UNIVERSIDAD NACIONAL DE SAN AGUSTÍN

ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN

COMPUTACION GRAFICA

Practica 11

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

Julio 2021

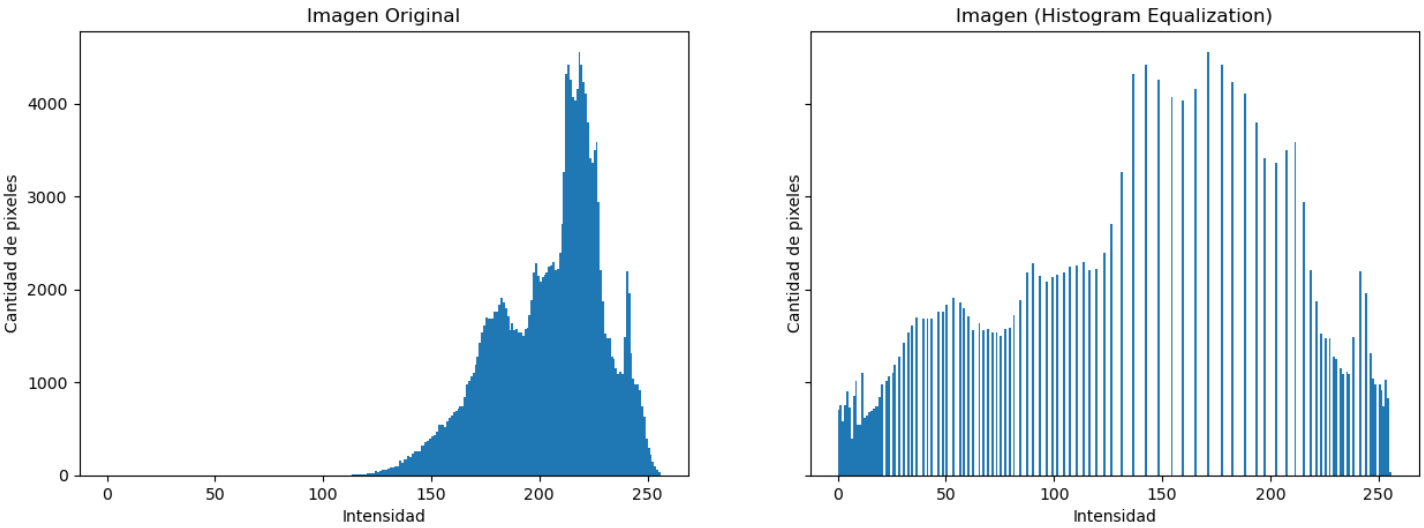
Índice

1. Algoritmo de computación gráfica a Ecualización de Histograma.

1.1. Implemente el algoritmo de ecualización de histograma y evalúe sus resultados con estas imágenes.

- **Histogram Equalization de la imagen 1**

  - **Histograma** :



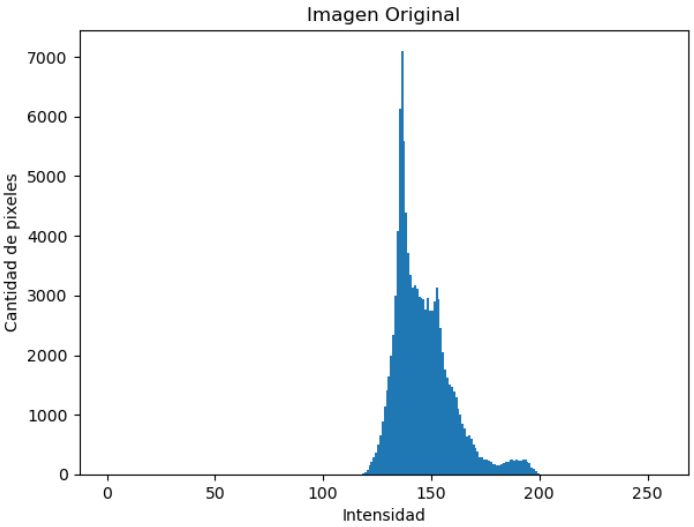
  - **Resultados**:

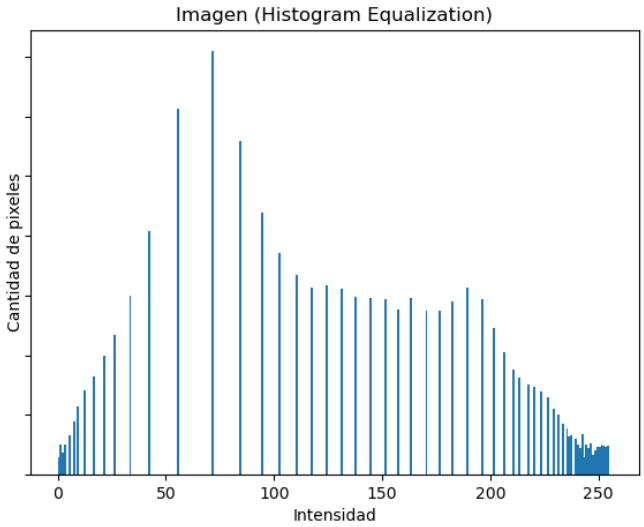
(a) Imagen Real (b) Imagen después de aplicar Histogram Equalization

Figura 1: Histogram Equalization

- **Histogram Equalization de la imagen 2**

  - **Histograma**:





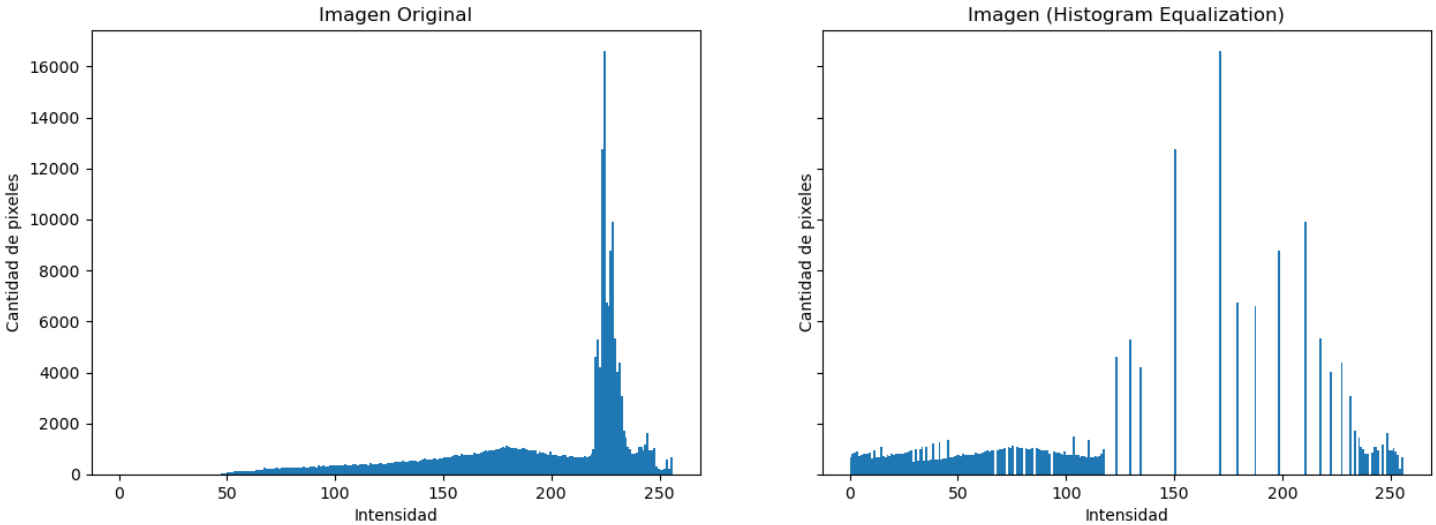
  - **Resultados**:

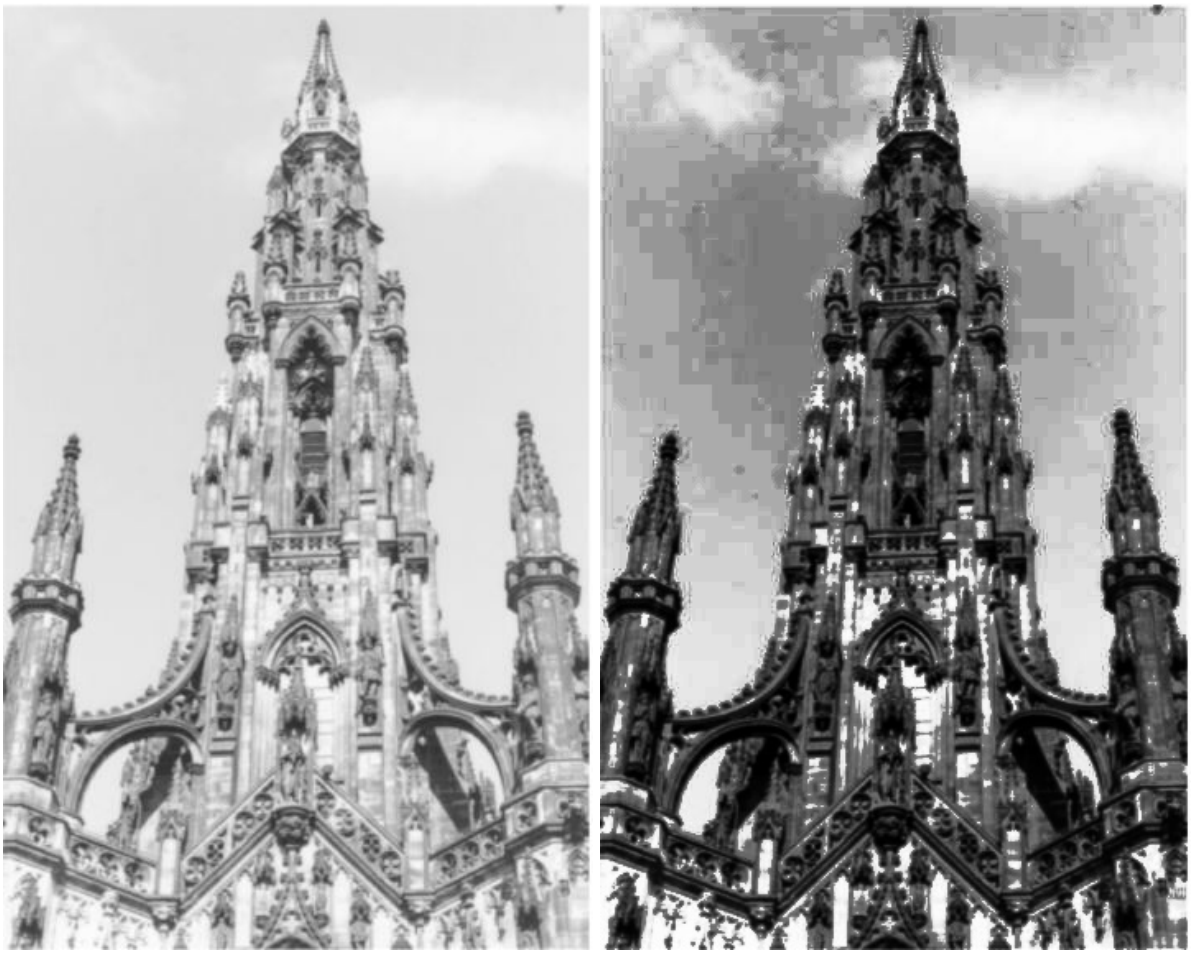
(a) Imagen Real (b) Imagen después de aplicar Histogram Equalization

Figura 2: Histogram Equalization

1.2. En la Figura 2 se muestra una iglesia antes y después de aplicar ecualización de histograma. Usted, debe obtener una subimagen como se ve en la Figura 3 y luego procesar sn (n es la intensidad de un píxel) con los datos de la sub imagen y obtener una imagen mejorada con la fórmula g[x, y] = sf[x,y].

- **Histogram Equalization de la capilla**
  - **Histograma**:





(a) Imagen Real (b) Imagen después de aplicar Histogram Equalization

Figura 3: Histogram Equalization

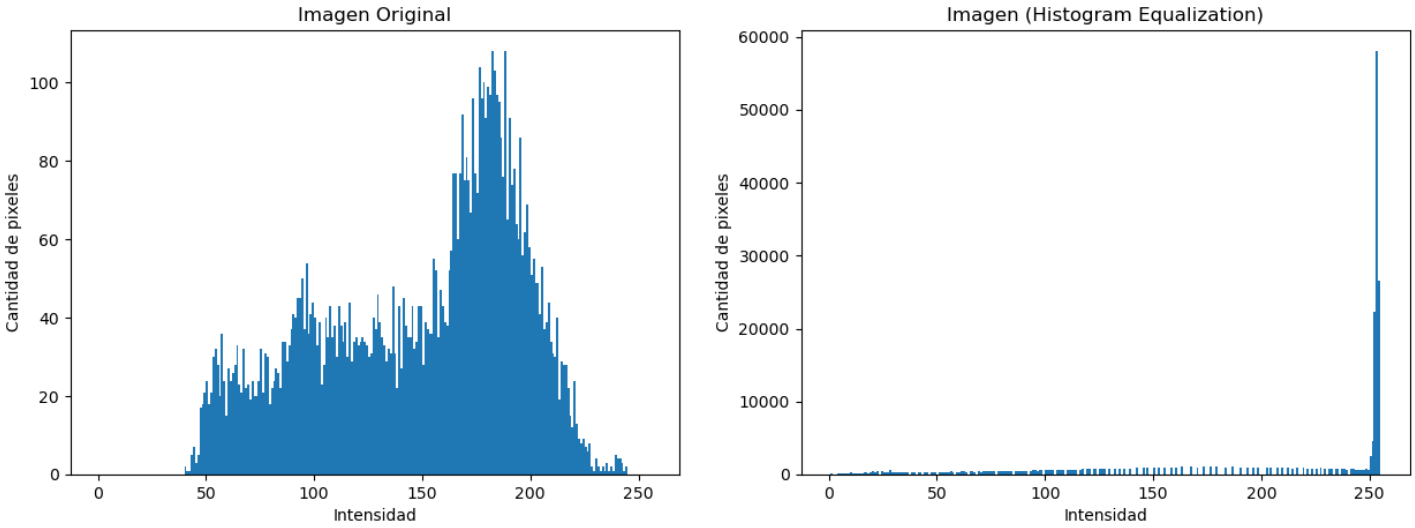
- **Histogram Equalization con los colores de la imagen real** :

- **Codigo**:

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

def HistogramEqualization(histogramData,row,column):
    totalPixel = row*column

    Pn = np.zeros(len(histogramData))
    for k in range(len(Pn)):
        Pn[k] = int(histogramData[k])/totalPixel

    Sn = np.zeros((len(histogramData),), dtype=int)

    for i in range(len(Sn)):
        L = 256
        sumPn = 0
        for j in range(i+1):
            sumPn += Pn[j]
        Sn[i] = math.floor((L-1)*sumPn)
    return Sn

imgReal = cv2.imread(sys.argv[1], cv2.IMREAD_GRAYSCALE) #8 bit por
    escala de grises

img = imgReal
#img = img[200:340,160:215]
#HISTOGRAMA
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True,figsize=(15, 5))
hist_mask = ax1.hist(img.ravel(),256,[0,256])
ax1.set_title('Imagen Original')
ax1.set_xlabel('Intensidad')
ax1.set_ylabel('Cantidad de pixeles')
cv2.imshow('Imagen Original',img)

histogramData = hist_mask[0]


Sn = HistogramEqualization(histogramData,img.shape[0],img.shape[1])

for i in range(len(img)):
    for j in range(len(img[i])):
        img[i][j] = np.uint8( Sn[img[i][j]] )#a la posicion del color
            agrega un nuevo color modificado

ax2.hist(img.ravel(),256,[0,256])
ax2.set_title('Imagen (Histogram Equalization)')
ax2.set_xlabel('Intensidad')
ax2.set_ylabel('Cantidad de pixeles')
cv2.imshow('Imagen (Histogram Equalization)',img)
# Filename
plt.savefig('Histograma3.png')
```

```
# Filename
filename = 'HistEquaCapilla.jpg'
cv2.imwrite(filename, img)

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

- **Histogram Equalization con la imagen recortada**



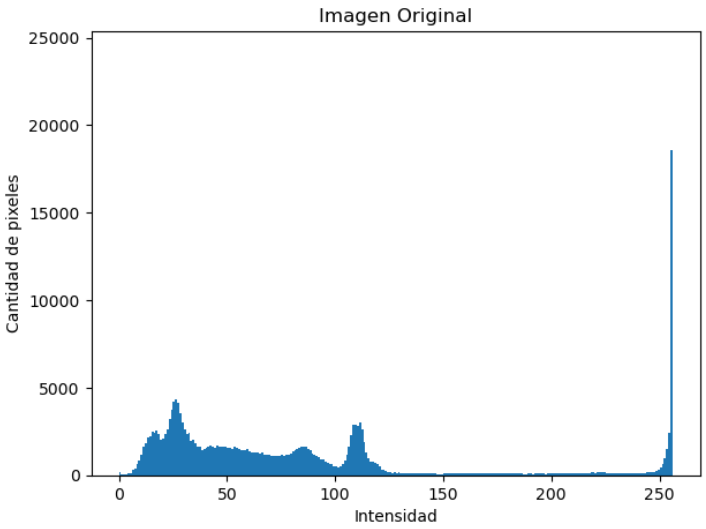
(a) Imagen Real (b) Imagen después de aplicar Histogram Equalization

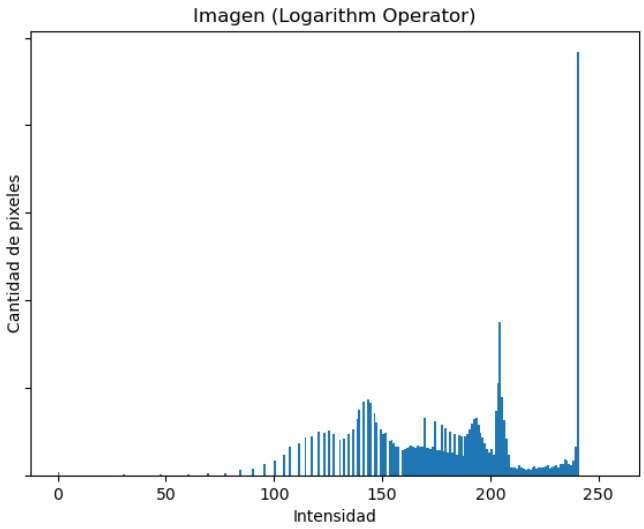
Figura 4: Histogram Equalization

2. Algoritmo de computación gráfica a Operador ´logarítmco y Operator Raiz

2.1. Implemente el operador logarítmico y evalúe sus resultados con estas imágenes (Figura 1). Escoja diferentes valores de c y comente con cuál obtuvo mejores resultados.

- **Aplicando Operador Logarítmic**
  - **Histograma**:



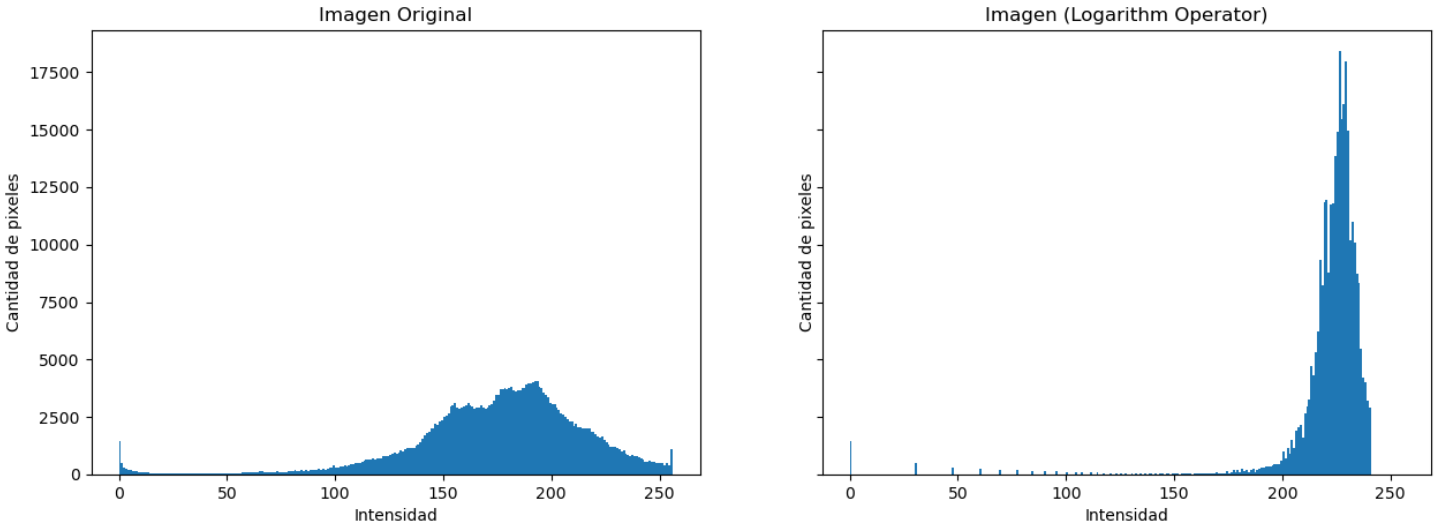


  - **Resultados**: En este imagen se utilizó una constante c igual a 100 el cual fue el mas optimo para subir la intensidad de los píxeles de la imagen

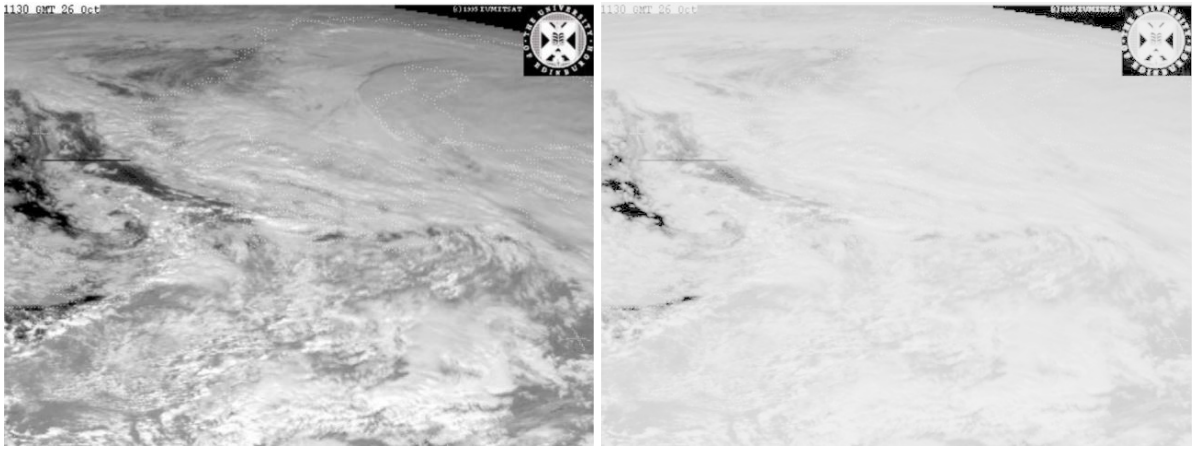
(a) Imagen Real (b) Imagen después de aplicar Operador Logarítmic

Figura 5: Operador Logarítmic

- **Histograma**:



- **Resultados**: En este imagen se utilizó una constante c igual a 100 , pero a pesar de usar una constante c mayor a 100 también los resultados no fueron óptimos por la pérdida de información ya que mejoraba la intensidad de los pixeles baja intensidad y esto hacía que aparezca el color blanco.En la imagen se muestra que existen algunas grietas que es importante y al aplicar Operator Logarítmic estos desaparecen

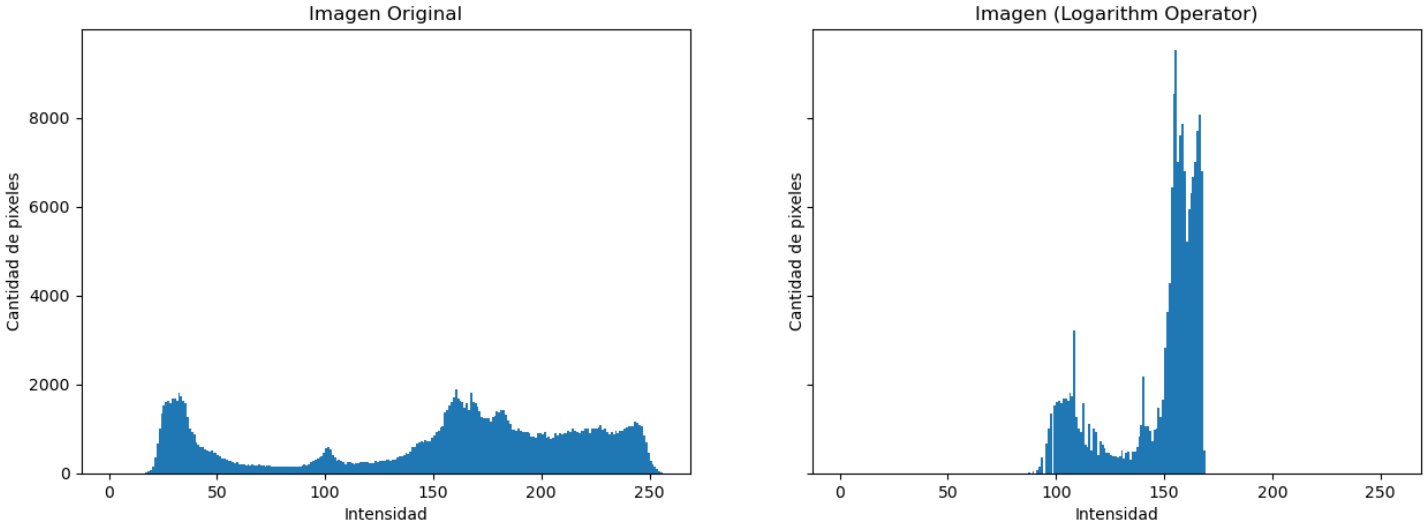


(a) Imagen Real (b) Imagen después de aplicar Operador Logarítmic

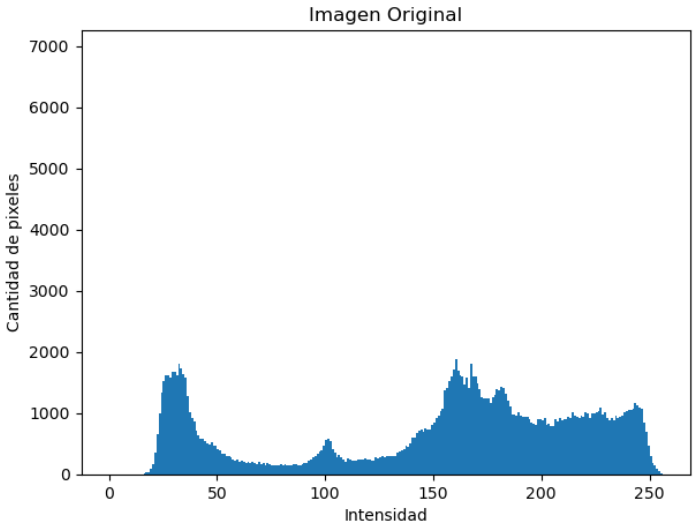
Figura 6: Operador Logarítmic

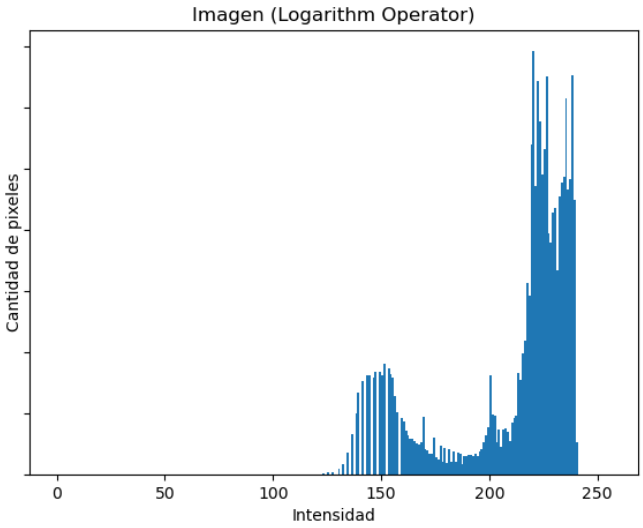
2.2. Aplique el operador logarítmico en la imagen de la Figura 2 y luego comente sus resultados.

- **Aplicando Operador Logarítmic**
  - **Histogramas**:
    Histograma con constante c = 70

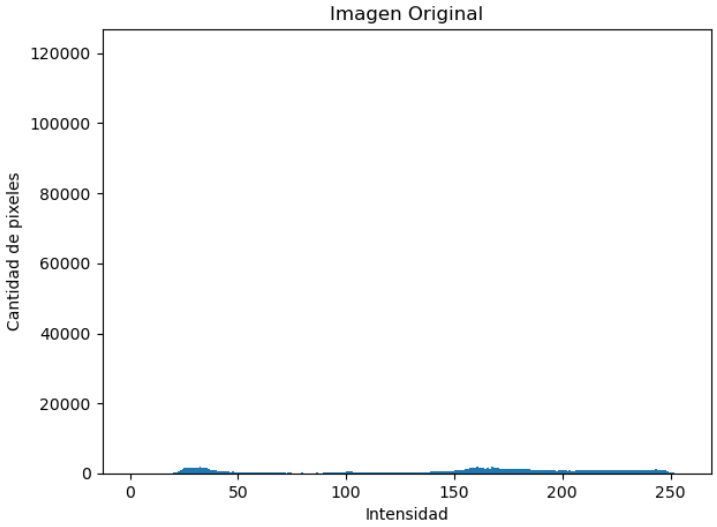


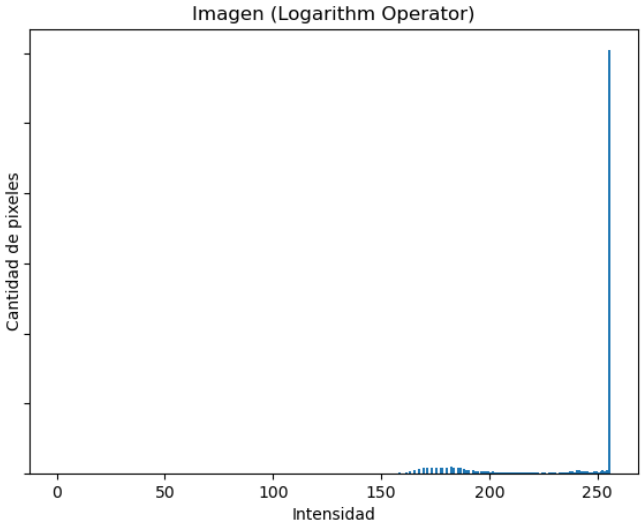
Histograma con constante c = 100



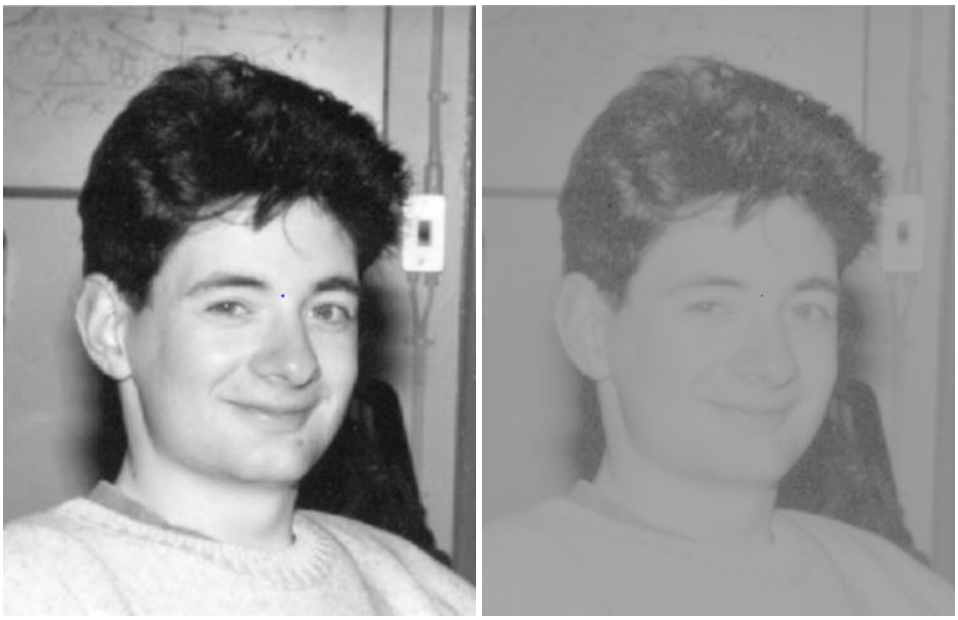


Histograma con constante c = 120





- **Resultados**: En esta imagen se utilizó una constante c igual a 70,100,120 y en ninguna de estas se encontró una mejora ya que el aumento de intensidad de cada pixel hace que se desaparezcan los detalles importantes del rostro.



(a) Imagen real

(b) Imagen despues de aplicar Operador Logarítmic con constante c = 70

Figura 7: Operador Logarítmic

(a) Imagen después de aplicar Operador Logarítmic con constante c = 100

(b) Imagen después de aplicar Operador Logarítmic con constante c = 120

Figura 8: Operador Logarítmic

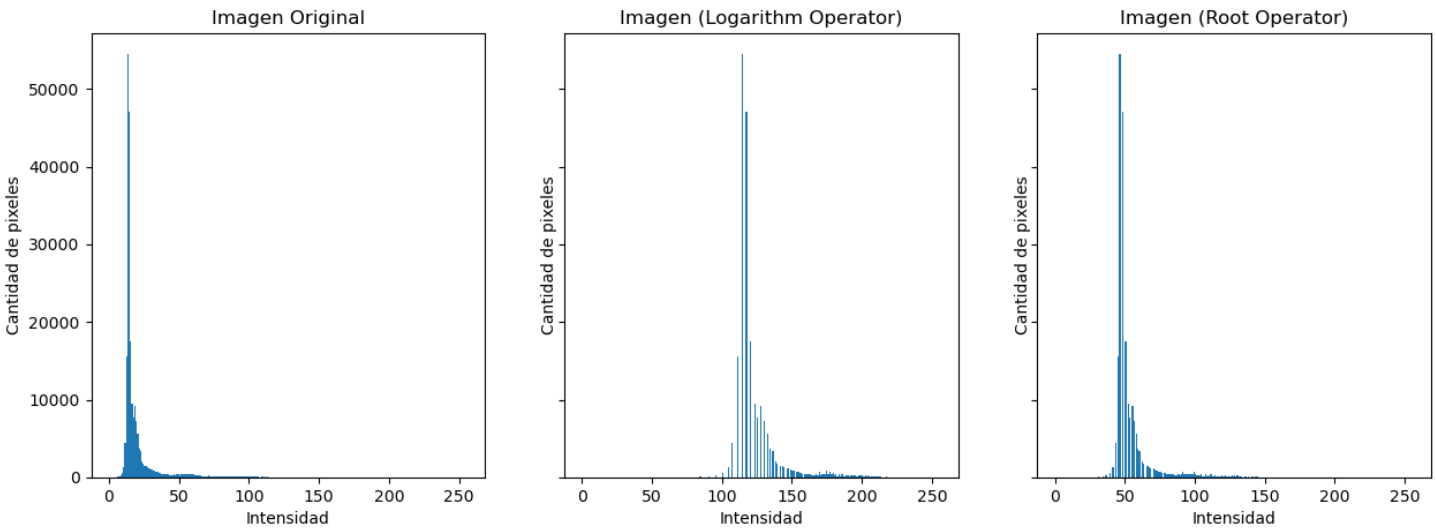
- **Código**:

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

imgReal = cv2.imread(sys.argv[1], cv2.IMREAD_GRAYSCALE) #8 bit por
    escala de grises
img = imgReal

#HISTOGRAMA
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True,figsize=(15, 5))
ax1.hist(img.ravel(),256,[0,256])
ax1.set_title('Imagen Original')
ax1.set_xlabel('Intensidad')
ax1.set_ylabel('Cantidad de pixeles')
cv2.imshow('Imagen Original',img)

#constant c
c = 100
for i in range(len(img)):
    for j in range(len(img[0])):
        new_pixel = c * math.log10(1 + int(img[i][j]))
        if (new_pixel > 255):
            new_pixel = 255
        img[i][j] = np.uint8(new_pixel)


ax2.hist(img.ravel(),256,[0,256])
ax2.set_title('Imagen (Logarithm Operator)')
ax2.set_xlabel('Intensidad')
ax2.set_ylabel('Cantidad de pixeles')
cv2.imshow('Imagen (Logarithm Operator)',img)

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

2.3. Al igual que el operador logarítmico, también se puede utilizar el operador raíz: g[x, y] = c ∗ sqrt(f[x, y]). Aplique el operador raíz para la imagen de la Figura 3 y haga una comparación con el operador logarítmico.

- **Aplicando Operador Logarítmic**

  - **Histogramas**:
    Histograma aplicando Operador logarítmico con una constante c = 100
    Histograma aplicando Operador raíz con una constante c = 20



- **Resultados**:

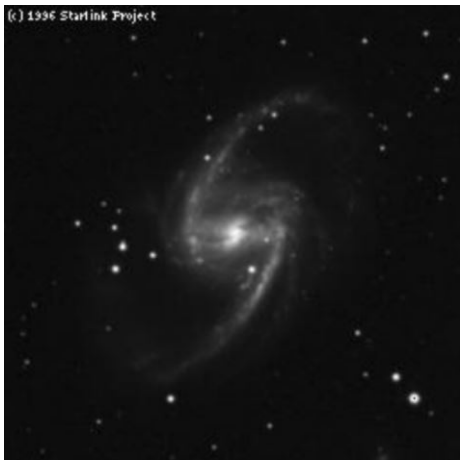
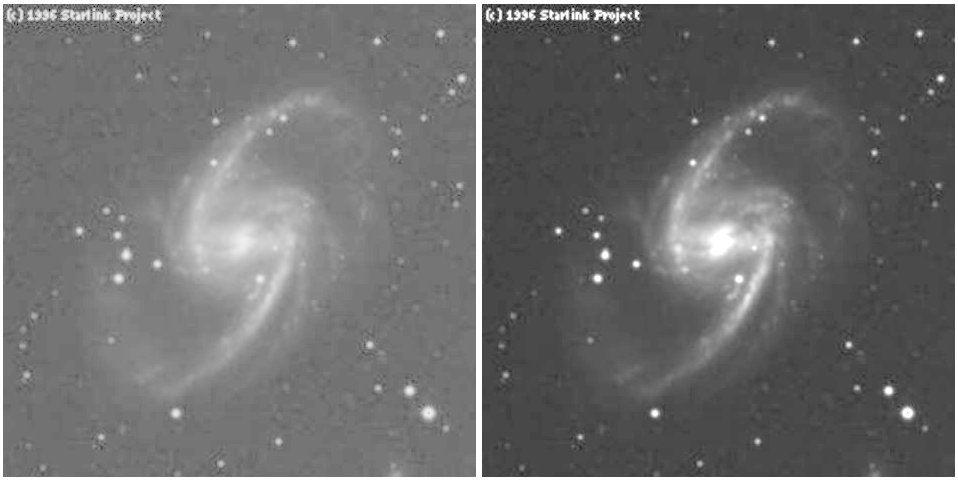


Figura 9: Imagen real

En esta imagen se utilizó una constante c igual a 20 para aplicar el algoritmo operador raíz, esto nos permite aumentar la intensidad de los píxeles hasta llegar a un estado aceptable donde no se pierda información, además de intensificar bien la galaxia ante la parte oscura de la imagen. Respecto al algoritmo del operador logarítmico se utilizó una constante c igual a 100 porque si se aumenta más se pierde información aumentando la intensidad sin importar detallar la galaxia.
Ambos algoritmos aclaran las imágenes pero el algoritmo del operador Raíz es aquel que conserva mas los detalles a diferencia del algoritmo del operador logarítmico que lo que hace es intensificar pero perdiendo información.
En la siguiente imagen se verifica la diferencia de estos algoritmos



(a) Imagen después de aplicar Operador Logarítmic con constante c = 100

(b) Imagen después de aplicar Operador raiz con constante c = 20

Figura 10: Logarítmic Operator and Root Operator

- **Codigo**:

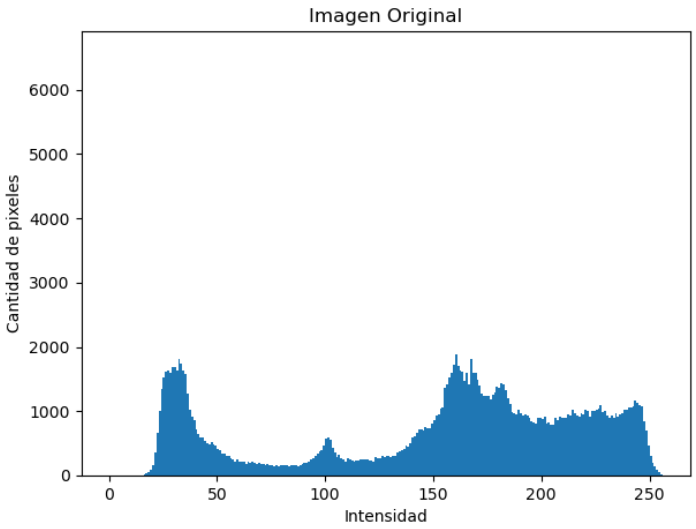
```
import cv2
import numpy as np
from matplotlib import pyplot as plt
import sys
import math

imgReal = cv2.imread(sys.argv[1], cv2.IMREAD_GRAYSCALE) #8 bit por
    escala de grises
img = imgReal.copy()

#HISTOGRAMA
f, (ax1, ax2,ax3) = plt.subplots(1, 3, sharey=True,figsize=(15, 5))
ax1.hist(img.ravel(),256,[0,256])
ax1.set_title('Imagen Original')
ax1.set_xlabel('Intensidad')
ax1.set_ylabel('Cantidad de pixeles')
cv2.imshow('Imagen Original',img)

#LOGARITMIC OPERATOR
#constant c
c = 100
for i in range(len(img)):
    for j in range(len(img[0])):
        new_pixel = c * math.log10(1 + int(img[i][j]))
        if (new_pixel > 255):
            new_pixel = 255
        img[i][j] = np.uint8(new_pixel)


ax2.hist(img.ravel(),256,[0,256])
ax2.set_title('Imagen (Logarithm Operator)')
```

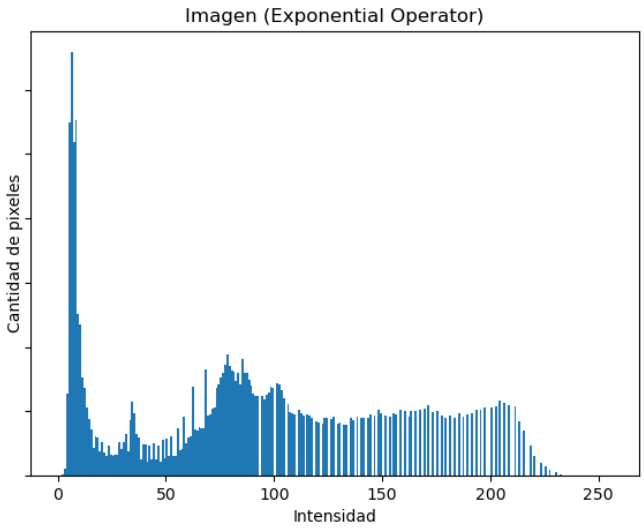
```
ax2.set_xlabel('Intensidad')
ax2.set_ylabel('Cantidad de pixeles')
cv2.imshow('Imagen (Logarithm Operator)',img)

#ROOT OPERATOR
#constant c
img2 = imgReal.copy()
c = 13
for i in range(len(img2)):
    for j in range(len(img2[0])):
        new_pixel = c * math.sqrt(int(img2[i][j]))
        if (new_pixel > 255):
            new_pixel = 255
        img2[i][j] = np.uint8(new_pixel)

ax3.hist(img2.ravel(),256,[0,256])
ax3.set_title('Imagen (Root Operator)')
ax3.set_xlabel('Intensidad')
ax3.set_ylabel('Cantidad de pixeles')
cv2.imshow('Imagen (Root Operator)',img2)

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

3. Algoritmo de computación gráfica Exponential Operator y Raise to the power Operator

3.1. Implemente el operador exponencial y evalue sus resultados con la Figura 1. Escoja diferentes valores de b, c y comente con cual obtuvo mejores resultados.

- **Aplicando Operador Exponencial**
  - **Histograma**:





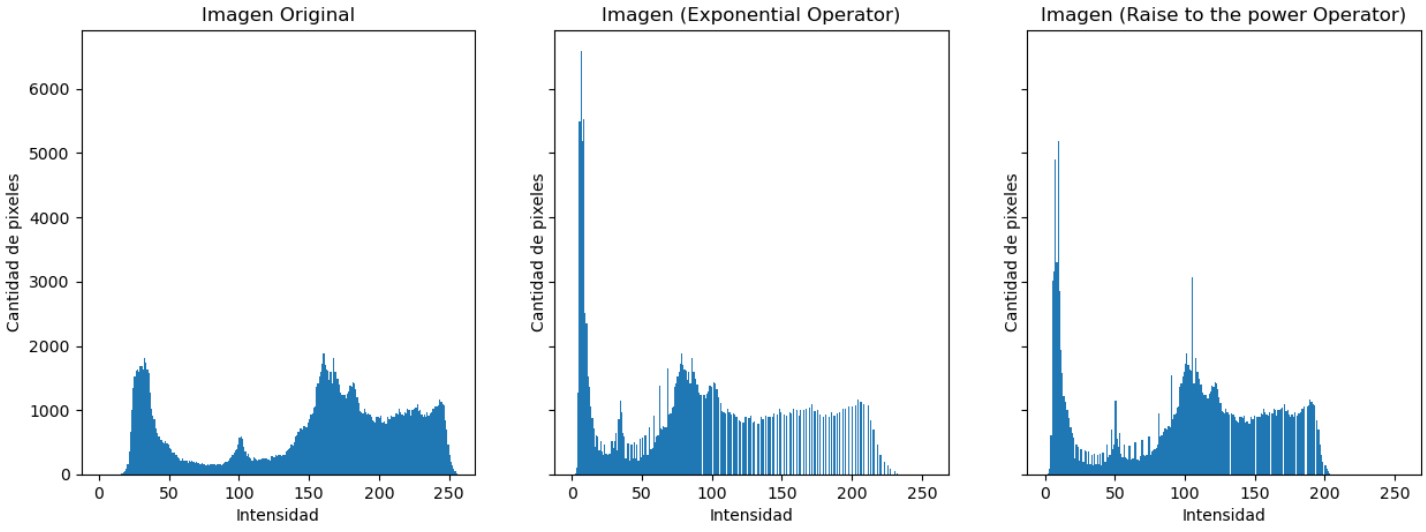
  - **Resultados**:
    - Aplicando Operador Exponencial con las constantes c = 20 y b = 1.01

(a) Imagen Real (b) Imagen después de aplicar Operador Exponencial

Figura 11: Operador Exponencial

3.2. Implemente el operador Raise to power y evalúe sus resultados con Figura 1. Escoja diferentes valores de r, c y comente con cuál obtuvo mejores resultados.

- **Aplicando Operador Exponencial y el Operador Raise to the power**
  - **Histograma**:



  - **Resultados**:

(a) Imagen después de aplicar Operador Exponencial

(b) Imagen después de aplicar Operador Raise to the power

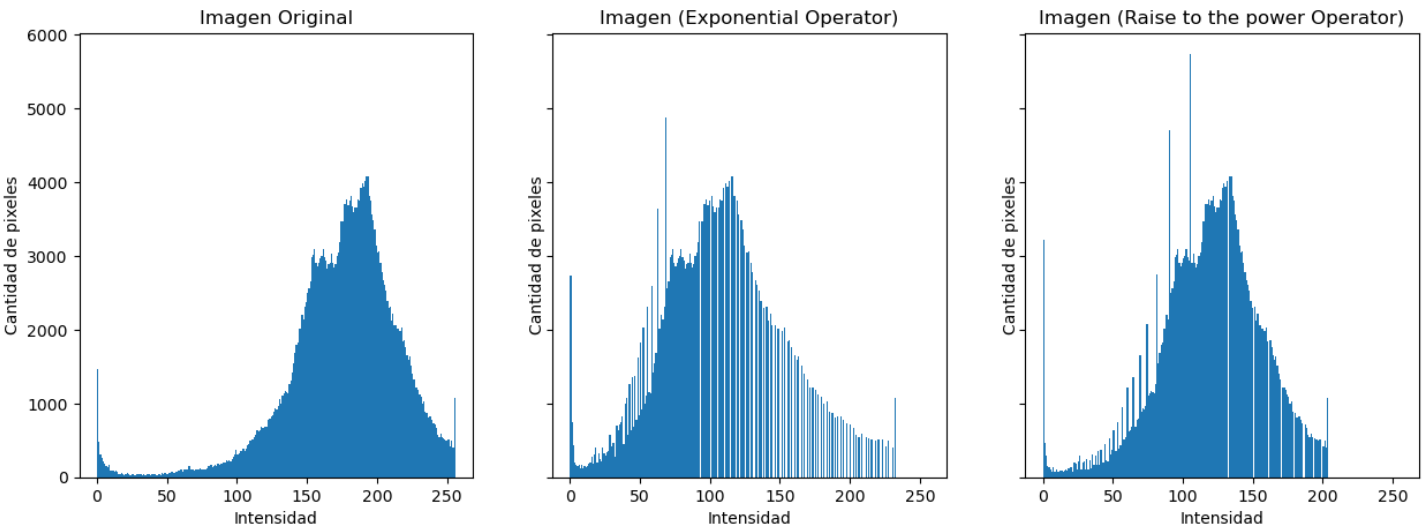
Figura 12: Operador Exponencial y Operador Raise to the power

- Aplicando Operador Exponencial con las constantes c = 20 y b = 1.01
- Aplicando Operador Raise to the power con las constantes c = 0.05 y b = 1.5

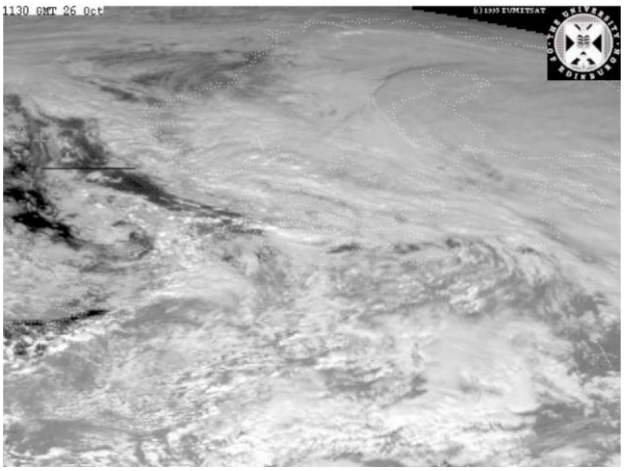
El algoritmo del Operador Exponencial es aquel que presenta mas contraste donde baja la intensidad de cada pixel. En cambio el algoritmo del Operador Raise to the power baja la intensidad pero no mucho . Donde se obtuvo mejor contraste fue utilizando el operador Exponencial sin embargo el operador Raise to the Power no tiene mucho contraste pero conserva los detalles de la imagen sin perder información

3.3. Evalúe el operador exponencial y Raise to power con la Figura 2, pruebe con diferentes valores de b, c y r. Luego comente si la imagen mejoró, y con qué operador se obtuvieron mejores resultados.

- **Aplicando Operador Exponencial y el Operador Raise to the power**
  - **Histograma**:

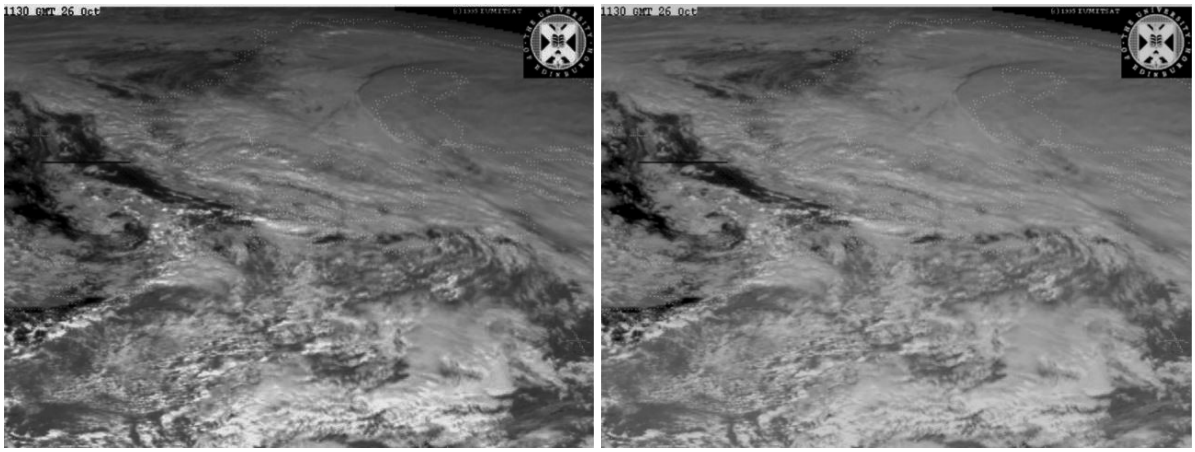


- **Resultados**:



  - Aplicando Operador Exponencial con las constantes c = 20 y b = 1.01
  - Aplicando Operador Raise to the power con las constantes c = 0.05 y b = 1.5

El resultado fue bien óptimo para ambos algoritmos ya que bajaron la intensidad de la imagen sin perder información de hecho ayuda mas a resaltar las grietas que se encuentran en dicha imagen. Aplicando Operador Exponencial a la imagen se nota una ligera superioridad respecto al contraste



(a) Imagen después de aplicar Operador Exponencial (b) Imagen después de aplicar Operador Raise to the power

Figura 13: Operador Exponencial y Operador Raise to the power

3.4. Evalúe el operador exponencial, Raise to power y logarítmico, con la Figura 3, Comente con que operador mejora, muestre sus resultados y comparaciones.

- **Aplicando Operador Exponencial ,el Operador Raise to the power y el Operador logarítmico**
  - **Histograma**:

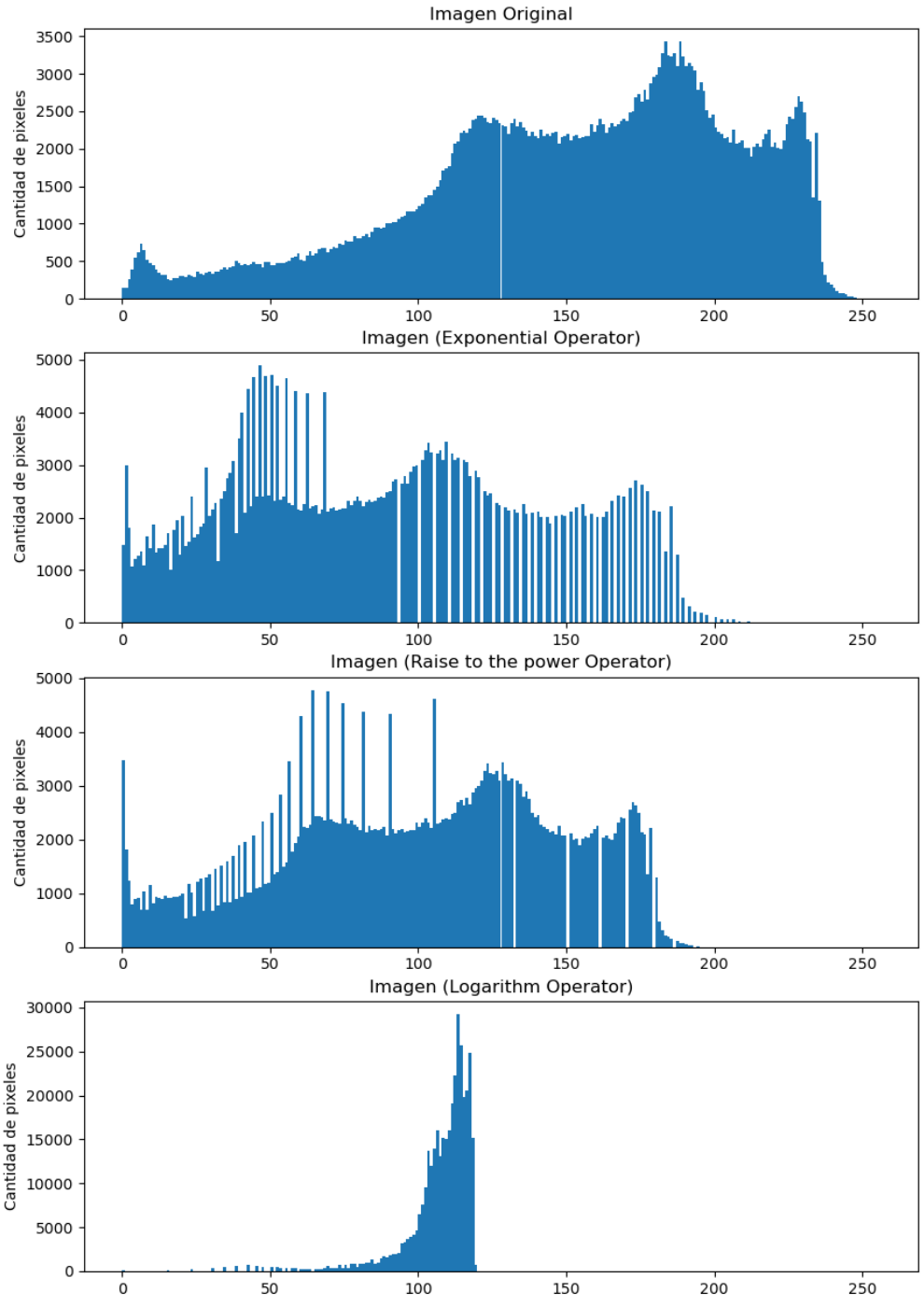


Imagen Original
Cantidad de pixeles
0
500
1000
1500
2000
2500
3000
3500
0
50
100
150
200
250
Imagen (Exponential Operator)
Cantidad de pixeles
0
1000
2000
3000
4000
5000
0
50
100
150
200
250
Imagen (Raise to the power Operator)
Cantidad de pixeles
0
1000
2000
3000
4000
5000
0
50
100
150
200
250
Imagen (Logarithm Operator)
Cantidad de pixeles
0
5000
10000
15000
20000
25000
30000
0
50
100
150
200
250

- **Resultados**:

  - Aplicando Operador Exponencial con las constantes c = 20 y b = 1.01
  - Aplicando Operador Raise to the power con las constantes c = 0.05 y b = 1.5
  - Aplicando Operador Logarítmico con la constante c = 50

El resultado fue óptimo en los operadores el cual disminuye la intensidad del color de la imagen ya que describe más detalladamente. Sin embargo el operador logarítmico al ser un método que se encarga de subir la intensidad de la imagen esto ocasiona que se pierda información porque la imagen en sí está clara ahí que el resultado al aplicar este método no es adecuado para esta imagen

(a) Imagen después de aplicar Operador Exponencial

(b) Imagen después de aplicar Operador Raise to the power

Figura 14: Operador Exponencial y Operador Raise to the power

(a) Imagen después de aplicar Operador Logarítmico

4. Link de los códigos en github

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