INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

Reconocimiento de Formas y Visión por Computadora

Tarea 2

Lozano Rivera Oscar

16 junio del 2022.

Elaborar un programa que permita leer un programa en Python que:

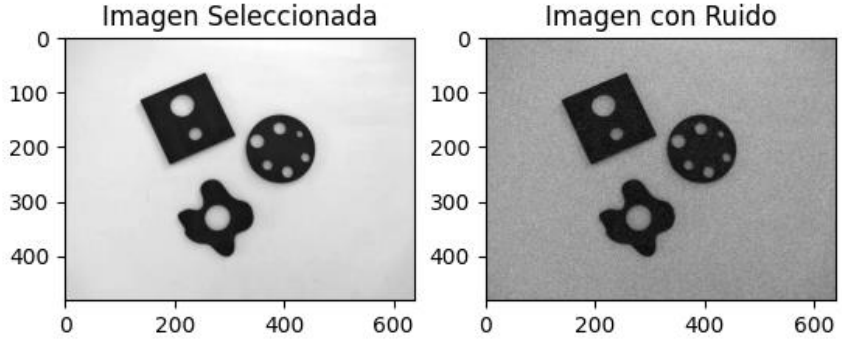
1. Permita leer imagen en niveles de gris de memoria de la computadora, que permita mostrarla en pantalla junto con sus características de tamaño y que permita agregarle algún tipo de ruido.

Código:

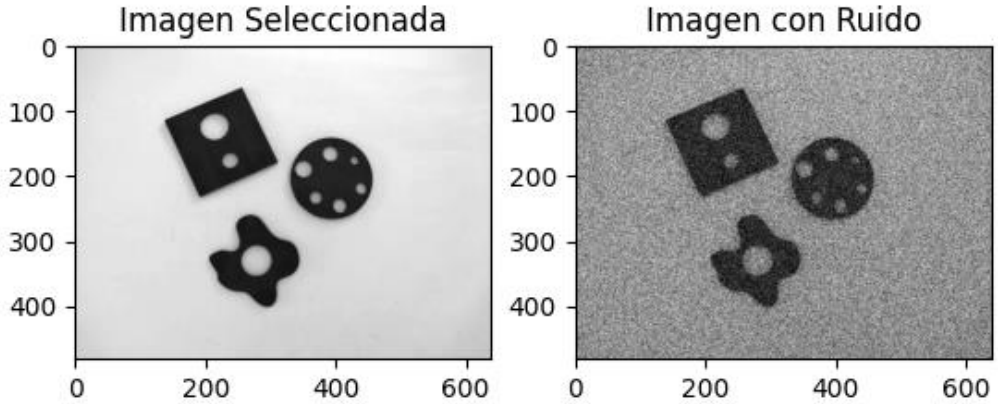
```
img = imread(imagen)                #Se carga la imagen a memoria
if opcion==1:                       #Opcion 1 seleccionada en la interfaz gráfica
    h = img.shape[0]                        #Obtener altura de la imagen en pixeles
    w = img.shape[1]                        #Obtener ancho de la imagen en pixeles
    valor=float(valorRuido.get())           #Valor de ruido a aplicar
    if ruidoSeleccionado.get()=="Gaussiano":                #Opcion Ruido Gaussiano
        cruido = skimage.util.random_noise(img, mode="gaussian", var=valor)
    elif ruidoSeleccionado.get()=="Sal":                    #Opcion Ruido Tipo Sal
        cruido = skimage.util.random_noise(img, mode="salt", amount=valor)
    elif ruidoSeleccionado.get()=="Pimienta":               #Opcion Ruido Tipo Pimienta
        cruido = skimage.util.random_noise(img, mode="pepper", amount=valor)
    elif ruidoSeleccionado.get()=="Sal y pimienta":         #Opcion Ruido Tipo Sal y Pimienta
        cruido = skimage.util.random_noise(img, mode="s&p", amount=valor ,salt_vs_pepper=0.5)
    #plt.imsave("ImagenConRuido.png",skimage.color.rgb2gray(cruido), cmap='gray') #Guardar imagen con Ruido
    fig = plt.figure()
    fig.tight_layout()
    ax1= fig.add_subplot(1,2,1)
    ax2= fig.add_subplot(1,2,2)
    titulo="Altura : " + str(h) + "       " + "Ancho: " + str(w) + "\n Tipo de Ruido:    "+ ruidoSeleccionado.get()
    titulo= titulo + "\nCantidad de ruido [0-1]:    " +  str(valor)
    fig.suptitle(titulo)                                    #Título
    ax1.imshow(skimage.color.rgb2gray(img), cmap='gray')    #Dibujar imagen seleccionada
    ax2.imshow(skimage.color.rgb2gray(cruido), cmap='gray') #Dibujar imagen con ruido
    fig.show()
```

Ejemplos:

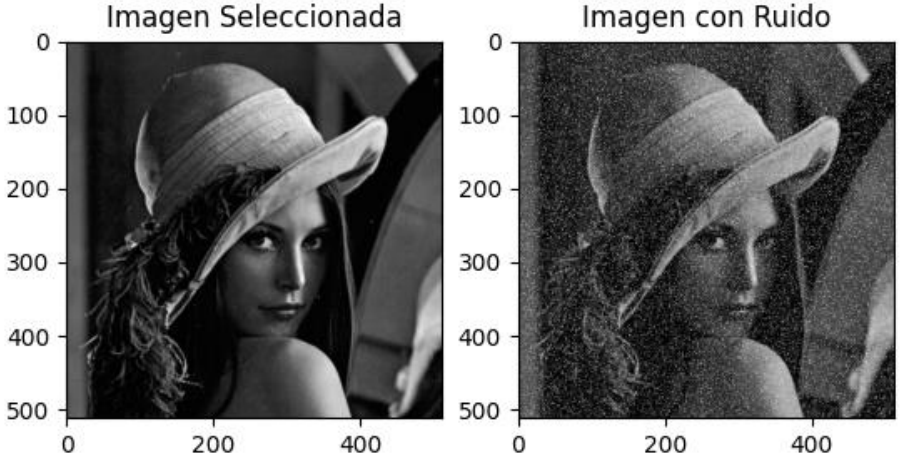
Altura : 480 Ancho: 640
Tipo de Ruido: Gaussiano
Cantidad de ruido [0-1]: 0.01



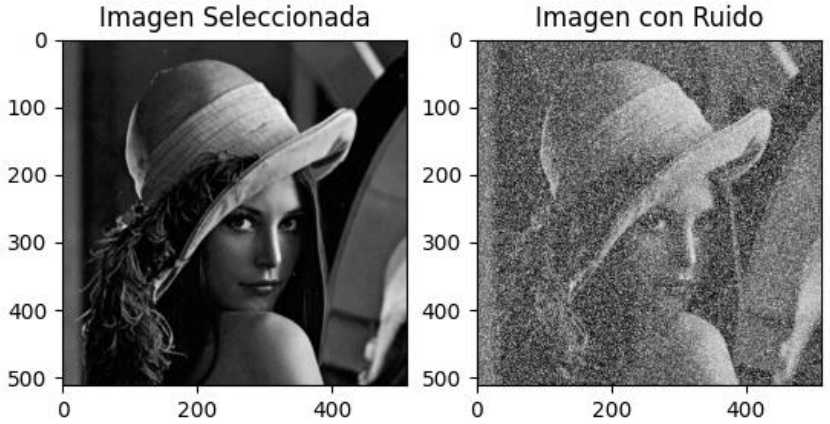
Altura : 480 Ancho: 640
Tipo de Ruido: Gaussiano
Cantidad de ruido [0-1]: 0.1



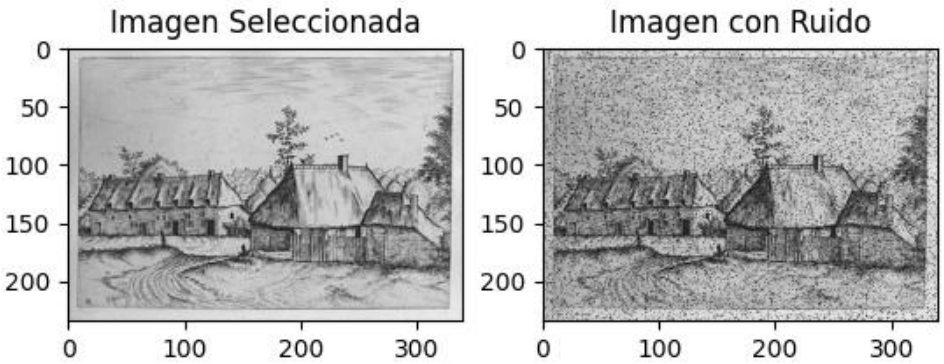
Altura : 512 Ancho: 512
Tipo de Ruido: Sal
Cantidad de ruido [0-1]: 0.1



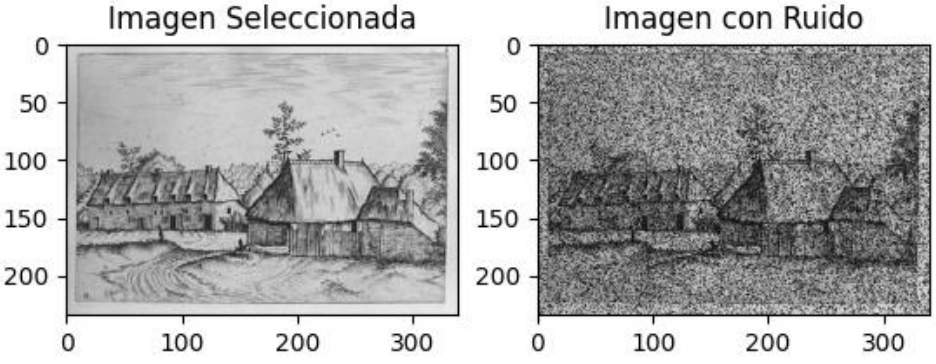
Altura : 512 Ancho: 512
Tipo de Ruido: Sal
Cantidad de ruido [0-1]: 0.3



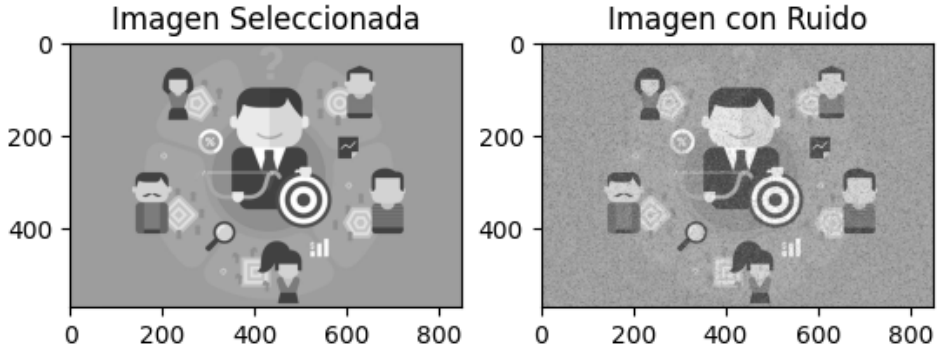
Altura : 234 Ancho: 340
Tipo de Ruido: Pimienta
Cantidad de ruido [0-1]: 0.1



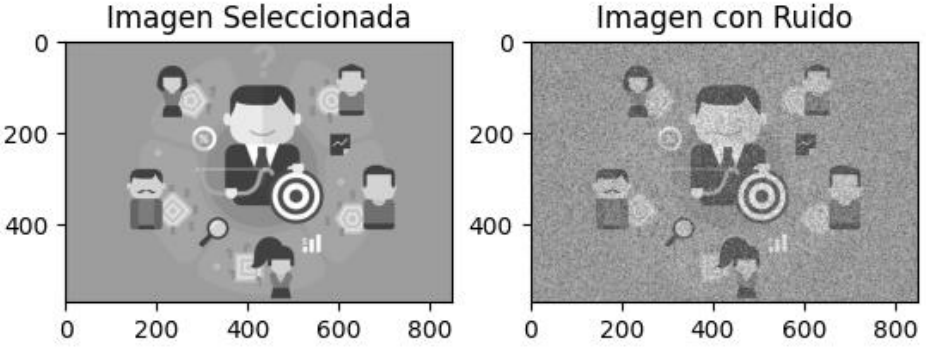
Altura : 234 Ancho: 340
Tipo de Ruido: Pimienta
Cantidad de ruido [0-1]: 0.3



Altura : 569 Ancho: 850
Tipo de Ruido: Sal y pimienta
Cantidad de ruido [0-1]: 0.1



Altura : 569 Ancho: 850
Tipo de Ruido: Sal y pimienta
Cantidad de ruido [0-1]: 0.3

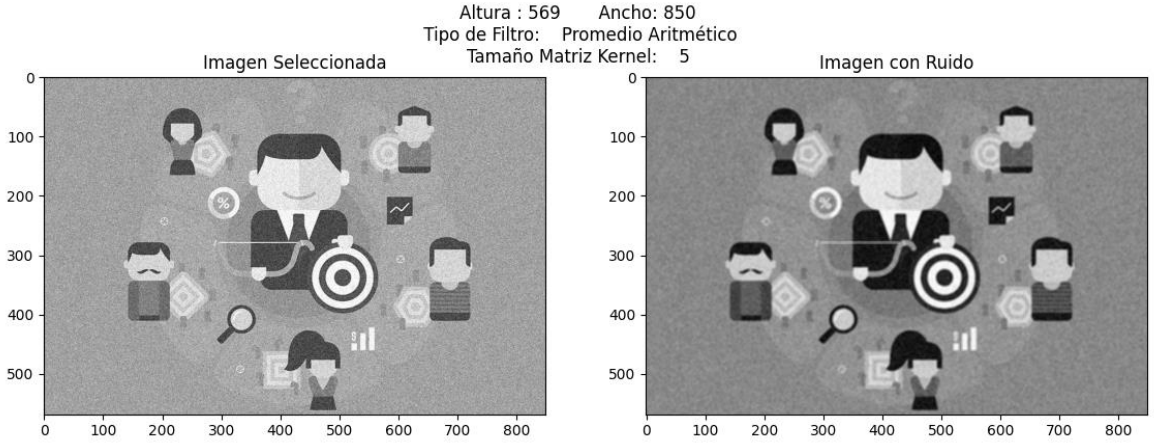


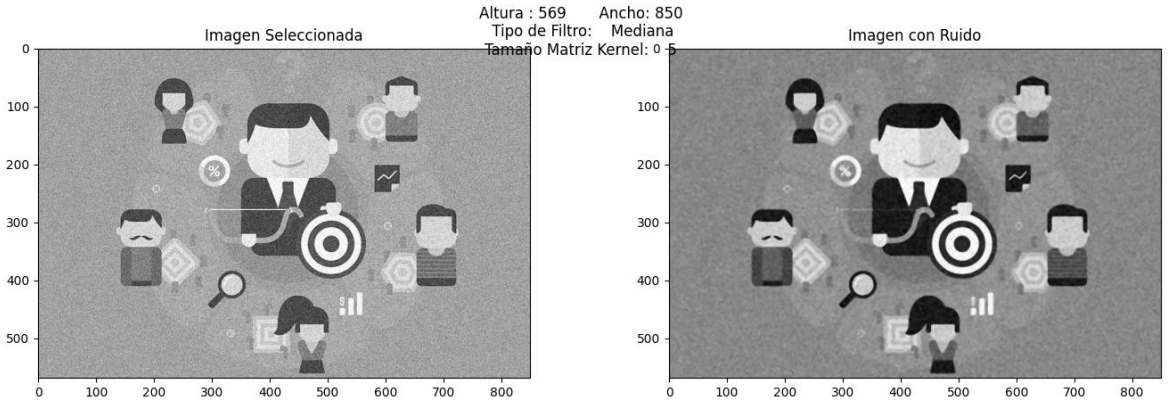
2. Permita pasar a la imagen ruido un tipo de filtro espacial.

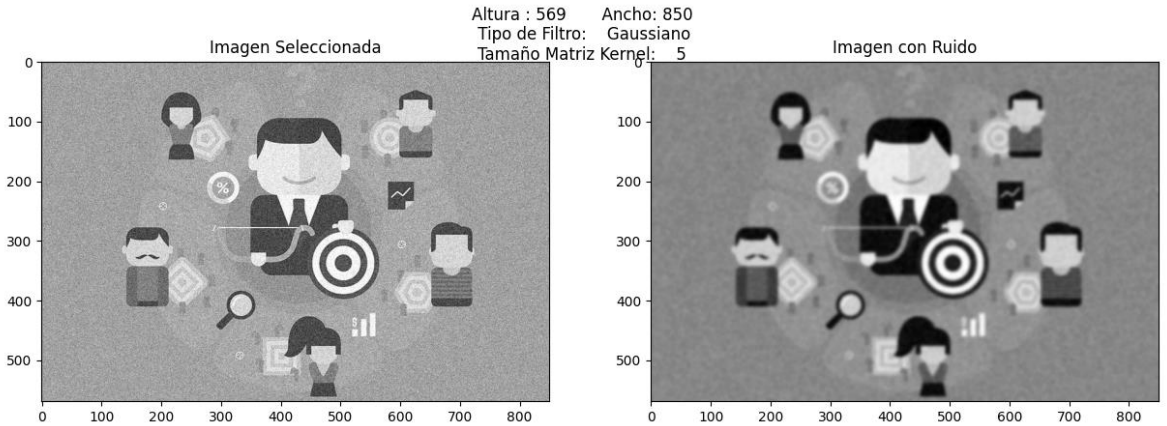
Código:

```
img = imread(imagen)                #Se carga la imagen a memoria
if opcion==2:                       #Opcion 2 seleccionada en la interfaz gráfica
    h = img.shape[0]                        #Obtener altura de la imagen en pixeles
    w = img.shape[1]                        #Obtener ancho de la imagen en pixeles
    imgrey=cvtColor(img, COLOR_BGR2GRAY)    #Convertir imagen de formato RGB a Escala de Grises
    valorKernel=int(valorFiltro.get())      #Obtener tamño de kernel
    if filtroSeleccionado.get()=="Mediana":         #Opcion Filtro Tipo Mediana
        imFiltro= cv2.medianBlur(imgrey,valorKernel)
    if filtroSeleccionado.get()=="Gaussiano":       #Opcion Filtro Gaussiano
        imFiltro= cv2.GaussianBlur(imgrey,(valorKernel*3,valorKernel*3),0)
    if filtroSeleccionado.get()=="Promedio Aritmético":         #Opcion Filtro Tipo Promedio Aritmético
        imFiltro= cv2.blur(imgrey,(valorKernel,valorKernel))
    fig = plt.figure()
    fig.tight_layout()
    ax1= fig.add_subplot(1,2,1)
    ax2= fig.add_subplot(1,2,2)
    titulo="Altura : " + str(h) + "       " + "Ancho: " + str(w) + "\n Tipo de Filtro:    "+ filtroSeleccionado.get()
    titulo= titulo + "\nTamaño Matriz Kernel:    " +  str(valorKernel)
    fig.suptitle(titulo)                                    #Título
    ax1.imshow(imgrey, cmap='gray')             #Dibujar imagen seleccionada
    ax1.set_title("Imagen Seleccionada")
    ax2.imshow(imFiltro, cmap='gray')       #Dibujar imagen con filtro aplicado
    ax2.set_title("Imagen con Ruido")
    fig.show()
```

Ejemplos:

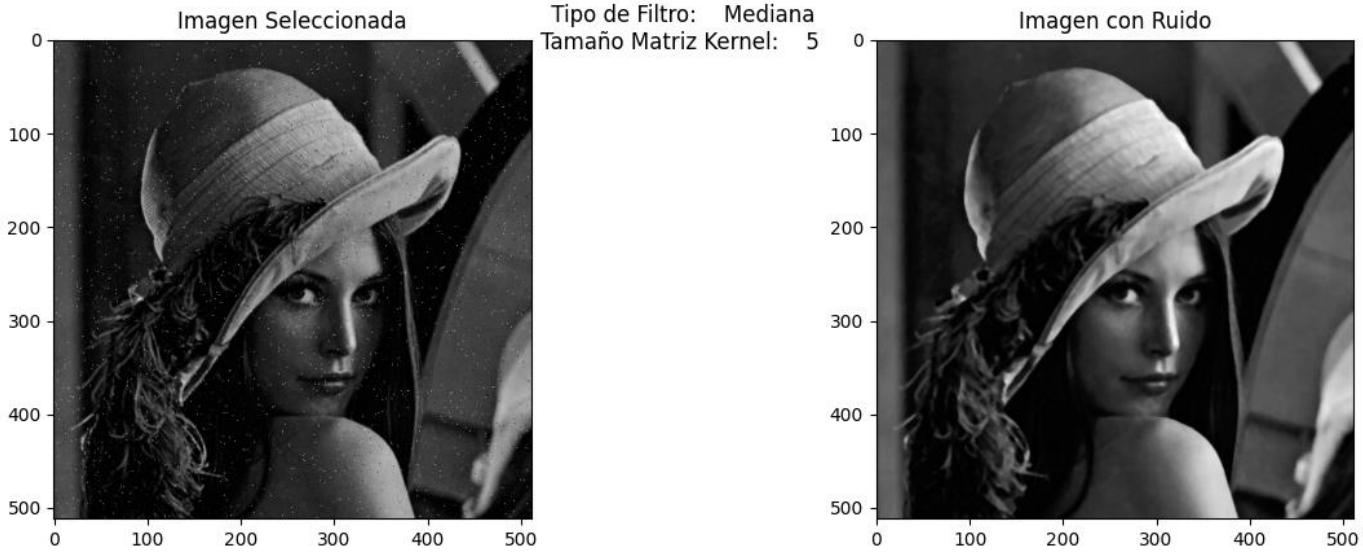




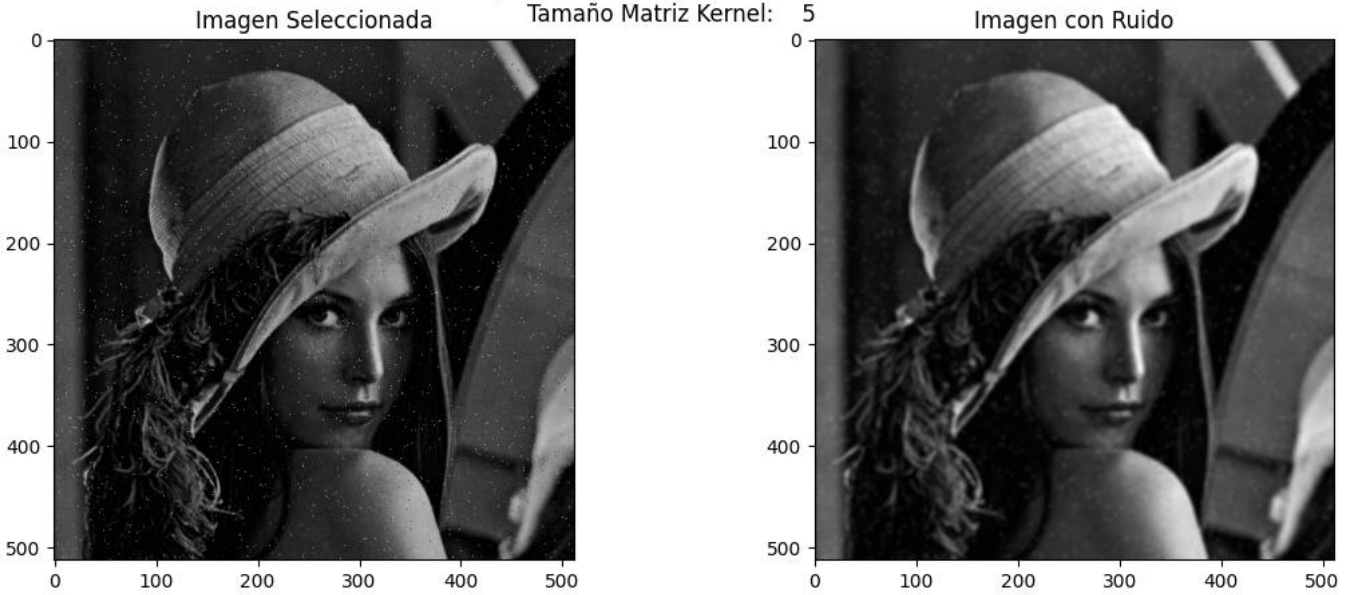


A esta imagen se le aplicó un ruido gaussiano, y después un filtro de cada tipo (aritmético, mediana y gaussiano).
Se puede observar que ninguno de estos filtros pudo reducir el efecto del ruido gaussiano, y cuando el tamaño de la matriz aumenta, la imagen se difumina aún más.

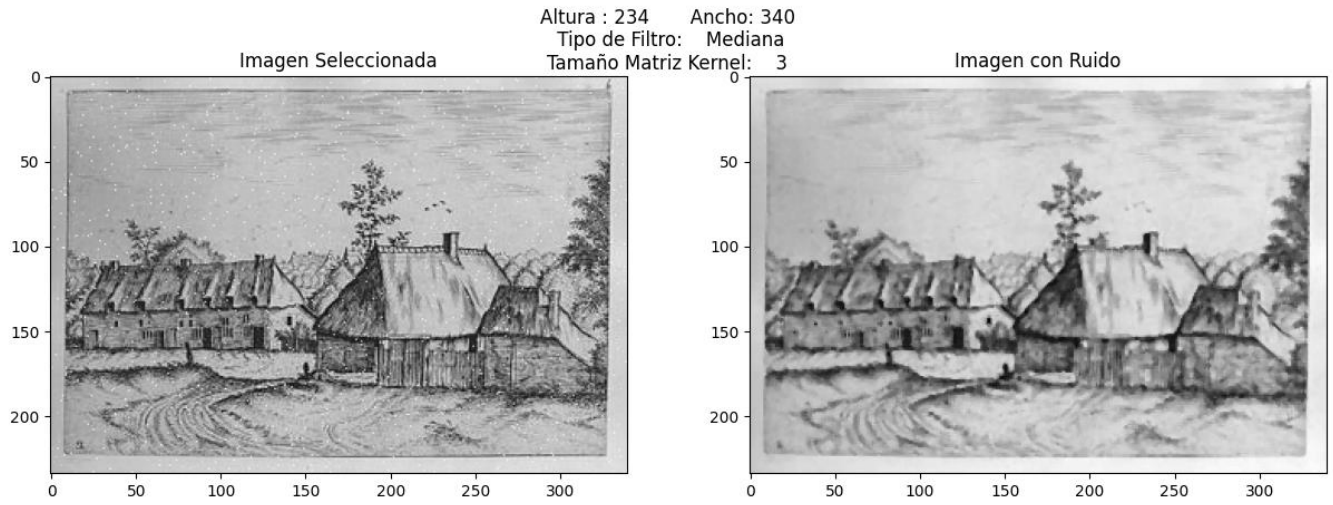
Altura : 512 Ancho: 512
Tipo de Filtro: Mediana
Tamaño Matriz Kernel: 5



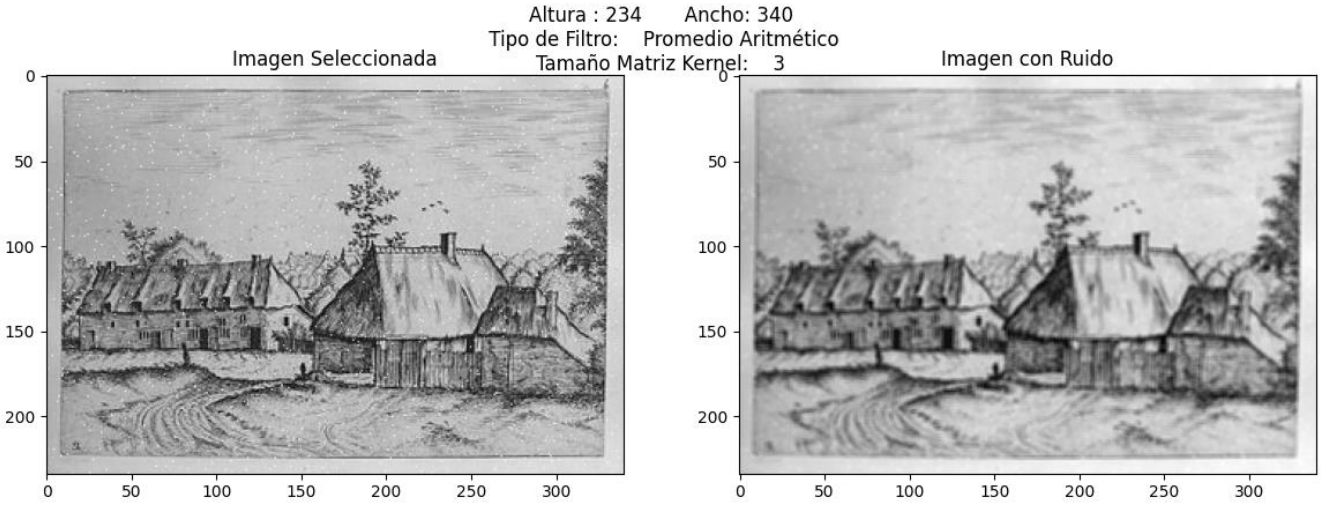
Altura : 512 Ancho: 512
Tipo de Filtro: Promedio Aritmético
Tamaño Matriz Kernel: 5



En esta imagen se encontraba un ruido de tipo sal y pimienta. El filtro mediana fue el que mejor redujo el ruido hasta con un kernel de tamaño 5x5, después se empezaba a difuminar. El filtro promedio aritmético reducía bien el ruido pimienta, pero el ruido tipo sal no.



Aquí se presenta una imagen con ruido tipo sal.
Como en el ejemplo anterior, el filtro tipo mediana reduje de mejor forma el ruido sal.



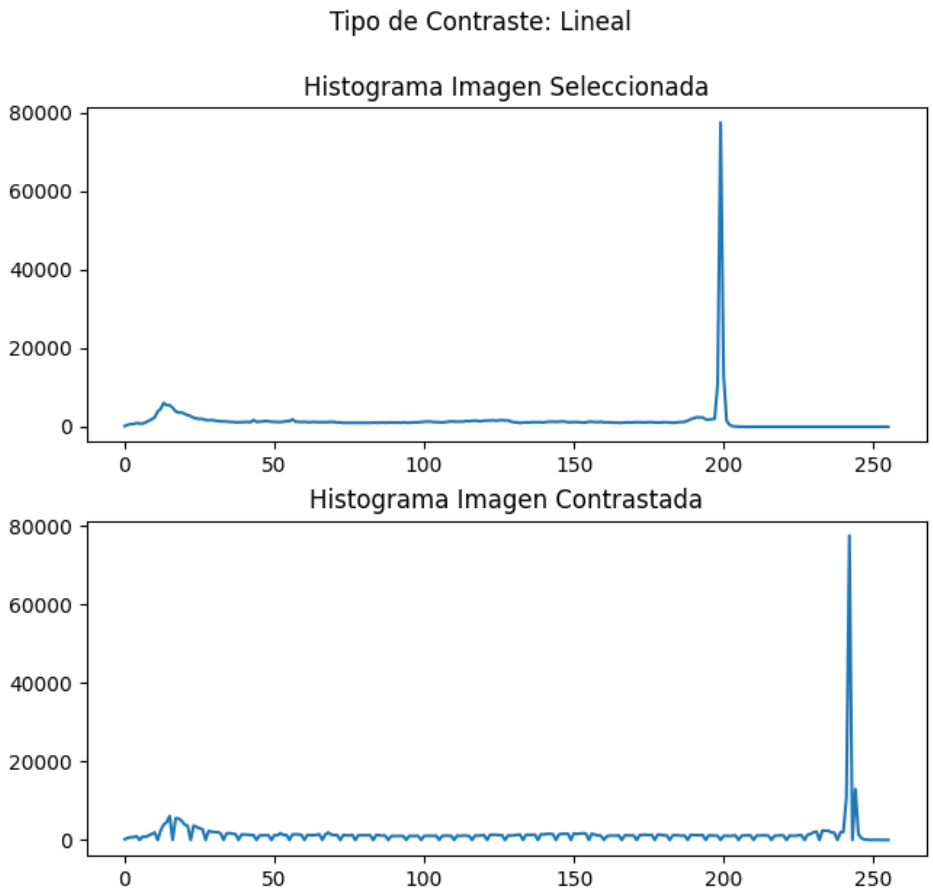
Mientras más chico sea el kernel del filtro, mejor detalle podrá retener la imagen, y mientras más grande sea, mayor difuminación tendrá la imagen resultante.
También noté que mientras más porcentaje de ruido tenga la imagen, más difícil será que algún filtro repare la imagen.

3. Permita aplicar sobre la imagen leída algún tipo de contrastado, puede ser lineal o no lineal.

Código:

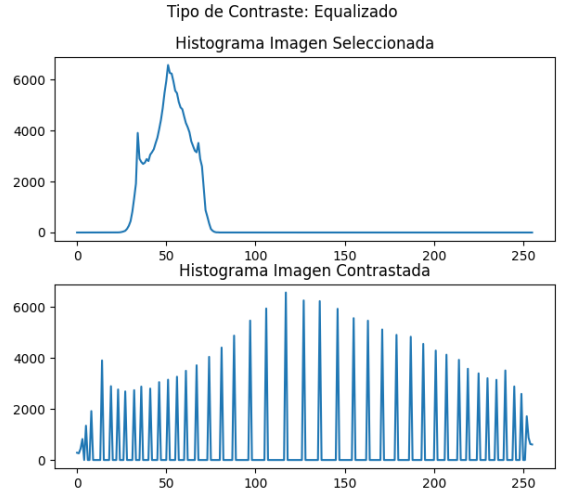
```
img = imread(imagen)                #Se carga la imagen a memoria
if opcion==3:                       #Opcion 3 seleccionada en la interfaz gráfica
    imgrey=cvtColor(img, COLOR_BGR2GRAY)    #Convertir imagen de formato RGB a Escala de Grises
    if contrasteSeleccionado.get()=="Lineal":
        imgrey=cvtColor(img, COLOR_BGR2GRAY)    #Convertir imagen de formato RGB a Escala de Grises
        arreglo= np.asarray(imgrey)
        #Obtener nivel máximo y mínimo de nivel de gris
        maximo=0
        minimo=255
        nuevo = deepcopy(arreglo)

        for fil,array in enumerate(nuevo):
            for col,a in enumerate(array):
                if a > maximo:
                    maximo = a

        for fil,array in enumerate(nuevo):
            for col,a in enumerate(array):
                if a < minimo:
                    minimo = a
        #Ajustar los valores del histograma para realizar un contraste lineal
        for fil,array in enumerate(nuevo):
            for col,a in enumerate(array):
                nuevo[fil,col]=int(a-minimo)*((255)/(maximo-minimo))

    if contrasteSeleccionado.get()=="Equalizado":
        nuevo = equalizeHist(imgrey)     #Funcion para equalizar el histograma

    hist = calcHist([imgrey], [0] , None, [256], [0,256]) #Calcular histograma de imagen seleccionada
    hist2 = calcHist([nuevo], [0] , None, [256], [0,256])   #Calcular histograma de imagen contrastada

    fig, axs = plt.subplots(2,2)
    fig.tight_layout()
    ax1= axs[0,0]
    ax2= axs[0,1]
    ax3= axs[1,0]
    ax4= axs[1,1]
    titulo="Tipo de Contraste: "+ contrasteSeleccionado.get()
    fig.suptitle(titulo)                                    #Título
    ax1.imshow(imgrey, cmap='gray')            #Dibujar imagen seleccionada
    ax1.set_title("Imagen Seleccionada")
    ax2.plot(hist)                              #Dibujar histograma imagen seleccionada
    ax2.set_title("Histograma Imagen Seleccionada")
    ax3.imshow(nuevo, cmap='gray')              #Dibujar imagen contrastada
    ax3.set_title("Imagen Contrastada")
    ax4.plot(hist2)                             #Dibujar histograma imagen contrastada
    ax4.set_title("Histograma Imagen Contrastada")
    fig.show()
```

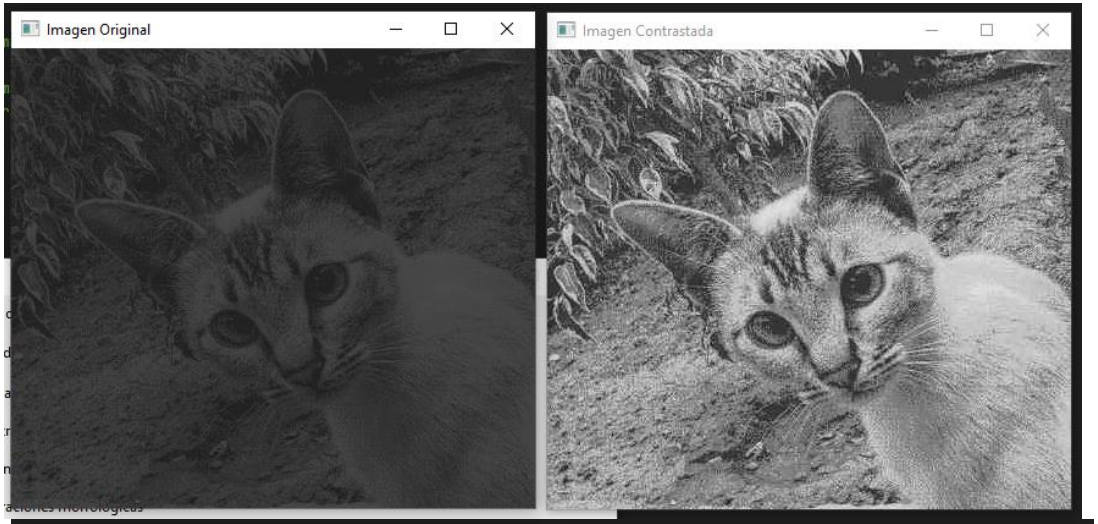
Ejemplos:

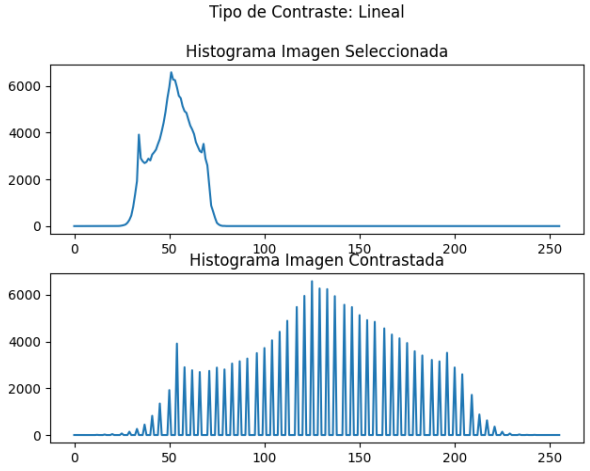


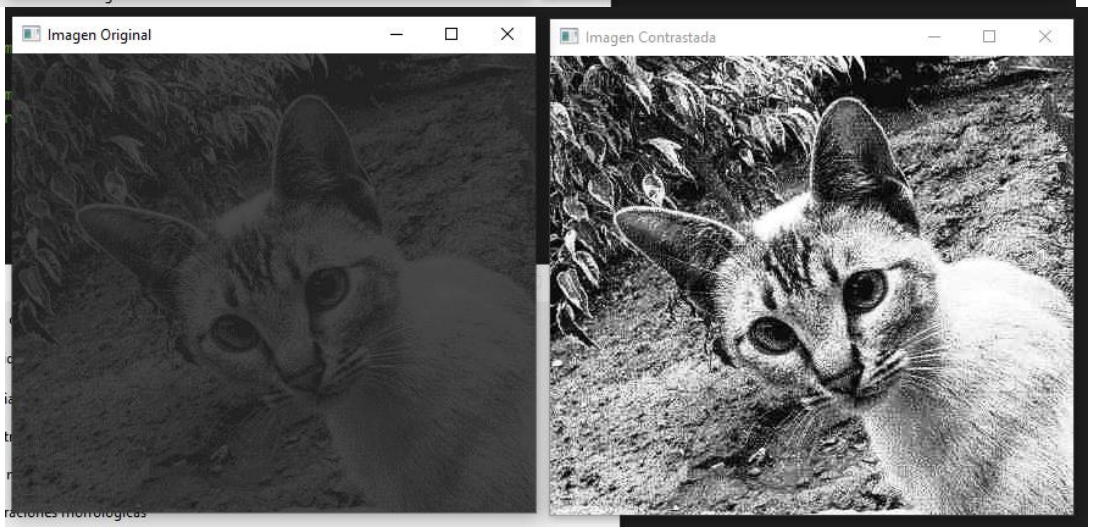
El tipo de contraste lineal permite que las zonas claras sean más claras, y las oscuras más oscuras; al hacer que los valores del histograma cubran completamente el rango de valores en la escala de grises. Esto se conoce como ensanchamiento de histograma.

El tipo de contraste lineal permite que las zonas claras sean más claras, y las oscuras más oscuras; al hacer que los valores del histograma cubran completamente el rango de valores en la escala de grises. Esto se conoce como ensanchamiento de histograma.









El contraste ecualizador, transforma el histograma de tal forma que todos los niveles de gris tengan a la salida una distribución de probabilidad uniforme.

En los dos ejemplos anteriores se puede observar la diferencia entre los dos tipos de contraste.
El primer ejemplo con contraste ecualizador, se observa como los valores están bien repartidos a través del histograma. Mientras que en el segundo la imagen se ve que los blancos son brillantes y los negros opacos.

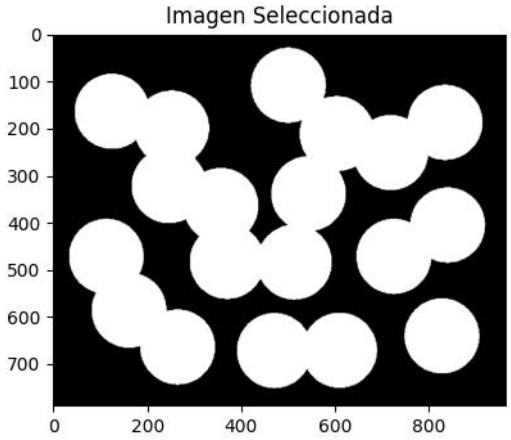
4. Permita determinar dentro de una imagen binaria el número de objetos redondos aproximadamente del mismo tamaño y en presencia de solapamientos cuantos objetos hay en dicha imagen.

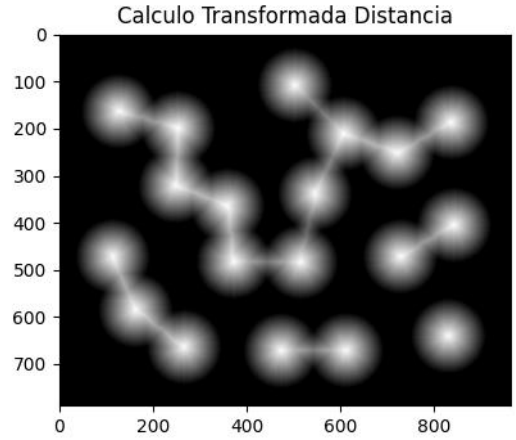
Código:

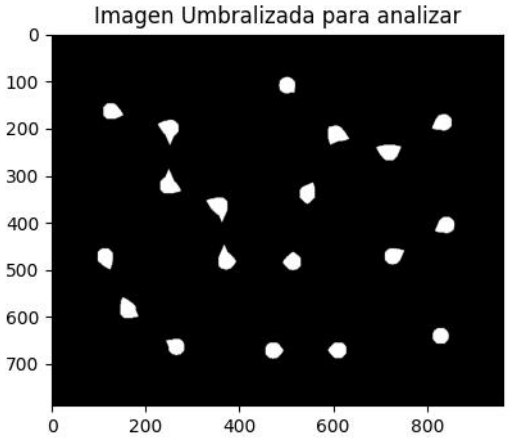
```
img = imread(imagen)                #Se carga la imagen a memoria
if opcion==4:                        #Opcion 4 seleccionada en la interfaz gráfica
    imgrey=cvtColor(img, COLOR_BGR2GRAY)    #Convertir imagen de formato RGB a Escala de Grises
    t2, imgbin = threshold(imgrey, 128, 256, THRESH_BINARY)     #Convertir imagen a binaria
    dist = distanceTransform(imgbin, DIST_L2 , 3)       #Función cálculo de la transformada distancia
    normalize(dist, dist, 0, 1.0, NORM_MINMAX)          #Normalizar la imagen resultante
    t2, imgbin2 = threshold(dist, float(umbralSel.get())/256, 1, THRESH_BINARY)     #Convertir imagen a binaria con el umbral seleccionado
    numObjetos = contarObjetos(imgbin2)                 #Contar objetos usando el método basado en vecindades

    fig, axs = plt.subplots(1,3)
    fig.tight_layout()
    ax1= axs[0]
    ax2= axs[1]
    ax3= axs[2]
    titulo="Objetos contados: " + str(numObjetos) + "\nUmbral: " + str(umbralSel.get())
    fig.suptitle(titulo)                                    #Título
    ax1.imshow(imgbin, cmap='gray')                     #Dibujar imagen seleccionada
    ax1.set_title("Imagen Seleccionada")
    ax2.imshow(dist, cmap='gray')                       #Dibujar imagen con el cálculo transformada distancia
    ax2.set_title("Calculo Transformada Distancia")
    ax3.imshow(imgbin2, cmap='gray')                    #Dibujar imagen umbralizada para analizar núm objetos
    ax3.set_title("Imagen Umbralizada para analizar")
    fig.show()
```

Ejemplos:

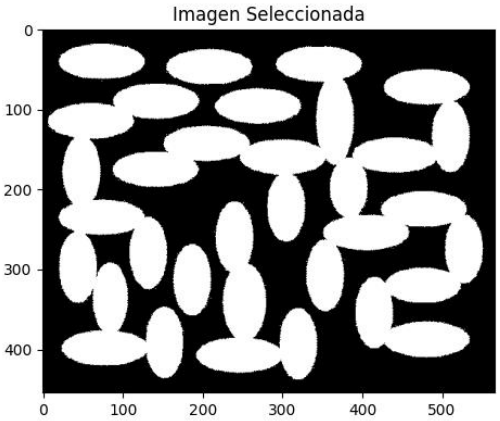
Objetos contados: 19
Umbral: 200

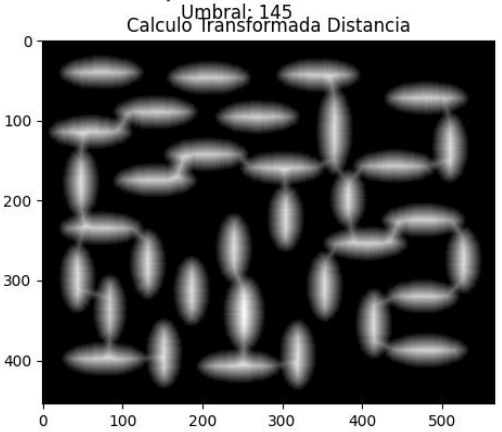


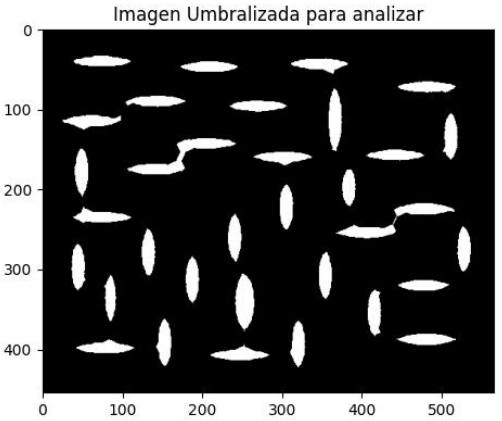




Objetos contados: 34
Umbral: 145







Este algoritmo resuelve fácilmente el encontrar objetos circulares pues se utiliza la transformada distancia para encontrar la distancia más corta de un píxel hacia el fondo de una imagen.

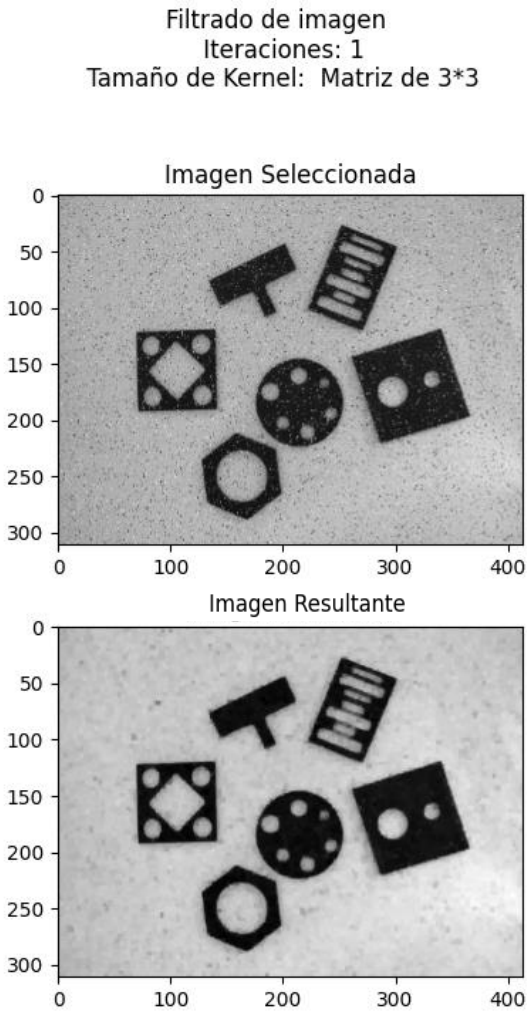
En los ejemplos anteriores podemos observar que se aplica el cálculo de la transformada distancia, después se normaliza la imagen resultante para que al umbralar la imagen se separen correctamente los objetos traslapados y se puedan contar.
Para contar los objetos se utiliza el método basado en vecindades.

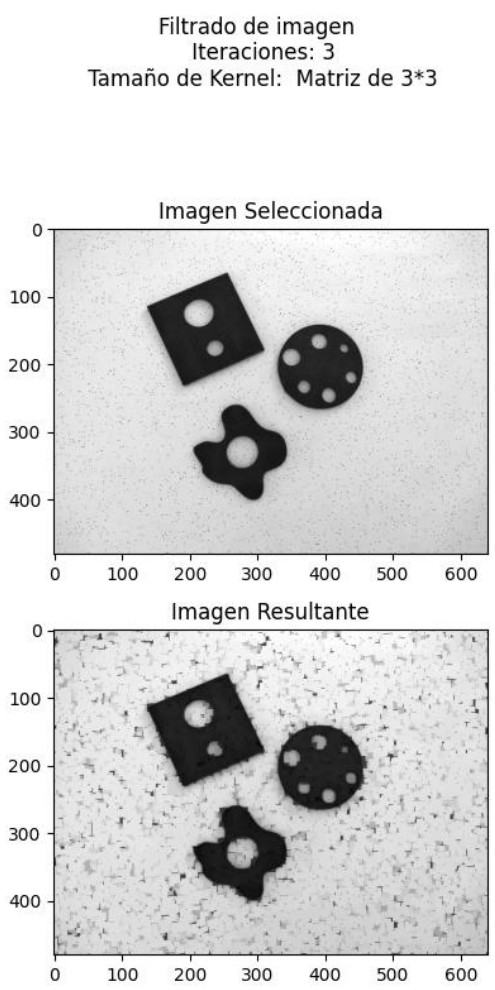
5. Permita sobre una imagen binaria o en niveles de gris, aplicar operaciones morfológicas para la realización de diversas tareas.

Código:

```
img = imread(imagen)                #Se carga la imagen a memoria
if opcion==5:                       #Opcion 5 seleccionada en la interfaz gráfica
    imgrey=cvtColor(img, COLOR_BGR2GRAY)    #Convertir imagen de formato RGB a Escala de Grises
    tamKer = int(kerSel.get())
    kernel = np.ones((tamKer,tamKer), np.uint8)     #Crear el kernel según el tamaño seleccionado
    if opSeleccionado.get()=="Filtrados":                   #Opcion Filtro
        iteraciones = int(iterSel.get())
        erosion = erode(imgrey,kernel,iterations=iteraciones)               #Erosionar y dilatar para eliminar ruido
        convertida = dilate(erosion,kernel,iterations=iteraciones)
        titulo = "Filtrado de imagen \n Iteraciones: "+ str(iteraciones) + "\n Tamaño de Kernel:  Matriz de " + str(tamKer) + "*" + str(tamKer)
    if opSeleccionado.get()=="Detección de contornos":       #Opcion Detección de contornos
        umbral = int(umbralSel.get())
        t2, imgbin = threshold(imgrey, umbral, 256, THRESH_BINARY)     #Convertir imagen a binaria
        erosion = erode(imgbin,kernel,iterations=1)             #Erosionar
        convertida = cv2.subtract(imgbin,erosion)          #Restar la imagen original a la erosionada para detectar contorno
        titulo = "Detección de contornos +\n Umbral: " +str(umbral) + "\n Tamaño de Kernel:  Matriz de " + str(tamKer) + "*" + str(tamKer)
    if opSeleccionado.get()=="Conteo de objetos":               #Opcion Conteo de objetoss
        umbral = int(umbralSel.get())
        t2, imgbin = threshold(imgrey, umbral, 256, THRESH_BINARY)     #Convertir imagen a binaria
        numIteraciones = int(iterSel.get())
        erosion = erode(imgbin,kernel,iterations=numIteraciones)               #Erosionar la imagen para separar uniones en caso de que haya
        convertida = dilate(erosion,kernel,iterations=int(numIteraciones/2))    #Dilatar para crear separaciones en un solo objeto
        numObjetos=contarObjetos(convertida)                                    #Contar objetos usando el método basado en vecindades
        titulo = "Conteo de objetos \n Objetos Contados: " + str(numObjetos)

    fig, axs = plt.subplots(2)
    ax1 = axs[0]
    ax2 = axs[1]
    fig.tight_layout()
    fig.suptitle(titulo)                                    #Título
    ax1.imshow(imgrey, cmap='gray')            #Dibujar imagen seleccionada
    ax1.set_title("Imagen Seleccionada")
    ax2.imshow(convertida, cmap='gray')       #Dibujar imagen con operaciones aplicadas
    ax2.set_title("Imagen Resultante")
    fig.show()
```

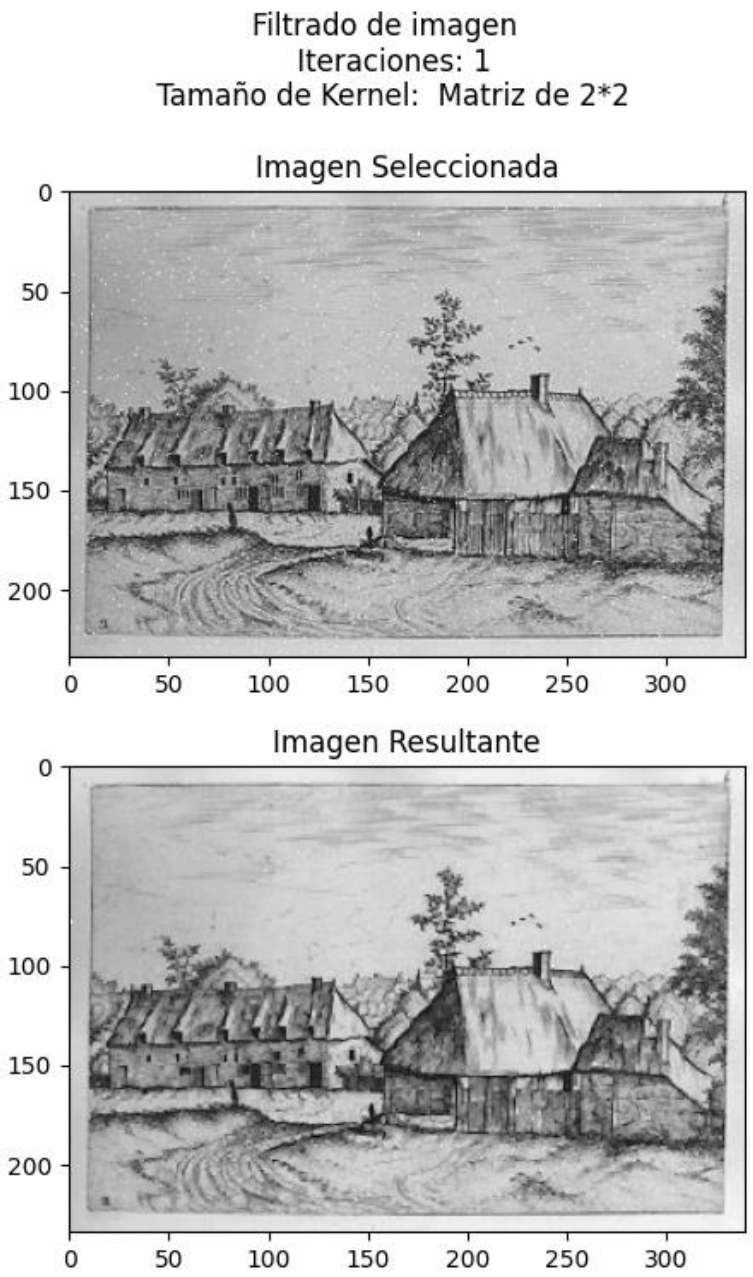
Ejemplos:

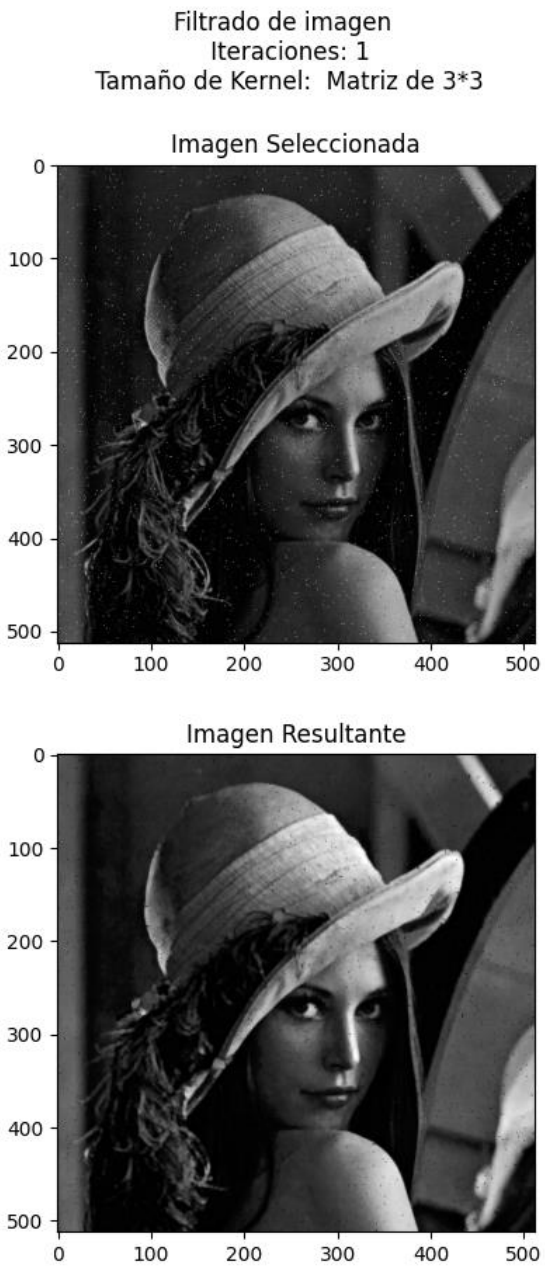




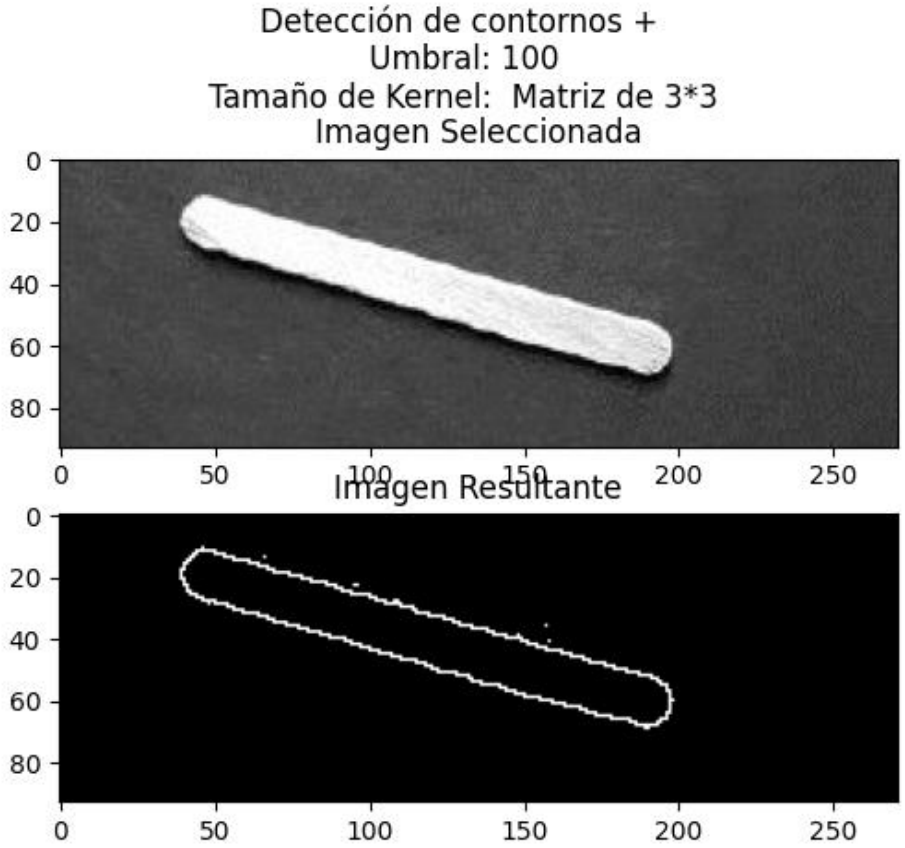
La primer aplicación de las operaciones morfológicas es Filtrado de Imágenes, Mediante erosionar y dilatar la imagen, se puede lograr en ciertos casos eliminar ruido de las imágenes,

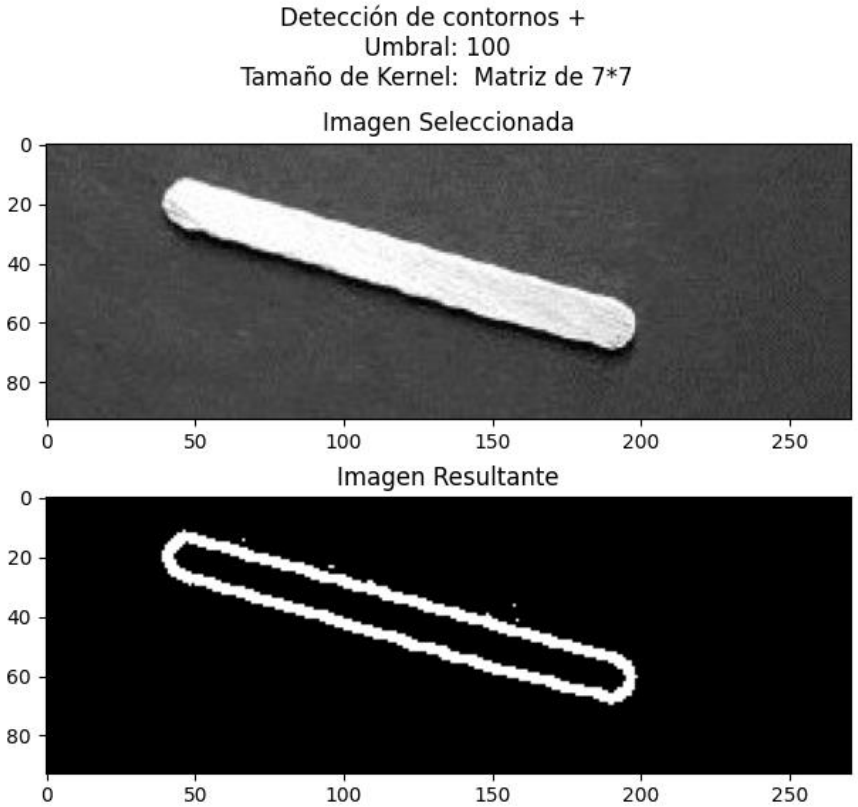
En este primer ejemplo se observa una imagen con ruido gaussiano. Al aplicar el filtro con una iteración, se elimina el ruido que se parece más al color de fondo, pero prevalece el ruido parecido al color de los objetos.
En la segunda imagen con tres iteraciones se hace más presente el ruido que quedó desde la primera iteración y empieza a distorsionarse el objeto dentro de la imagen.





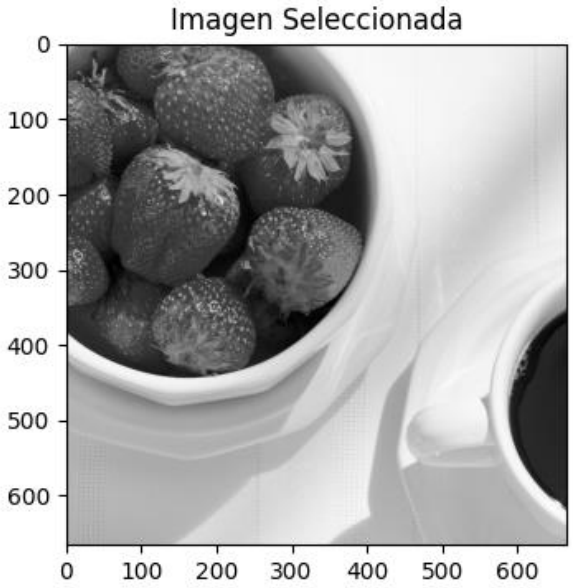
Para el ruido tipo sal presente en la primera imagen, se requiere solamente una iteración para librar la imagen de la mayoría del ruido presente.
Mientras que en el ruido tipo sal y pimienta, no se elimina el ruido pimienta,

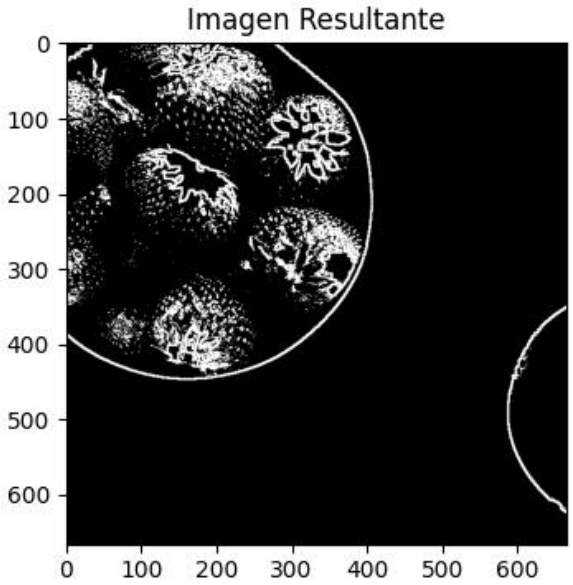




En la detección de umbrales se utilizan dos operaciones morfológicas, un erosionado y una resta de imágenes.
La primera para desgastar ligeramente el contorno de los objetos y la segunda para que al restar y umbralar la imagen solamente permanezca el borde.
Mientras más grande sea el kernel para la erosión, mayor será el borde presente después de la resta de imágenes, así como se presenta en los ejemplos anteriores.

Detección de contornos +
Umbral: 100
Tamaño de Kernel: Matriz de 7*7

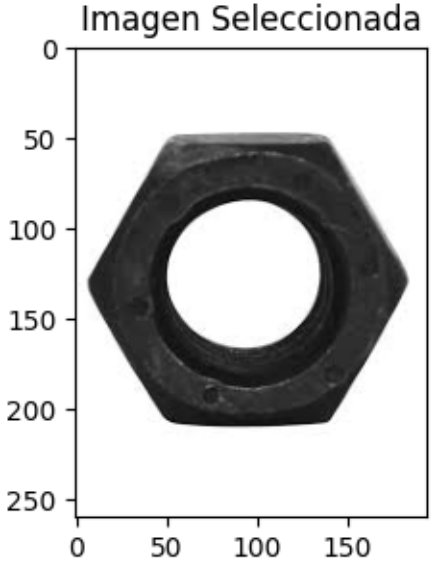


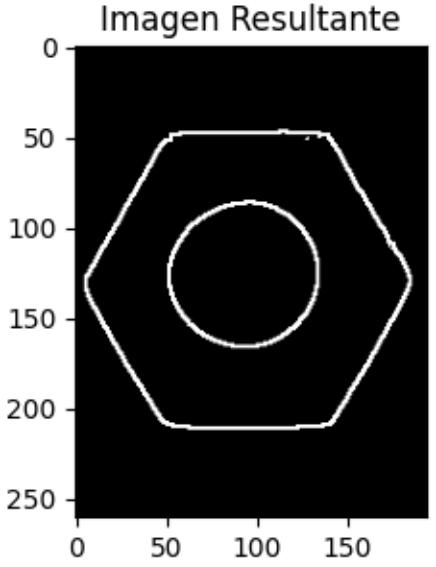


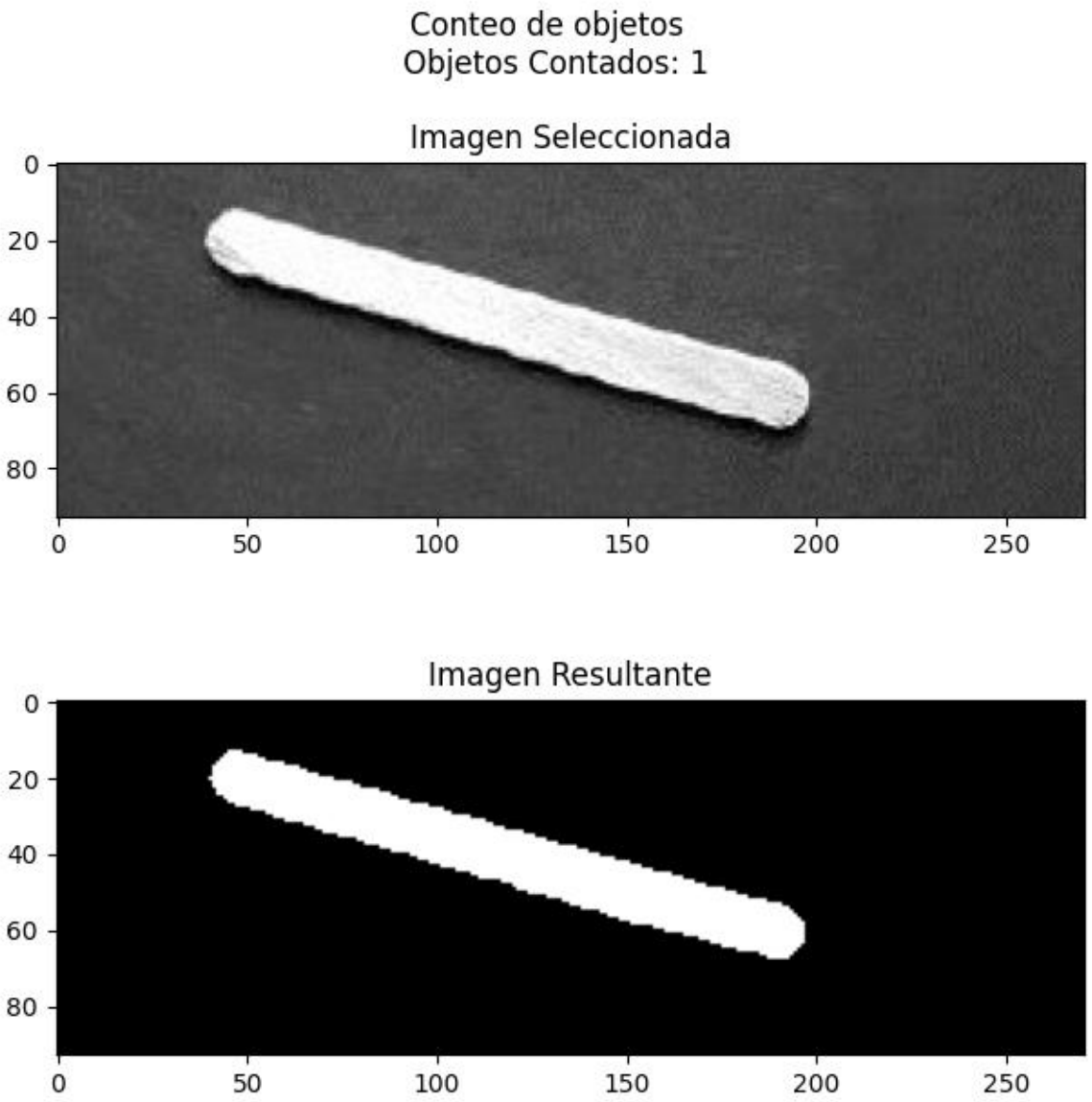
Para la imagen de las fresas no se presenta el contorno de las fresas tan claramente, pues mediante este método es difícil que se identifique, sin embargo, los contornos de las tazas si se presentan claramente. No así de los platos por la similitud con el fondo blanco.

A diferencia de la imagen de la tuerca, donde está claramente definido el contorno respecto al fondo blanco.

Detección de contornos +
Umbral: 150
Tamaño de Kernel: Matriz de 5*5



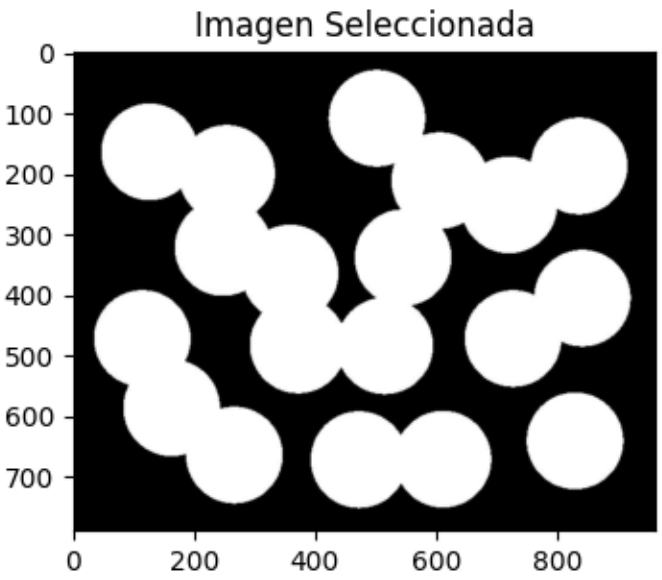


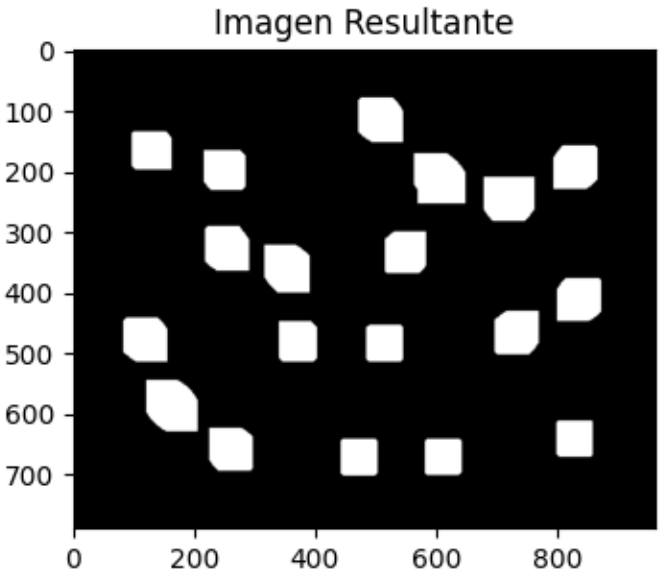


La última aplicación con operaciones morfológicas es contar objetos de una imagen. El proceso inicia umbralando la imagen, después durante un determinado número de iteraciones se erosionará la imagen para separará aquellos objetos traslapados y al finalizar se dilatará la mitad de esas veces para recuperar la forma de los objetos sin traslapes.

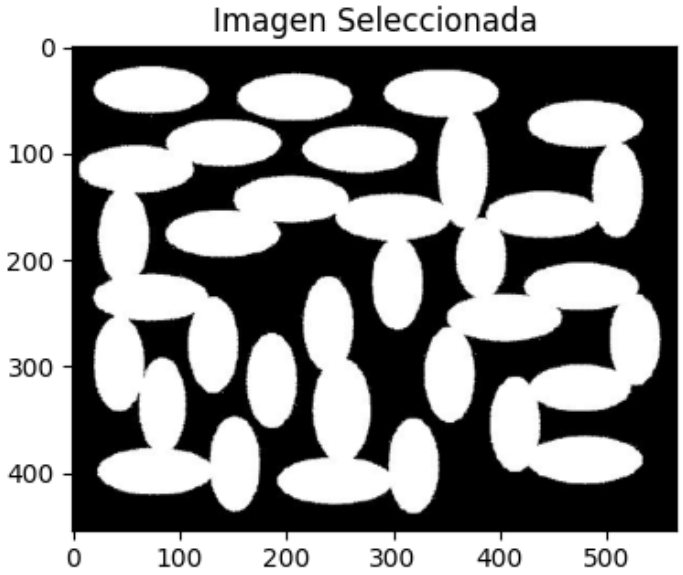
Finalmente, para determinar el número de objetos en la imagen se utilizará el método basado en vecindades,

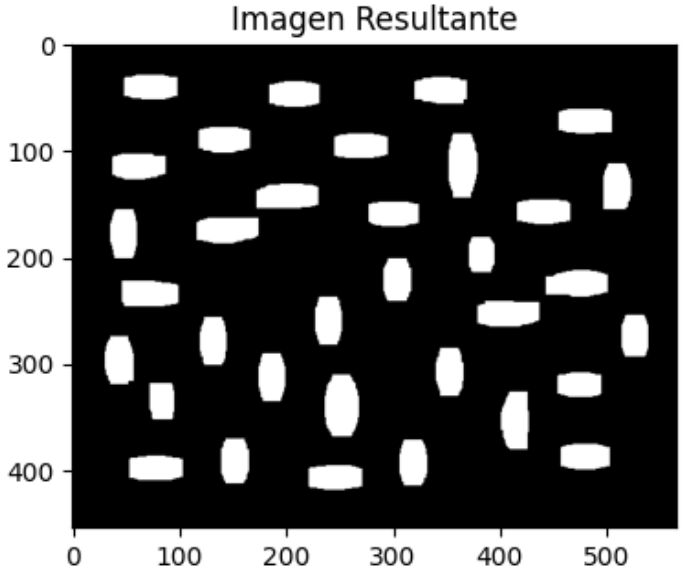
Conteo de objetos+
Iteraciones: 27
Tam Kernel:5
Objetos Contados: 19





Conteo de objetos+
Iteraciones: 17
Tam Kernel:3
Objetos Contados: 34





A diferencia del cálculo de la transformada distancia para identificar objetos, este método también sirve para objetos no solo circulares. Como se puede observar en las imágenes, se separó correctamente e identificó el número de objetos en cada imagen.

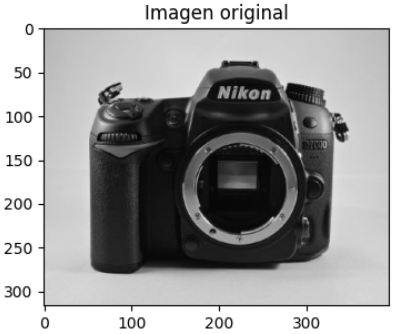
6. Programe el método de umbralado de Kapur y aplíquelo a varias imágenes en niveles de gris.

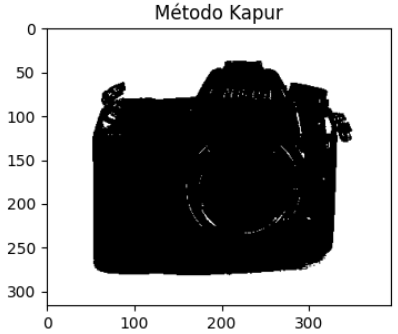
Código:

```
img = imread(imagen,0)                #Se carga la imagen a memoria
if opcion==6:                        #Opcion 6 seleccionada en la interfaz gráfica
    imgrey=imread(imagen, IMREAD_GRAYSCALE)    #Convertir imagen de formato RGB a Escala de Grises
    hist = calcHist([img], [0] , None, [256], [0,256])  #Calcular histograma en escala de grises
    h = img.shape[0]                        #Obtener altura de la imagen en pixeles
    w = img.shape[1]                        #Obtener ancho de la imagen en pixeles
    Nt=h*w                                  #Número total de pixelse
    Ht=[]
    for Pr in hist:
            Ht.append(float(Pr)/Nt)                    #Histograma normalizado por nivel de gris
    hc1= []
    hc2= []
    hc1.append(Ht[0])
    hc2.append(1-Ht[0])
    for i,a in enumerate(Ht[1:]):                  #Se calculan las entropias para cada nivel de umbral posible
            hc1.append(hc1[i-1]+Ht[i])
            hc2.append(1-hc1[i])
    umbral=0
    entropiaTotal = 0
    maxEntropia = 0
    for i,P in enumerate(hist):                #Se encuentra en que umbral la entropía tiene su valor máximo
        entropiaFondo = 0
        entropiaObjeto = 0
        for u in range(i):                              #Se calcula la entropia de ambas clases, objeto y fondo
            if float(hist[u])!=0 and hc1[i]!=0:         # se suman para encontrar la entropía total y
                A=log(Ht[u]/hc1[i])                     # definir en qué umbral la entropía es mayor
                entropiaFondo -= ((Ht[u]/hc1[i]) * A[0])
        for u in range(i+1,256):
            if float(hist[u])!=0 and hc2[i]!=0:
                #print("a",Ht[u]/hc2[i])
                B=log(Ht[u]/hc2[i])
                entropiaObjeto -= ((Ht[u]/hc2[i]) * B[0])

        entropiaTotal = entropiaFondo + entropiaObjeto
        if maxEntropia < entropiaTotal:
            maxEntropia = entropiaTotal
            umbral = i
                #Convertir imagen a binaria según el umbral encontrado
    t2, imgbin = threshold(imgrey, umbral, 256, THRESH_BINARY)
    maximo = int(max(hist))
    fig, axs = plt.subplots(1,3)
    fig.tight_layout()
    ax1= axs[0]
    ax2= axs[1]
    ax3= axs[2]
    titulo="Umbral seleccionado por método Kapur: " + str(umbral)
    fig.suptitle(titulo)                                    #Título
    ax1.imshow(imgrey, cmap='gray')            #Dibujar imagen seleccionada
    ax1.set_title("Imagen original")
    ax2.imshow(imgbin, cmap='gray')            #Dibujar imagen binarizada con el umbral encontrado
    ax2.set_title("Método Kapur")
    ax3.plot([x*umbral for x in np.ones(maximo)],np.arange(maximo))     #Se dibuja el umbral encontrado
    ax3.hist(imgrey.ravel(),256,[0,256])       #Dibujar histograma de la imagen en escala de grises
    ax3.set_title("Histograma con umbral")
    fig.show()
```

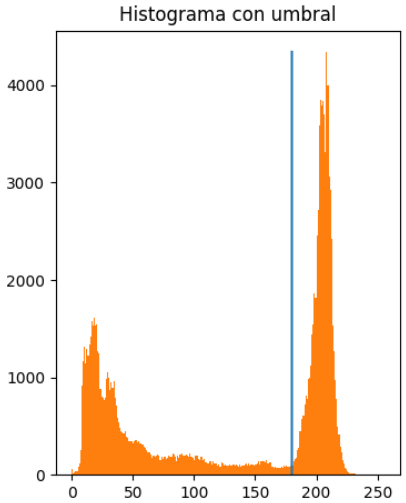
Ejemplos:

1)

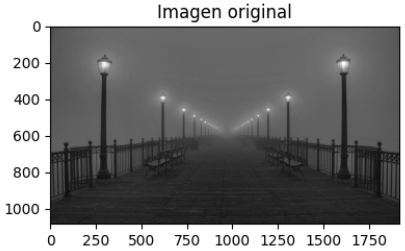


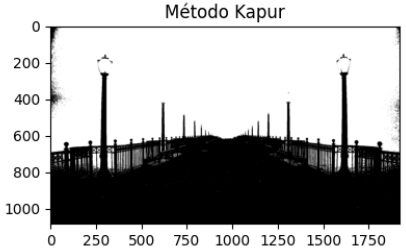


Umbral seleccionado por método Kapur: 180

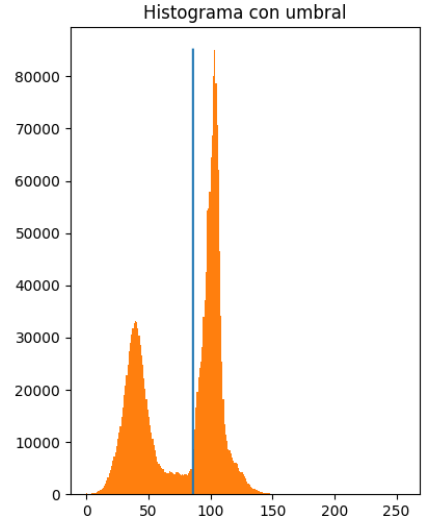


2)





Umbral seleccionado por método Kapur: 86



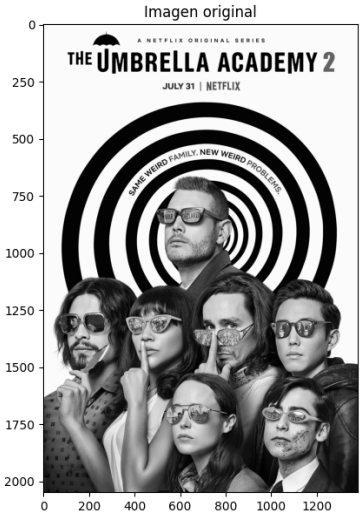
3)

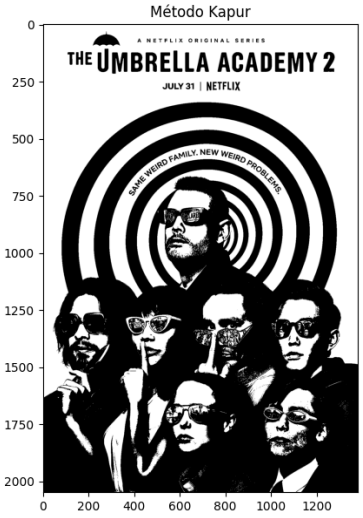
Para la implementación del método Kapur para la umbralización mediante el cálculo de la máxima entropía, se utilizaron imágenes que en su histograma de escala de grises tuviera dos clases principales.

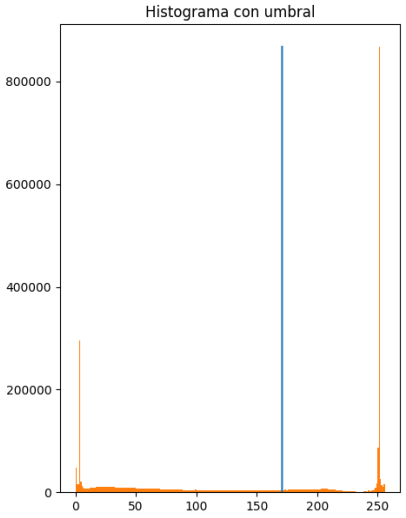
En el ejemplo 1) Se separa eficazmente el objeto del fondo, solamente las partes con más brillo de la cámara no se umbralizaron como parte del objeto.

En la segunda imagen, toda la parte del puerto se separa del fondo, así como los postes, la acera y la parte metálica de los candiles.

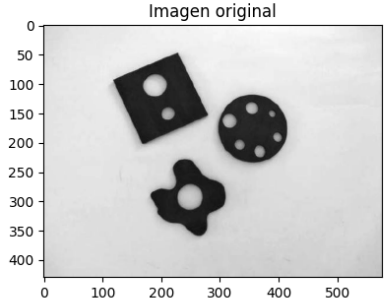
Umbral seleccionado por método Kapur: 171

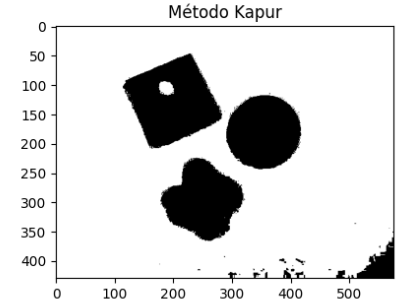


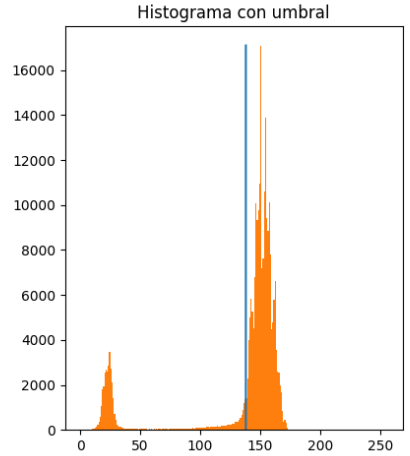




Umbral seleccionado por método Kapur: 138







4)

En el ejemplo 3) no se logró separar correctamente la silueta de las personas del fondo, por que el objeto y el fondo comparten valores en ambas clases (en ambas hay valores blancos y negros).

En la última imagen se logró separar el objeto del fondo, sin embargo, una parte del fondo se tomó como parte del objeto.

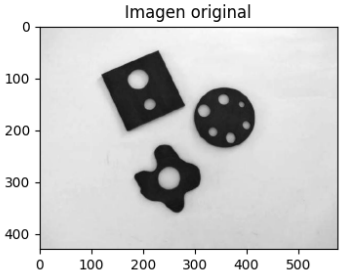
7. Programe el método de umbralado de Cheng y aplíquelo a varias imágenes en niveles de gris.

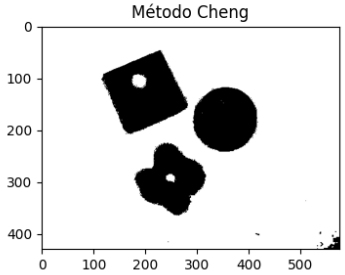
Código:

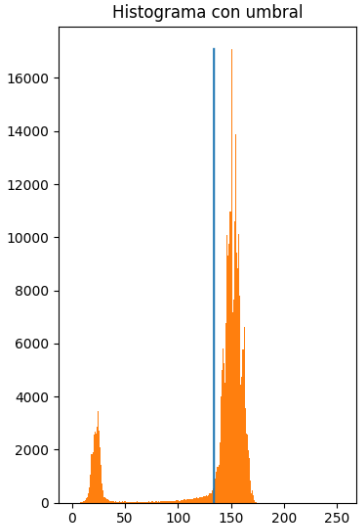
```
img = imread(imagen,0)               #Se carga la imagen a memoria
if opcion ==7:
    imgrey=imread(imagen, IMREAD_GRAYSCALE)    #Convertir imagen de formato RGB a Escala de Grises
    hist = calcHist([img], [0] , None, [256], [0,256])  #Calcular histograma en escala de grises
    h = img.shape[0]                          #Obtener altura de la imagen en pixeles
    w = img.shape[1]                          #Obtener ancho de la imagen en pixeles
    Nt=h*w                                    #Número total de pixelse
    Ht=[]

    hc1= []
    hc2= []
    hc1.append(Ht[0])
    hc2.append(1-Ht[0])
    for i,a in enumerate(Ht[1:]):                  #Se calculan las distribuciones de probabilidad para
            hc1.append(hc1[i-1]+Ht[i])             # cada nivel de umbral posible
            hc2.append(1-hc1[i])
    umbral=0
    entropiaTotal = 0
    maxEntropia = 0
    for i,P in enumerate(hist):                #Se encuentra en que umbral la correlación tiene su valor máximo
        entropiaFondo = 0
        entropiaObjeto = 0
        for u in range(i):                                  #Se calcula la correlación de ambas clases, objeto y fondo
            if float(hist[u])!=0 and hc1[i]!=0:             # se suman para encontrar la correlación total y
                A=pow(hist[u]/hc1[i],2)                         # definir en qué umbral la correlación es mayor
                entropiaFondo -= log(A)[0]
        for u in range(i+1,256):
            if float(hist[u])!=0 and hc2[i]!=0:             #No se toman los valores donde el histograma
                #print("a",Ht[u]/hc2[i])                    # no tenga incidencias
                B=pow(hist[u]/hc2[i],2)                         # definir en qué umbral la correlación es mayor
                entropiaObjeto -= log(B)[0]

        entropiaTotal = entropiaFondo + entropiaObjeto
        if maxEntropia > entropiaTotal:
            maxEntropia = entropiaTotal
            umbral = i
                #Convertir imagen a binaria según el umbral encontrado
    t2, imgbin = threshold(imgrey, umbral, 256, THRESH_BINARY)
    maximo = int(max(hist))
    fig, axs = plt.subplots(1,3)
    fig.tight_layout()
    ax1= axs[0]
    ax2= axs[1]
    ax3= axs[2]
    titulo="Umbral seleccionado por método Cheng: " + str(umbral)
    fig.suptitle(titulo)                                    #Título
    ax1.imshow(imgrey, cmap='gray')            #Dibujar imagen seleccionada
    ax1.set_title("Imagen original")
    ax2.imshow(imgbin, cmap='gray')            #Dibujar imagen binarizada con el umbral encontrado
    ax2.set_title("Método Cheng")
    ax3.plot([x*umbral for x in np.ones(maximo)],np.arange(maximo))     #Se dibuja el umbral encontrado
    ax3.hist(imgrey.ravel(),256,[0,256])       #Dibujar histograma de la imagen en escala de grises
    ax3.set_title("Histograma con umbral")
    fig.show()
```

Ejemplos:

Umbral seleccionado por método Cheng: 134



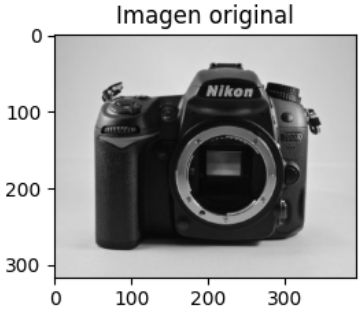


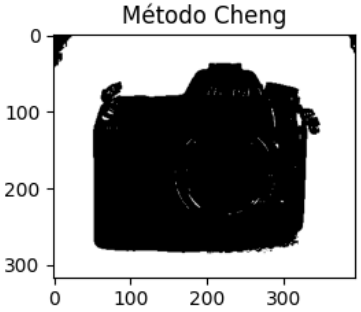


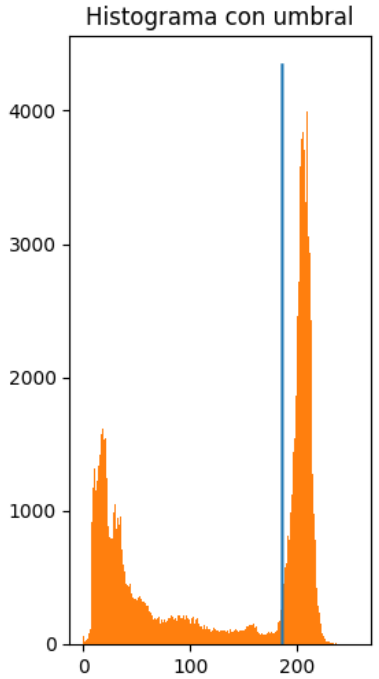
El umbralado con el método de Cheng arrojó resultados bastante parecidos a los encontrados con el método de Kapur.
En la primera y segunda imagen, se separa bastante bien el objeto del fondo.

En la tercera imagen, hace un muy buen trabajo separando la silueta completa de las personas del fondo

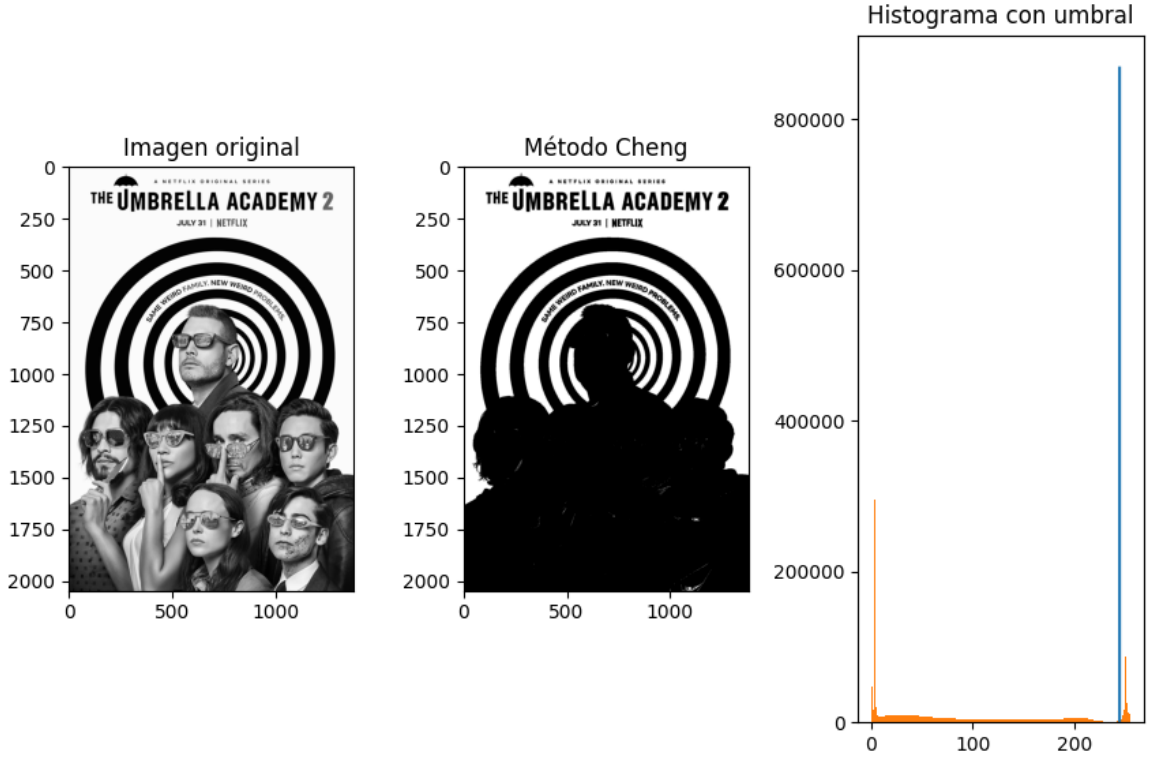
Umbral seleccionado por método Cheng: 186



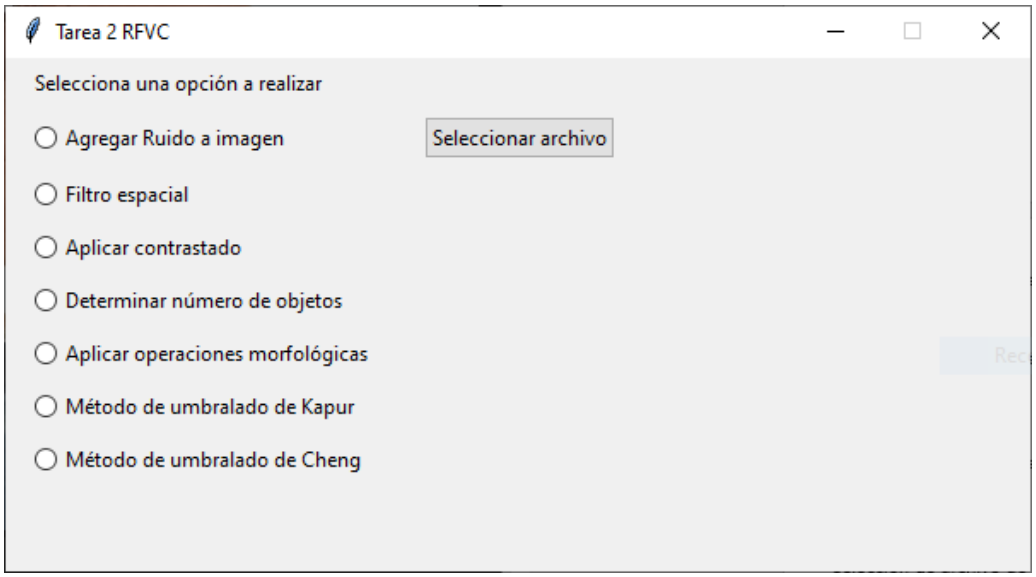




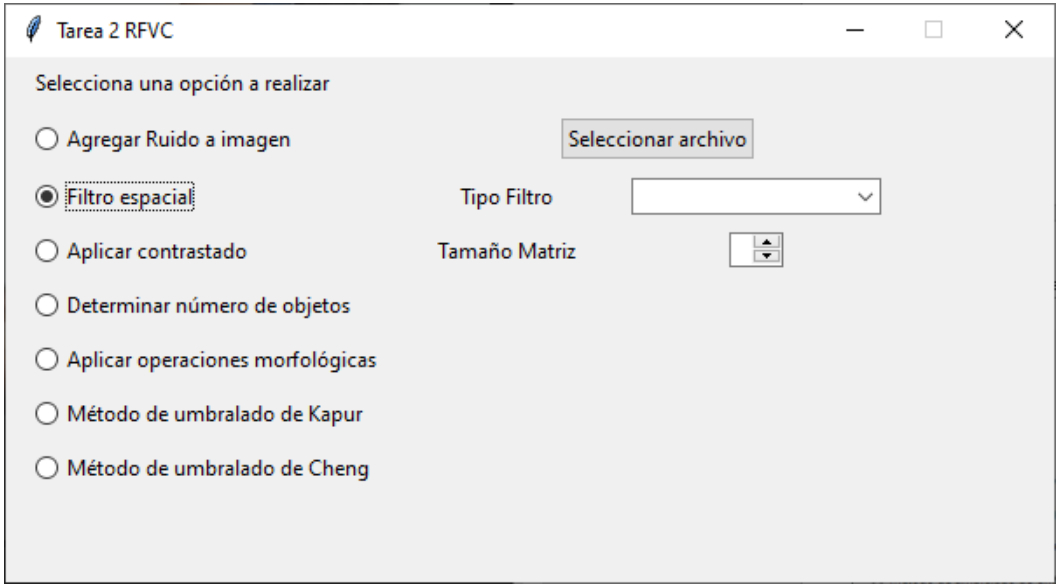
Umbral seleccionado por método Cheng: 245



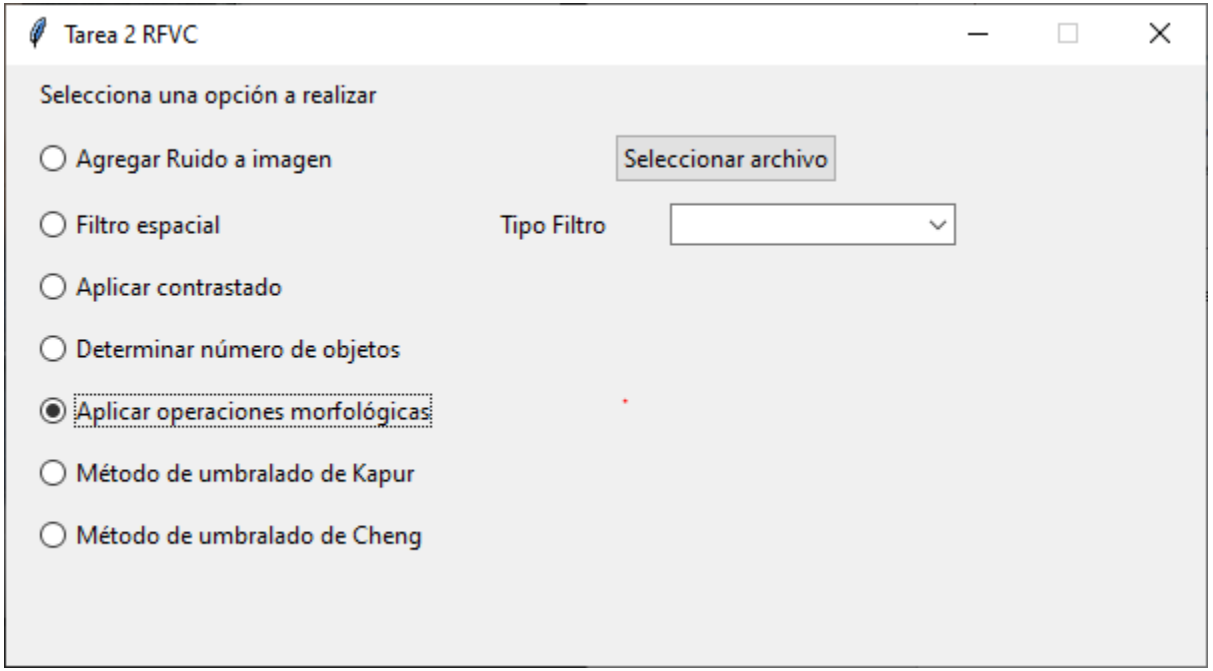
Interfaz Gráfica realizada con la biblioteca de Python *tkinter* (1):



Interfaz Gráfica realizada con la biblioteca de Python *tkinter* (2):



Interfaz Gráfica realizada con la biblioteca de Python *tkinter* (3):



Selección de archivo de tipo imagen (JPG, JPEG, PNG):