



Trabajo Práctico 1

Procesamiento de imágenes y visión por computadora

Nicolás Seivane
11 de Septiembre de 2025

Ejercicio 1: Función de potencia γ

1.1 Descripción función de potencia γ

Se describe la función de γ , donde se oscurecen los píxeles con valor más bajo y se aclaran los píxeles con valores de intensidad mayores.

Este tipo de operación es puntual, ya que se realiza una transformación pixel a pixel.

Operador puntual

$f(x, y) = r$ siendo f la imagen original y r el valor del píxel (x, y) .

$g(x, y) = s$ siendo g la imagen de salida y s el valor del píxel (x, y) .

Por lo tanto:

$$T(r) = s$$

1.1.2 Descripción función de potencia γ

Entonces definimos el método de la función de potencia γ .

Función de potencia γ

$$T(r) = c r^\gamma, \quad 0 < \gamma < 2, \gamma \neq 1$$

$$c = (255)^{1-\gamma}$$

Por lo tanto:

$$T(r) = (255)^{1-\gamma} r^\gamma, \quad 0 < \gamma < 2, \gamma \neq 1$$

1.2.1 Código para función de potencia γ

Entonces definimos el código de la función de potencia γ .

```
def funcion_y_previw():

    global imagen_operativa

    src = obtener_fuente_operacion()

    gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)

    gamma = slider_y.get()

    constante = 255 ** (1 - gamma)

    imagen_corregida = gray.astype(np.float32)

    height, width = gray.shape
```

1.2.2 Código para función de potencia γ

Entonces definimos el código de la función de potencia γ .

```
for fila in range(height):

    for columna in range(width):

        imagen_corregida[fila, columna] = (gray[fila,
column] ** gamma) * constante

        imagen_corregida = ( (imagen_corregida - np.min(
imagen_corregida)) / (np.max(imagen_corregida) - np.min(
imagen_corregida)) ) * 255

    imagen_operativa = imagen_corregida.astype(np.uint8)

mostrar_imagen(imagen_operativa, lblOutputImage)
```

1.3 Comparación de imágenes

Original



Gamma 0.3



1.3 Comparación de imágenes

Gamma 1.1



Gamma 1.5



Ejercicio 2: Negativo de una imagen

2.1 Descripción función de negativo

Se describe la función de negativos, donde los píxeles con valor más bajo se aclaran y los píxeles con valores de intensidad mayores se oscurecen.

Este tipo de operación es puntual, ya que se realiza una transformación pixel a pixel.

Función Negativo

$$T(r) = 255 - r$$

2.2 Código para función de negativos

Entonces definimos el código de la función de negativos.

```
def fnegativo():
    global imagen_operativa
    src = obtener_fuente_operacion()

    bgr_src = asegurar_bgr(src)
    imagen_negativa = np.zeros_like(bgr_src)

    height, width, channels = bgr_src.shape

    for fila in range(height):
        for columna in range(width):
            for canal in range(channels):

                imagen_negativa[fila, columna, canal] = 255 -
                bgr_src[fila, columna, canal]

    imagen_operativa = imagen_negativa.astype(np.uint8)
    mostrar_imagen(imagen_operativa, lblOutputImage)
```

2.3 Comparación de imágenes

Original



Negativo



2.3 Comparación de imágenes

Original



Negativo



2.3 Comparación de imágenes

Original



Negativo



2.3 Comparación de imágenes

Original



Negativo



Ejercicio 3: Histograma en niveles de gris

3.1 Descripción función de histogramas de grises

Es la frecuencia relativa de los niveles de gris presentes en la imagen.

Función Negativo

$$h_i = \frac{n_i}{N.M}, i = 0, \dots, 255$$

donde;

- n_i : cantidad de pixels que tienen ese valor i dentro de la imagen.
- NM : cantidad de pixels totales que tiene la imagen.

3.2.1 Código para función de histogramas

Entonces definimos el código de la función de histogramas.

```
def histograma_grises():

    src = obtener_fuente_operacion()
    gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)

    height, width = gray.shape
    pixeles_totales = height * width
    valores = np.arange(256)
    cuenta = [0] * 256

    for fila in range(height):
        for columna in range(width):
            cuenta[gray[fila, columna]] += 1
```

3.2.2 Código para función de histogramas

Entonces definimos el código de la función de histogramas.

```
for i in range(len(cuenta)):
    cuenta[i] = cuenta[i] / pixeles_totales

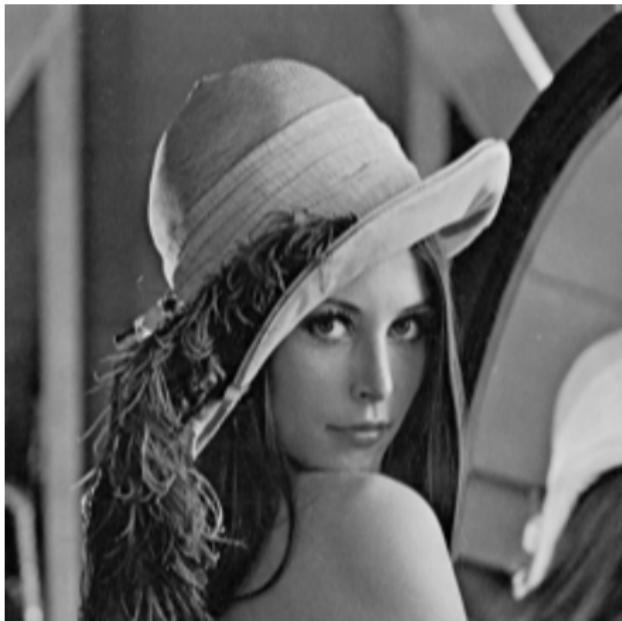
plt.figure()
plt.title('Histograma de Grises')
plt.xlabel('Nivel de Gris')
plt.ylabel('Frecuencia Relativa') # 0 'Frecuencia
Relativa',

plt.bar(valores, cuenta, width=1, align='center', color=
'gray')
plt.xlim([-1, 256])

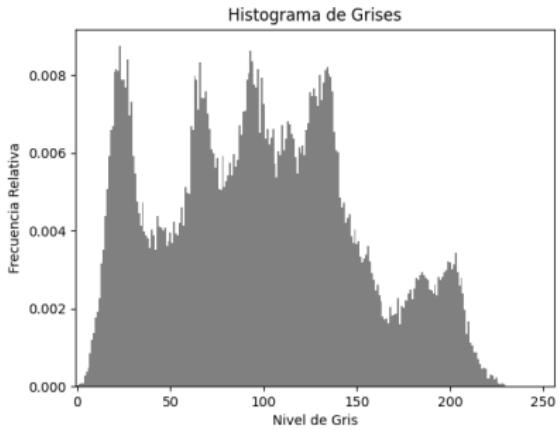
plt.show()
set_status("Mostrando histograma de la imagen de trabajo
.")
```

3.3 Comparación de imágenes

Original



Histograma

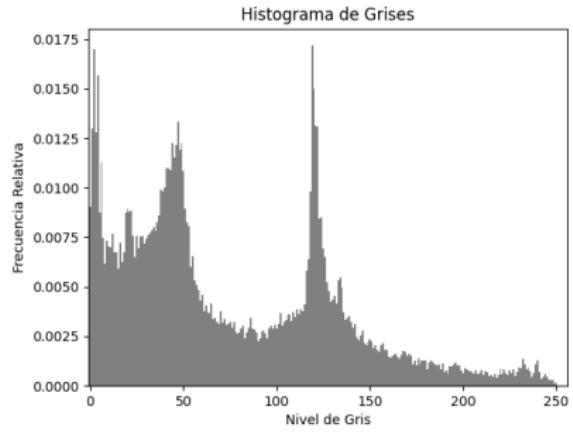


3.3 Comparación de imágenes

Original



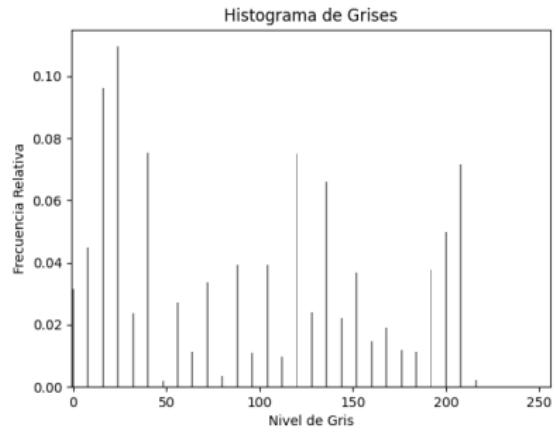
Histograma



3.3 Comparación de imágenes

Histograma

Original



3.3 Comparación de imágenes

Original



Histograma



Ejercicio 4: Umbral para una imagen binaria

4.1 Descripción función de umbralización

Se describe la función de umbralización, donde si el valor del pixel es mayor o menor a un umbral se le asigna un valor 255 o 0.

Este tipo de operación es puntual, ya que se realiza una transformación pixel a pixel.

Función umbral

donde $u \in [0, \dots, 255]$

$$T(r) = \begin{cases} 255 & \text{si } r \geq u \\ 0 & \text{si } r < u \end{cases}$$

4.2 Código para función de umbralización

Entonces definimos el código de la función de umbralización.

```
def aplicar_umbral_preview():
    global imagen_operativa
    src = obtener_fuente_operacion()

    gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
    u = slider_umbral.get()
    binaria = np.zeros_like(gray)

    height, width = gray.shape
    for y in range(height):
        for x in range(width):
            if gray[y, x] > u:
                binaria[y, x] = 255
            else:
                binaria[y, x] = 0

    imagen_operativa = binaria
    mostrar_imagen(imagen_operativa, lblOutputImage)
```

4.3 Comparación de imágenes

Original



Umbral 20



Umbral 181



4.3 Comparación de imágenes

Original



Umbral 36



Umbral 144



Ejercicio 5: Ecualización de un Histograma

5.1.1 Ecualización de un Histograma

La idea es realizar una distribución mas uniforme de los niveles de gris. Se basa en controlar la función de densidad de probabilidad de los niveles de gris, de manera que sea mas uniforme.

Sumatoria de frecuencia relativa

$$s_k = T(r_k) = \sum_{i=0}^k \frac{n_i}{n}$$

- r_k es el k-esimo nivel de gris
- n_i es la cantidad de pixels con el k-esimo nivel de gris
- n es el numero total de pixeles de la imagen, es NM
- s_k es la distribución acumulada del histograma, donde es la sumatoria de frecuencia relativa hasta el k-esimo nivel de gris.

5.1.2 Ecualización de un Histograma

Luego de obtener lo anterior hay que discretizar para todos los niveles de gris

Discretización

Sabemos que $s_{minimo} = < s_k = < 1$. Necesitamos que $s_k \in [0, \dots, 255]$

$$s'_k = 255 * \left[\frac{s_k - s_{min}}{1 - s_{min}} \right]$$

5.2.1 Código para función de ecualización

Entonces definimos el código de la función de ecualización.

```
def ecualizacion():
    global imagen_operativa
    src = obtener_fuente_operacion()

    grises = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
    height, width = grises.shape
    imagen_ecualizada = np.zeros_like(grises)

    total = [0]*256
    for i in range(height):
        for j in range(width):
            total[grises[i,j]] += 1
    total_pixels = height * width

    cdf = [0]*256
    cdf[0] = total[0] / total_pixels
    for i in range(1, 256):
        cdf[i] = total[i]/total_pixels + cdf[i-1]
```

5.2.2 Código para función de ecualización

Entonces definimos el código de la función de ecualización.

```
min_index = 0
for i in range(256):
    if cdf[i] > 0:
        min_index = i
        break
min_val = cdf[min_index]
for i in range(height):
    for j in range(width):
        val = grises[i,j]
        transformacion = ((cdf[val]-min_val)/(1-min_val))*255
        valor_final = transformacion
        if valor_final < 0:
            valor_final = 0
        elif valor_final > 255:
            valor_final = 255
        imagen_ecualizada[i, j] = int(valor_final)
imagen_operativa = imagen_ecualizada
mostrar_imagen(imagen_operativa, lblOutputImage)
```

5.3 Comparación de imágenes

Original

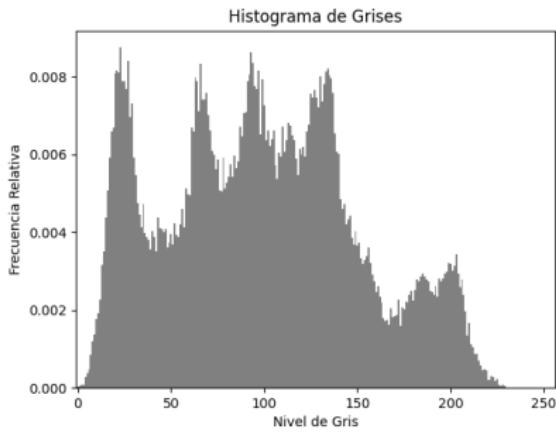


Ecualización

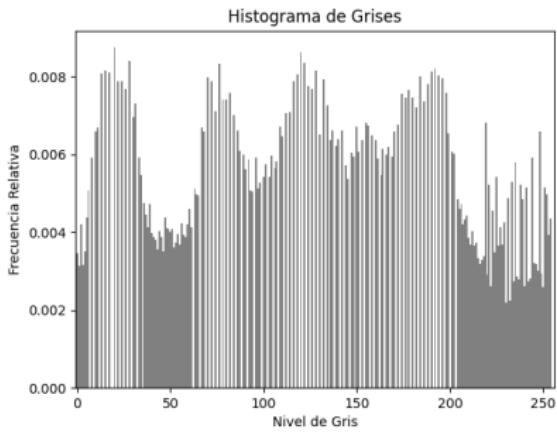


5.3 Comparación de imágenes

Original histograma



Ecualización histograma



5.3 Comparación de imágenes

Original

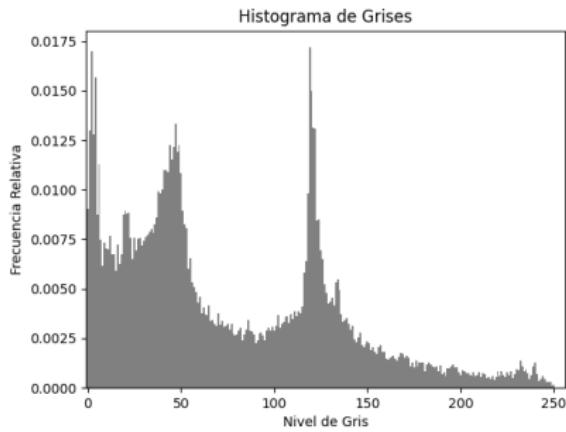


Ecualización

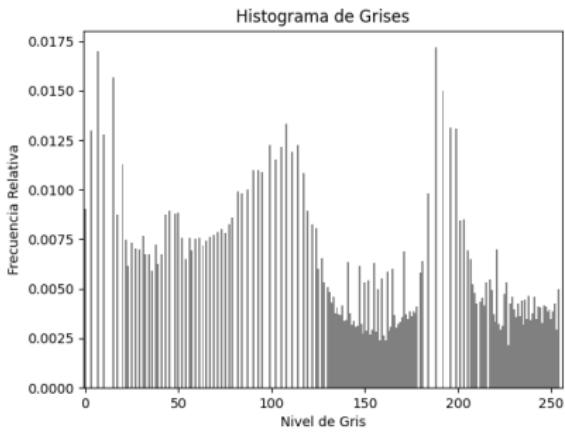


5.3 Comparación de imágenes

Original histograma



Ecualización histograma



Ejercicio 6: Ecualización por segunda vez de un Histograma

6 Comparación de imágenes

1ra Ecualización

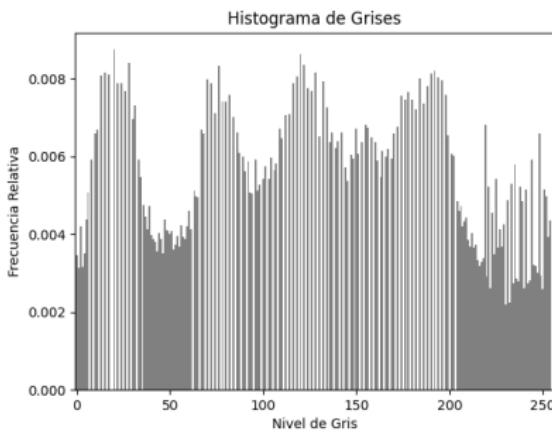


2da Ecualización

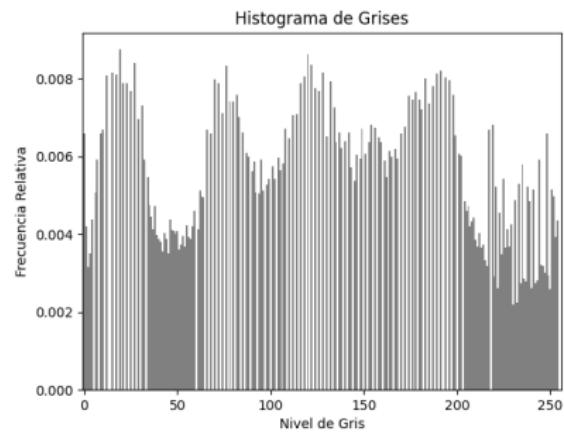


6 Comparación de imágenes

1er Ecualización histograma



2do Ecualización histograma



6 Comparación de imágenes

1ra Ecualización

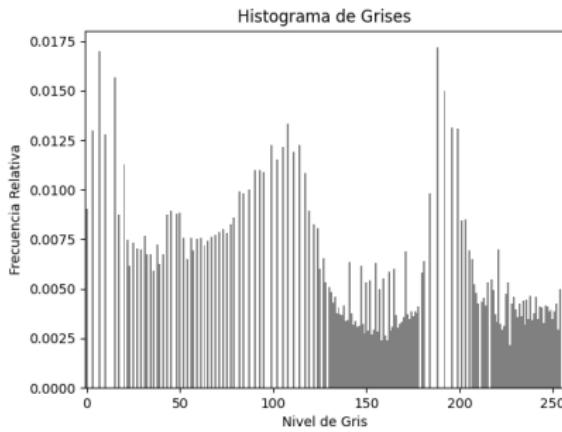


2da Ecualización

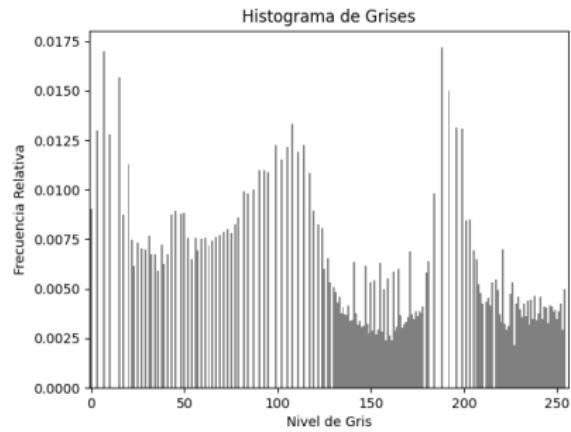


6 Comparación de imágenes

1er Ecualización histograma



2do Ecualización histograma



Ejercicio 7: Generadores de números aleatorios

7.1 Números Aleatorios

La posibilidad de generar números aleatorios que tengan distintas distribuciones.

Variables globales de entrada

```
media_gaussiana_valor = None  
desvio_gaussiano_valor = None  
rayleigh_valor = None  
exponencial_valor = None  
tipo_ruido = None  
cantidad_ruido = None
```

Son variables de entrada que luego se piden con funciones de tkinter.

7.2.1 Generador de aleatoriedad Gaussiano

La forma de obtener valores aleatorios segun la formula de Gauss

Formula Gauss

$$G_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \quad \mu \in \mathbb{R}, \sigma \in \mathbb{R}_{>0}.$$

Código

```
def generador_ruido(cantidad):
    global tipo_ruido, media_gaussiana_valor,
           desvio_gaussiano_valor, rayleigh_valor,
           exponencial_valor

    if tipo_ruido == "Gaussiano":
        return np.random.normal(media_gaussiana_valor,
                               desvio_gaussiano_valor, cantidad)
```

7.2.2 Generador de aleatoriedad Exponencial

La forma de obtener valores aleatorios segun el metodo Exponencial
Exponencial

$$f_{\lambda}(y) = 1 - e^{-\lambda y}$$

Código

```
elif tipo_ruido == "Exponencial":  
  
    return np.random.exponential(exponencial_valor,  
cantidad)
```

7.2.3 Generador de aleatoriedad Rayleigh

La forma de obtener valores aleatorios según el método Rayleigh

Rayleigh

$$n \sim \text{Rayleigh}(\sigma), \quad F_N(n) = 1 - \exp\left(-\frac{n^2}{2\sigma^2}\right), \quad n \geq 0$$

Código

```
elif tipo_ruido == "Rayleigh":  
  
    return np.random.rayleigh(rayleigh_valor, cantidad)
```

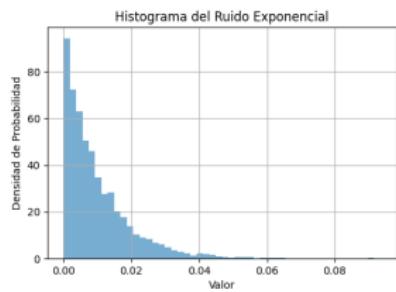
7.3 Generador de histograma para ruido

La forma de obtener valores aleatorios según el método Rayleigh

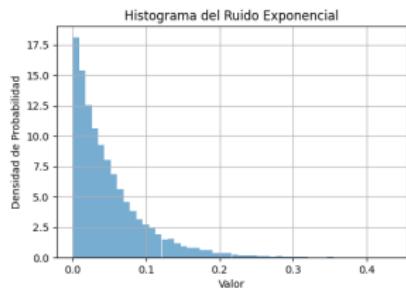
```
def mostrar_histograma(datos_ruido, tipo_ruido_str):  
  
    fig, ax = plt.subplots(figsize=(6, 4))  
    ax.hist(datos_ruido, bins=50, density=True, alpha=0.6)  
    ax.set_title(f'Histograma del Ruido {tipo_ruido_str}')  
    ax.set_xlabel('Valor')  
    ax.set_ylabel('Densidad de Probabilidad')  
    ax.grid(True)  
  
    ventana_hist = tk.Toplevel()  
    ventana_hist.title(f"Histograma de Ruido {tipo_ruido_str}")  
    canvas = FigureCanvasTkAgg(fig, master=ventana_hist)  
    canvas.draw()  
    canvas.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH,  
                               expand=1)
```

7.3 Comparación de histogramas

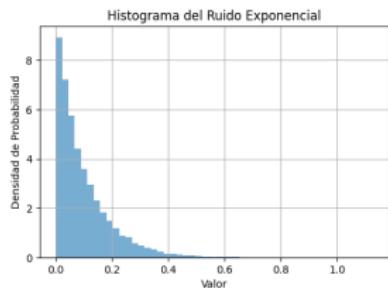
Exponencial 0.01



Exponencial 0.05

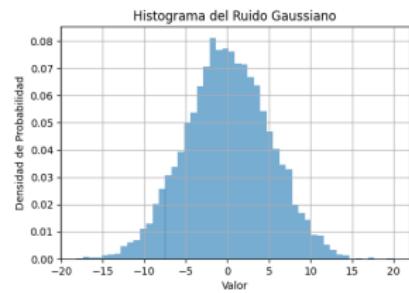


Exponencial 0.1

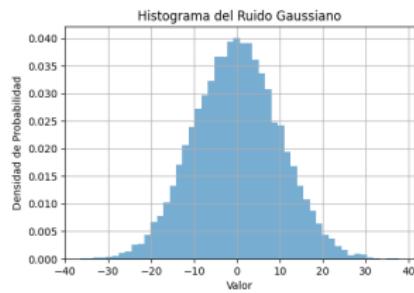


7.3 Comparación de histogramas

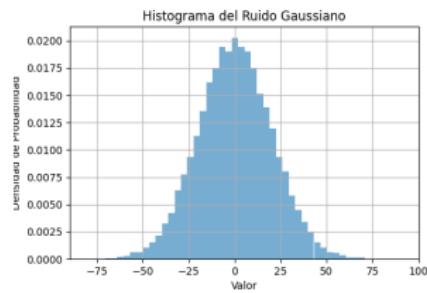
Media 0, Desvio = 5



Media 0, Desvio = 10

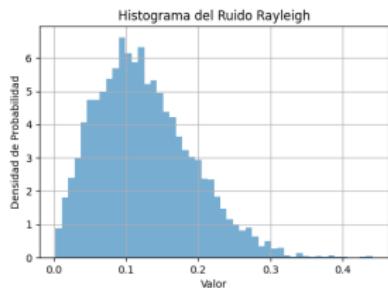


Media 0, Desvio = 20

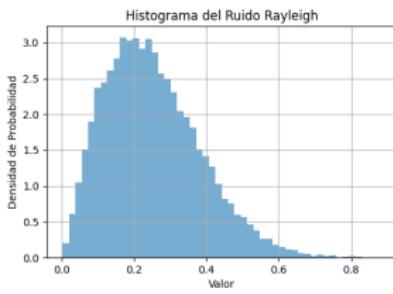


7.3 Comparación de histogramas

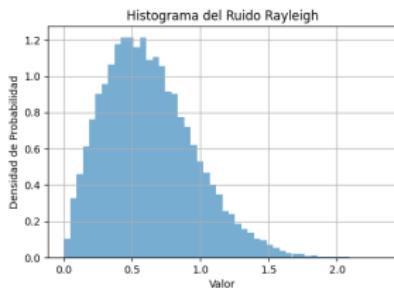
Rayleigh 0.1



Rayleigh 0.2



Rayleigh 0.5



Ejercicio 8: Contaminar imagen con ruido

8.1 Aplicar ruido a una imagen

Se aplica al píxel seleccionado al azar el ruido.

Ruido

Es la información no deseada que contamina la imagen.

Tipos de ruido

Ruido Aditivo: $Z = X + Y$

Ruido Multiplicativo: $Z = X \cdot Y$

Z es la imagen real, X es la imagen ideal e Y es el ruido puro que calculamos antes.

8.2 Variables globales

Variables globales para la función.

Variables globales

```
ruido = None  
porcentaje_ruido = None  
ruido_sal_pimineta = None
```

8.3.1 Inicio de la función

Definición de la función para aplicar ruido a la imagen.

Inicio de la función

```
def aplicar_ruido():
    global imagen_operativa, porcentaje_ruido,
    tipo_ruido, ruido
    grises = cv2.cvtColor(imagen_operativa.copy(), cv2.
    COLOR_BGR2GRAY)

    imagen_ruidosa = grises.copy().astype(np.float32)
    cantidad_ruido = int(imagen_ruidosa.size * (
    porcentaje_ruido / 100))
    ruido = generador_ruido(cantidad_ruido)

    mostrar_histograma(ruido, tipo_ruido)

    height, width = imagen_ruidosa.shape
    indices_pixels = np.random.choice(imagen_ruidosa.
    size, cantidad_ruido, replace=False)
```

8.3.2 Aplicación según el tipo de ruido

Utilización del ruido aditivo Gaussiano. Donde D son los indicespixels, e y_k es el k-esimo valor del ruido = generadoruido(...)

$$I_C(i, j) = \begin{cases} I(i, j) + y_k, & \text{si } (i, j) \in D \\ I(i, j), & \text{si } (i, j) \notin D \end{cases}$$

```
flat_img = imagen_ruidosa.ravel()

if tipo_ruido == "Gaussiano":
    for i in range(len(ruido)):
        flat_img[indices_pixels[i]] += ruido[i]
```

8.3.2 Aplicación según el tipo de ruido

Utilización del ruido multiplicativo Exponencial. Donde D son los indicespixels, e y_k es el k-esimo valor del ruido = generadoruido(...)

$$I_C(i, j) = \begin{cases} I(i, j) * y_k, & \text{si } (i, j) \in D \\ I(i, j), & \text{si } (i, j) \notin D \end{cases}$$

```
elif tipo_ruido == "Exponencial":  
    for i in range(len(ruido)):  
        flat_img[indices_pixels[i]] *= ruido[i]
```

8.3.2 Aplicación según el tipo de ruido

Utilización del ruido multiplicativo Rayleigh. Donde D son los indicespixels, e y_k es el k-esimo valor del ruido = generadoruido(...)

$$I_C(i, j) = \begin{cases} I(i, j) * y_k, & \text{si } (i, j) \in D \\ I(i, j), & \text{si } (i, j) \notin D \end{cases}$$

```
elif tipo_ruido == "Rayleigh":  
    for i in range(len(ruido)):  
        flat_img[indices_pixels[i]] *= ruido[i]
```

8.3.3 Continuación de la función

Definición de la función para aplicar ruido a la imagen.

Inicio de la función

```
imagen_ruidosa = ( (imagen_ruidosa - np.min(imagen_ruidosa)) / (np.max(imagen_ruidosa) - np.min(imagen_ruidosa)) ) * 255

imagen_ruidosa = imagen_ruidosa.astype(np.uint8)
## Aquí va Sal y Pimienta
imagen_operativa = imagen_ruidosa
mostrar_imagen(imagen_operativa, lblOutputImage)
```

8.4 Comparación de ruidos

Exponencial 0.01



Exponencial 0.05



Exponencial 0.1



8.4 Comparación de ruidos

Media 0, Desvio = 5 Media 0, Desvio = 10 Media 0, Desvio = 20



8.4 Comparación de ruidos

Media 0, Desvio = 5 Media 0, Desvio = 10 Media 0, Desvio = 20



8.4 Comparación de ruidos

Rayleigh 0.1



Rayleigh 0.2



Rayleigh 0.5



Ejercicio 9: Generador ruido Sal y Pimienta

9.1 Función Sal y Pimienta

Dentro de la función aplicarruido() se agrega un condicional al final antes de terminar la función

```
if tipo_ruido == "Sal y Pimienta":  
    p = ruido_sal_pimineta ## Es la elección la densidad  
    for fila in range(height):  
        for columna in range(width):  
            x = np.random.uniform(0, 1)  
            if x <= p:  
                imagen_ruidosa[fila, columna] = 0  
            elif x > 1 - p:  
                imagen_ruidosa[fila, columna] = 255  
    imagen_operativa = imagen_ruidosa  
    mostrar_imagen(imagen_operativa, lblOutputImage)
```

9.2 Comparación de densidades

Original



Densidad 0.01



Densidad 0.05

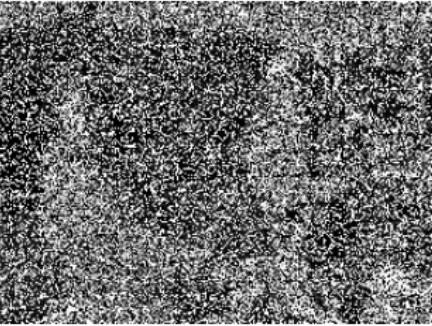


9.2 Comparación de densidades

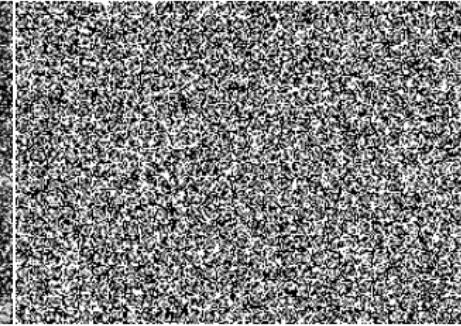
Densidad 0.1



Densidad 0.3



Densidad 0.5



Ejercicio 10: Aplicar Mascaras tamaño variables y cuadradas

10.1.1 Filtrado en el Dominio Espacial

El filtrado en el dominio espacial consiste en pasar un kernel, matriz o máscara sobre una imagen. Se reemplaza el valor del centro de la máscara con la suma de los valores originales de los píxeles multiplicados por los pesos de la máscara.

Código Python

```
def mascara():
    global imagen_operativa
    src = obtener_fuente_operacion()
    gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
    nueva_imagen = np.zeros_like(gray, dtype=np.float32)
    altura, ancho = gray.shape
```

10.1.2 Filtrado en el Dominio Espacial

Continuación de la función mascara.

Código Python

```
for fila in range(altura):
    for columna in range(ancho):
        nueva_imagen[fila, columna] = aplicar_mascara(gray
, [fila, columna], mascara_kernel, tipo_kernel_aplicar)

imagen_operativa = ((nueva_imagen - np.min(nueva_imagen)) /
(np.max(nueva_imagen) - np.min(nueva_imagen))) * 255

imagen_operativa = imagen_operativa.astype(np.uint8)
mostrar_imagen(imagen_operativa, lblOutputImage)
```

10.2.1 Aplicación según máscara

Parte de cómo se aplica el kernel de la media. Se establece como promedio de los píxeles que rodean al núcleo del kernel.

Código Python

```
def aplicar_mascara(imagen, pixel, mascara, tipo):
    ancho = int(mascara.shape[0] // 2)
    nuevo_pixel = 0
    if tipo == "Media":
        for i in range(-ancho, ancho + 1):
            for j in range(-ancho, ancho + 1):
                x, y = pixel[0] + i, pixel[1] + j
                if 0 <= x < imagen.shape[0] and 0 <= y < imagen.
shape[1]:
                    nuevo_pixel += imagen[x, y] * mascara[i + ancho, j
+ ancho]

    nuevo_pixel = nuevo_pixel / (mascara.shape[0] * mascara.
shape[1])
```

10.2.1 Aplicación según máscara

El filtro Gaussiano devuelve un promedio pesado de cada pixel en la vecindad, con el mayor peso dado al valor del pixel central.

Código Python

```
elif tipo == "Gaussiano":  
    for i in range(-ancho, ancho + 1):  
        for j in range(-ancho, ancho + 1):  
            x, y = pixel[0] + i, pixel[1] + j  
  
            if 0 <= x < imagen.shape[0] and 0 <= y < imagen.shape[1]:  
  
                nuevo_pixel += imagen[x, y] * mascara[i + ancho, j + ancho]  
suma = mascara.sum()  
if suma != 0:  
    nuevo_pixel = nuevo_pixel / suma
```

10.2.1 Aplicación según máscara

El filtro de realce destaca los detalles de una imagen o intensifica aquellos que han sido borroneados.

Código Python

```
elif tipo == "Realce":  
    for i in range(-ancho, ancho + 1):  
        for j in range(-ancho, ancho + 1):  
            x, y = pixel[0] + i, pixel[1] + j  
  
            if 0 <= x < imagen.shape[0] and 0 <= y < imagen.shape[1]:  
  
                nuevo_pixel += imagen[x, y] * mascara[i + ancho, j + ancho]  
nuevo_pixel = nuevo_pixel / (mascara.shape[0]*mascara.shape[1])
```

10.2.1 Aplicación según máscara

El kernel de la Mediana consiste en ordenarlos y luego tomar el valor que esta en el medio

Código Python

```
elif tipo == "Mediana":  
    vecinos = []  
    for i in range(-ancho, ancho + 1):  
        for j in range(-ancho, ancho + 1):  
            x, y = pixel[0] + i, pixel[1] + j  
            if 0 <= x < imagen.shape[0] and 0 <= y < imagen.  
shape[1]:  
                vecinos.append(imagen[x, y])  
    vecinos.sort()  
    nuevo_pixel = vecinos[len(vecinos)//2]
```

10.2.1 Aplicación según máscara

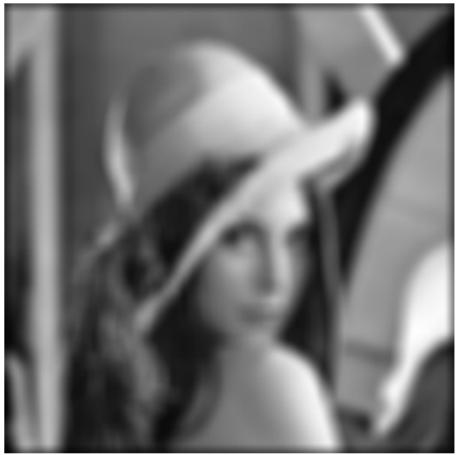
El kernel de la Mediana Ponderada consiste en generar el pixel hallando la mediana de los valores del entorno del pixel, repetidos tantas veces como indique una determinada matriz de números enteros.

Código Python

```
elif tipo == "Mediana Ponderada":  
    vecinos = []  
    for i in range(-ancho, ancho + 1):  
        for j in range(-ancho, ancho + 1):  
            x, y = pixel[0] + i, pixel[1] + j  
            if 0 <= x < imagen.shape[0] and 0 <= y < imagen.  
shape[1]:  
                for k in range(int((mascara[i + ancho, j + ancho]))  
):  
  
                    vecinos.append(imagen[x, y])  
    vecinos.sort()  
    nuevo_pixel = vecinos[len(vecinos)//2]
```

10.3 Comparación de filtros

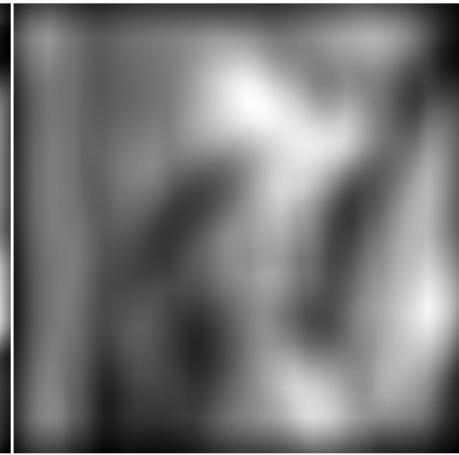
Gauss 5 desvio



Gauss 10 desvio



Gauss 20 desvio



10.3 Comparación de filtros

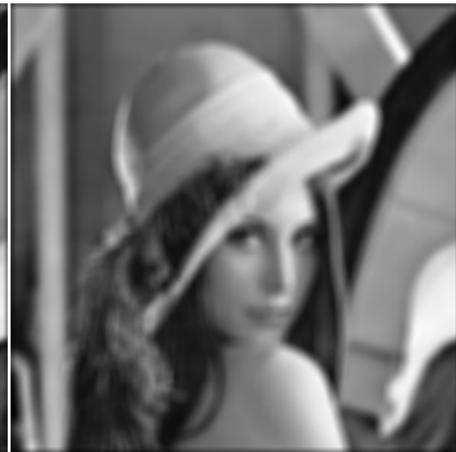
Media 3x3



Media 5x5



Media 7x7



10.3 Comparación de filtros

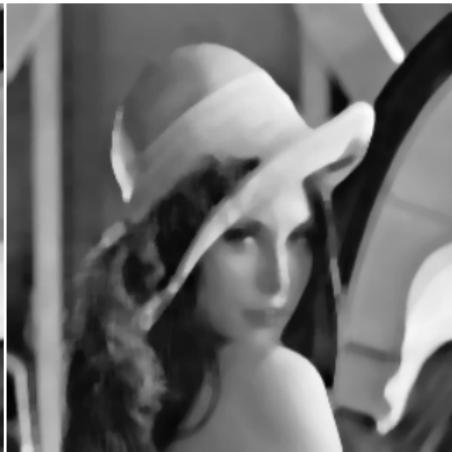
Mediana 3x3



Mediana 5x5



Mediana 7x7



10.3 Comparación de filtros

Mediana ponderada 3x3 Mediana ponderada 5x5 Mediana ponderada 7x7



10.3 Comparación de filtros

Realce 3x3



Realce 5x5



Realce 7x7



Ejercicio 11: Aplicar Mascaras para imágenes contaminadas

11 Imágenes con ruido

Rayleigh 0.2



Rayleigh 0.5



11 Imágenes con ruido

Media 0, Desvio = 10



Media 0, Desvio = 20



11.2 Mascaras aplicadas a Rayleigh 0.2

Gauss desvio 1



Gauss desvio 10



Media 3x3



11 Mascaras aplicadas a Rayleigh 0.2

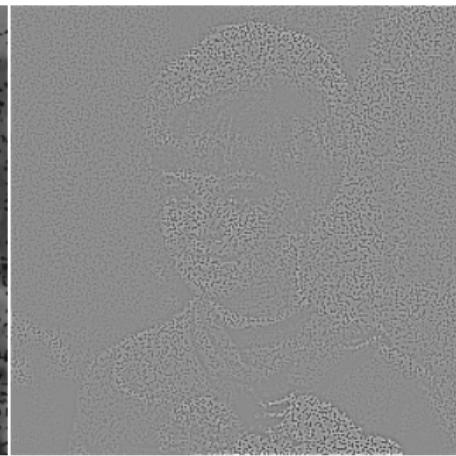
Mediana



Mediana Ponderada



Realce



11 Mascaras aplicadas a Rayleigh 0.5

Gauss desvio: 1



Gauss desvio: 10



Media 3x3



11 Mascaras aplicadas a Rayleigh 0.5

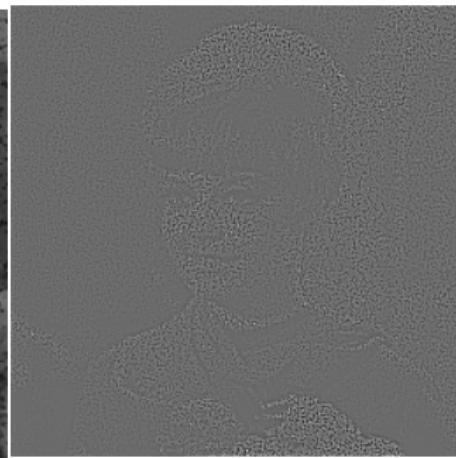
Mediana



Mediana ponderada



Realce



11 Mascaras aplicadas a Gauss con desvio:10

Gauss desvio: 1



Gauss desvio: 10



Media 3x3



11 Mascaras aplicadas a Gauss con desvio:10

Mediana



Mediana ponderada



Realce



11 Mascaras aplicadas a Gauss con desvio: 20

Gauss desvio: 1



Gauss desvio: 10



Media 3x3



11 Mascaras aplicadas a Gauss con desvio: 20

Mediana



Mediana ponderada



Realce



Ejercicio 12: Aplicar filtro de media y mediana a Sal y Pimienta

11 Mascaras aplicadas a Sal y pimienta densidad: 0.01

Mediana 3x3



Mediana ponderada 3x3



11 Mascaras aplicadas a Sal y pimienta densidad: 0.01

Mediana 5x5



Mediana ponderada 5x5



11 Mascaras aplicadas a Sal y pimienta densidad: 0.05

Mediana 3x3



Mediana ponderada 3x3



11 Mascaras aplicadas a Sal y pimienta densidad: 0.05

Mediana 5x5



Mediana ponderada 5x5



11 Mascaras aplicadas a Sal y pimienta densidad: 0.1

Mediana 3x3



Mediana ponderada 3x3



11 Mascaras aplicadas a Sal y pimienta densidad: 0.1

Mediana 5x5

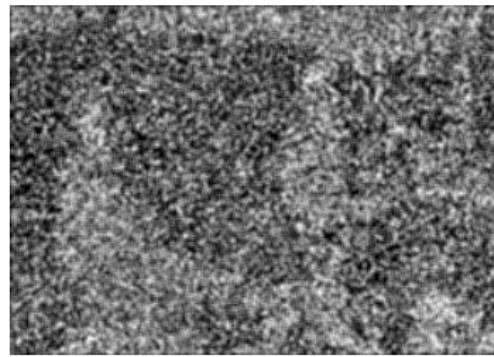


Mediana ponderada 5x5

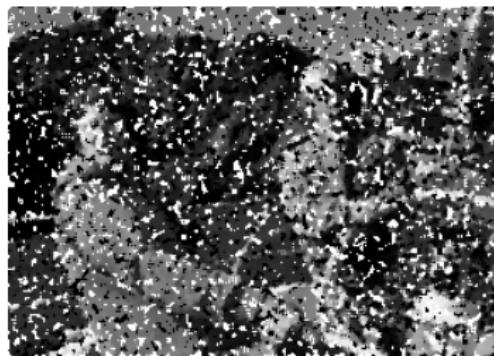


11 Mascaras aplicadas a Sal y pimienta densidad: 0.3

Mediana 3x3

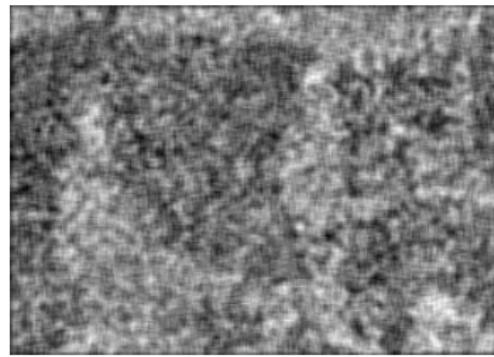


Mediana ponderada 3x3



11 Mascaras aplicadas a Sal y pimienta densidad: 0.3

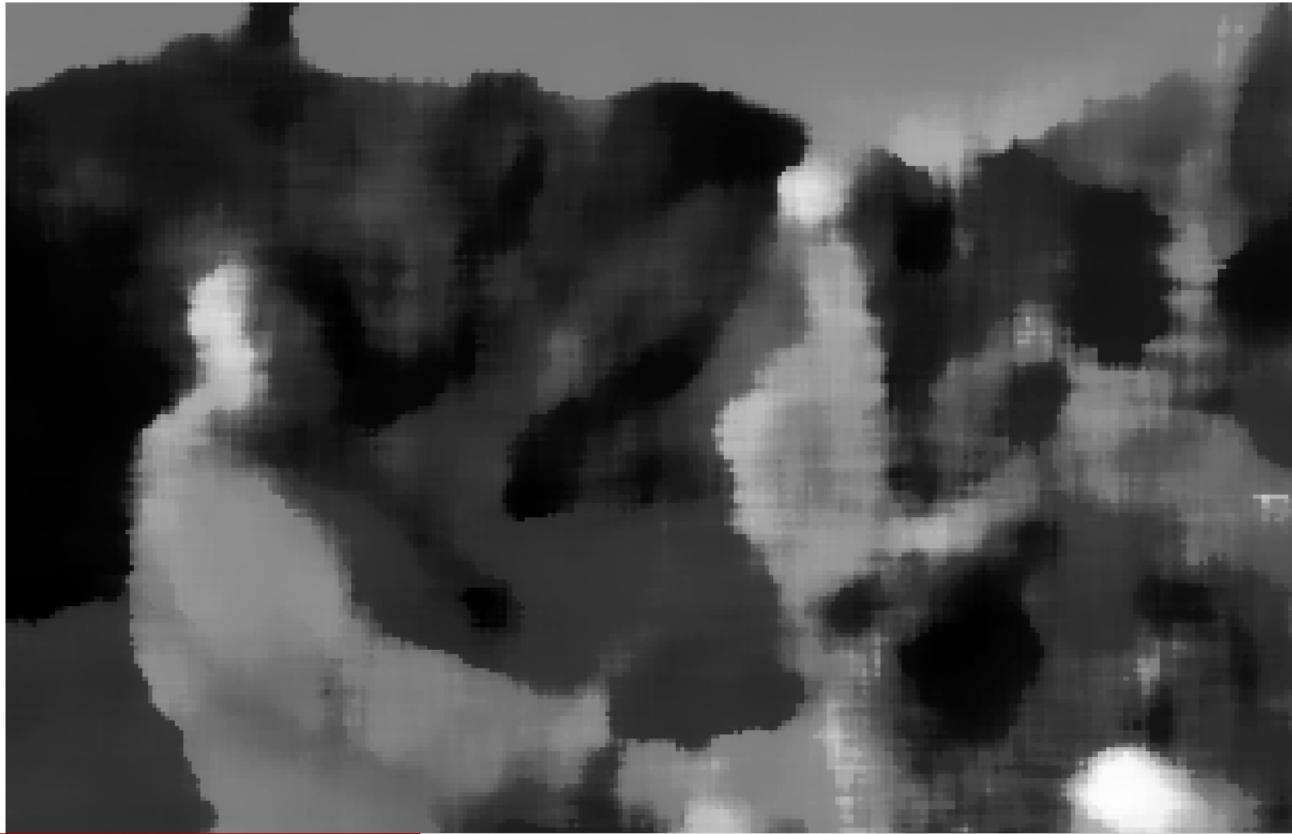
Mediana 5x5



Mediana ponderada 5x5



Mediana 13x13



Espero que haya gustado!!

