThres1 = getThresholding(img1,threshold)
cv2.imshow("Resultado Thresolding de la IMG1", imgThres1)

imgThres2 = getThresholding(img2,threshold)
cv2.imshow("Resultado Thresolding de la IMG2", imgThres2)


# ======================
# OPERADOR OR
# ======================

imgResult = imgThres1.copy()

for i in range(len(imgThres1)):
    for j in range(len(imgThres1[0])):
        result = imgThres1[i,j] | imgThres2[i,j]
        imgResult[i,j] = result
cv2.imshow("Resultado Operador OR", imgResult)
filename = 'ResultadoOperadorOR.jpg'
cv2.imwrite(filename, imgResult)

plt.show()
cv2.waitKey(0)
cv2.destroyAllWindows()
```

4.3. Implemente el operador XOR con las imágenes de la Figura 1 para detectar los cambios.

- Histograma para analizar y definir un Thresold
  Se uso un thresold de 98



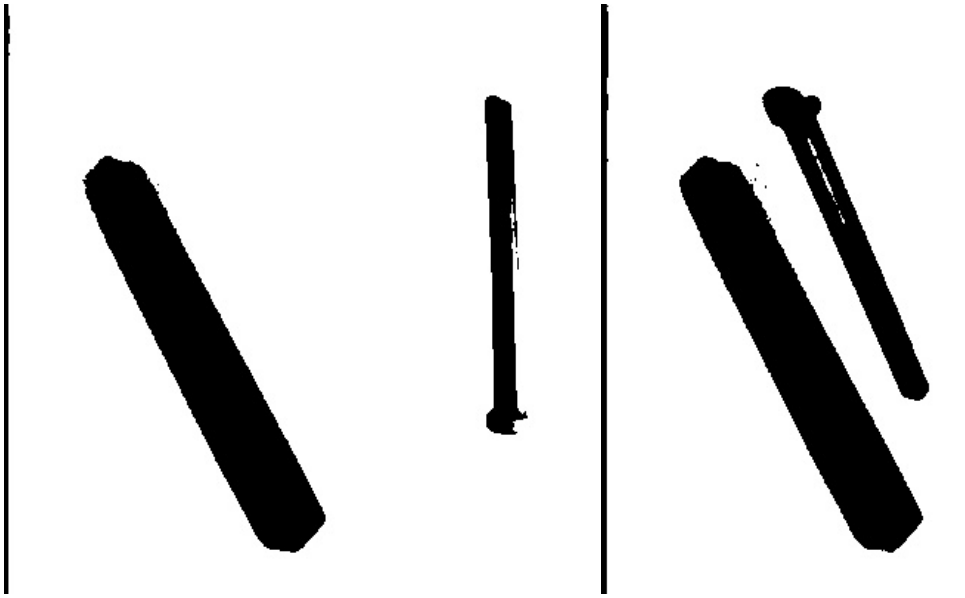
- Imagenes despues despues de invertir los colores



(a) Imagen 1 (b) Imagen 2

Figura 22: Invercion de colores en las imagenes

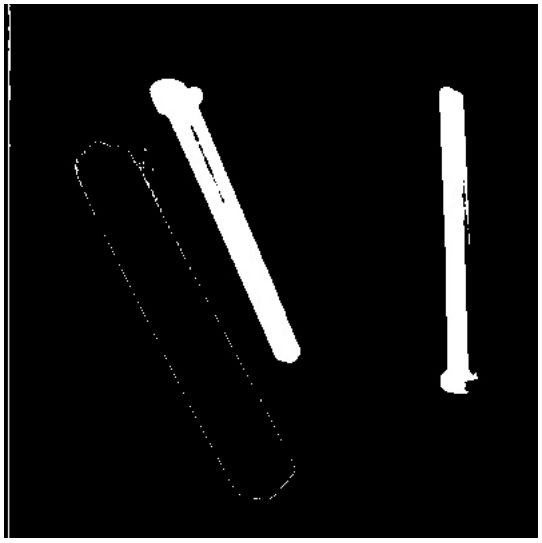
- Imagenes despues de aplicar Thresolding



(a) Imagen 1 (b) Imagen 2

Figura 23: Thresolding

- Resultados despues de aplicar el operador XOR
  Al aplicar el operador OR a cada pixel de las imagenes en su forma binaria se puede llegar a un resultado en la cual se muestra en la figura. La figura resultante muestra al objeto que tuvo ciertos cambios dando como resultado el grafico de los cambios del dicho objeto



- **Codigo**:

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


def getThresholding(img,threshold):
    modifiedImg = img.copy()
    # print(img)
    for i in range(len(img)):
        for j in range(len(img[i])):
            if(img[i][j] < threshold):
                modifiedImg[i][j] = np.uint8(255)
            else:
                modifiedImg[i][j] = np.uint8(0)
    return modifiedImg

imgReal1 = cv2.imread("pernos1.jpg", cv2.IMREAD_GRAYSCALE) #8 bit por
    escala de grises
imgReal1 = cv2.resize(imgReal1,(400,400))
img1 = imgReal1.copy()

imgReal2 = cv2.imread("pernos2.jpg", cv2.IMREAD_GRAYSCALE) #8 bit por
    escala de grises
imgReal2 = cv2.resize(imgReal2,(400,400))
img2 = imgReal2.copy()

img1 = 255-img1
img2 = 255-img2

cv2.imshow("Image Original 1", img1)
cv2.imshow("Image Original 2", img2)


f, (ax1,ax2) = plt.subplots(1, 2,figsize=(15, 5))
ax1.hist(img1.ravel(),256,[0,256])
ax1.set_title("Histograma IMG1")
ax1.set_xlabel('Intensidad')
ax1.set_ylabel('Cantidad de pixeles')
ax2.hist(img2.ravel(),256,[0,256])
ax2.set_title("Histograma IMG2")
ax2.set_xlabel('Intensidad')
ax2.set_ylabel('Cantidad de pixeles')


threshold = 96
imgThres1 = getThresholding(img1,threshold)
cv2.imshow("Resultado Thresolding de la IMG1", imgThres1)
# filename = 'ResultadoThresoldingPernos1_2.jpg'
# cv2.imwrite(filename, imgThres1)

imgThres2 = getThresholding(img2,threshold)
cv2.imshow("Resultado Thresolding de la IMG2", imgThres2)
# filename = 'ResultadoThresoldingPernos2_2.jpg'
```

```
# cv2.imwrite(filename, imgThres2)


# ====================
# OPERADOR XOR
# ====================

imgResult = imgThres1.copy()

for i in range(len(imgThres1)):
    for j in range(len(imgThres1[0])):
        result = imgThres1[i,j] ^ imgThres2[i,j]
        imgResult[i,j] = result
cv2.imshow("Resultado Operador XOR", imgResult)
# filename = 'ResultadoOperadorXOR.jpg'
# cv2.imwrite(filename, imgResult)

plt.show()
cv2.waitKey(0)
cv2.destroyAllWindows()
```

5. Link de los códigos en github

https://github.com/kpzaolod6000/Graphics-Computing/tree/main/parcial_3/lab_10