

# Portfolio Vision Computer - Javier González Barreda

Enlace a hoja de cálculo de ejercicios	<a href="https://docs.google.com/spreadsheets/d/1NMjHDUi6VeJz6V75HCIDqCKuzQmrJiUKsGOoYvR8kTc/edit?usp=sharing">https://docs.google.com/spreadsheets/d/1NMjHDUi6VeJz6V75HCIDqCKuzQmrJiUKsGOoYvR8kTc/edit?usp=sharing</a>
Enlace scripts de python de las prácticas	<a href="https://drive.google.com/drive/folders/13ikgiApSSFrB2G5g4EnVTI--mhinVkr?usp=sharing">https://drive.google.com/drive/folders/13ikgiApSSFrB2G5g4EnVTI--mhinVkr?usp=sharing</a>

## P6 - Reflexiones de aprendizaje - 13/12/2023

El objetivo de esta práctica es la estimación del flujo óptico, un campo importante para aplicaciones donde la información visual sufre cambios dinámicos con el tiempo, ya que permite comprender el movimiento y la dinámica dentro de una escena. El laboratorio explora específicamente la implementación del algoritmo clásico de Lucas-Kanade, un método fundamental para el cálculo del flujo óptico.

En concreto en esta práctica se emplea el algoritmo de Lucas-Kanade, el cual destaca por su simplicidad y eficacia en determinados escenarios. El algoritmo, basado en información de gradiente local, supone que el flujo es esencialmente constante dentro de un vecindario local. Esto lo hace muy adecuado para escenarios donde el movimiento es relativamente suave y puede aproximarse mediante un modelo lineal. En este laboratorio se nos han dado las pautas a seguir para comprender e implementar este algoritmo clásico, lo que nos ha proporcionado una base sólida para comprender los principios del cálculo del flujo óptico.

## P6 - Ejercicios 1, 2, 3 y 4 - 13/12/2023

En estos ejercicios se trata de completar la función `optical_flow()`, para calcular el flujo óptico ( $u$ ,  $v$ ) entre el par de imágenes ( $I_1$ ,  $I_2$ ) con el algoritmo de Lucas-Kanade (LK), utilizando ventanas cuadradas locales de longitud de lado  $w$  (tamaño\_ventana) y  $\tau$  (tau) como valor mínimo de los valores propios de la matriz  $A^T A$ .

En primer lugar, suavizamos las imágenes de entrada con un filtro gaussiano y luego calculamos los gradientes,  $I_x$ ,  $I_y$  e  $I_t$  en las imágenes suavizadas. Para ello, definimos los tres núcleos correspondientes para las direcciones  $x$ ,  $y$  y  $t$ , y luego aplicamos convoluciones 2D.

En el método LK, para cada ventana local de  $n$  píxeles, tenemos que  $Au = -b$ , donde  $A$  es la matriz  $nx2$  de los  $n$  gradientes espaciales, y  $b$  es el vector  $nx1$  de derivadas temporales. Para conocer el flujo óptico para esa ventana tenemos que  $u = (A^T A)^{-1} A^T b$ .

Finalmente, usamos `np.linalg.lstsq(C,d)` para calcular la solución de mínimos cuadrados  $x$  para  $Cx = d$ . Donde la matriz  $C$  es  $A^T A$  y el vector  $d$  es  $A^T b$ .

Sólo podemos calcular el flujo óptico en aquellas ubicaciones de la imagen donde  $A^T A$  está bien condicionado. Por tanto, antes debemos verificar esto comprobando que el valor propio más pequeño para  $A^T A$  es mayor que  $\tau$ , o que  $A^T A$  tiene rango 2, empleando `np.linalg.eigvals(AtA)` y `np.linalg.matrix_rank(AtA)`, respectivamente.

Código:

```
def optical_flow(I1g, I2g, window_size, tau=1e-2, bDisplay=False):

    kernel_x = np.array([[[-1., 1.], [-1., 1.]]])
    kernel_y = np.array([[[-1., -1.], [1., 1.]]])
    kernel_t = np.ones((4, 4)) / 16

    w = int(window_size/2) # window_size is odd, all the pixels with offset in between [-w, w] are inside the window

    I1g = gaussian(I1g, sigma=5, truncate=1/5)
    I2g = gaussian(I2g, sigma=5, truncate=1/5)

    # Implement Lucas-Kanade
    # for each image point, calculate I_x, I_y, I_t
    mode = 'same'
    fx = signal.convolve2d(I1g, kernel_x, boundary='symm', mode=mode)
    fy = signal.convolve2d(I1g, kernel_y, boundary='symm', mode=mode)
    ft = signal.convolve2d(I2g, kernel_t, boundary='symm', mode=mode) - signal.convolve2d(I1g, kernel_t, boundary='symm', mode=mode)
    if bDisplay:
        for f in [fx,fy,ft]:
            plt.imshow(f,cmap='gray')
            plt.show(block=True)

    u = np.zeros(I1g.shape)
    v = np.zeros(I1g.shape)

    # iterate for each image window of size window_size * window_size
    M,N = I1g.shape
    for i in range(w, M-w):
        for j in range(w, N-w):
            Ix = fx[i-w:i+w+1, j-w:j+w+1].flatten()
            Iy = fy[i-w:i+w+1, j-w:j+w+1].flatten()
            It = ft[i-w:i+w+1, j-w:j+w+1].flatten()

            AtA = np.matmul(np.vstack((Ix, Iy)), np.vstack((Ix, Iy)).transpose())
            Atb = np.matmul(np.vstack((Ix, Iy)), -np.reshape(It, (It.shape[0], 1)))

            if np.min(np.linalg.eigvals(AtA)) > tau or np.linalg.matrix_rank(AtA) == 2:
                nu, residuals, rank, s = np.linalg.lstsq(AtA, Atb, rcond=None)
                u[i,j]=nu[0]
                v[i,j]=nu[1]

    return u,v
```

## P5 - Reflexiones de aprendizaje - 11/12/2023

Las actividades propuestas en este laboratorio tienen como objetivo proporcionar una comprensión introductoria de la segmentación de imágenes utilizando imágenes simples adquiridas en condiciones controladas.

El objetivo principal de la segmentación de imágenes es mejorar la comprensión de la escena. Al dividir una imagen en segmentos significativos, obtenemos información sobre los diferentes componentes dentro de una escena y esta práctica nos permite descifrar información visual compleja e identificar distintas regiones de interés.

Además, una vez que la imagen se divide en segmentos, el recuento se vuelve factible, lo cual es muy útil en escenarios donde es necesario determinar la cantidad de objetos o entidades específicas en una imagen.

La segmentación también permite caracterizar diferentes regiones en función de diversos atributos como el color, la textura o la forma.

De esta forma, en esta práctica se nos va guiando paso a paso a través de estas técnicas de segmentación preparando el escenario para realizar tareas de clasificación, es decir, una vez identificadas las regiones, se pueden clasificar según criterios predefinidos.

Trabajar en condiciones relativamente controladas en este laboratorio es una estrategia que proporciona una base para comprender los conceptos esenciales sin las complejidades asociadas con los escenarios no controlados del mundo real.

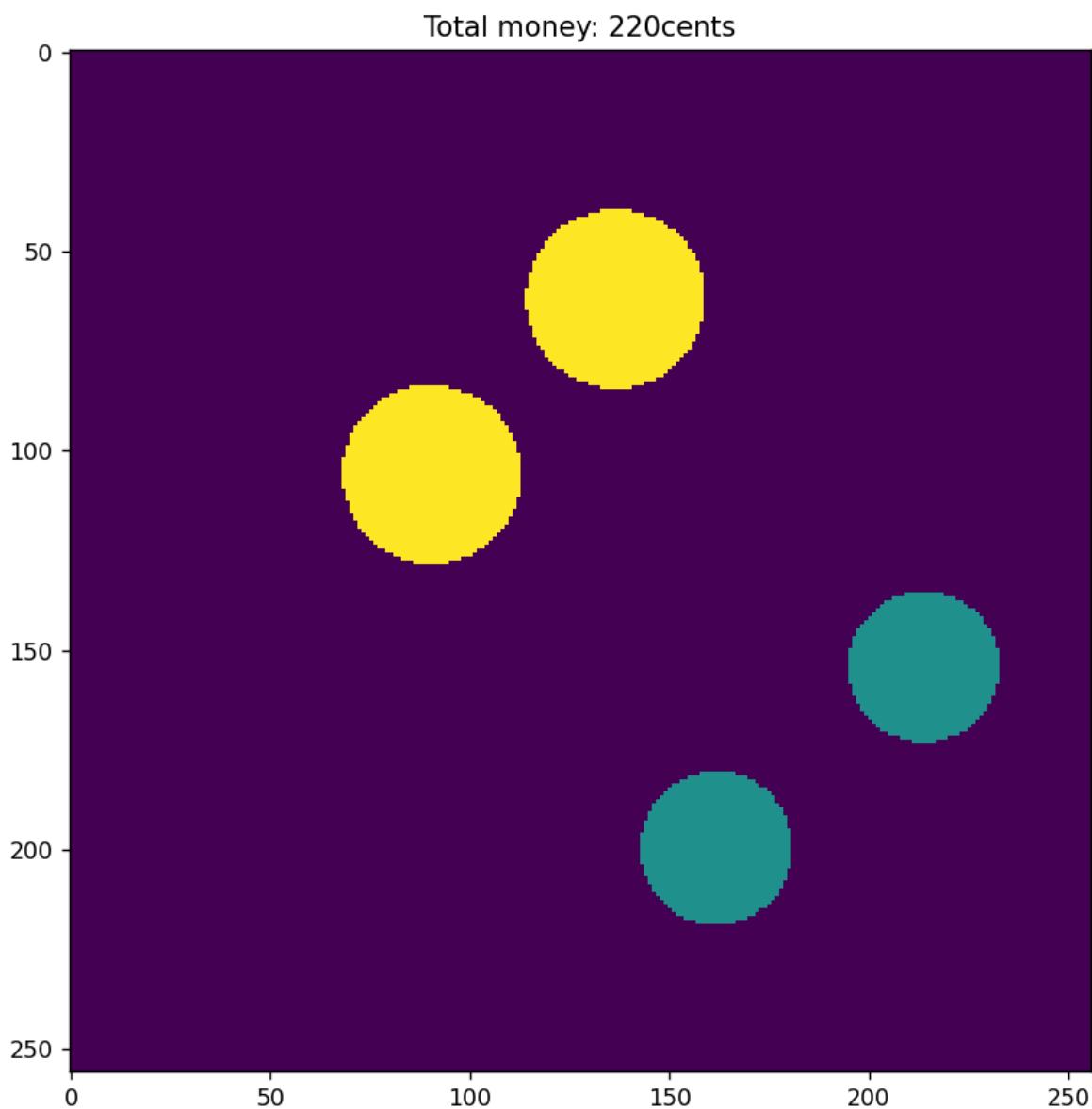
#### P5- Ejercicio 4 - 11/12/2023

En este ejercicio, una vez se ha detectado cuantas monedas se tienen de cada tipo, simplemente se añade el siguiente cálculo para obtener el total de dinero:

```
total_money = euro_coin * 100 + cent_coin * 10
```

Resultados:

```
Total coins detected: 4
Total 1 euro coins detected: 2
Total 10 cents coins detected: 2
Total money: 220 cents
```



### P5 - Ejercicio 3 - 11/12/2023

Utilizando el resultado del Ejercicio 2 como punto de partida, se ha encontrado qué regiones corresponden a cada tipo de moneda (1 euro, 10 céntimos) sabiendo que:

- Las imágenes de las monedas se tomaron con 50 píxeles por pulgada (1 pulgada = 2,54 cm).
- Los diámetros de las monedas de 1 euro y 10 céntimos son 23 mm y 19,5 mm respectivamente

Código:

```

def reportPropertiesRegions(im, labelIm, title):
    print("* * " + title)
    regions = measure.regionprops(labelIm)
    coin_count = 0 # Counter for the number of coins
    euro_coin = 0
    cent_coin = 0

    # Create a new image with colors for circular and non-circular regions
    result_image = np.zeros_like(im, dtype=np.uint8)

    for r, region in enumerate(regions):
        print("Region", r + 1, "(label", str(region.label) + ")")
        print("\t area:", region.area)
        print("\t perimeter:", round(region.perimeter, 1))

        # Determine coin type based on circularity and diameter
        diameter_pixels = np.sqrt(region.area / np.pi) * 2 # Diameter in pixels
        diameter_inches = diameter_pixels / 50.0 # Convert pixels to inches
        diameter_mm = diameter_inches * 25.4 # Convert inches to mm

        print("\t diameter:", round(diameter_mm, 1))

        # Check circularity based on the ratio of perimeter to the square root of area
        circularity = (4 * np.pi * region.area) / (region.perimeter ** 2)
        print("\t circularity:", round(circularity, 3))

        # If circularity is above the threshold, consider it a coin
        if circularity >= 0.9 and circularity <= 1.1:
            print("\t Coin detected!")
            coin_count += 1

            # Set color based on circularity
            #color = 255 #(255, 0, 0)

            if 22 < diameter_mm < 24: # Assuming 1 euro coin diameter = 23 mm
                euro_coin += 1
                color = 255

            elif 18.5 < diameter_mm < 20.5: # Assuming 10 cents coin diameter = 19.5 mm
                cent_coin += 1
                color = 128

            else:
                color = 0 #(0, 255, 0)

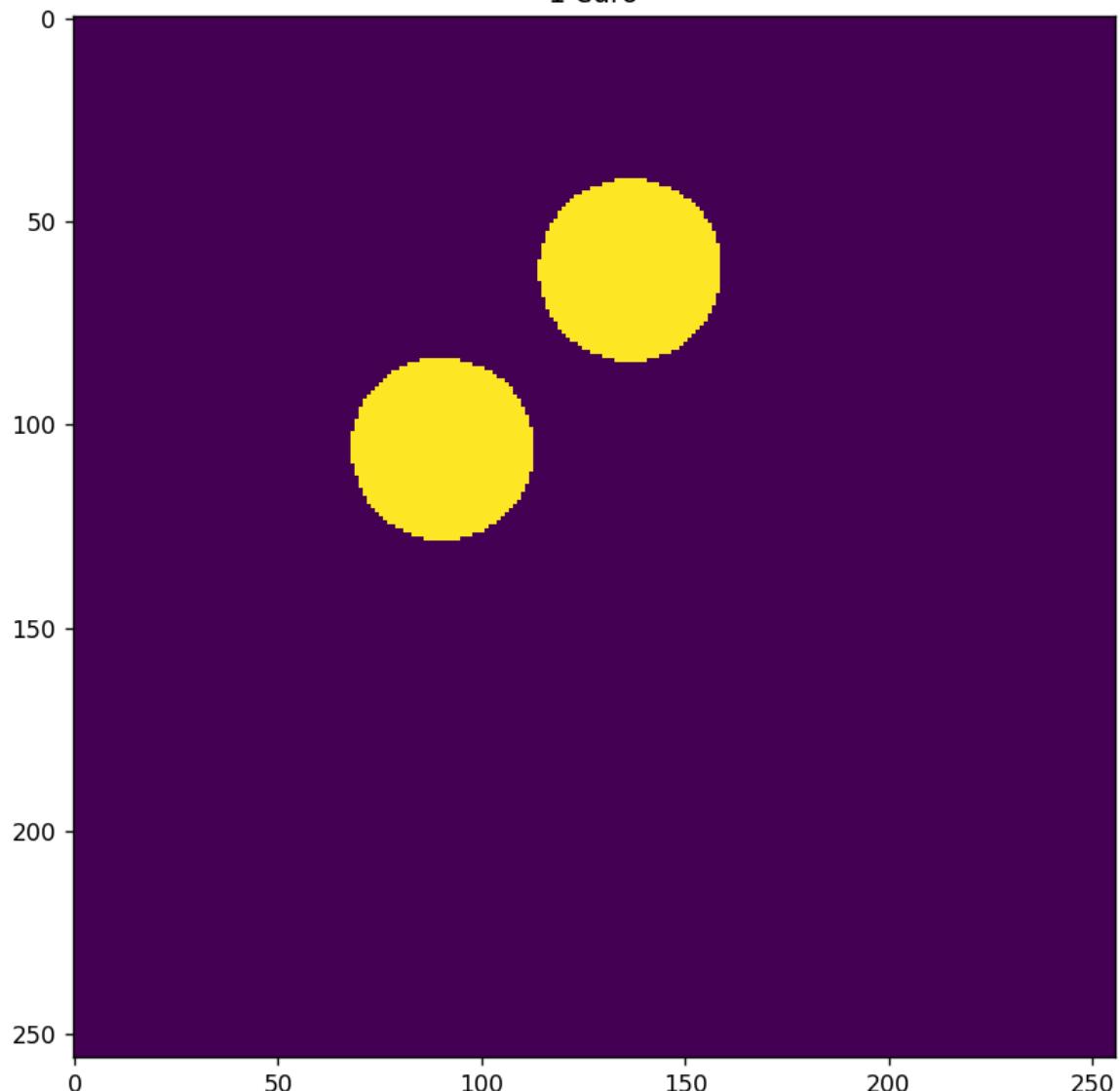
            # Assign color to the region in the result image
            result_image[labelIm == region.label] = color

    plotCoins(result_image)
    print("Total coins detected:", coin_count)
    print("Total 1 euro coins detected:", euro_coin)
    print("Total 10 cents coins detected:", cent_coin)

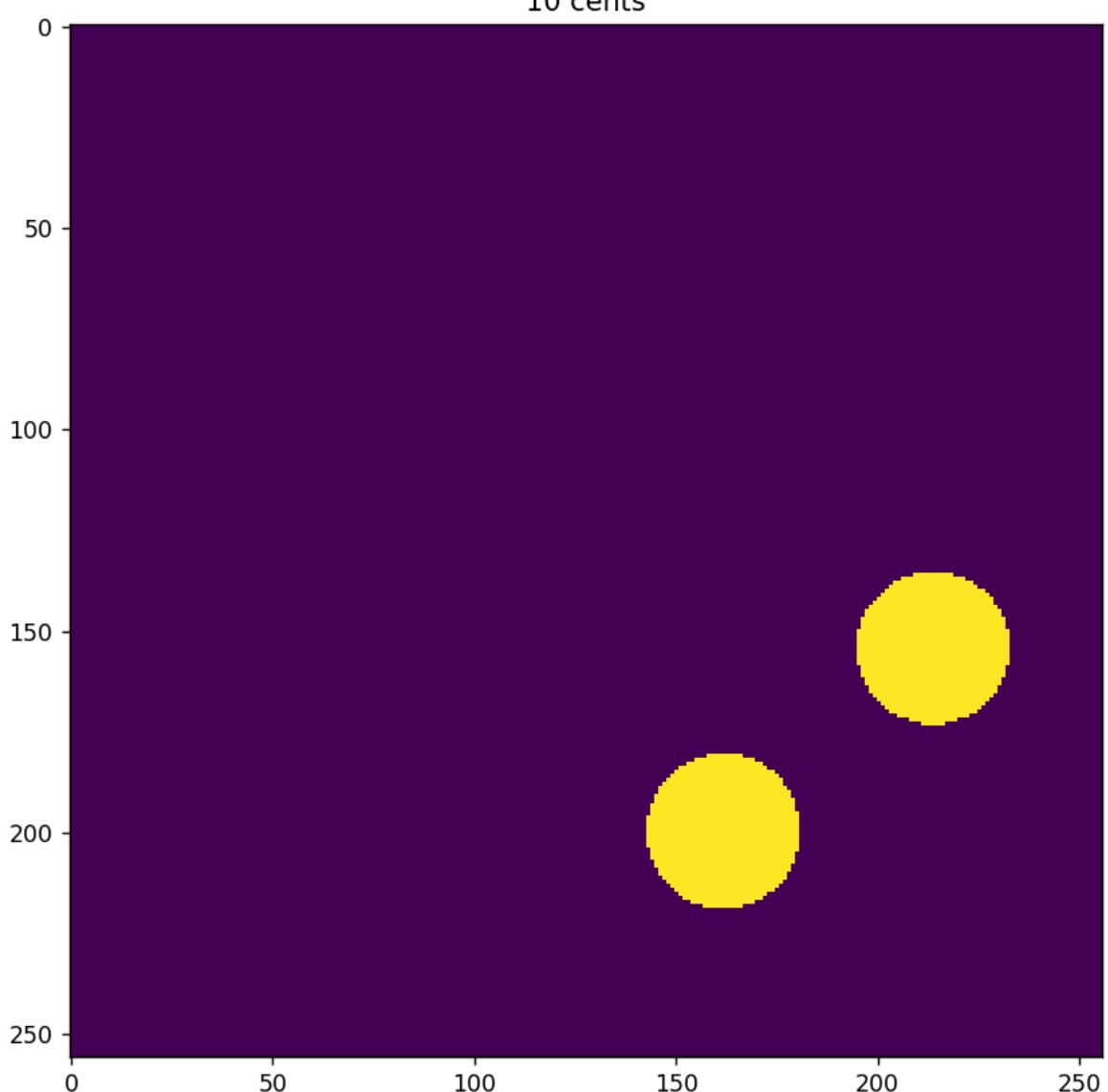
```

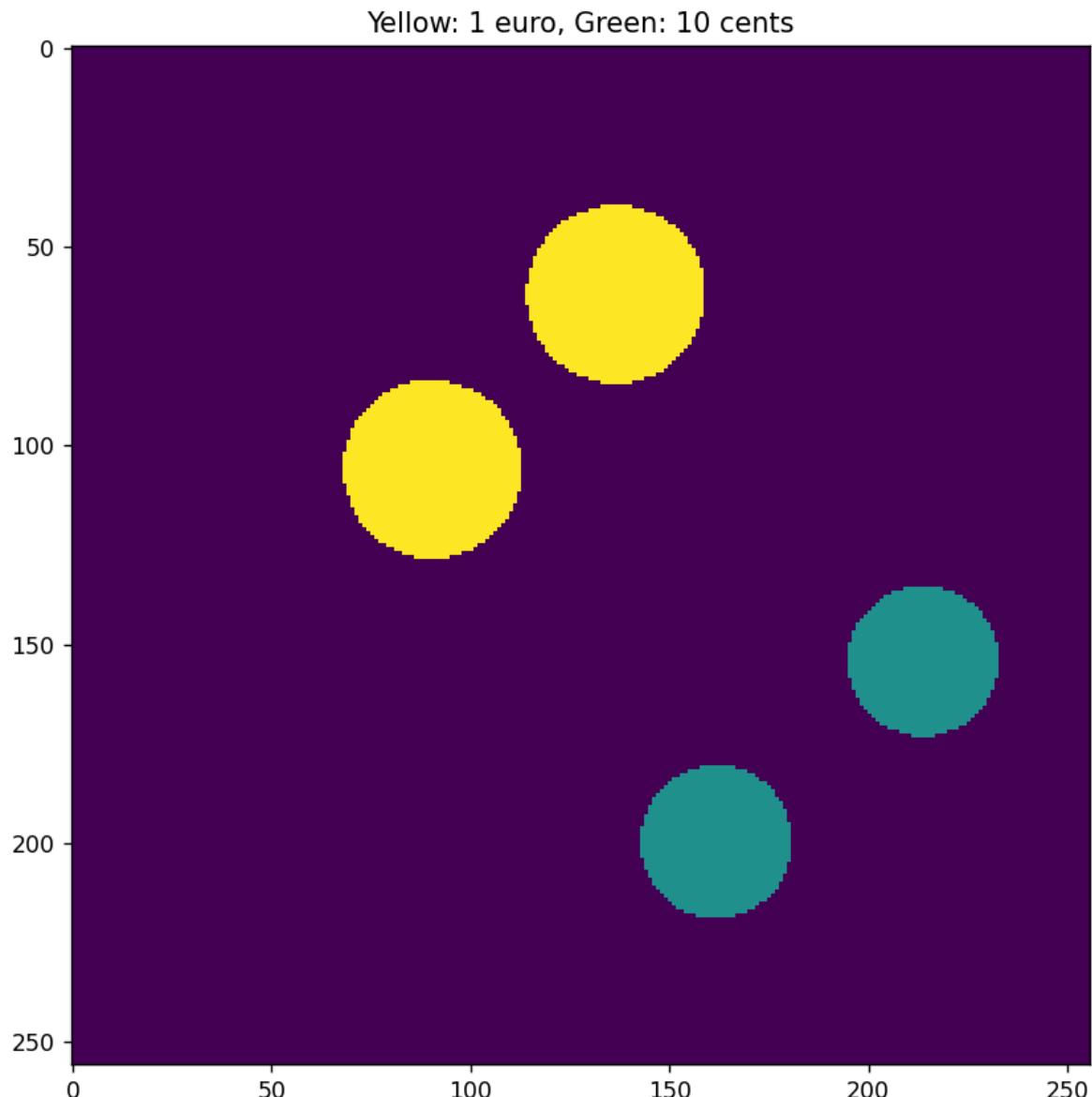
Resultados:

1 euro



10 cents





### P5 - Ejercicio 2 - 11/12/2023

A partir de la imagen anterior segmentada, se usa el perímetro y el área de las regiones encontradas para identificar aquellas que tienen una forma circular aproximada (que, en nuestro caso, corresponde a monedas). Finalmente se representa el resultado usando un color para la región circular y otro para las que no lo son y se cuentan la cantidad de monedas.

Código:

```

def reportPropertiesRegions(im, labelIm, title):
    print("* * " + title)
    regions = measure.regionprops(labelIm)
    coin_count = 0 # Counter for the number of coins

    # Create a new image with colors for circular and non-circular regions
    result_image = np.zeros_like(im, dtype=np.uint8)

    for r, region in enumerate(regions):
        print("Region", r + 1, "(label", str(region.label) + ")")
        print("\t area:", region.area)
        print("\t perimeter:", round(region.perimeter, 1))

        # Check circularity based on the ratio of perimeter to the square root of area
        circularity = (4 * np.pi * region.area) / (region.perimeter ** 2)
        print("\t circularity:", round(circularity, 3))

        # If circularity is above the threshold, consider it a coin
        if circularity >= 0.9 and circularity <= 1.1:
            print("\t Coin detected!")
            coin_count += 1

            # Set color based on circularity
            color = 255 #(255, 0, 0)

        else:
            color = 0 #(0, 255, 0)

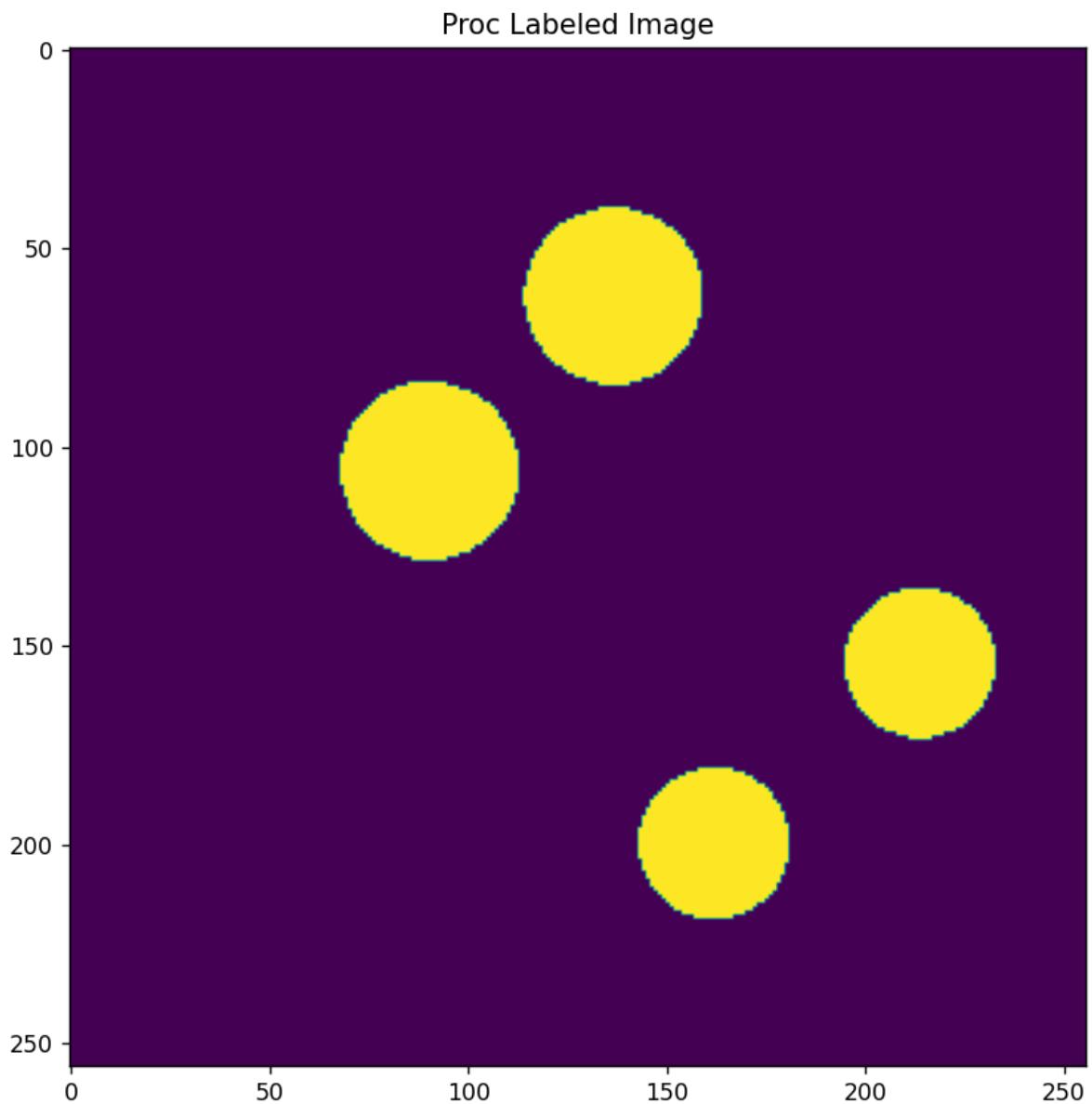
        # Assign color to the region in the result image
        result_image[labelIm == region.label] = color

    plotCoins(result_image)

    print("Total coins detected:", coin_count)

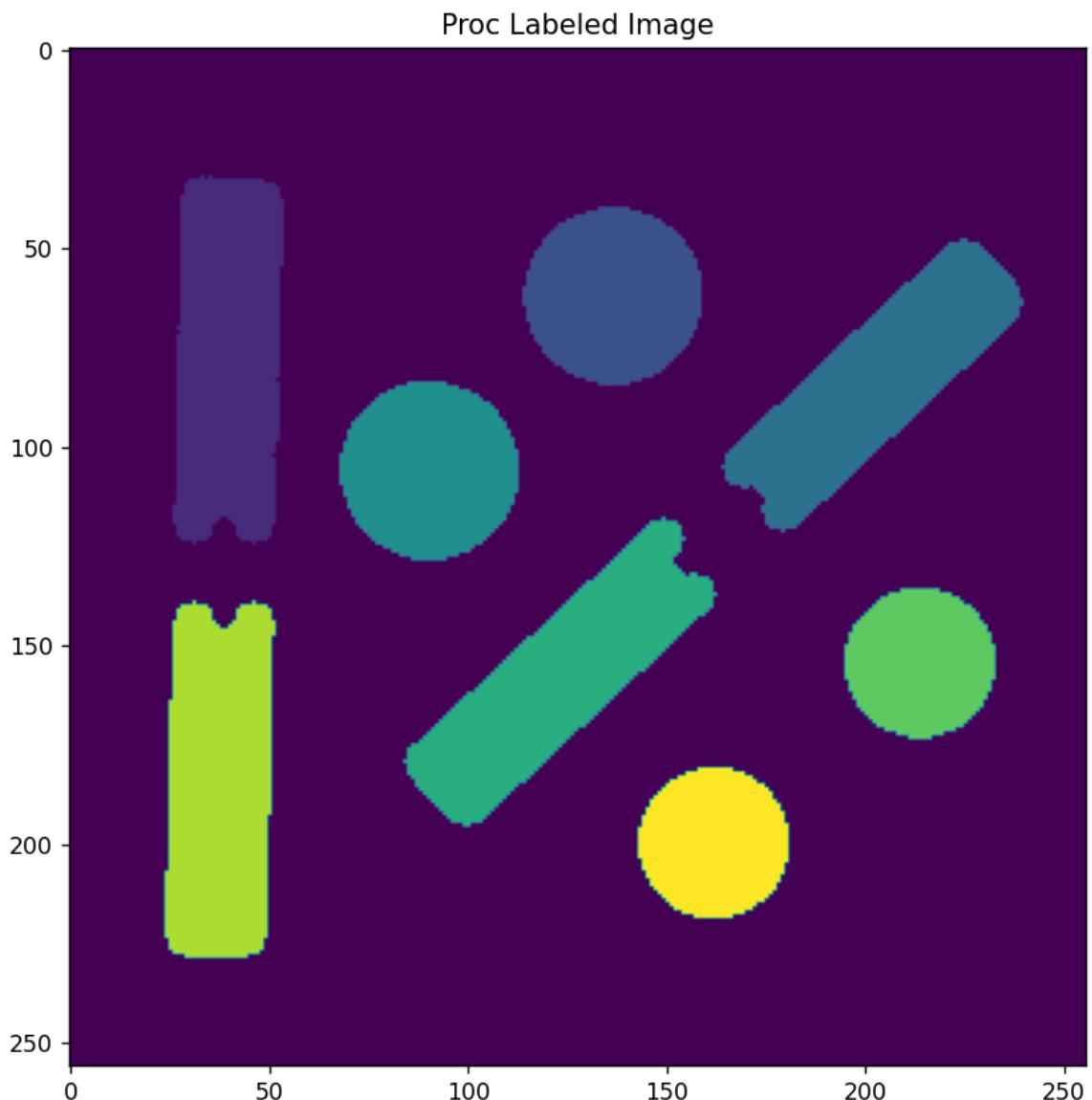
```

Resultados:



**P5 - Ejercicio 1 - 11/12/2023**

En este ejercicio se pide realizar una segmentación de la imagen de monedas, para lo cual realizamos el procedimiento anterior. Obteniendo el siguiente resultado luego de procesar la imagen binarizada.



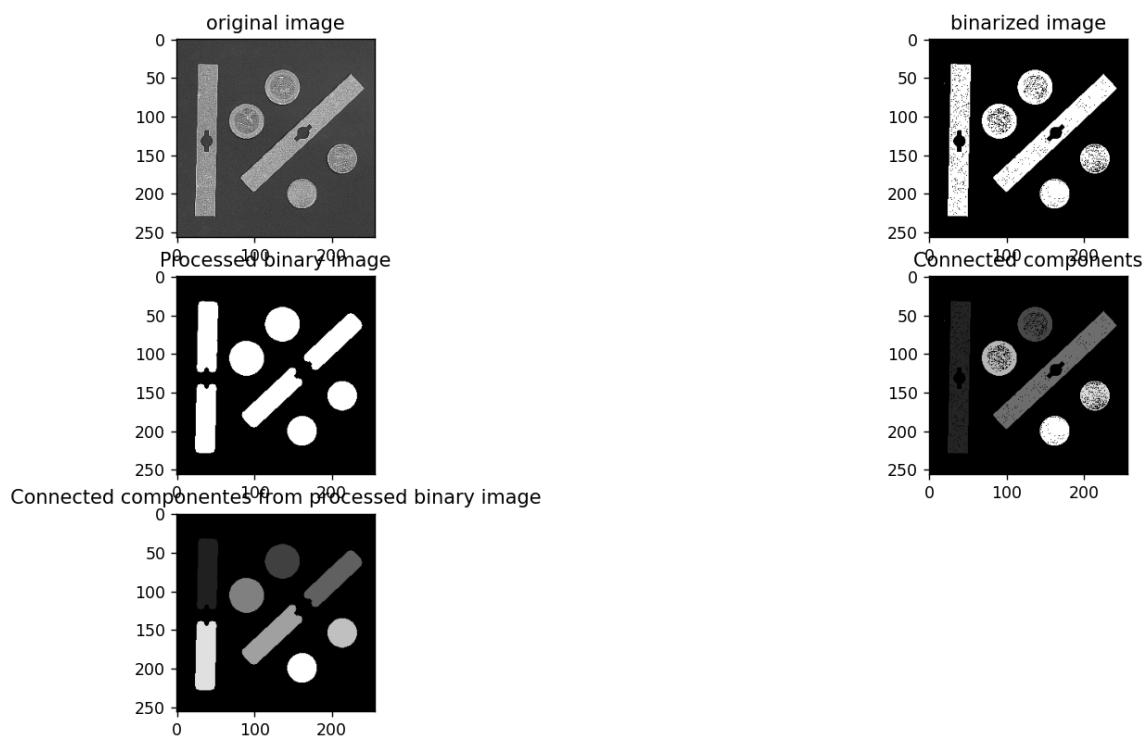
### P5- Etiquetado de regiones (intro) - 11/12/2023

Si mostramos una imagen binarizada, podemos percibir diferentes “regiones” y la forma tradicional de identificarlas automáticamente es mediante un algoritmo de etiquetado de componentes conectados.

En el código proporcionado, el algoritmo de etiquetado de componentes conectados se aplica mediante la función `measure.label(binIm, background=0)`, que forma parte del módulo `skimage.measure`. El parámetro `background` se establece en 0, lo que indica que 0 representa el fondo en la imagen binaria. Si cambia a 1, significa que está especificando que 1 representa el fondo en la imagen binaria. Esto da lugar a una interpretación diferente de la imagen binaria que afecta al etiquetado de los componentes conectados, dando como resultado una segmentación incorrecta de los objetos de la imagen.

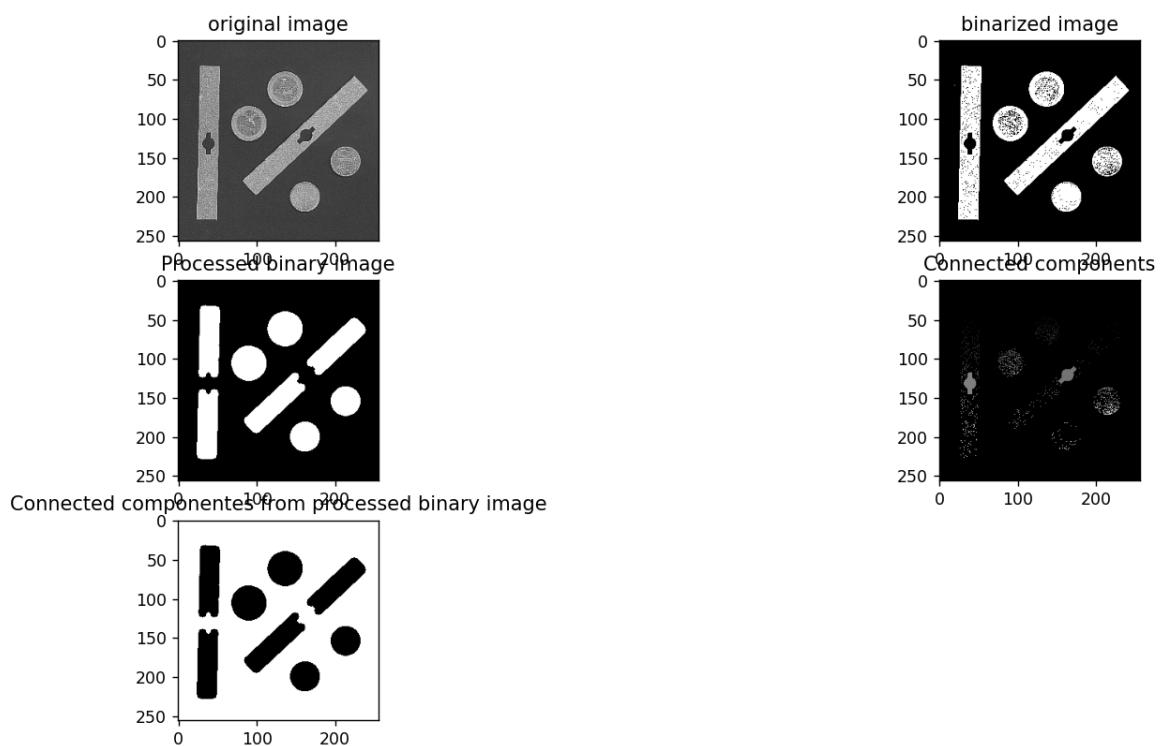
Background=0

Labelling connected components

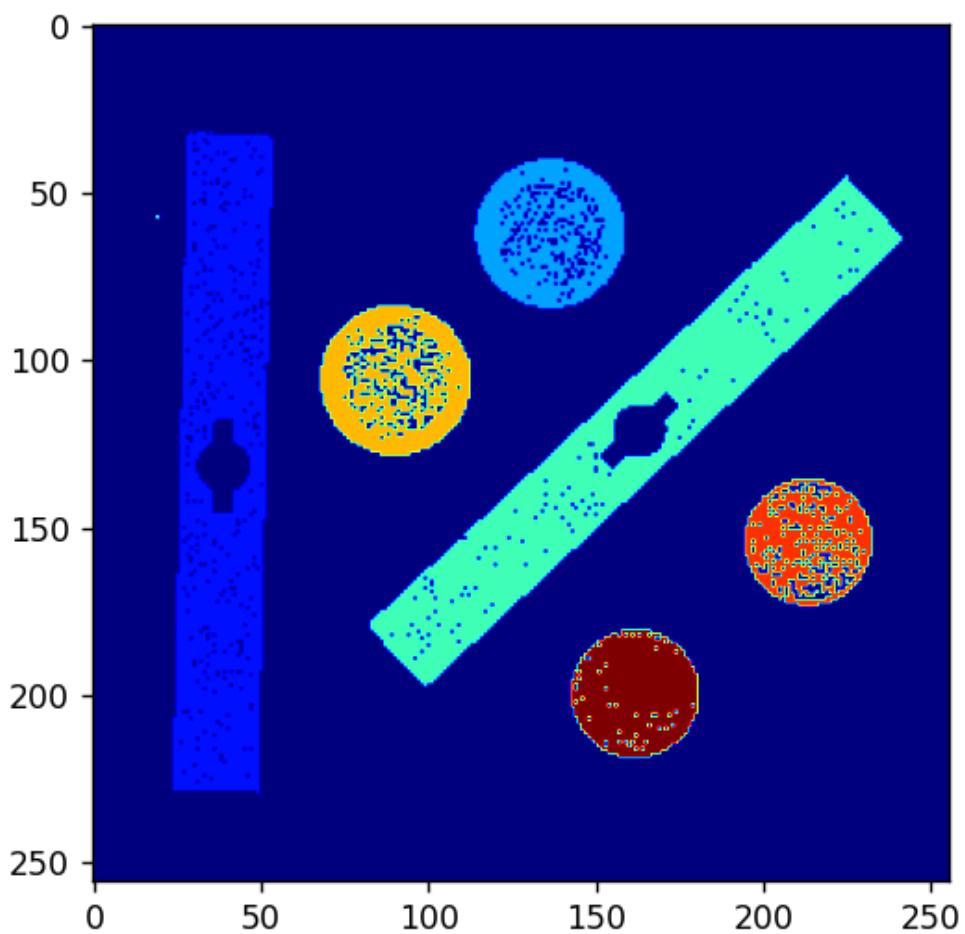


Background=1

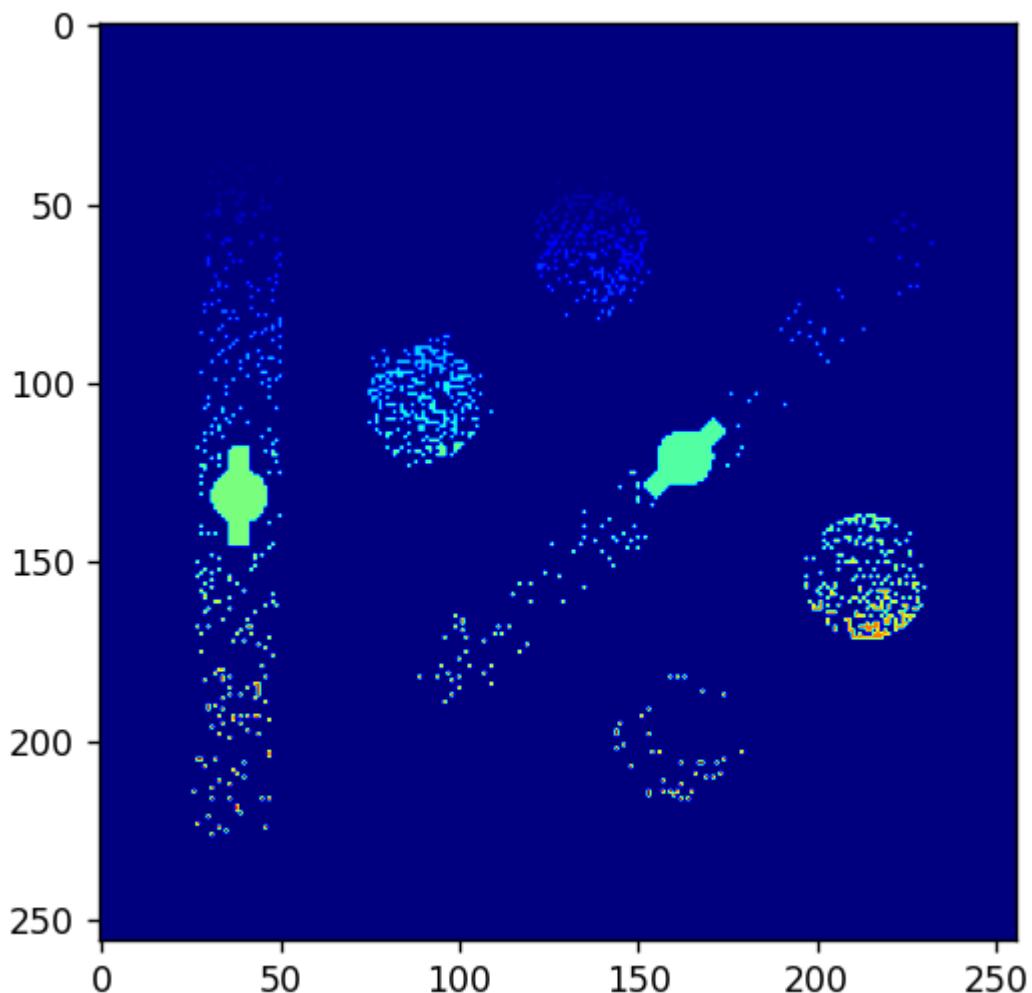
Labelling connected components



Background=0



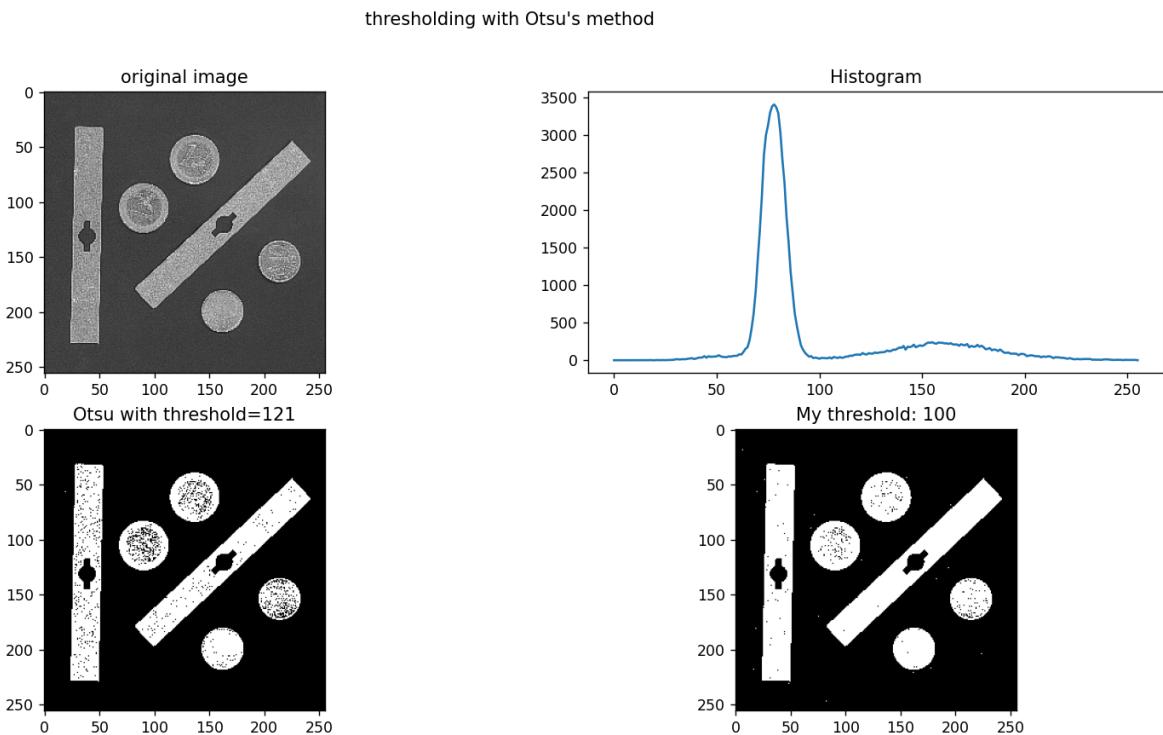
Background=1



### P5 - Binarización (intro) - 11/12/2023

Se ha considerado la imagen monedas.pgm y se ha mostrado su histograma para verificar que es claramente bimodal. A partir de esa observación, se ha elegido manualmente un umbral (100) y se ha comparado la segmentación utilizando este umbral con la de Otsu. Se ha escogido 100, ya que un buen umbral es aquel que maximiza la separación entre los dos modos en el histograma. En este caso, el umbral más adecuado es el elegido por Otsu (121), ya que es el que mejor separa los objetos del fondo.

El método Otsu se aplica utilizando la función `skimage.filters.threshold_otsu`. En el código, la imagen se binariza usando la expresión `im > threshold_otsu(im)`. Esto crea una imagen binaria donde los valores de píxeles mayores que el umbral calculado se establecen en True (1), lo que representa el primer plano (objetos), y los valores de píxeles menores o iguales al umbral se establecen en False (0), lo que representa el fondo.



#### P4 - Reflexiones de aprendizaje - 8/12/2023

Las actividades de este laboratorio implican la utilización de técnicas de detección de bordes, específicamente los detectores Sobel y Canny, y la exploración de la transformada de Hough para detectar líneas rectas.

Por un lado, los detectores Sobel y Canny destacan por su eficacia a la hora de capturar diferentes aspectos de la información de los bordes. El operador Sobel es conocido por su simplicidad y capacidad para enfatizar bordes verticales y horizontales. Mientras que, el detector de Canny destaca por detectar bordes con poco ruido y una localización precisa. Al trabajar con estos detectores, se ha obtenido información sobre las ventajas y desventajas entre simplicidad, tolerancia al ruido y precisión, lo cual es esencial a la hora de seleccionar un método de detección de bordes adecuado para aplicaciones específicas.

Por otro lado, la transformada de Hough es un mecanismo, basado en un esquema de votación, que proporciona un enfoque versátil para identificar estructuras lineales en una imagen. Estas actividades han servido para comprender el funcionamiento y la aplicabilidad de la transformada de Hough, concluyendo que no solo es una herramienta válida para detectar líneas sino también otras formas geométricas. Sin embargo, conlleva costos computacionales, especialmente cuando se trata de un espacio de parámetros grande, por tanto, la utilidad de esta transformación debe considerarse cuidadosamente.

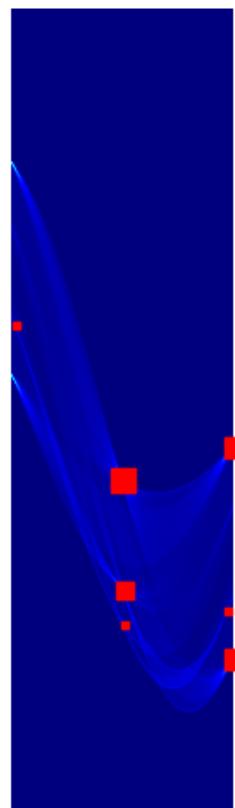
#### P4 - Ejercicio 3 - 8/12/2023

En este ejercicio se ha habilitado la variable booleana `bRotate` para rotar la imagen de entrada algunos grados  $\alpha$  (0, 15, 30, 45, 90). A continuación, se ha ejecutado el HT para estas orientaciones y se ha visto que no se encuentran el mismo número de picos en el espacio del acumulador (Hough) en todos los casos. Por otro lado, saber si existe una

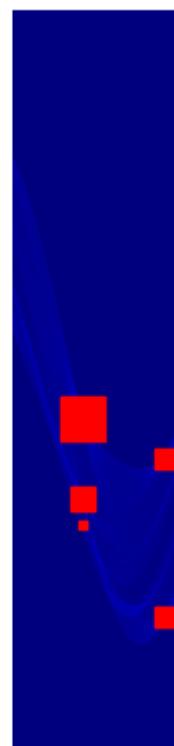
relación entre la coordenada  $\theta$  del pico y la rotación aplicada, implica examinar cómo cambian los ángulos máximos con diferentes rotaciones. Es decir, se tiene que observar si hay un cambio o un patrón constante en los ángulos máximos a medida que se gira la imagen.

Resultados:

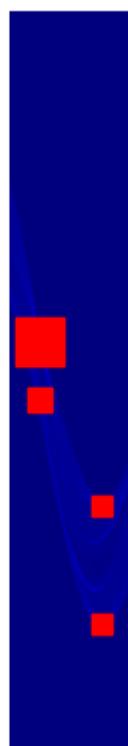
0:



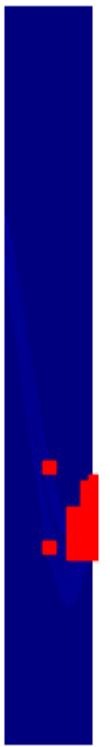
15:



30:



45:



90:



#### P4 - Ejercicio 2 - 8/12/2023

$\sigma$  es la desviación estándar del filtro gaussiano y determina la cantidad de suavizado aplicado a la imagen antes de la detección de bordes. Los valores de  $\sigma$  más altos dan como resultado un suavizado más extenso, lo que reduce el ruido de alta frecuencia pero potencialmente difumina los bordes. Los valores de  $\sigma$  más bajos conservan detalles más finos pero también pueden retener más ruido.

T1 establece el umbral inferior para las magnitudes de gradiente. Los píxeles con magnitudes de gradiente inferiores a T1 se consideran bordes débiles. Es decir, los valores más bajos de T1 aumentan la sensibilidad a los bordes débiles y al ruido. Un T1 bajo permite incluir más bordes débiles (y potencialmente ruido) en el mapa de bordes final.

T2 establece el umbral más alto para las magnitudes de gradiente. Los píxeles con magnitudes de gradiente superiores a T2 se consideran bordes fuertes. Es decir, los valores de T2 más altos reducen la probabilidad de que se incluyan bordes débiles en el mapa de bordes final. Ayuda a filtrar los bordes más débiles y el ruido, asegurando que sólo se conserven los bordes fuertes y bien definidos.

Por tanto, las relaciones quedarían de la siguiente forma:

- 1: b
- 2: a
- 3: c
- 4: d

#### P4 - Ejercicio 1 - 8/12/2023

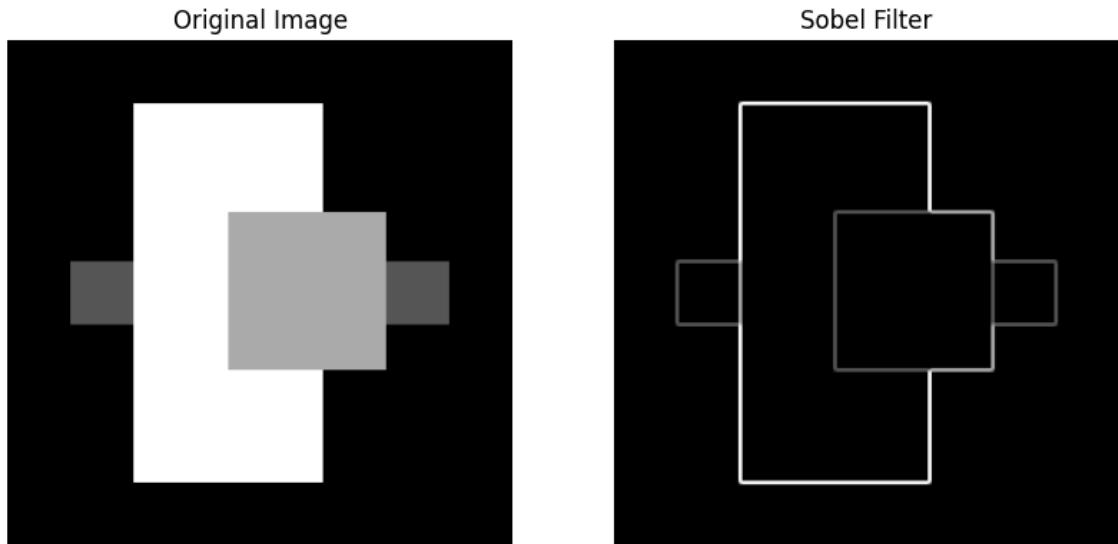
En este ejercicio, en lugar de utilizar la función sobel de los filtros del módulo en `scipy.ndimage`, se ha aplicado el detector Sobel a través de las convoluciones correspondientes. Sin embargo, como se puede observar el resultado no es el mismo (no tengo claro porque pasa esto).

Código:

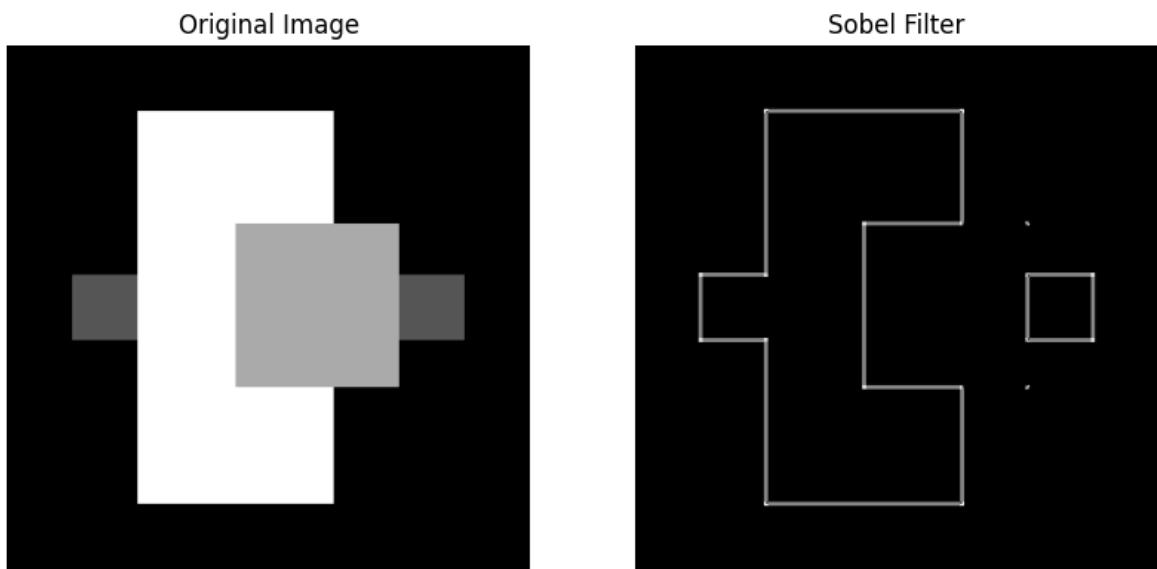
```
def sobel_convolution(im):  
    # Sobel filter kernels for x and y directions  
    sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])  
    sobel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])  
  
    # Perform convolutions  
    gx = signal.convolve2d(im, sobel_x, mode='same', boundary='symm', fillvalue=0)  
    gy = signal.convolve2d(im, sobel_y, mode='same', boundary='symm', fillvalue=0)  
  
    # Compute gradient magnitude  
    magnitude = np.sqrt(gx**2 + gy**2)  
  
    return magnitude
```

Resultados:

Con convoluciones:



Con `scipy.ndimage`:



#### P4 - Transformada de Hough (intro) - 8/12/2023

Se ha probado la Transformada de Hough para líneas sobre la imagen de cuadros, donde el primer paso consiste en calcular el espacio del acumulador a partir de un mapa de bordes binario.

La función que utilizamos para obtener este espacio es `skimage.transform.hough_line`, qué tiene por parámetros:

- `image`: el mapa de bordes binario obtenido a partir de un algoritmo de detección de bordes.

- `'theta'`: Los ángulos en los que se calcula la transformada de Hough.

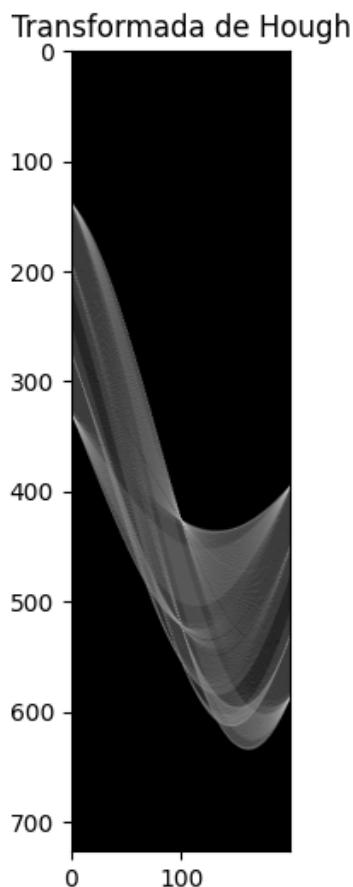
Y qué se devuelve:

- `'hspace'`: El espacio acumulador de Hough, donde cada elemento representa una línea particular en la imagen.

- `angles`: La matriz de ángulos utilizados en el cálculo.
- `distances`: El conjunto de distancias desde el origen hasta el punto más cercano de las líneas.

En el código, la variable correspondiente al espacio del acumulador es `H` (`hspace`). El tamaño de `H` está determinado por el número de ángulos (`angles`) y distancias (`distances`). Por tanto, el tamaño de `H` es `(len(distances), len(angles))`, que representa la cuadrícula de rhos y thetas en el espacio del acumulador de Hough.

Si observamos el resultado mostrado, el eje x normal representa los ángulos ("thetas") en el espacio de Hough, mientras que el eje y representa las distancias ("rhos") desde el origen hasta el punto más cercano de las líneas. Cada elemento en "H" representa la fuerza o el recuento de votos para una línea particular en la imagen correspondiente a una combinación específica de ángulo y distancia. Es decir, los valores más altos en "H" indican una mayor probabilidad de que exista una línea con el ángulo y la distancia correspondientes en la imagen.



Una vez calculado el espacio del acumulador, podemos proceder a identificar sus picos. Los picos en el espacio del acumulador proporcionan información sobre la presencia de líneas en la imagen original. Cada pico corresponde a una línea de potencial, caracterizada por un ángulo y una distancia específicos desde el origen, es decir, la posición de un pico indica los parámetros (ángulo y distancia) de una línea detectada en la imagen.

Los valores de los picos en el espacio del acumulador dependen de la fuerza de la línea en la imagen y del número de votos que recibió durante la Transformada de Hough. Por tanto, los valores más altos indican líneas que tienen más evidencia de respaldo (votos) en el mapa de borde.

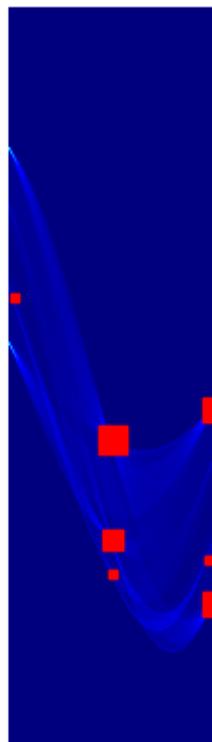
En el código proporcionado, la función utilizada para localizar picos en el espacio del acumulador es `skimage.transform.hough_line_peaks`.

Parámetros:

- `H`: El espacio del acumulador obtenido de la Transformada de Hough.
- `thetas`: el conjunto de ángulos utilizados en la transformada de Hough.
- `rhos`: el conjunto de distancias desde el origen hasta el punto más cercano de las líneas.
- `num_peaks`: el número máximo de picos para identificar.
- `umbral`: El valor mínimo de votos requeridos para que se considere un pico.
- `min_angle`: el ángulo mínimo (en grados) entre picos adyacentes.
- `min_distance`: la distancia mínima entre picos adyacentes.

Devoluciones:

- `hough_peaks`: una tupla que contiene matrices que representan los índices de fila, los índices de columna y los valores acumulados de los picos detectados.



#### P4 - Detector de Canny (intro) - 8/12/2023

Se han probado distintos valores de desviación estándar (1, 2 y 3) para el detector de Canny para dos imágenes distintas primero con ruido y, a continuación, sin ruido.

Observando los resultados mostrados a continuación, se pueden realizar una serie de conclusiones.

En primer lugar, el efecto de  $\sigma$  en la detección de bordes Canny. A medida que varía el valor de  $\sigma$  para el filtro gaussiano en la detección de bordes de Canny, se observan cambios en los bordes resultantes. Los valores de  $\sigma$  más pequeños dan lugar a que se capturen detalles más finos en los bordes, mientras que los valores de  $\sigma$  más grandes dan como resultado bordes más suaves.

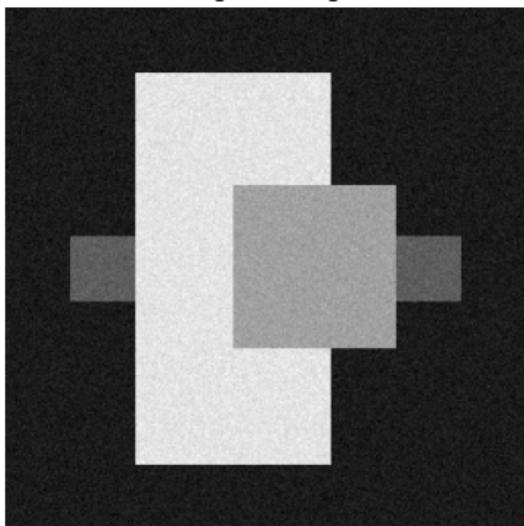
En segundo lugar, el efecto del ruido en la detección de bordes Canny. Agregar ruido a las imágenes afecta el rendimiento de la detección de bordes de Canny, de forma que, los niveles de ruido más altos pueden introducir bordes falsos u oscurecer los bordes verdaderos, lo que dificulta la detección.

Por otro lado comparando los resultados del filtro de Sobel con los del detector de Canny se puede observar que los bordes de Sobel tienden a ser más delgados en comparación con los bordes de Canny, es decir, captura las regiones de gradiente de alta intensidad en la imagen, y los bordes Canny son generalmente más gruesos debido al seguimiento basado en histéresis, lo que proporciona segmentos de borde más continuos y conectados. Además, Sobel es sensible al ruido y el ruido puede afectar significativamente la calidad de la detección de bordes, mientras que Canny, con su suavizado gaussiano y umbral de histéresis, es menos sensible al ruido, lo que lo hace más robusto en presencia de condiciones ruidosas. Finalmente, los bordes de Sobel pueden aparecer fragmentados, especialmente en presencia de ruido, mientras que los bordes Canny están más conectados y forman curvas más continuas, lo que proporciona una representación más coherente de los límites de los objetos.

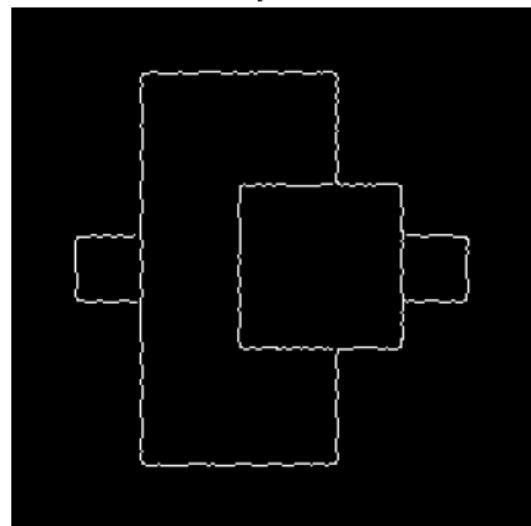
Por último, el detector de bordes Canny utiliza dos umbrales para determinar qué bordes conservar y cuáles descartar durante el seguimiento de bordes por histéresis. Estos umbrales son el umbral bajo (`low_threshold`) y el umbral alto (`high_threshold`). Un valor más bajo para el umbral bajo permite incluir bordes más débiles en el resultado, por tanto, si el umbral bajo es demasiado bajo, es posible que se incluyan más ruido y bordes débiles, lo que dará lugar a un mapa de bordes menos específico. Un valor más alto para el umbral alto da como resultado que se incluyan menos bordes fuertes en el resultado final, por tanto, aumentar el umbral alto puede hacer que se pierdan bordes débiles, lo que reduce la sensibilidad general del detector.

Con ruido:

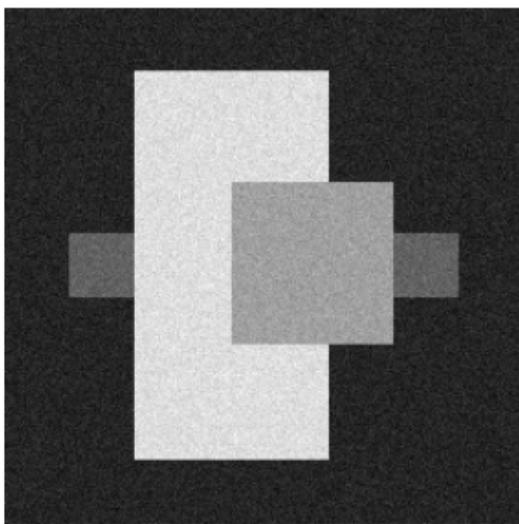
Original Image



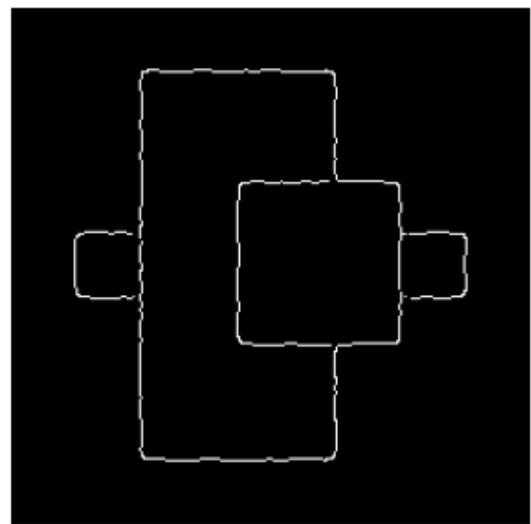
Canny ( $\sigma=1$ )



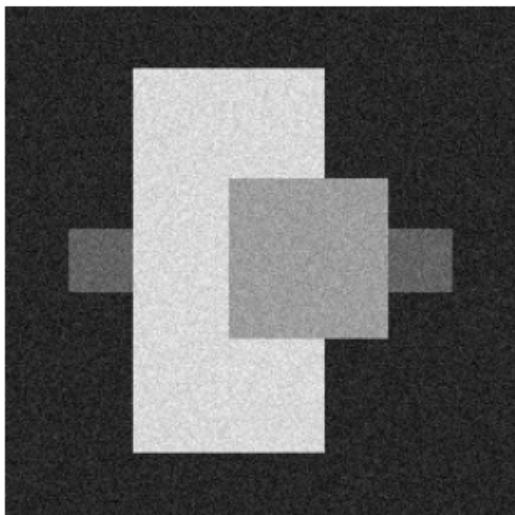
Original Image



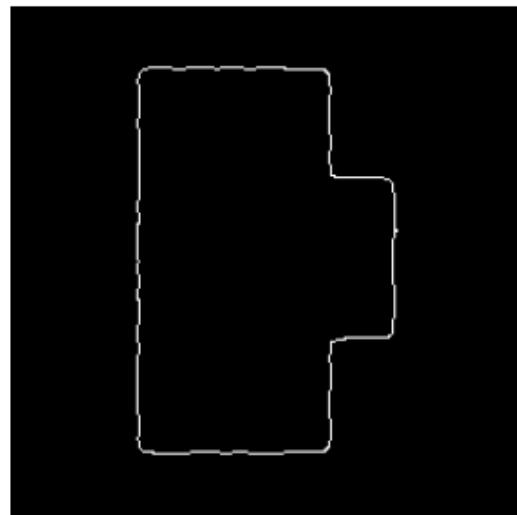
Canny ( $\sigma=2$ )



Original Image



Canny ( $\sigma=3$ )



Original Image



Canny ( $\sigma=1$ )



Original Image



Canny ( $\sigma=2$ )



Original Image

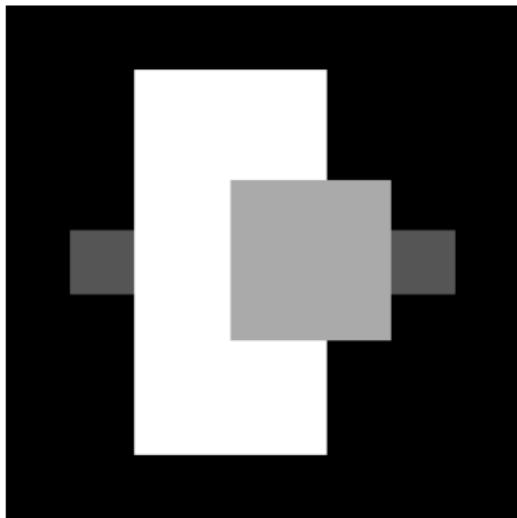


Canny ( $\sigma=3$ )

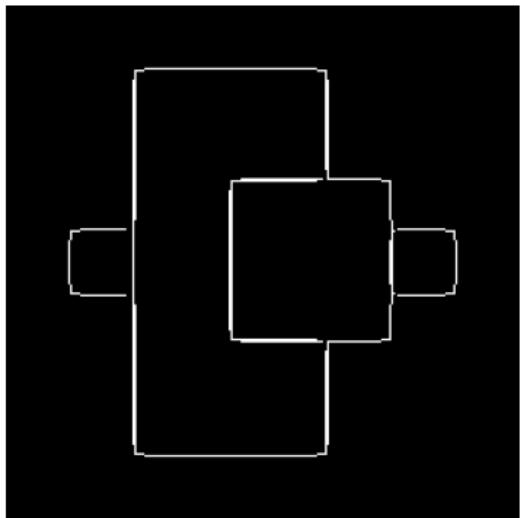


Sin ruido:

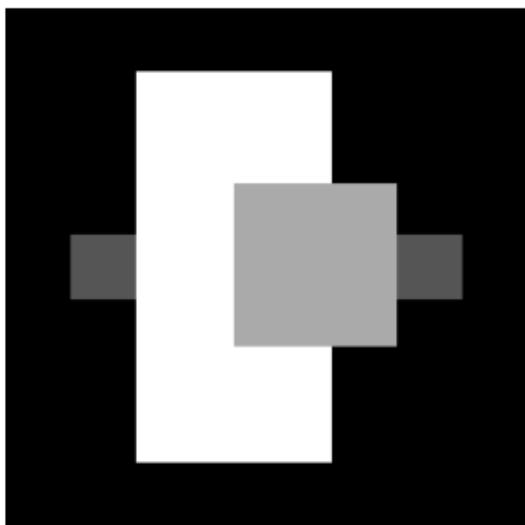
Original Image



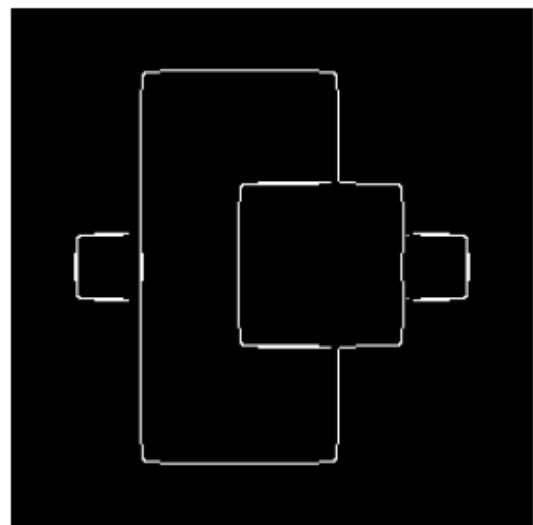
Canny ( $\sigma=1$ )



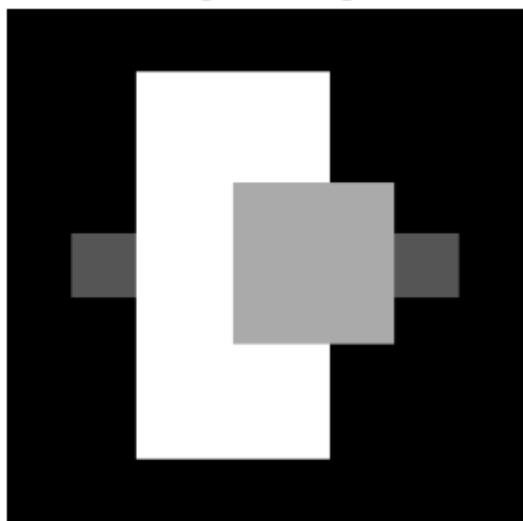
Original Image



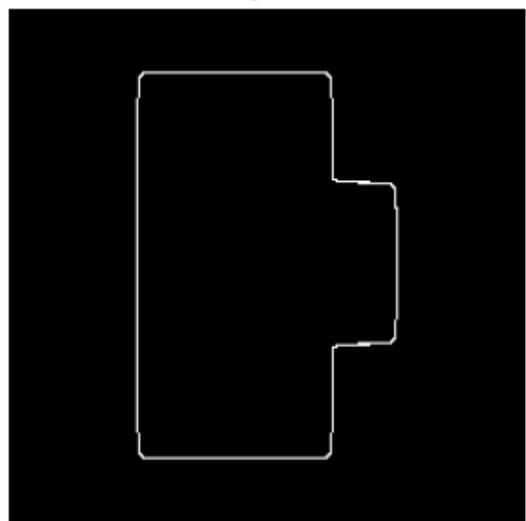
Canny ( $\sigma=2$ )



Original Image



Canny ( $\sigma=3$ )



Original Image



Canny ( $\sigma=1$ )



Original Image



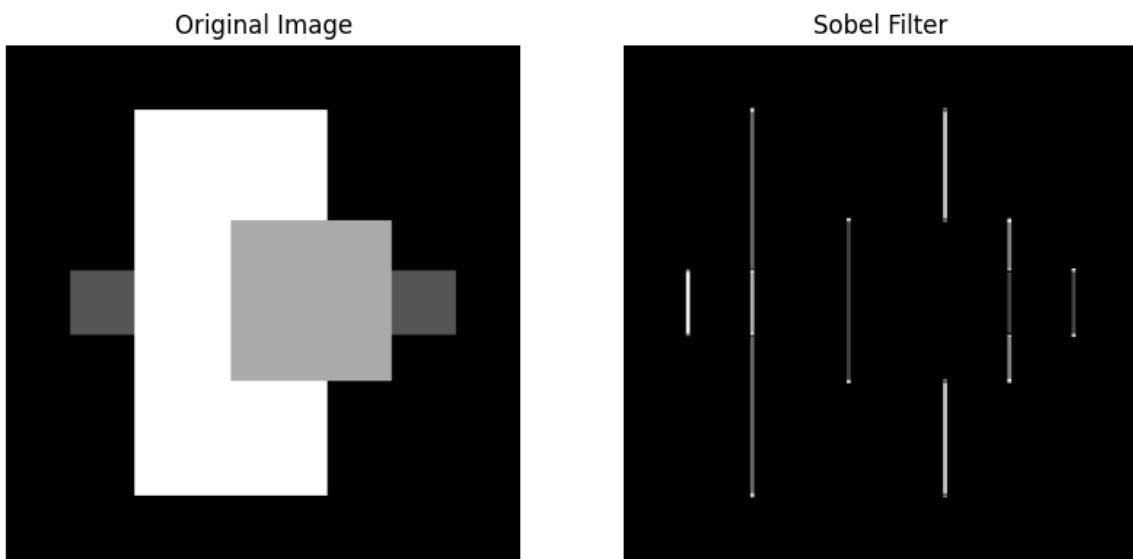
Canny ( $\sigma=2$ )





#### P4 - Filtro de Sobel (intro) - 8/12/2023

Se ha ejecutado el código proporcionado para aplicar el filtro de Sobel a la imagen cuadros.png. Como se puede observar, el resultado corresponde al gradiente de la imagen en una sola dirección (vertical).

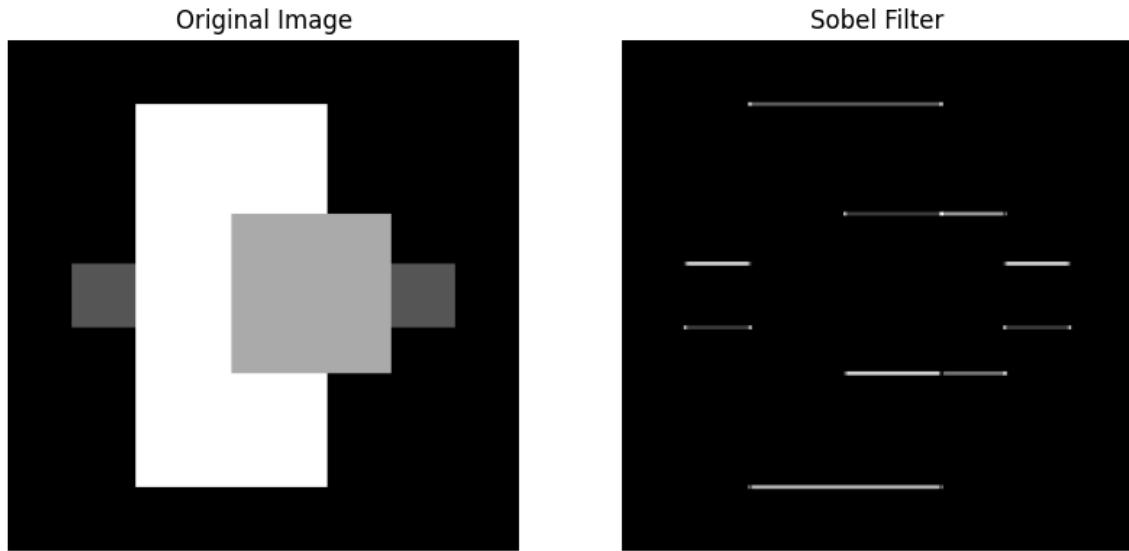


En el resultado mostrado, los niveles de gris corresponden a la magnitud de los valores de gradiente en la imagen. El filtro Sobel se utiliza para calcular el gradiente a lo largo de una dirección particular (generalmente horizontal o vertical). El gradiente representa la tasa de cambio de intensidad en la imagen por tanto:

- Regiones oscuras: magnitud de gradiente baja. Esto corresponde a áreas de la imagen donde la intensidad cambia gradualmente o no cambia en absoluto.
- Regiones claras: magnitud de gradiente alta. Esto corresponde a áreas de la imagen donde la intensidad cambia rápidamente.

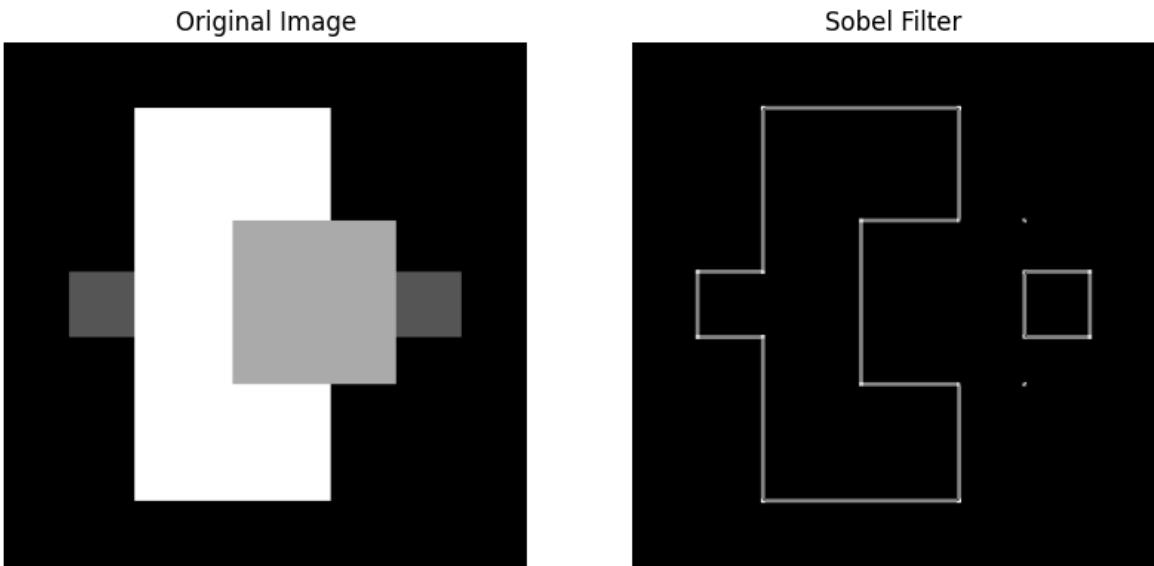
Por ello, en las imágenes filtradas por Sobel, los píxeles oscuros indican regiones con un pequeño cambio de intensidad, y los píxeles claros indican regiones con un cambio significativo de intensidad.

Si cambiamos el parámetro ‘axis’ de 1 a 0: `gx = filters.sobel(im, 1) -> gy = filters.sobel(im, 0)`, entonces obtenemos el filtro de Sobel de la imagen en la dirección horizontal.

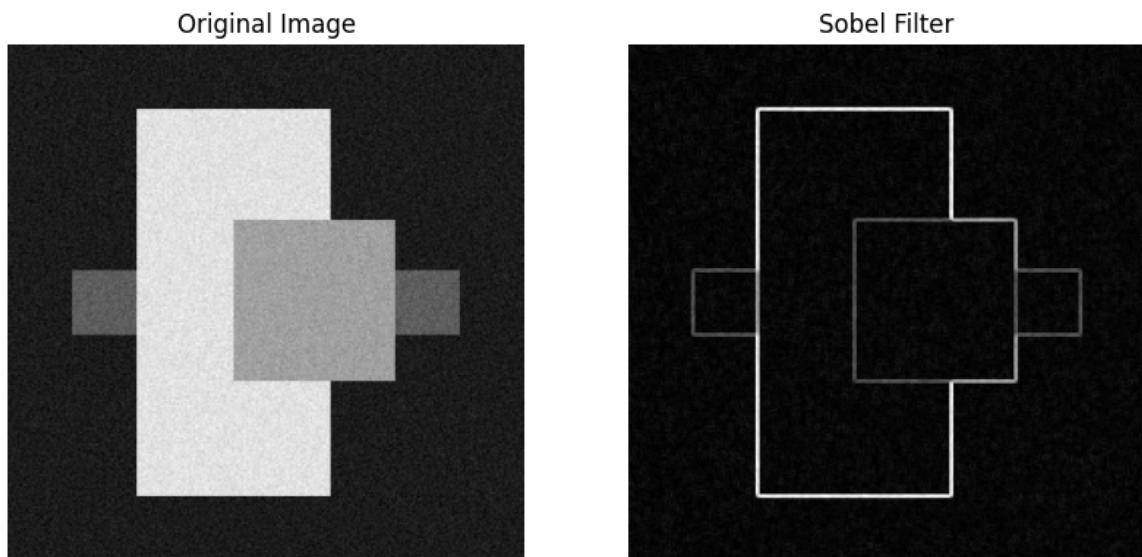


Finalmente, obtenemos la magnitud del filtro de la siguiente forma:

```
def testSobel(im, params=None):
    gx = filters.sobel(im, 1)
    gy = filters.sobel(im, 0)
    magnitude = np.sqrt(gx**2 + gy**2)
    plotSobel(im, magnitude)
    return [magnitude]
```



Con ruido:



### P3 - Reflexiones de aprendizaje - 5/12/2023

El objetivo de esta práctica es la exploración del dominio de la frecuencia, un ámbito menos intuitivo que el dominio espacial pero que ofrece distintas ventajas en determinadas situaciones. Las actividades propuestas en esta práctica de laboratorio implican trabajar con la Transformada de Fourier para aplicar, manipular y visualizar representaciones de frecuencia de imágenes. Además, el laboratorio tiene como objetivo validar experimentalmente el teorema de convolución y familiarizarse con el diseño y la aplicación de filtros en el dominio de la frecuencia.

Por tanto, a través de estas actividades se ha aprendido a aplicar, manipular y visualizar la Transformada de Fourier en imágenes, comprendiendo la información que se proporciona sobre los componentes de frecuencia de una imagen.

Por otro lado, la verificación experimental del teorema de convolución no sólo refuerza el conocimiento teórico sino que también proporciona confianza en la utilización de operaciones en el dominio de la frecuencia.

Finalmente, diseñar y aplicar filtros en el dominio de la frecuencia es otro aspecto clave de este laboratorio. Ciertos filtros, pueden aplicarse de forma más natural en el dominio de la frecuencia, ofreciendo ventajas sobre sus homólogos del dominio espacial. Esta práctica permite comprender cómo manipular las representaciones de frecuencia para lograr objetivos de filtrado específicos.

### P3 - Ejercicio 5 - 5/12/2023

En este ejercicio se cargan dos imágenes, se calculan sus transformadas de Fourier, se combinan usando diferentes valores de  $\lambda$  y, finalmente, se visualizan los resultados. La propiedad que se ilustra es la linealidad de la transformada de Fourier, es decir, la transformada inversa de una combinación lineal de transformadas de Fourier es igual a la combinación lineal de las transformadas inversas.

Código:

```
def inverse_transform_combination(F1, F2, lmbda):
    return np.real(fft.ifft2(lmbda * F1 + (1 - lmbda) * F2))

def exercise5(im1, im2):
    F1 = fft.fft2(im1)
    F2 = fft.fft2(im2)

    lambda_values = np.linspace(0, 1, 5)

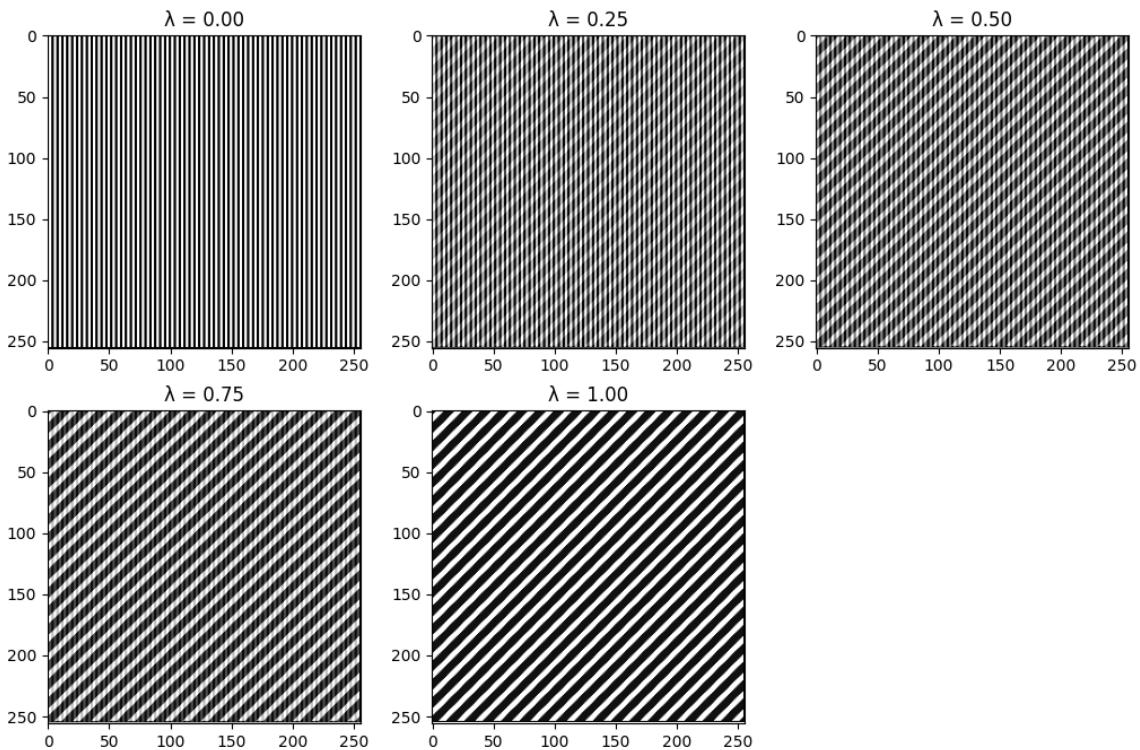
    plt.figure(figsize=(15, 10))

    for i, lmbda in enumerate(lambda_values, 1):
        combined_image = inverse_transform_combination(F1, F2, lmbda)

        plt.subplot(2, 3, i)
        plt.imshow(combined_image, cmap='gray')
        plt.title(f'\u03bb = {lmbda:.2f}')

    plt.show()
```

Resultados:



### P3 - Ejercicio 4 - 5/12/2023

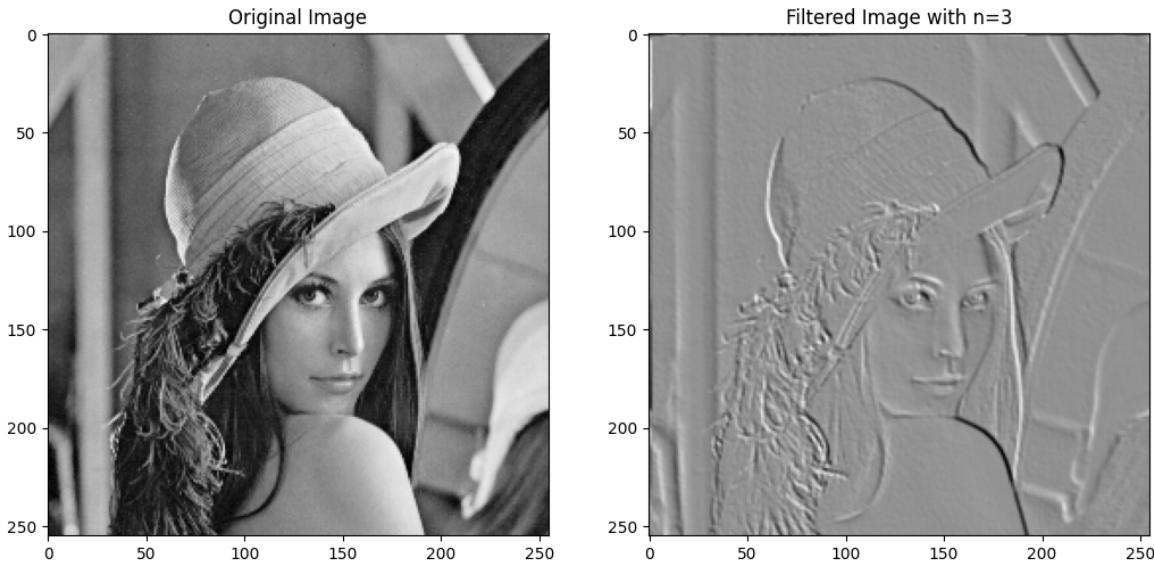
En este ejercicio se ha creado una matriz que dado un tamaño  $n \times n$ , la diagonal sean ceros, la parte superior menos unos y la parte inferior unos. Esta matriz se emplea como máscara para filtrar la imagen en el dominio de la frecuencia.

Código:

```
def my_mask(n):
    matrix = np.zeros((n, n), dtype=int)
    matrix[np.triu_indices(n, k=1)] = -1
    matrix[np.tril_indices(n, k=-1)] = 1
    return matrix

def my_filter(im, n):
    mask = my_mask(n) # Get the mask from my_mask function
    im_fft = fft.fft2(im) # FFT of the image
    mask_fft = fft.fft2(mask, s=im.shape) # FFT of the mask
    im_filtered_fft = im_fft * mask_fft # Apply the filter in the frequency domain
    im_filtered = np.real(fft.ifft2(im_filtered_fft)) # Inverse FFT to obtain the filtered image
    return im_filtered
```

Resultados:



### P3 - Ejercicio 3 - 5/12/2023

En este ejercicio se ha realizado un estudio de los tiempos de ejecución para diferentes tamaños de filtro ( $3 \times 3$ ,  $5 \times 5$ ,  $7 \times 7$  y  $9 \times 9$ ), y para diferentes tamaños de imagen ( $255 \times 255$  y  $512 \times 512$ ). Se ha realizado un gráfico de los tiempos medidos para apreciar la influencia del tamaño de la imagen para un tamaño de filtro determinado ( $7 \times 7$ ), y otro gráfico donde el tamaño de la imagen es fijo ( $255 \times 255$ ) y el tamaño del filtro varía. Se ha empleado el filtro medio y se han realizado 1000 experimentos para cada gráfico, de los cuales se ha obtenido la media de los tiempos de ejecución. Finalmente, como se observa en las curvas, se puede concluir que un tamaño de filtro mayor implica un tiempo de ejecución más grande. De la misma forma, un tamaño de imagen más grande, implica un mayor tiempo de ejecución.

Código:

```

def plot_time_mean_filter_frequency_response(imfiles, filterSizes):
    time_lst_filters = []
    time_lst_im = []
    im = imfiles[0] # 255x255
    size = filterSizes[1] #7x7
    im_sizes = ['255x255', '512x512']
    n = 1000

    time_mean_filters = []
    for filterSize in filterSizes:
        for i in range(n):
            initTime = time.time()
            imFiltFreq, FG, filterFT = averageFilterFrequency(im, filterSize)
            endTime = time.time()
            time_lst_filters.append(endTime - initTime)
        time_mean_filters.append(np.mean(time_lst_filters))

    time_mean_im = []
    for imfile in imfiles:
        for i in range(n):
            initTime = time.time()
            imFiltFreq, FG, filterFT = averageFilterFrequency(imfile, size)
            endTime = time.time()
            time_lst_im.append(endTime - initTime)
        time_mean_im.append(np.mean(time_lst_im))

    # Crear el primer gráfico
    plt.subplot(2, 1, 1) # 2 filas, 1 columna, primer gráfico
    plt.plot(filterSizes, time_mean_filters, marker='o', linestyle='--')
    plt.title('image size fixed (255x255)')
    plt.xlabel('Filter Sizes')
    plt.ylabel('Time')

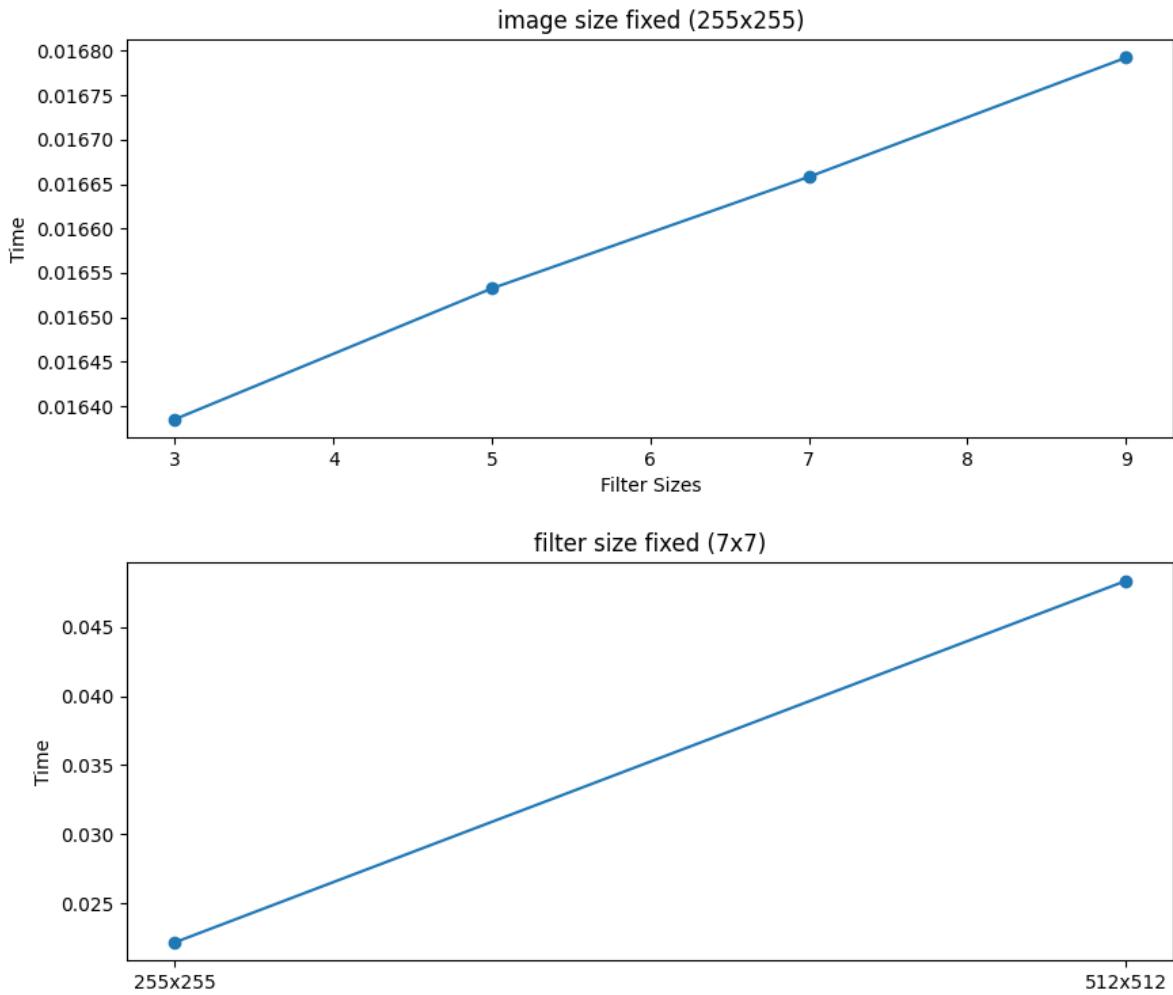
    plt.tight_layout()

    plt.subplot(2, 1, 2) # 2 filas, 1 columna, segundo gráfico
    plt.plot(im_sizes, time_mean_im, marker='o', linestyle='--')
    plt.title('filter size fixed (7x7)')
    plt.xlabel('Image sizes')
    plt.ylabel('Time')

    plt.show()

```

Resultados:



### P3 - Ejercicio 2 -5/12/2023

En este ejercicio se define la función del filtro gaussiano, y se aplica el filtro en el dominio espacial y de frecuencia, del mismo modo que el filtro medio. Finalmente, se muestran los resultados junto con la transformada de Fourier del filtro y el producto por elementos de las transformadas de Fourier del filtro y de la imagen, y como se puede observar las imágenes filtradas en el dominio del espacio y en el dominio de la frecuencia son iguales, tal y como cabía esperar.

Por otro lado, se ha obtenido la transformada de fourier del filtro medio para tamaños de filtro de 3x3, 5x5, 7x7 y 9x9, y se ha observado que cuanto menor sea la extensión espacial de un filtro (es decir, el tamaño del filtro), mayor será su dispersión en el dominio de la frecuencia, y viceversa. Es decir, un filtro medio más pequeño corresponde a una respuesta de frecuencia más amplia y un filtro medio más grande corresponde a una respuesta de frecuencia más estrecha.

Por último, se ha obtenido la transformada de fourier del filtro gaussiano para un tamaño de filtro de 7x7 y desviaciones estándar de 1, 2, 3 y 4, y se ha observado que una desviación estándar más pequeña da como resultado una respuesta de frecuencia más amplia, lo que permite que el filtro capture una gama más amplia de frecuencias. Por el contrario, una desviación estándar mayor reduce la respuesta de frecuencia, centrándose en un rango de frecuencias más específico.

Código:

```
def gaussianFilter(filterSize, sigma):
    x = np.arange(-filterSize // 2 + 1., filterSize // 2 + 1.)
    y = np.arange(-filterSize // 2 + 1., filterSize // 2 + 1.)
    xx, yy = np.meshgrid(x, y, sparse=True)
    filterMask = np.exp(-(xx**2 + yy**2) / (2. * sigma**2))
    return filterMask / np.sum(filterMask)

def gaussianFilterSpace(im, filterSize, sigma):
    return filters.convolve(im, gaussianFilter(filterSize, sigma))

def gaussianFilterFrequency(im, filterSize, sigma):
    filterMask = gaussianFilter(filterSize, sigma)
    filterBig = np.zeros_like(im, dtype=float)

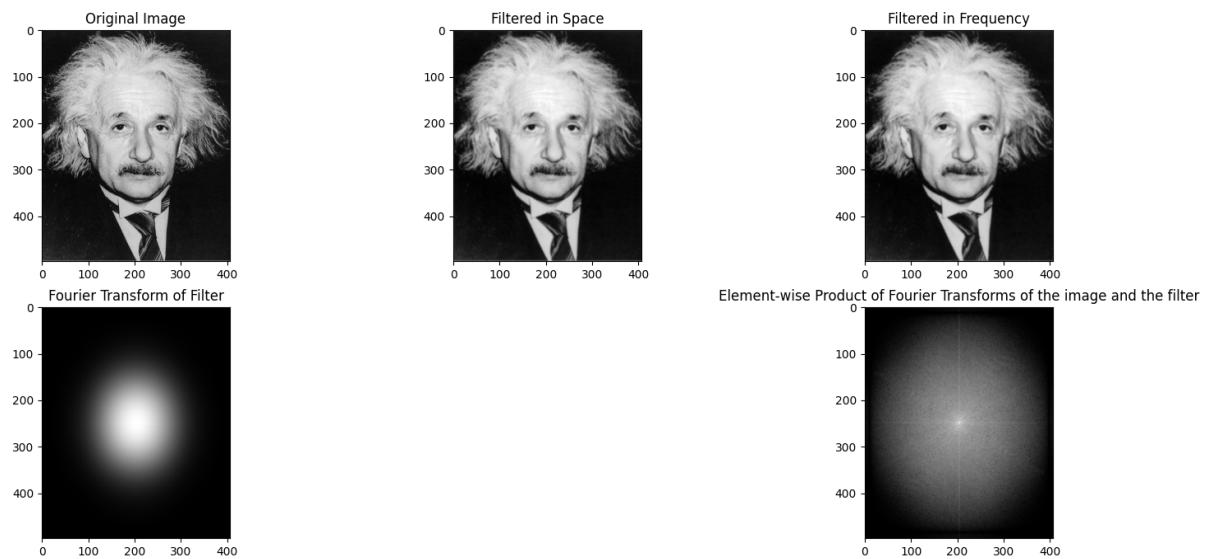
    w, h = filterMask.shape
    w2, h2 = w / 2, h / 2
    W, H = filterBig.shape
    W2, H2 = W / 2, H / 2

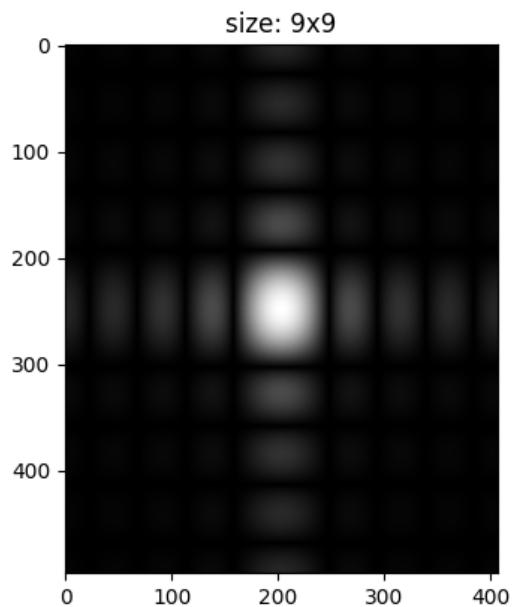
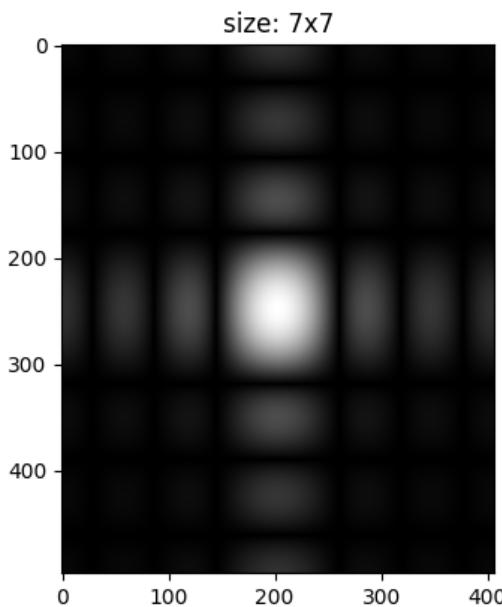
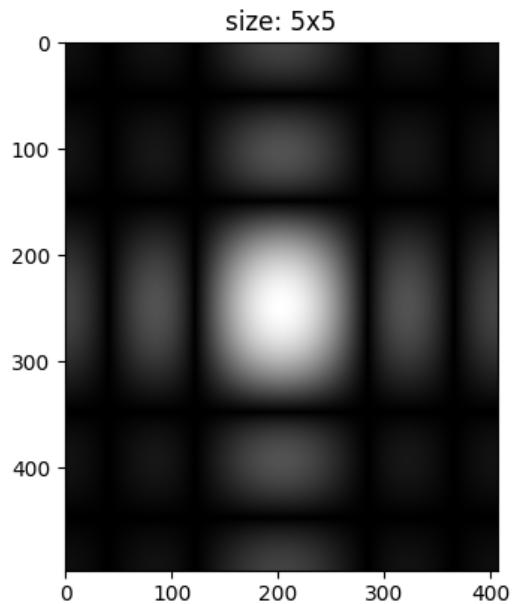
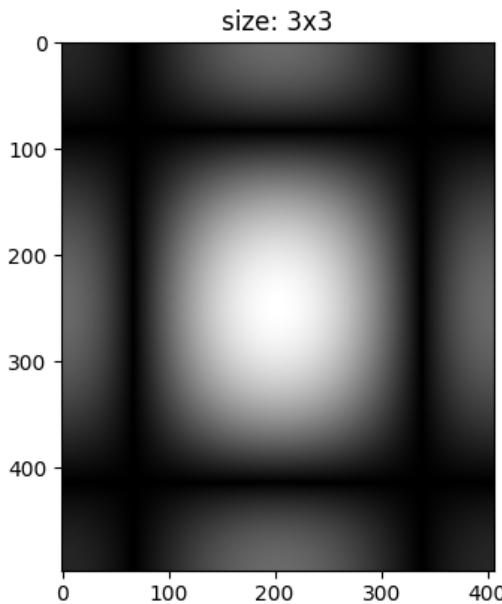
    filterBig[int(W2 - w2):int(W2 + w2), int(H2 - h2):int(H2 + h2)] = filterMask
    filterBig = fft.ifftshift(filterBig)

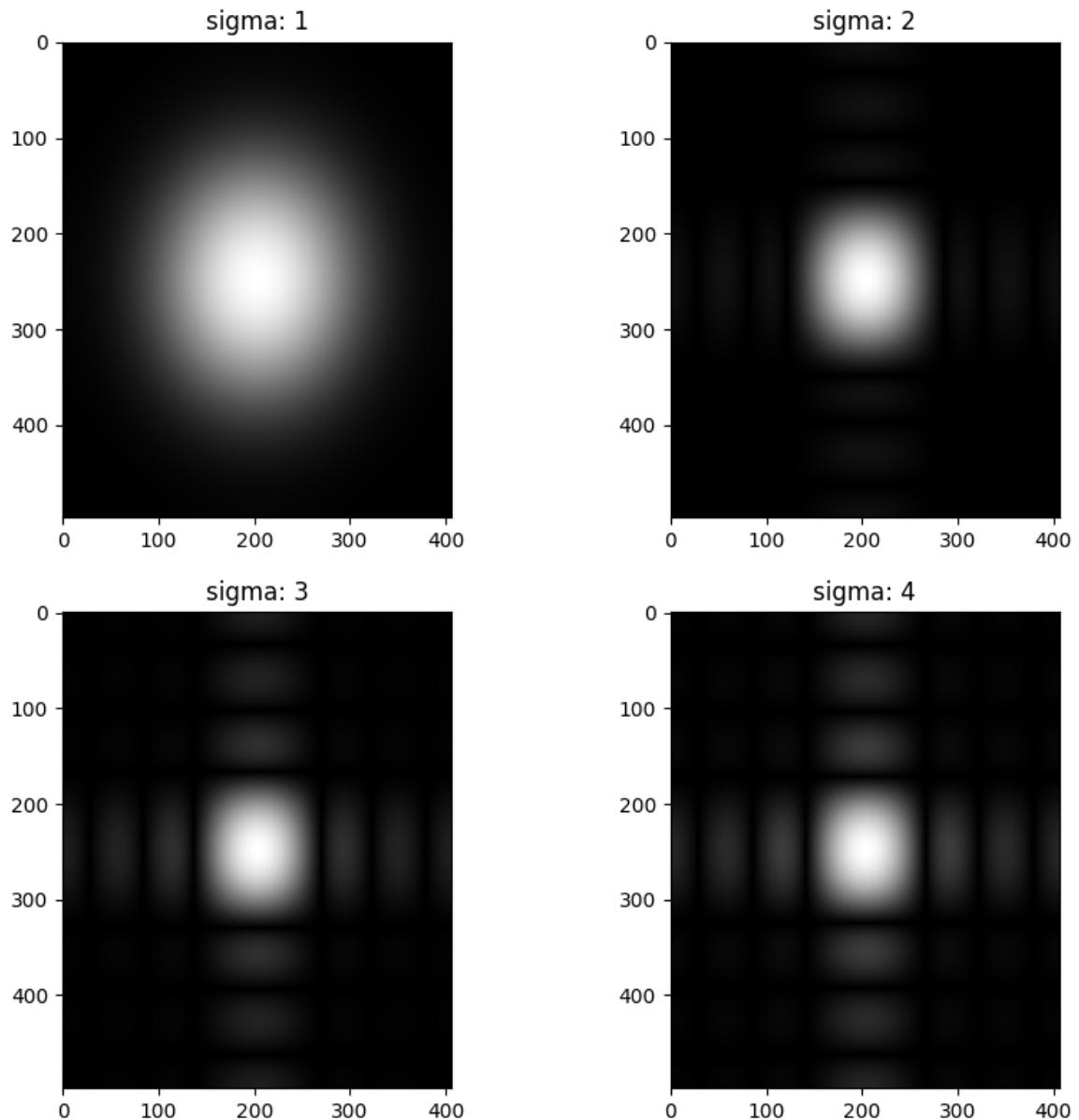
    imFiltFreq = np.absolute(IFT(FT(im) * FT(filterBig)))
    FG = np.absolute(FT(im) * FT(filterBig))
    filterFT = FT(filterBig)

    return imFiltFreq, FG, filterFT
```

Resultados:







### P3 - Ejercicio 1 - 5/12/2023

En este ejercicio, se calculan las magnitudes más baja y más alta de la imagen de Einstein. A continuación, se muestra un diagrama de caja de los valores de magnitud y se traza un histograma de los valores de fase.

Código:

```
def exercise1(result):
    # (a) Compute the lowest and highest magnitudes
    lowest_magnitude = np.min(result[0])
    highest_magnitude = np.max(result[0])

    # (b) Display a boxplot with the magnitude values
    plt.boxplot(result[0].flatten())
    plt.title('Magnitude Boxplot')
    plt.xlabel('Magnitude')
    plt.show()

    # (c) Plot a histogram of the phase values
    plt.hist(result[1].flatten(), bins=50, edgecolor='black')
    plt.title('Phase Histogram')
    plt.xlabel('Phase')
    plt.ylabel('Frequency')
    plt.show()

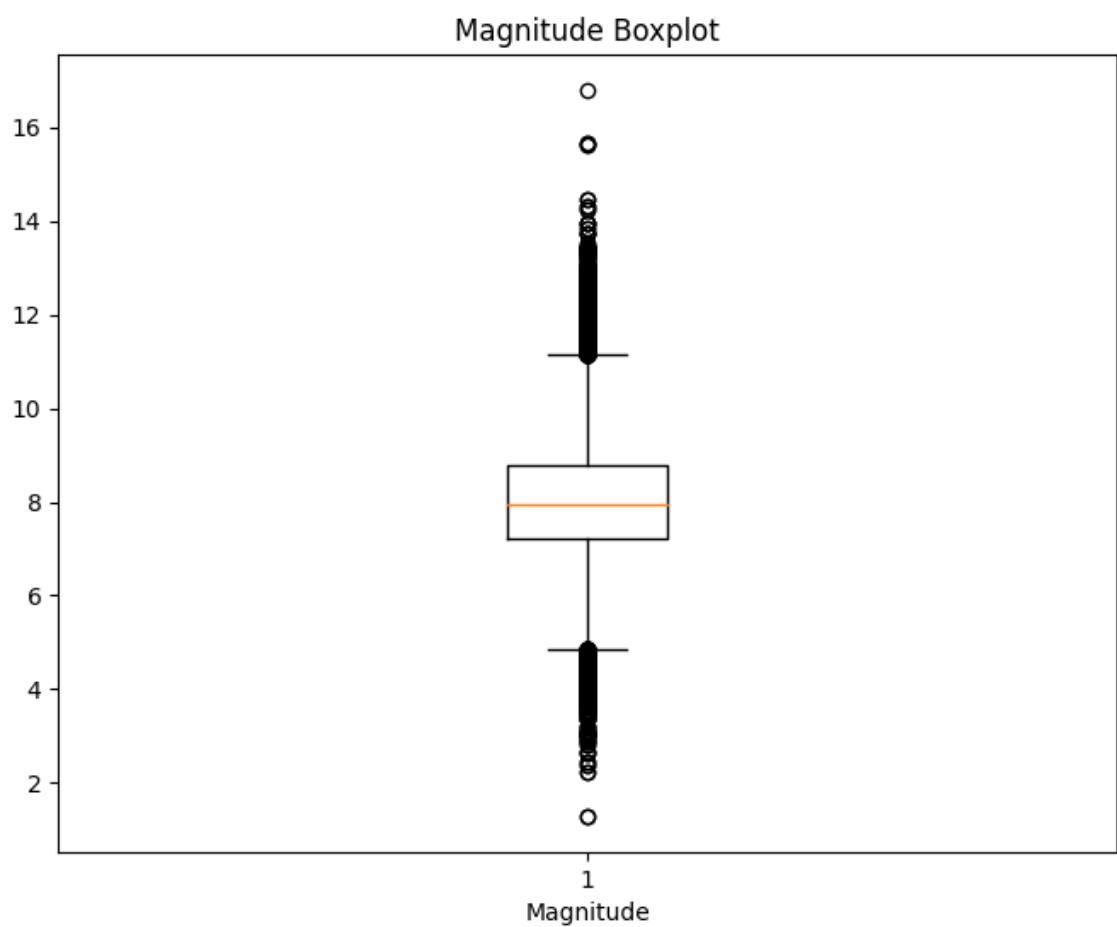
    print(f'(a) Lowest Magnitude: {lowest_magnitude}')
    print(f'(a) Highest Magnitude: {highest_magnitude}')
```

Resultados:

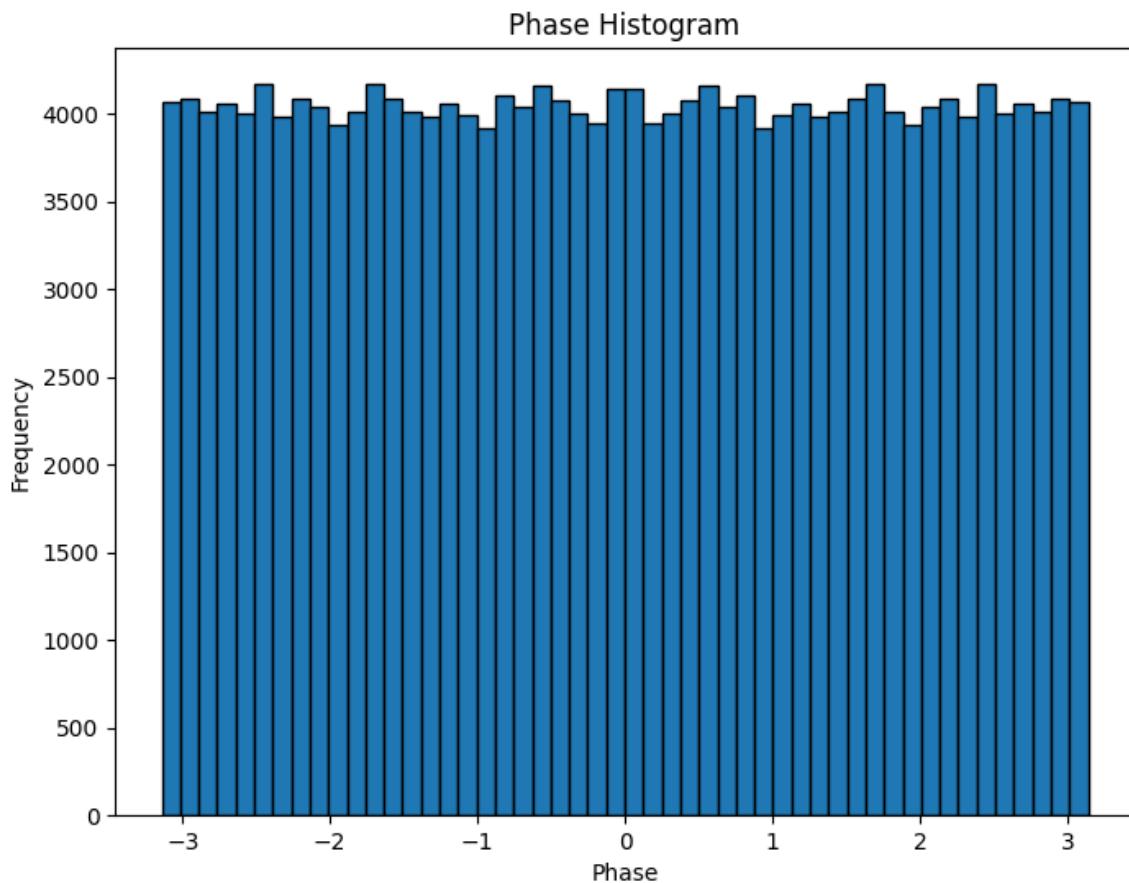
a) Lowest Magnitude: 1.2929587708613193

Highest Magnitude: 16.781096307101873

b)



c)



### P2 - Reflexiones de aprendizaje - 2/12/2023

El enfoque de esta práctica radica en la introducción deliberada de varios tipos y niveles de ruido en la imagen, seguida de la exploración de técnicas de filtrado de dominio espacial para mitigar o eliminar este ruido. Las actividades propuestas en este laboratorio permiten avanzar paso a paso en la comprensión de la generación y reducción de ruido.

La introducción intencionada de ruido en la imagen tiene un doble propósito. En primer lugar, proporciona una experiencia práctica al lidiar con diferentes tipos de ruido que se encuentran comúnmente en escenarios del mundo real. De forma que se llega a comprender las características de ruidos como el gaussiano o el de sal pudiendo reconocer y diferenciar estos ruidos.

En segundo lugar, introducir ruido de forma intencionada crea un entorno controlado para estudiar los efectos de diversas técnicas de filtrado en el dominio espacial. Lo cual permite obtener información sobre las fortalezas y limitaciones de los diferentes filtros. Al simular tipos de ruido específicos, podemos evaluar el rendimiento de los filtros en condiciones predefinidas, lo que permite una comprensión de su eficacia.

### P2 - Ejercicio 5 - 2/12/2023

En este ejercicio, se pretende crear una imagen cociente mediante la división en píxeles de una imagen  $I$  y su versión borrosa  $I * G_\sigma$  (donde  $G_\sigma$  es un núcleo gaussiano 2D con desviación estándar  $\sigma$ )

En el código se define una función `quotientImage()` que toma una imagen de entrada y la desviación estándar. Luego calcula la versión borrosa de la imagen utilizando un núcleo gaussiano 2D y calcula el cociente de la imagen mediante división de píxeles.

La imagen cociente puede resultar útil en varios escenarios, como mejorar bordes o detalles de la imagen, ya que resalta las diferencias entre la imagen original y su versión borrosa. Puede emplearse en tareas de procesamiento de imágenes como la extracción de características, donde es importante identificar variaciones locales.

Código:

```
def gaussian_kernel(size, sigma):
    #Generate a 2D Gaussian kernel.
    x, y = np.mgrid[-size//2 + 1:size//2 + 1, -size//2 + 1:size//2 + 1]
    g = np.exp(-((x**2 + y**2)/(2.0*sigma**2)))
    return g / g.sum()

def quotientImage(im, sigma):
    #Compute the quotient image by dividing the image by its blurred version.
    # Create a 2D Gaussian kernel
    kernel = gaussian_kernel(6*sigma, sigma)

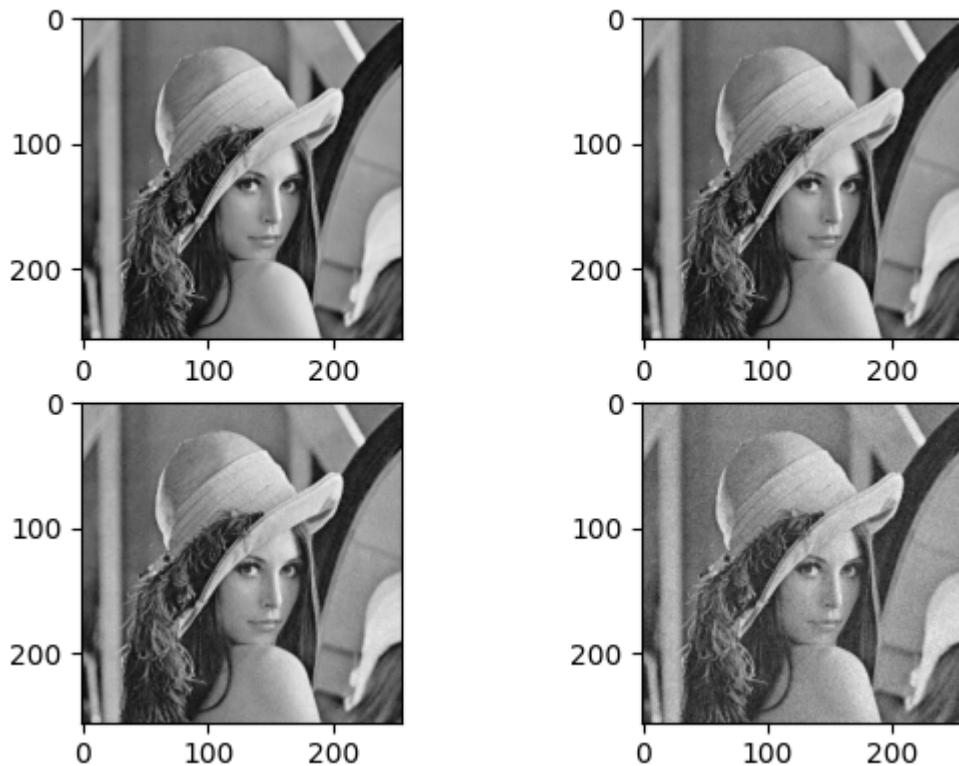
    # Blur the image using the Gaussian kernel
    blurred_im = convolve(im, kernel, mode='constant', cval=0.0)

    # Compute the quotient image by pixel-wise division
    quotient_im = im / (blurred_im + 1e-8) # Adding a small constant to avoid division by zero

    return quotient_im
```

Resultados:

quotient image,  $\sigma$ : [3, 5, 10]



## P2- Ejercicio 4 - 2/12/2023

En este ejercicio, no era necesario hacer ninguna modificación sobre la función original (`addGaussianNoise()`) ya que la función `np.random.normal()` permite trabajar con imágenes a color, siempre que se especifique el tamaño de la imagen.

Código:

```
def addGaussianNoise(im, sd=5):
    noise = np.random.normal(loc=0, scale=sd, size=im.shape)
    noisy_im = im + noise
    # Clip the values to the valid range [0, 255]
    noisy_im = np.clip(noisy_im, 0, 255)

    return noisy_im.astype(np.uint8)
```

Si quisiéramos aplicar el filtro a cada una de los canales de color, lo haríamos de la siguiente forma (aunque no es necesario):

```

def addGaussianNoise(im, sd=5):
    if len(im.shape) == 2: # Gray-level image
        return im + np.random.normal(loc=0, scale=sd, size=im.shape)

    elif len(im.shape) == 3: # Color image
        result = np.zeros_like(im, dtype=np.float64)

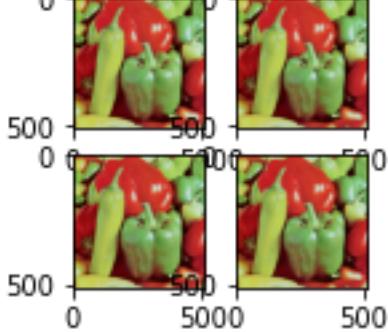
        for channel in range(im.shape[2]):
            result[:, :, channel] = im[:, :, channel] + np.random.normal(loc=0, scale=sd, size=(im.shape[0],im.shape[1]))

        result = np.clip(result, [0, 0, 0], [255, 255, 255])
        return result

```

Resultados:

Gaussian noise,  $\sigma$ : [3, 5, 10]



### P2 - Ejercicio 3 - 2/12/2023

En este ejercicio, primero hemos utilizado la función `scipy.signal.windows.gaussian()` para generar un vector gaussiano 1D (`gv1d`). A continuación, se ha generado una matriz gaussiana 2D (`gv2d`). Finalmente, se ha utilizado `gv2d` como máscara (kernel) para convolucionar la imagen. Además, se ha aplicado una convolución con dos máscaras 1D (versión separable).

El objetivo, por tanto, es mostrar cómo realizar una convolución 2D y separable con filtros gaussianos, partiendo de vectores gaussianos 1D y generando matrices gaussianas 2D.

Código:

```

def gaussianFilterExplicit(im, sigma=5):
    # (a) Generate 1D Gaussian vector
    n = 101 # Size of the vector
    gv1d = signal.windows.gaussian(n, sigma)

    # (b) Generate 2D Gaussian matrix
    gv2d = np.outer(gv1d, gv1d)

    # (c) Convolve the image using gv2d as a mask
    im_convolved_2d = convolve(im, gv2d, mode='constant', cval=0.0)

    # (d) Apply convolution with two 1D masks
    im_convolved_sep = convolve2d(convolve2d(im, gv1d.reshape(-1, 1), mode='same', boundary='fill', fillvalue=0.0),
                                  gv1d.reshape(1, -1), mode='same', boundary='fill', fillvalue=0.0)

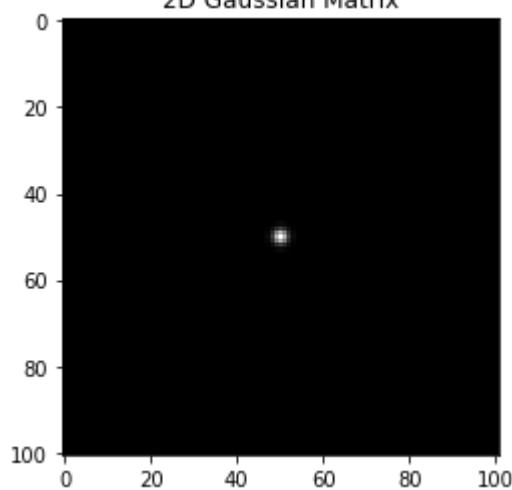
```

Resultados:

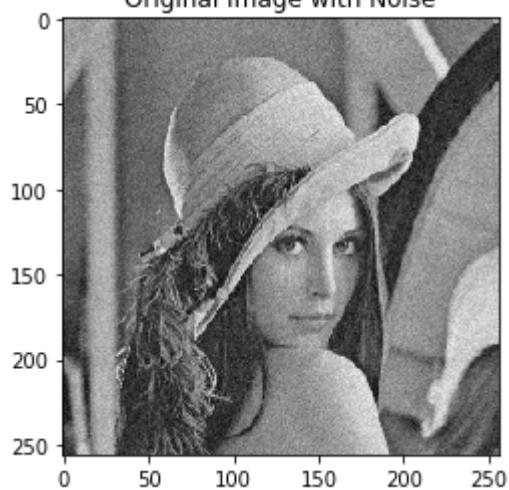
1D Gaussian Vector

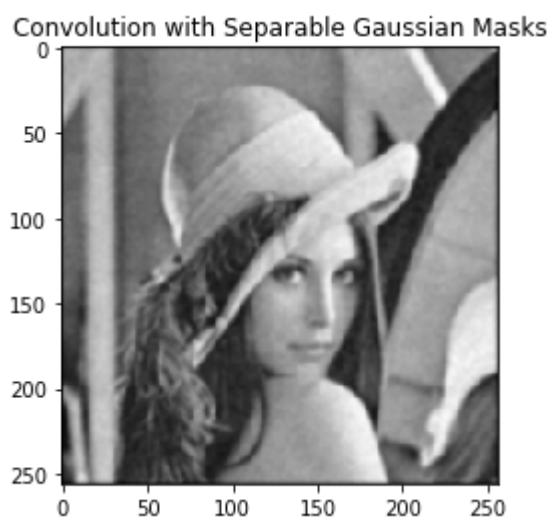
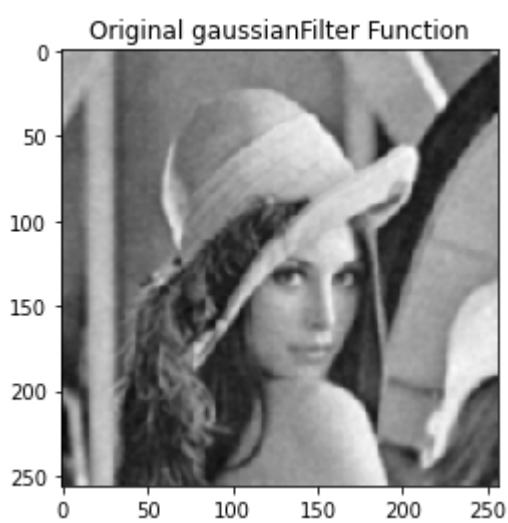
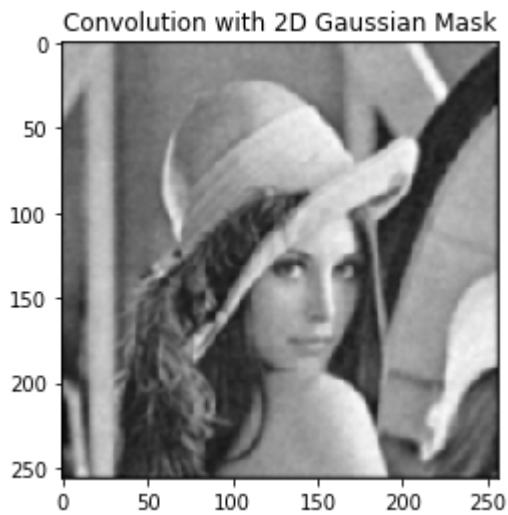


2D Gaussian Matrix



Original Image with Noise





```

def testAverageFilter(im_clean, params):
    runtimes_2d = []
    runtimes_sep = []
    imgs = []
    imgsep = []
    results = []
    for sp_pctg in params['sp_pctg']:
        im_dirty = addSPNoise(im_clean, sp_pctg) # salt and
        for filterSize in params['filterSizes']:
            imgs.append(np.array(im_dirty))
            start_time = time.time()
            im = averageFilter(im_dirty, filterSize)
            imgs.append(im)
            elapsed_time = time.time() - start_time
            runtimes_2d.append(elapsed_time)

            imgsep.append(np.array(im_dirty))
            start_time = time.time()
            imsep = averageFilterSep(im_dirty, filterSize)
            imgsep.append(imsep)
            elapsed_time = time.time() - start_time
            runtimes_sep.append(elapsed_time)

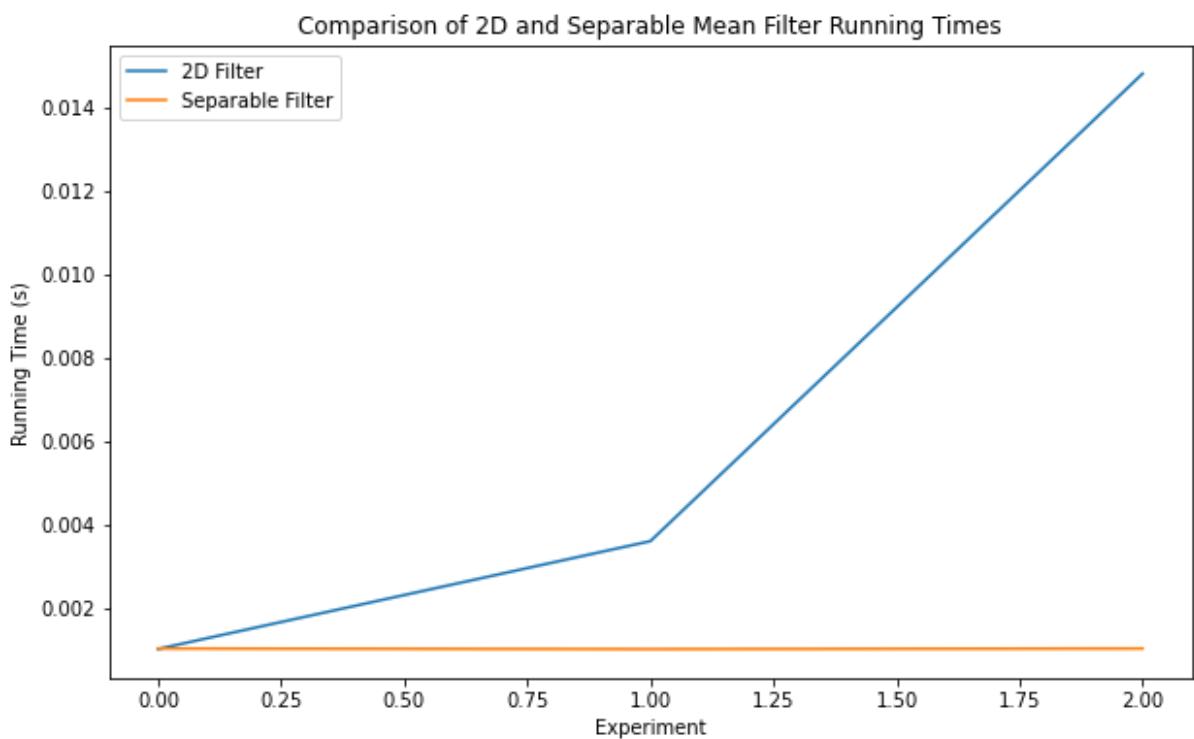
            results.append({
                'Original with Noise': im_dirty,
                '2D Filtered Image': im,
                'Separable Filtered Image': imsep
            })

    plotRunningTimes(runtimes_2d, runtimes_sep)
    plotResults(results)

    return imgs

```

Resultados:





## P2 - Ejercicio 2 - 2/12/2023

En este ejercicio se define una nueva función que aplica el filtro medio a una imagen aplicando un filtro 1D sobre las filas y, a continuación, sobre las columnas. `convolve1d()` se utiliza para realizar convoluciones 1D a lo largo de filas y columnas por separado. El parámetro del eje (`axis`) se establece en 0 para filas y 1 para columnas. Además se monitorizan los tiempos de ejecución de cada modo de filtrado para los distintos tamaños de filtro. Finalmente, se representan los resultados.

Código:

```
def averageFilterSep(im, filterSize):
    # Create a 1D filter along rows
    row_filter = np.ones((filterSize,)) / filterSize

    # Convolve along rows
    im_filtered_rows = convolve1d(im, row_filter, axis=0, mode='constant', cval=0.0)

    # Convolve along columns using the transposed result from the previous step
    im_filtered_sep = convolve1d(im_filtered_rows, row_filter, axis=1, mode='constant', cval=0.0)

    return im_filtered_sep
```

## P1 - Reflexiones de aprendizaje - 30/11/2023

A medida que profundizamos en este práctica, se hace evidente la importancia de la programación en Python y el uso de bibliotecas como NumPy en el contexto de la visión por computador.

Python, con su simplicidad y legibilidad, es un lenguaje de programación ideal para implementar herramientas de procesamiento de imágenes. Además, la integración de NumPy agiliza aún más la manipulación de datos de imágenes, facilitando el desarrollo de código eficiente.

En conclusión, la primera práctica de laboratorio sirve como una sólida introducción a la relación entre la mejora de imágenes y la programación Python en visión por computador.

## P1 - Ejercicio 5 - 30/11/2023

Se define una función `multiHist()` que calcula histogramas en diferentes niveles dividiendo recursivamente la imagen en cuatro cuadrantes en cada nivel. El caso base es cuando se

alcanza el nivel deseado (n) y calcula el histograma de la imagen actual. Luego, los histogramas se acumulan en una lista de histogramas.

Código:

```
multiHist(im, n, bins):
    histograms = []

def recursive_histogram(image, level, bins):
    # Calculate histogram at the current level
    histogram = np.histogram(image.flatten(), bins=bins)[0]
    histograms.append(histogram)

    # Base case: stop recursion when the desired level is reached
    if level == n:
        return

    # Recursive case: partition the image into four quadrants
    height, width = image.shape
    half_height, half_width = height // 2, width // 2

    # Top-left quadrant
    recursive_histogram(image[:half_height, :half_width], level + 1, bins)
    # Top-right quadrant
    recursive_histogram(image[:half_height, half_width:], level + 1, bins)
    # Bottom-left quadrant
    recursive_histogram(image[half_height:, :half_width], level + 1, bins)
    # Bottom-right quadrant
    recursive_histogram(image[half_height:, half_width:], level + 1, bins)

recursive_histogram(im, 1, bins)

return im, histograms
```

Resultado para la imagen de la iglesia con 3 bins y 2 niveles: Histograms list: [array([ 25837, 53944, 193375], dtype=int64), array([ 1811, 3782, 62593], dtype=int64), array([ 3835, 7701, 56650], dtype=int64), array([ 9006, 19530, 39856], dtype=int64), array([10921, 21955, 35516], dtype=int64)]

### P1 - Ejercicio 4 - 30/11/2023

Se define una función, en primer lugar, se asegura de que la imagen de entrada es una matriz NumPy 2D que representa una imagen en escala de grises. A continuación, crea una máscara booleana para el patrón de tablero de ajedrez de tamaño m x n, invirtiendo los píxeles en celdas alternas y reconstruyendo la imagen final del tablero de ajedrez.

La función `fromfunction()` se utiliza para crear una máscara booleana que representa el patrón de tablero de ajedrez. Luego, la función `np.where()` se utiliza para invertir píxeles selectivamente según esta máscara. Este enfoque evita bucles `for` explícitos y aprovecha las operaciones de matriz de NumPy para mejorar el rendimiento.

Código:

```
def checkBoardImg(im, m, n):
    # Ensure the image is 2D (grayscale)
    if len(im.shape) == 2:
        # Get image dimensions
        height, width = im.shape

        # Create an empty image with the same size and data type
        checkerboard_im = np.zeros_like(im)

        # Calculate the size of each cell
        cell_height = height // m
        cell_width = width // n

        # Create a boolean mask for the checkerboard pattern
        checkerboard_mask = np.fromfunction(lambda i, j: (i // cell_height + j // cell_width) % 2, (height, width), dtype=int)

        # Use the mask to invert the pixels in alternate cells
        checkerboard_im = np.where(checkerboard_mask, 255 - im, im)

    return checkerboard_im

else:
    return im
```

Resultado:



Para generalizar la función de brillo, definimos una nueva función que detecte la dimensionalidad del array de la imagen. Si se trata de un array 2D, es una imagen en escala de grises y se realizan las mismas operaciones que en la función de brillo original. Si el array es 3D, es una imagen a color y, por tanto, realizamos la misma operación de brillo pero para cada uno de los canales de la imagen. Procedemos de forma similar para la función de oscurecer.

Código:

```
def brightenImg2(im, p=2):
    if len(im.shape) == 3:
        result = np.zeros_like(im, dtype=np.float64)

        for channel in range(im.shape[2]):
            result[:, :, channel] = np.power(255.0 ** (p - 1) * im[:, :, channel], 1. / p)

    else:
        result = np.power(255.0 ** (p - 1) * im, 1. / p)

    return result
```

```
def darkenImg2(im, p=2):
    if len(im.shape) == 3:
        result = np.zeros_like(im, dtype=np.float64)

        for channel in range(im.shape[2]):
            result[:, :, channel] = (im[:, :, channel] ** float(p)) / (255 ** (p - 1))

    else:
        result = (im ** float(p)) / (255 ** (p - 1))

    return result
```

Resultado:





### P1 - Ejercicio 2 - 30/11/2023

En este ejercicio, simplemente, englobamos dentro de una función las últimas líneas del código que se encargan de convertir los arrays de salida en imágenes y de guardarlas en el directorio './imgs-out-P1/'. Para ello, le pasamos la imagen original, la imagen procesada y la transformación que se va ha hacer sobre la imagen.

Código:

```
def saveImg(imfile, im2, test):
    dirname, basename = os.path.dirname(imfile), os.path.basename(imfile)
    fname, fext = os.path.splitext(basename)
    #print(dname, basename)
    pil_im = Image.fromarray(im2.astype(np.uint8)) # from array to Image
    pil_im.save(path_output+'//'+fname + suffixFiles[test] + fext)
```

```
def doTests():
    print("Testing on", files)
    for imfile in files:
        im = np.array(Image.open(imfile)) # from Image to array

        #im = np.array(Image.open(imfile).convert('L')) # from Image to array
        for test in tests:
            out = eval(test)(im)
            im2 = out[0]
            vpu.showImgsPlusHists(im, im2, title=nameTests[test])
            if len(out) > 1:
                vpu.showPlusInfo(out[1],"cumulative histogram" if test=="testHistEq" else None)
            if bSaveResultImgs:
                saveImg(imfile, im2, test)
```

**P2 - Ejercicio 1 - 19/09/2023**

	Ruido 'salt-and-pepper'	Ruido Gaussiano	Comentarios
Filtro de media	No efectivo	Efectivo.	Puede causar borrosidad y pérdida de detalles de la imagen. Adecuado para suavizar.
Filtro Gaussiano	No efectivo	Muy efectivo	Puede causar borrosidad y pérdida de detalles de la imagen. Adecuado para suavizar.
Filtro de media	Muy efectivo	Efectivo	Conserva los bordes y los detalles