Tecnicatura Universitaria en Inteligencia Artificial
Facultad De Ciencias Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario

Trabajo Práctico N°02
Procesamiento de Imágenes I

Integrantes: Brisa Moresco, Julian Britos, Timoteo Garcia y Sofia Rondini
Fecha: 01/11/2023 - 20/11/2023

Tecnicatura Universitaria en Inteligencia Artificial
Facultad De Ciencias Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario

PROBLEMA 1 – Detección y clasificación de Monedas y Dados

La imagen monedas.jpg, adquirida con un smart phone, consiste de monedas de distinto valor y tamaño, y de dados sobre un fondo de intensidad no uniforme (ver Figura 1).

a) Procesar la imagen de manera de segmentar las monedas y los dados de manera automática.

b) Clasificar los distintos tipos de monedas y realizar un conteo, de manera automática.

c) Determinar el número que presenta cada dado mediante procesamiento automático.

PLANTEAMIENTO

La idea general de como resolver el codigo es:

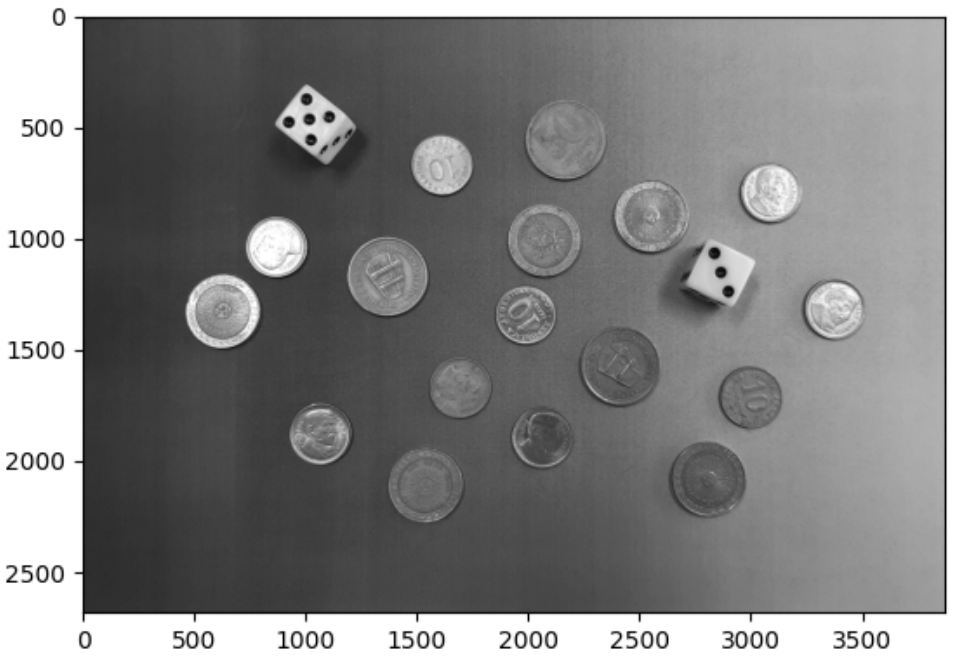
- Filtros y Canny para detectar objetos
- Componentes conectadas para bordes
- Itero por cada contorno y los divido entre monedas y dados
- Diferencio las monedas según área
- Contabilizo los tipos de moneda
- Recorto los dados
- Realizo los mismos pasos para detectar contornos
- Itero nuevamente y elijo los contornos (puntos de los dados) que necesito
- Los cuento e imprimo en pantalla la conclusión

PROCEDIMIENTO

Cargo la imagen. La paso a RGB y luego a escala de grises.

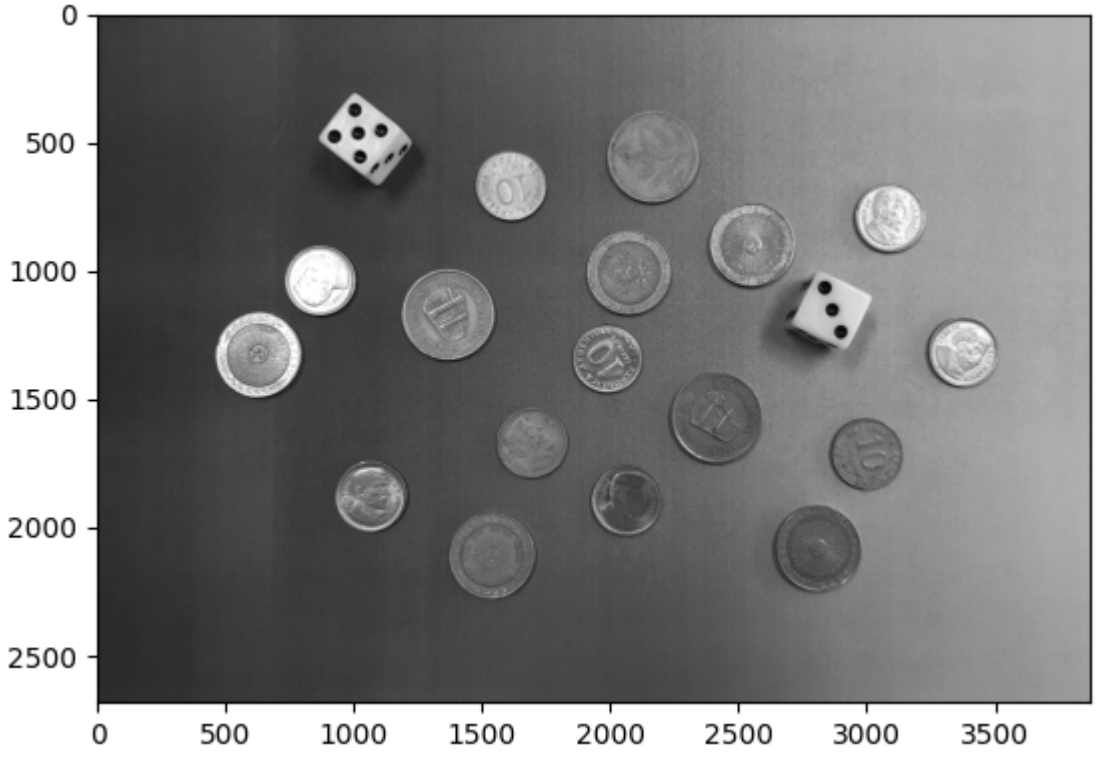
```
img = cv2.imread('monedas.jpg', cv2.IMREAD_COLOR)
plt.figure(), plt.imshow(img, cmap='gray'), plt.show(block=False)
img_original = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.figure(), plt.imshow(img_original, cmap='gray'), plt.show(block=False)
```


```
img_gris = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.figure(), plt.imshow(img_gris, cmap='gray'), plt.show(block=False)
```

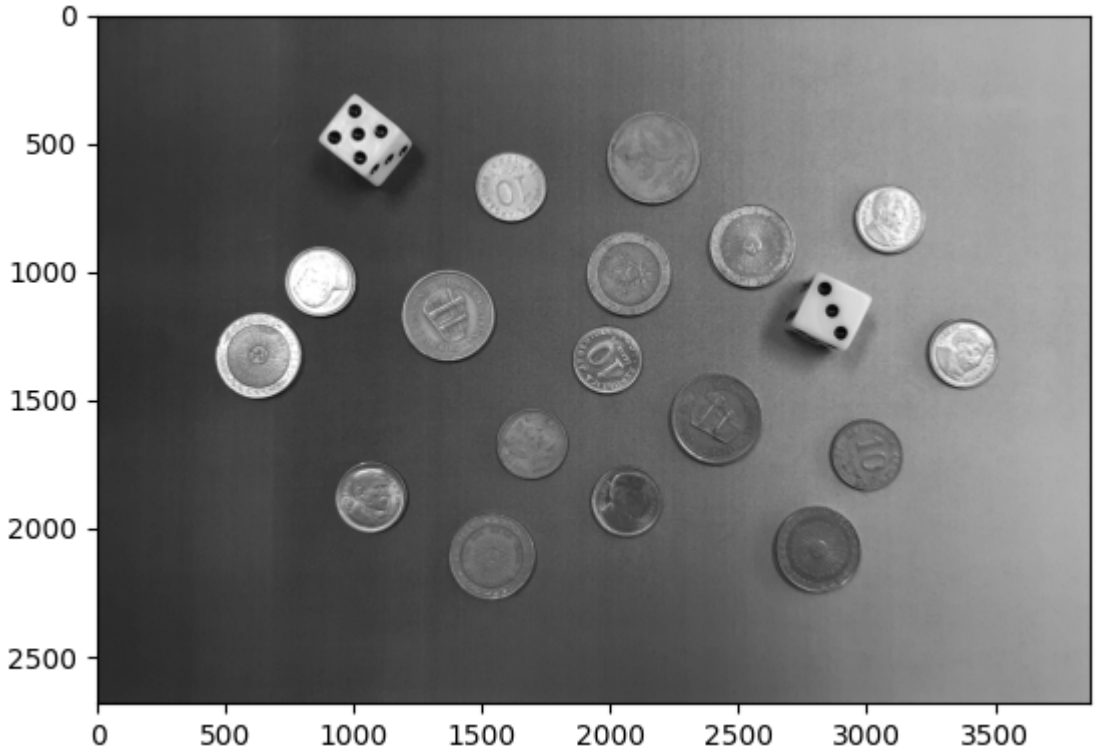


Luego aplico filtros para mejorar la imagen antes de realizar Canny

```
img_fil = cv2.medianBlur(img_gris, 5)
plt.figure(), plt.imshow(img_fil, cmap='gray'), plt.show(block=False)
```

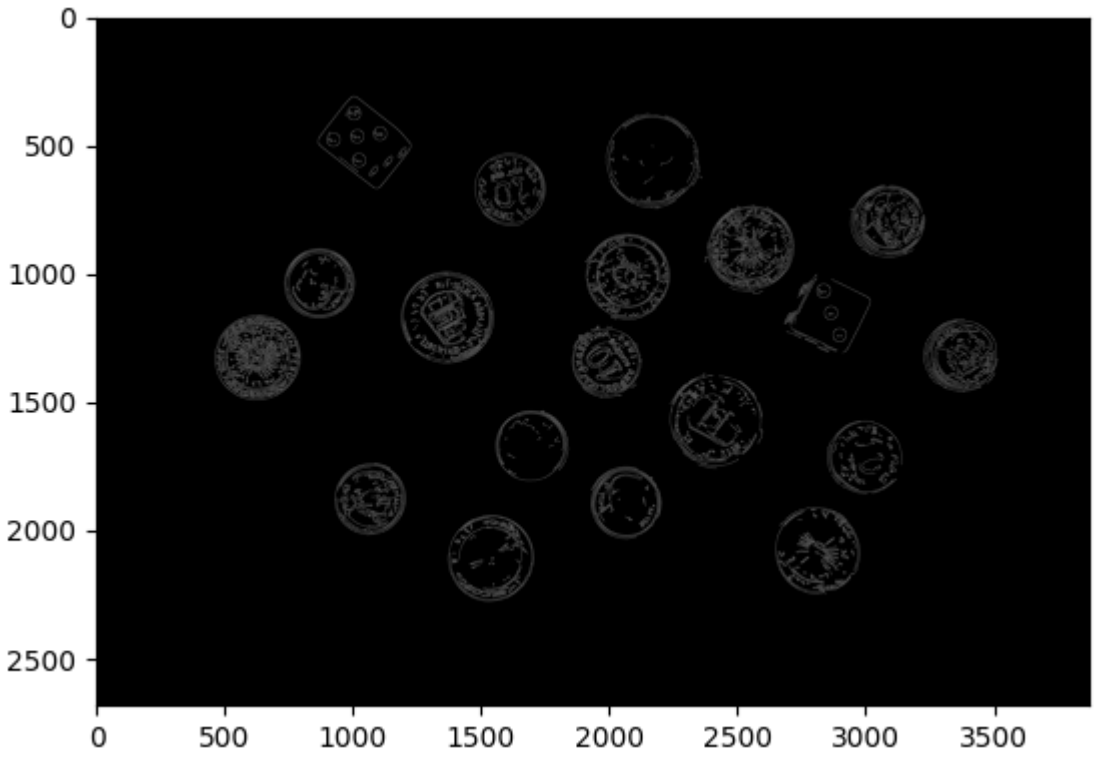



```
img_pasabajo = cv2.blur(img_fil, (5, 5))
plt.figure(), plt.imshow(img_fil, cmap='gray'), plt.show(block=False)
```



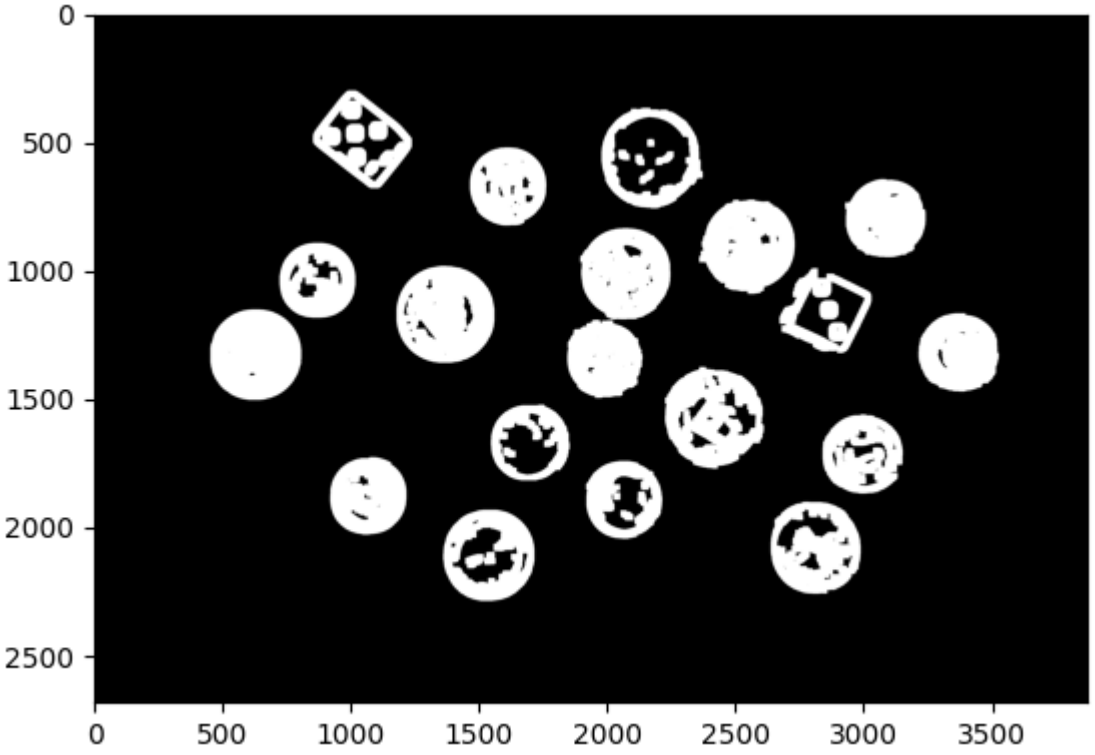
Ahora vamos a hacer una detección de bordes suando Canny

```
bordes = cv2.Canny(img_pasabajo, 10, 80)
plt.figure(), plt.imshow(bordes, cmap='gray'), plt.show(block=False)
```



Veo que los circulos no estan del todo cerrados y por eso voy a dilatar con una matriz muy grande.

```
kernel = np.ones((23,23), np.uint8)
dilatado = cv2.dilate(bordes, kernel, iterations=1)
plt.figure(), plt.imshow(dilatado, cmap='gray'), plt.show(block=False)
```

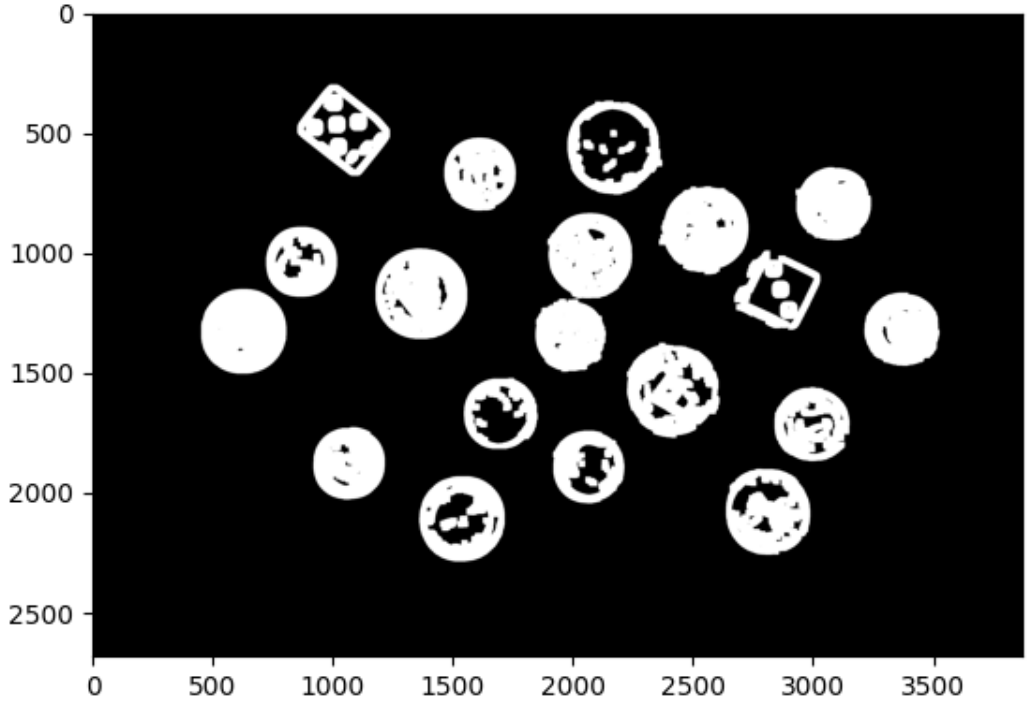


Genero una mascara. Hago componentes conectadas.

```
num_labels, etiquetas= cv2.connectedComponents(dilatado)
mascara = np.zeros_like(dilatado)
```

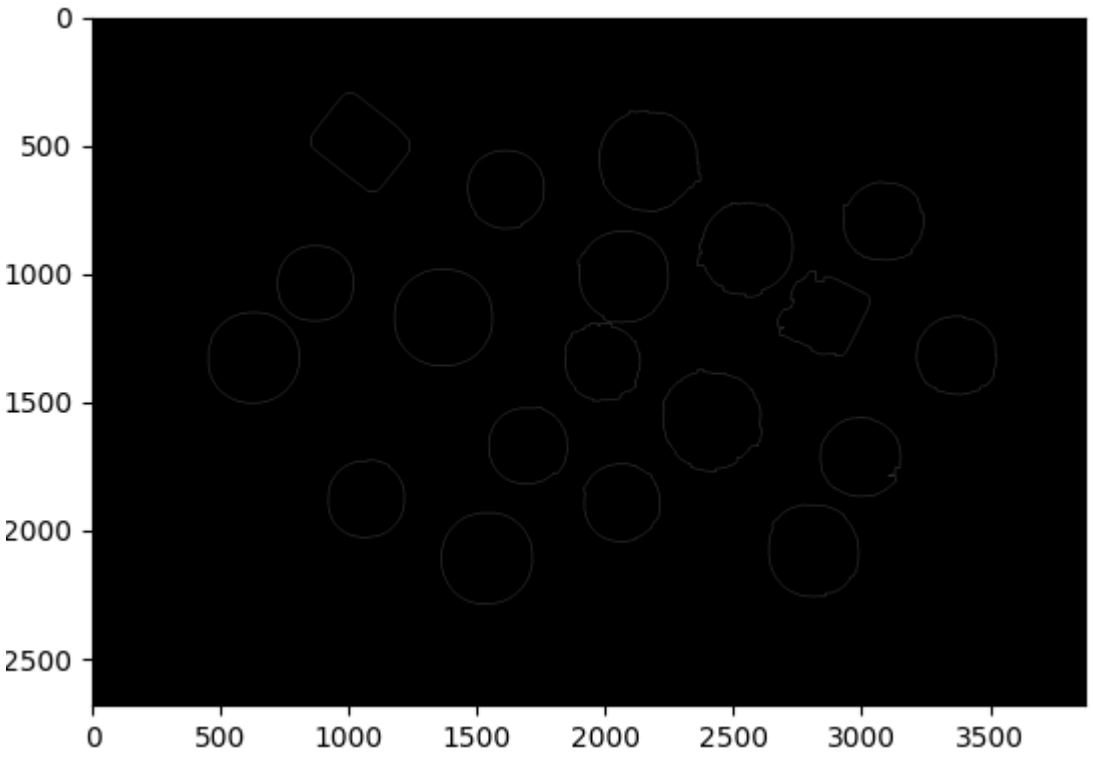
Dibujo los objetos en la máscara

```
for label in range(1, num_labels):
    mascara[etiquetas == label] = 255
plt.figure(), plt.imshow(mascara, cmap='gray'), plt.show(block=False)
```



Ahora voy a buscar de cada objeto los contornos. Luego los dibujo en una máscara.

```
contornos, _ = cv2.findContours(mascara, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
bordes_imagen = np.zeros_like(mascara)
cv2.drawContours(bordes_imagen, contornos, -1, 255, 1)
plt.figure(), plt.imshow(bordes_imagen, cmap='gray'), plt.show(block=False)
```



Voy a contar cuantos contornos genero hasta ahora

```
len(contornos)
```
19

De contornos tengo una tupla, la voy a pasar a lista para trabajar más fácil

```
contornos = list(contornos)
print(type(contornos))
```
<class 'list'>

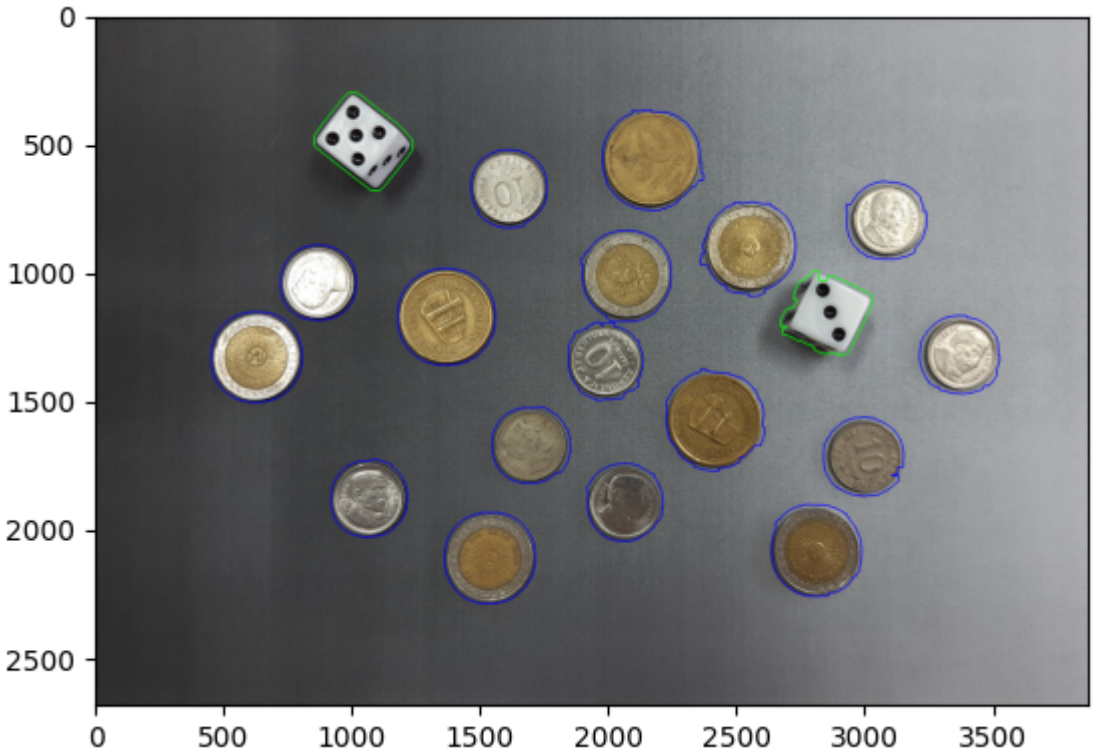
Voy a clasificar por forma, usando la excentricidad, dividir cuales contornos con circulares y cuales no

```
img_final = img_original.copy()

# --- Consulto la excentricidad de todos los contornos
for contorno in contornos:
    elipse = cv2.fitEllipse(contorno)
    excentricidad = elipse[1][0] / elipse[1][1]
    print(f"Excentricidad: {excentricidad}")
```

Excentricidad: 0.9834956310704087

```
Excentricidad: 0.962306720099717
Excentricidad: 0.9891550758340281
Excentricidad: 0.9736273543695863
Excentricidad: 0.9801981356998817
Excentricidad: 0.9656610596903042
Excentricidad: 0.9835016347784251
Excentricidad: 0.9564145045580156
Excentricidad: 0.9640194693785565
Excentricidad: 0.9847362137006007
Excentricidad: 0.9179957211896657
Excentricidad: 0.9822992814249099
Excentricidad: 0.9866101667482672
Excentricidad: 0.9778544601914901
Excentricidad: 0.9711638128668959
Excentricidad: 0.9809758892109834
Excentricidad: 0.9817280201879145
Excentricidad: 0.9961179850340228
Excentricidad: 0.7730697380498083

# Veo que la mayoría de los valores son superiores a 0.95 y hay algunos menores, entonces voy a generar la división en este punto



# --- Genero las listas
monedas = []
dados = []
areas = []

# Itero sobre los contornos, busco la excentricidad y divido dos subgrupos, los circulos (excentricidad mayor a 0.95) y las otras figuras
k = -1
for contorno in contornos:
    k += 1
    elipse = cv2.fitEllipse(contorno)
    excentricidad = elipse[1][0] / elipse[1][1]
    if excentricidad > 0.95:
        # Agrego el contorno a la lista
        monedas.append(contorno)
        # Calculo el area y la agrego a la lista
        area = cv2.contourArea(contorno)
        areas.append(area)
        # Dibujo los contronos en la imagen y los diferencio con colores
```

```
        cv2.drawContours(img_final, contornos, k, (0,0,255),3)
    else:
        dados.append(contorno)
        cv2.drawContours(img_final, contornos, k, (0,255,0),3)
plt.figure(), plt.imshow(img_final), plt.title('Contornos Pintados'), plt.show(block=False)
```



Vamos a ver como quedo formada las lista

```
print(areas)
[101368.5, 101494.0, 71811.0, 71977.5, 76278.0, 72900.5, 116503.0, 69949.0, 75959.5, 100701.0,
115624.0, 69997.5, 99093.5, 103009.5, 75938.0, 72070.5, 119558.5]

# Veo que las areas son [101368.5, 101494.0, 71811.0, 71977.5, 76278.0, 72900.5, 116503.0, 69949.0,
75959.5, 100701.0, 115624.0, 69997.5, 99093.5, 103009.5, 75938.0, 72070.5, 119558.5]
areas.sort()
print(areas)
# Sabiendo que tenemos 3 grupos de monedas, tenemos que identificar 3 "saltos" en las areas
# Entonces vamos a dividir en 3 grupos segun los valores ordenados de las areas
# La lista ordenada queda:
[69949.0, 69997.5, 71811.0, 71977.5, 72070.5, 72900.5, 75938.0, 75959.5, 76278.0, 99093.5, 100701.0,
101368.5, 101494.0, 103009.5, 115624.0, 116503.0, 119558.5]
# chica <= 90000
# 90000 < mediana <= 1070000
```

```
# 107000 < grande
```

Ahora vamos a inicializar las listas nuevamente, sumandole donde vamos a contabilizar según tamaño

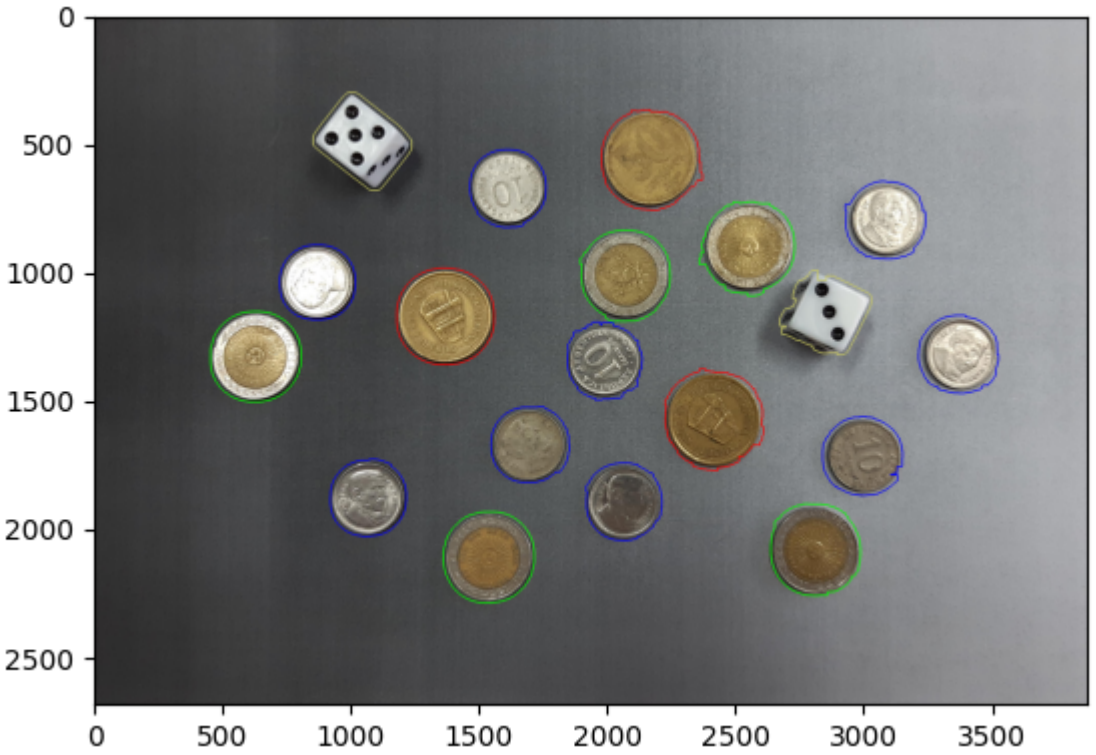
```
monedas = []
dados = []
monedas_10_centavos = 0
monedas_50_centavos = 0
monedas_1_peso = 0
```

Genero una máscara para usar luego

```
img_negra = np.zeros_like(img_original)
```

Repito parte del proceso, sumando la informacion de las divisiones de las areas y las diferencio con colores

```
k = -1
for contorno in contornos:
    k += 1
    elipse = cv2.fitEllipse(contorno)
    excentricidad = elipse[1][0] / elipse[1][1]
    if excentricidad > 0.95:
        monedas.append(contorno)
        area = cv2.contourArea(contorno)
        if area <= 90000:
            monedas_10_centavos += 1
            cv2.drawContours(img_final, contornos, k, (0,0,255),3)
        elif 90000 < area <= 107000:
            monedas_50_centavos += 1
            cv2.drawContours(img_final, contornos, k, (0,255,0),3)
        else:
            monedas_1_peso += 1
            cv2.drawContours(img_final, contornos, k, (255,0,0),3)
    else:
        dados.append(contorno)
        cv2.drawContours(img_final, contornos, k, (200,200,100),3)
        cv2.drawContours(img_negra, contornos, k, (255, 255, 255), thickness=cv2.FILLED)
plt.figure(), plt.imshow(img_final), plt.show(block=False)
```

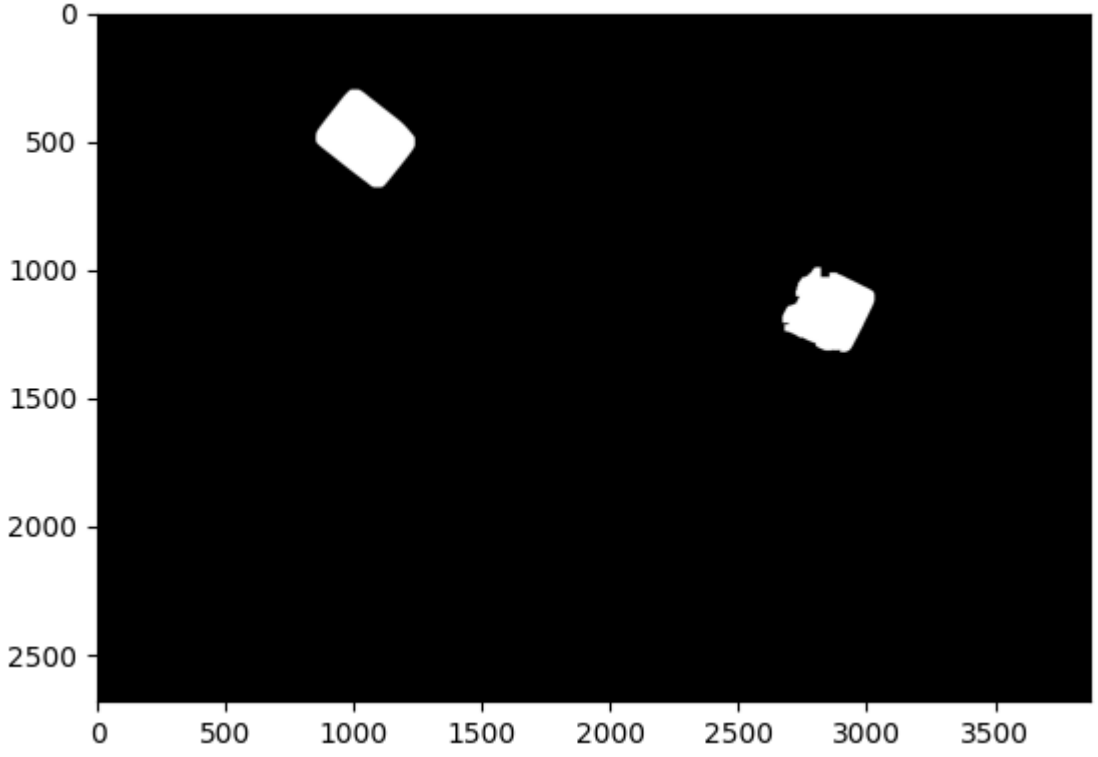


print(f"Hay {monedas_10_centavos} monedas de 10 centavos, {monedas_50_centavos} de 50 centavos y {monedas_1_peso} de 1 peso.")

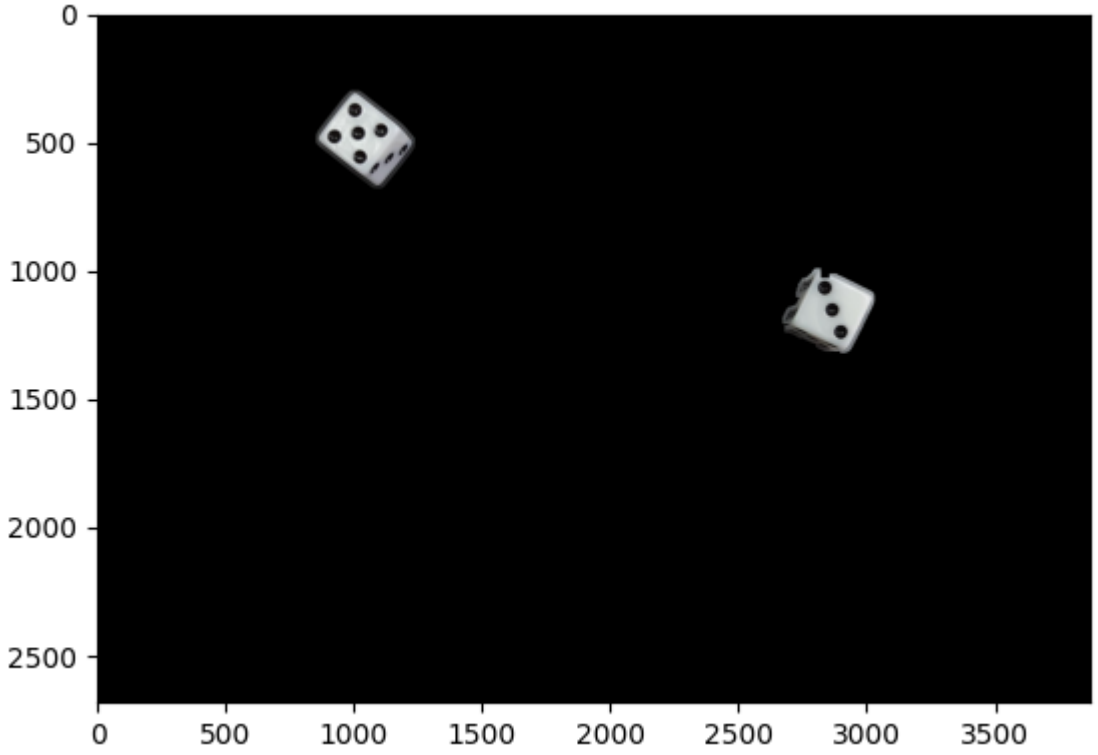
Hay 9 monedas de 10 centavos, 5 de 50 centavos y 3 de 1 peso.

A la imagen con los contornos de los dados le agrego la original solo en esas partes.

plt.figure(), plt.imshow(img_negra), plt.show(block=False)

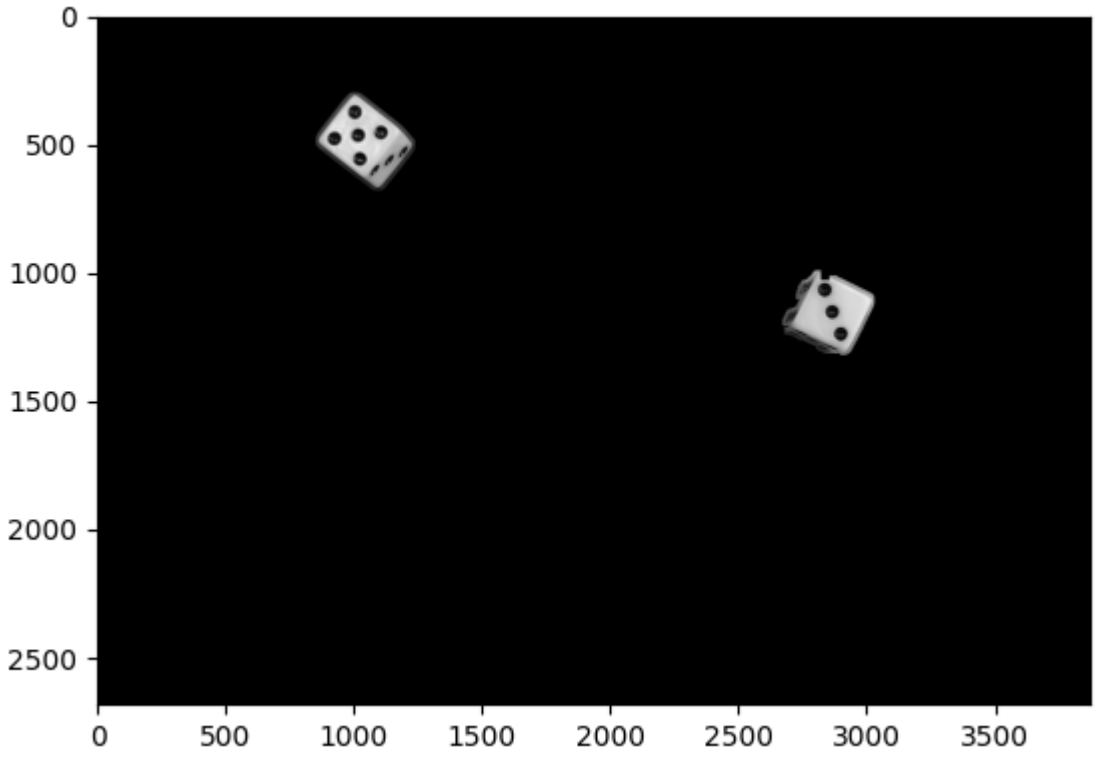



```
img_pintada = cv2.bitwise_and(img_original, img_negra)
plt.figure(), plt.imshow(img_pintada), plt.show(block=False)
```



Pasamos a escala de grises

```
img_pintada_gris = cv2.cvtColor(img_pintada, cv2.COLOR_BGR2GRAY)
plt.figure(), plt.imshow(img_pintada_gris, cmap='gray'), plt.show(block=False)
```

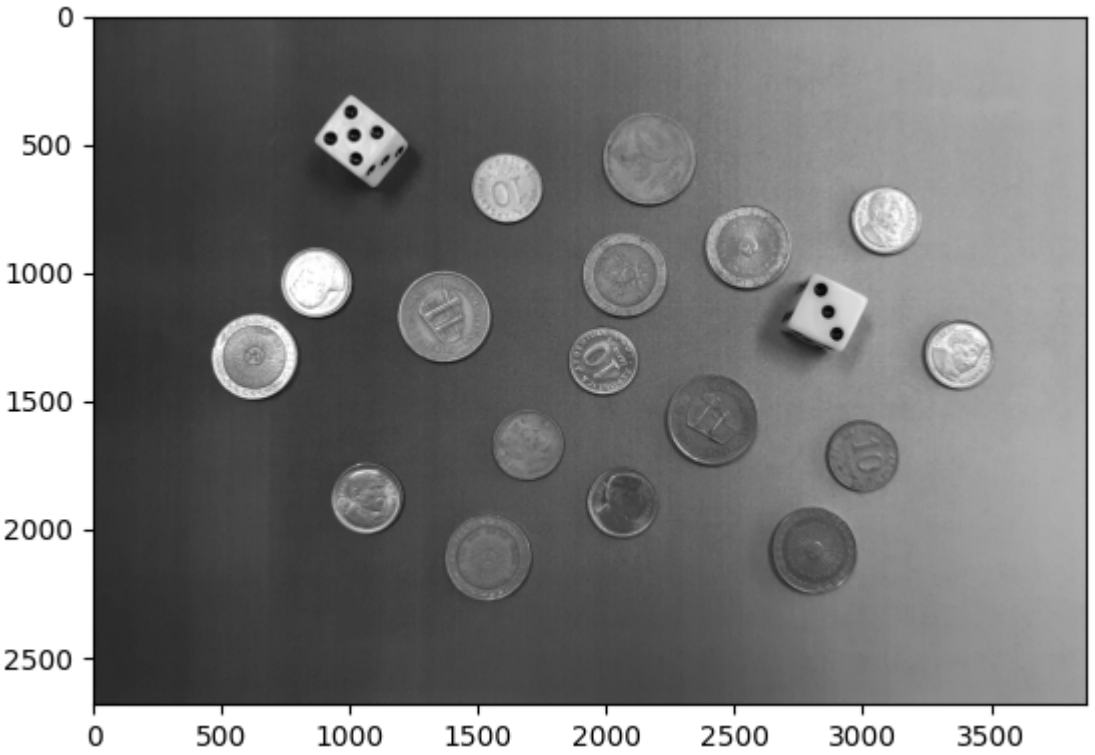



```
img_negra = np.zeros_like(img_original)
```

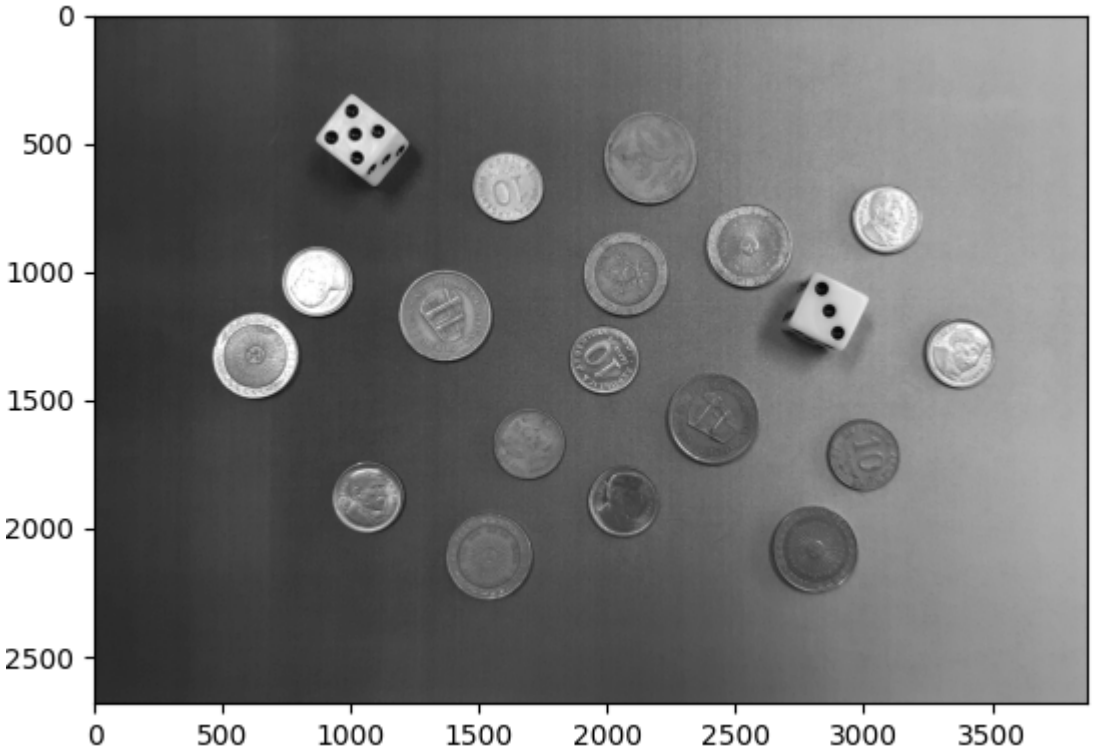
Repito el bucle de todos los contornos pero me enfoco en los dados

```
k = -1
m = 0 # Con este contador voy a seguir a cada dado
for contorno in contornos:
    k += 1
    elipse = cv2.fitEllipse(contorno)
    excentricidad = elipse[1][0] / elipse[1][1]

    if excentricidad > 0.95:
        continue

    else:
        m += 1
        x, y, w, h = cv2.boundingRect(contorno)
```

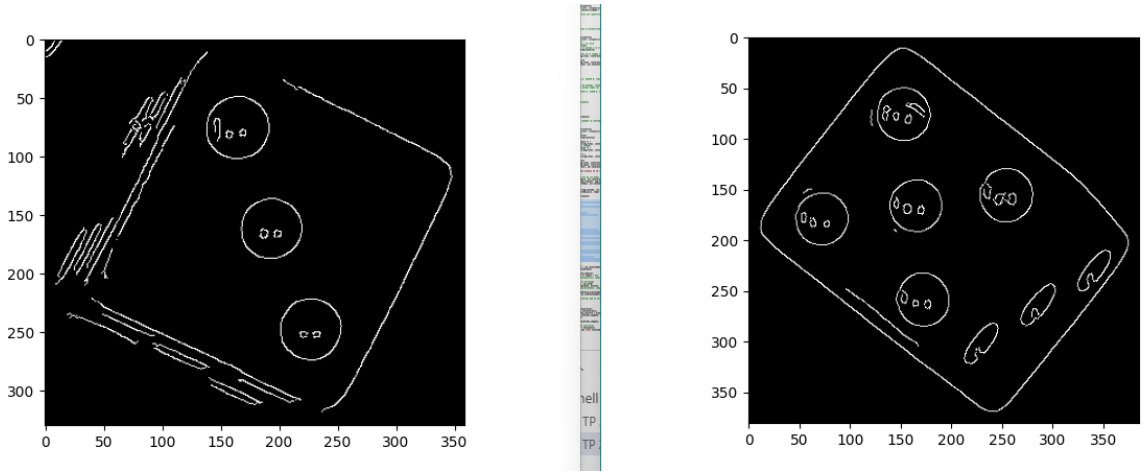
```
region_recortada = img_original[y:y + h, x:x + w]
region_recortada_gris = cv2.cvtColor(region_recortada, cv2.COLOR_BGR2GRAY)

# Repito los pasos del principio para detectar los contornos dentro de cada dado
# --- Suavizo
img_fil2 = cv2.medianBlur(region_recortada_gris, 5)
plt.figure(), plt.imshow(img_fil, cmap='gray'), plt.show(block=False)
```



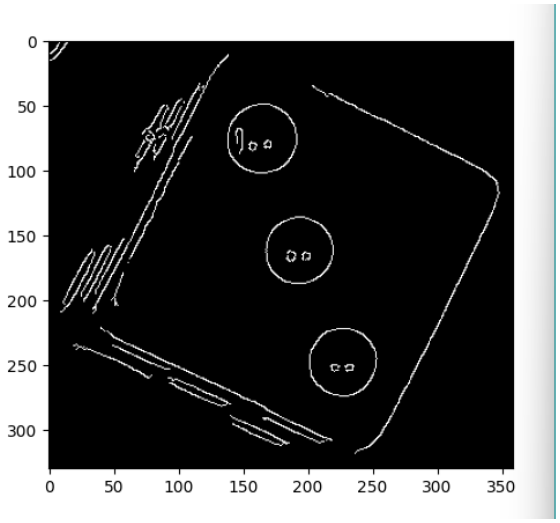

```
# --- Filtro pasabajo
img_pasabajo2 = cv2.blur(img_fil2, (5, 5))
plt.figure(), plt.imshow(img_fil, cmap='gray'), plt.show(block=False)
```

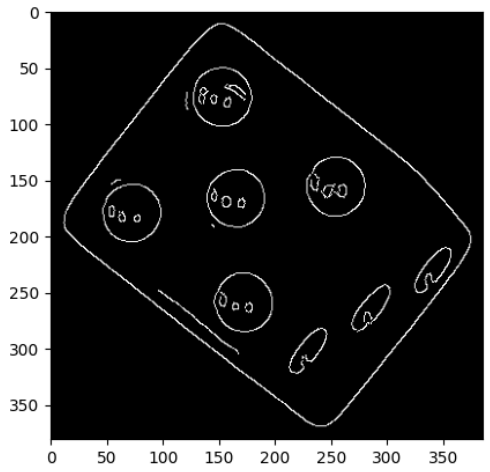



```
# --- Detección de bordes con Canny
bordes2 = cv2.Canny(img_pasabajo2, 10, 80)
plt.figure(), plt.imshow(bordes2, cmap='gray'), plt.show(block=False)
```

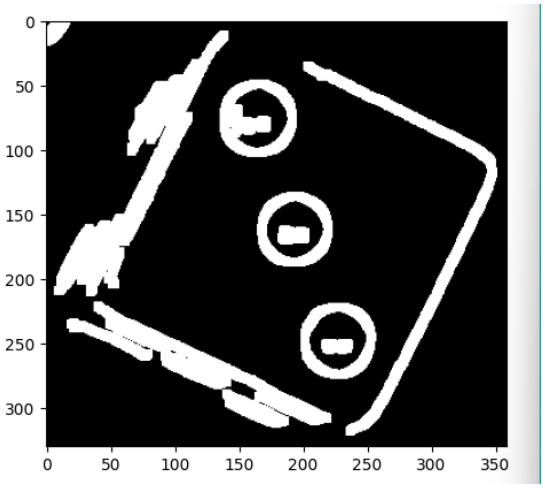


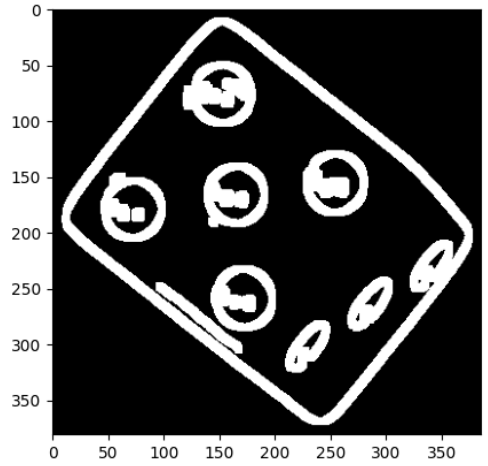

```
# --- Genero la máscara
num_labels2, etiquetas2 = cv2.connectedComponents(bordes2)
mascara2 = np.zeros_like(bordes2)

for label in range(1, num_labels2):
    mascara2[etiquetas2 == label] = 255
plt.figure(), plt.imshow(mascara2, cmap='gray'), plt.show(block=False)
```





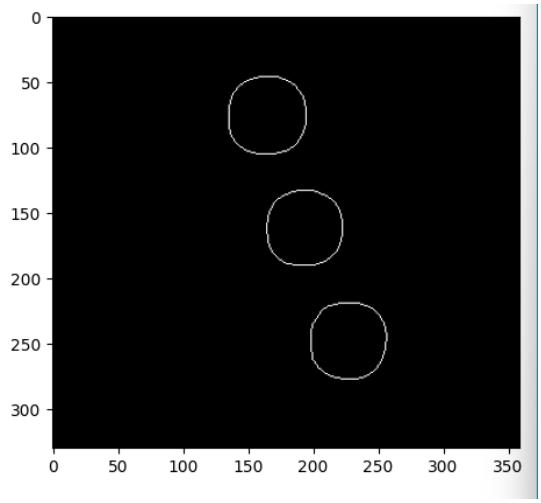
```
# --- Cierro los circulos dilatando
kernel2 = np.ones((7,7), np.uint8)
dilatado2 = cv2.dilate(bordes2, kernel2, iterations=1)
plt.figure(), plt.imshow(dilatado2, cmap='gray'), plt.show(block=False)
```

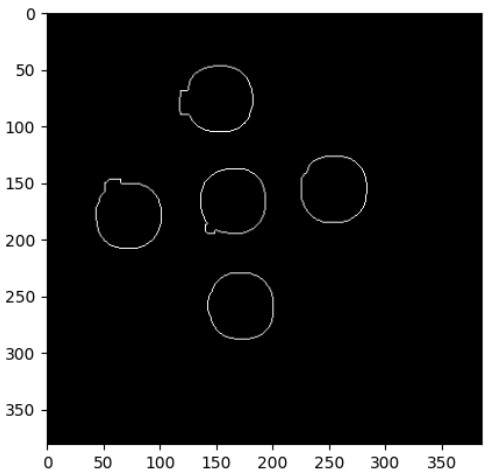




```
contornos2, _ = cv2.findContours(dilatado2, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
bordes_imagen2 = np.zeros_like(dilatado2)

# Voy a "elegir" los contornos que yo necesito para analizar, filtrando por area
area_minima = 2700
area_maxima = 2950
puntos = []

for i, c in enumerate(contornos2):
    area_contorno = cv2.contourArea(c)
    if area_minima < area_contorno < area_maxima:
        cv2.drawContours(bordes_imagen2, [c], -1, 255, 1)
        puntos.append(c)
```

```
plt.figure(), plt.imshow(bordes_imagen2, cmap='gray'), plt.show(block=False)
```






```
# Muestro en pantalla la conclusión
cantidad_puntos_negros = len(puntos)
print(f"El dado número {m} tiene {cantidad_puntos_negros} puntos negros.")
```

El dado número 1 tiene 3 puntos negros.

El dado número 2 tiene 5 puntos negros.

Tecnicatura Universitaria en Inteligencia Artificial
Facultad De Ciencias Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario

PROBLEMA 2 – Detección de patentes

La carpeta Patentes contiene imágenes de la vista anterior o posterior de diversos vehículos donde se visualizan las correspondientes patentes. En Figura 2 puede verse una de las imágenes.

a) Implementar un algoritmo de procesamiento de las imágenes que detecte automáticamente las patentes y segmente las mismas. Informar las distintas etapas de procesamiento y mostrar los resultados de cada etapa.

b) Implementar un algoritmo de procesamiento que segmente los caracteres de la patente detectada en el punto anterior. Informar las distintas etapas de procesamiento y mostrar los resultados de cada etapa

PLANTEAMIENTO

- **Carga de la imagen**: Se carga la imagen de la patente del vehículo a procesar.
- **Preprocesamiento**: Se realiza un preprocesamiento de la imagen para prepararla para la detección de la patente. Esto puede incluir la conversión a escala de grises, la eliminación de ruido, la mejora del contraste, etc.
- **Binarización**: Se binariza la imagen para separar los caracteres de la patente del fondo.
- **Identificación de componentes conectados**: Se identifican los elementos conectados en la imagen binarizada. Cada elemento conectado puede ser un carácter de la patente o ruido.
- **Filtrado por área**: Se filtran los elementos conectados basándose en su área. Se eliminan los elementos cuyo área es demasiado grande o demasiado pequeña para ser un carácter de la patente.
- **Filtrado por relación de aspecto**: Se filtran los elementos conectados basándose en su relación de aspecto (altura / ancho). Se eliminan los elementos cuyo aspecto no se ajusta al de un carácter de la patente.
- **Filtrado por cercanía**: Se filtran los elementos conectados basándose en su cercanía a otros elementos. Se eliminan los elementos que están demasiado lejos de cualquier otro elemento, ya que los caracteres de una patente suelen estar cerca unos de otros.
- **Detección de la patente**: Los elementos conectados que quedan después de todos los filtros forman la patente del vehículo.
- **Segmentación de los caracteres**: Se segmentan los caracteres de la patente. Esto se puede hacer identificando los elementos conectados en la imagen de la patente.

- **Visualización de los resultados**: Se muestran los resultados de cada etapa del procesamiento, incluyendo la imagen original, la imagen binarizada, la imagen después de cada filtro, la patente detectada y los caracteres segmentados.

PROCEDIMIENTO

```
# Definimos una función llamada 'imshow' que se utiliza para mostrar imágenes.
# Esta función toma varios argumentos:
# - img: La imagen que se va a mostrar.
# - new_fig: Un booleano que determina si se debe crear una nueva figura para mostrar la imagen. Por defecto es True.
# - title: El título de la figura. Por defecto es None.
# - color_img: Un booleano que determina si la imagen es en color. Si es False, la imagen se mostrará en escala de grises. Por defecto es False.
# - blocking: Un booleano que determina si la función 'show' de matplotlib debe bloquear la ejecución del resto del código hasta que se cierre la figura. Por defecto es False.
# - colorbar: Un booleano que determina si se debe mostrar una barra de colores junto a la imagen. Por defecto es False.
# - ticks: Un booleano que determina si se deben mostrar las marcas en los ejes x e y. Por defecto es False.
def imshow(img, new_fig=True, title=None, color_img=False, blocking=False, colorbar=False, ticks=False):
    # Si new_fig es True, se crea una nueva figura.
    if new_fig:
        plt.figure()
    # Si color_img es True, se muestra la imagen en color. Si es False, se muestra en escala de grises.
    if color_img:
        plt.imshow(img)
    else:
        plt.imshow(img, cmap='gray')
    # Se establece el título de la figura.
    plt.title(title)
    # Si ticks es False, se eliminan las marcas de los ejes x e y.
    if not ticks:
        plt.xticks([]), plt.yticks([])
    # Si colorbar es True, se muestra una barra de colores junto a la imagen.
    if colorbar:
        plt.colorbar()
    # Si new_fig es True, se muestra la figura.
    if new_fig:
        plt.show(block=blocking)
```

```
# --- Cargo Imagen ----------------------------------------
# Cerramos todas las figuras existentes
plt.close('all')
# Leemos la imagen desde el archivo img07.png en la carpeta Patentes
I = cv2.imread(f"Patentes/img07.png")
# Convertimos la imagen de BGR (Blue, Green, Red) a RGB (Red, Green, Blue)
I = cv2.cvtColor(I, cv2.COLOR_BGR2RGB)
# Mostramos la imagen con el título "Imagen Original"
imshow(I, title="Imagen Original")

# --- Paso a escalas de grises ----------------------------
# Convertimos la imagen de RGB a escala de grises
Ig = cv2.cvtColor(I, cv2.COLOR_RGB2GRAY)
# Mostramos la imagen en escala de grises con el título "Imagen en escala de grises"
imshow(Ig, title="Imagen en escala de grises")

# --- Binarizo --------------------------------------------
# Aplicamos un umbral a la imagen en escala de grises para obtener una imagen binaria
th, Ibw = cv2.threshold(Ig, 121, 255, cv2.THRESH_BINARY)
# Mostramos la imagen binaria con el título "Umbralado"
imshow(Ibw, title="Umbralado")

# --- Elementos conectados --------------------------------
# Establecemos la conectividad a 6
connectivity = 6
# Utilizamos la función connectedComponentsWithStats para obtener los componentes conectados en
la imagen
# Esta función también devuelve estadísticas (como el área) y los centroides de los componentes
num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(Ibw, connectivity,
cv2.CV_32S)
# Mostramos los componentes conectados
imshow(labels, title="Componentes conectados")

# --- Observo las áreas de todos los objetos --------------------------------
# Creamos una lista con las áreas de todos los componentes
areas = [st[cv2.CC_STAT_AREA] for st in stats]
# Ordenamos las áreas
areas_sorted = sorted(areas)
# Imprimimos las áreas ordenadas
print(areas_sorted)
```

```
# Iteramos sobre las áreas ordenadas e imprimimos cada una con su índice
for ii, vv in enumerate(areas_sorted):
    print(f"{ii:3d}): {vv:8d}")

# --- Filtro por area ---------------------------------------------------------
# Creamos una copia de la imagen binaria
Ibw_filtArea = Ibw.copy()
# Iteramos sobre cada componente conectado (ignorando el fondo)
for jj in range(1,num_labels):
    # Si el área del componente es mayor a 100 o menor a 15
    if (stats[jj, cv2.CC_STAT_AREA]>100) or (stats[jj, cv2.CC_STAT_AREA]<15):
        # Eliminamos el componente de la imagen (lo hacemos negro)
        Ibw_filtArea[labels==jj] = 0
# Mostramos la imagen después de filtrar por área
imshow(Ibw_filtArea, title="Filtrado por Area (imagen binaria)")

# --- Filtro por relacion de aspecto ------------------------------------------
# Establecemos la conectividad a 8
connectivity = 8
# Utilizamos la función connectedComponentsWithStats para obtener los componentes conectados en
la imagen
# Esta función también devuelve estadísticas (como el área) y los centroides de los componentes
num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(Ibw_filtArea, connectivity,
cv2.CV_32S)
# Mostramos los componentes conectados
imshow(labels, title="Componentes Conectados")

# Creamos una copia de la imagen filtrada por área
Ibw_filtAspect = Ibw_filtArea.copy()
# Iteramos sobre cada componente conectado (ignorando el fondo)
for jj in range(1,num_labels):
    # Calculamos la relación de aspecto del componente (altura / ancho)
    rel_asp = stats[jj, cv2.CC_STAT_HEIGHT] / stats[jj, cv2.CC_STAT_WIDTH]
    # Imprimimos la relación de aspecto
    print(f"{jj:3d}) {rel_asp:5.2f}")
    # Si la relación de aspecto es menor a 1.5 o mayor a 3.0
    if (rel_asp<1.5) or (rel_asp >3.0):
        # Eliminamos el componente de la imagen (lo hacemos negro)
        Ibw_filtAspect[labels==jj] = 0
# Mostramos la imagen después de filtrar por relación de aspecto
```

```
imshow(Ibw_filtAspect, title="Filtrado por relación de aspecto")

# --- Resultado parcial ------------------------------------------------
# Establecemos la conectividad a 8
connectivity = 8
# Utilizamos la función connectedComponentsWithStats para obtener los componentes conectados en la imagen
# Esta función también devuelve estadísticas (como el área) y los centroides de los componentes
num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(Ibw_filtAspect, connectivity, cv2.CV_32S)
# Mostramos los componentes conectados
imshow(labels, title="Componentes Conectados")

# Creamos una imagen de 3 canales a partir de la imagen filtrada por relación de aspecto
Ipatente = cv2.merge((Ibw_filtAspect, Ibw_filtAspect, Ibw_filtAspect))

# Iteramos sobre cada componente conectado (ignorando el fondo)
for ii in range(1,num_labels):
    # Dibujamos un rectángulo alrededor del componente en la imagen
    cv2.rectangle(Ipatente, tuple(stats[ii,0:2]), tuple(stats[ii,0:2]+stats[ii,2:4]), (255,0,0), 1)
    # Añadimos un texto con el índice del componente en el centroide del componente
    cv2.putText(Ipatente, f"{ii}", tuple(centroids[ii].astype(int)),
fontFace=cv2.FONT_HERSHEY_TRIPLEX, fontScale=0.7, color=(255, 0, 0), thickness=1)

# Mostramos la imagen con los componentes conectados, los rectángulos y las etiquetas
imshow(Ipatente, title="Componentes Conectados + BBOX + label")

# --- Analizo las relaciones de aspecto de todos los objetos -----------------------------------------------
# Iteramos sobre cada componente conectado (ignorando el fondo)
for ii in range(1,num_labels):
    # Calculamos la relación de aspecto del componente (altura / ancho)
    rel_asp = stats[ii, cv2.CC_STAT_HEIGHT] / stats[ii, cv2.CC_STAT_WIDTH]
    # Imprimimos el índice del componente y su relación de aspecto
    print(f"{ii:3d}) {rel_asp:5.2f}")

# --- Corroboro cercania de otro caracter ----------------------------
# Establecemos un umbral de distancia
DIST_TH = 20
# Creamos una copia de la imagen filtrada por relación de aspecto
Ipatente_cercania = Ibw_filtAspect.copy()
```

```
# Iteramos sobre cada componente conectado (ignorando el fondo)
for ii in range(1, num_labels):
    # Obtenemos el centroide del componente actual
    ch = centroids[ii,:]

    # --- Obtengo los centroides de los demás caracteres ---------------------------
    # Creamos una copia de los centroides y eliminamos el centroide del componente actual
    objs = np.delete(centroids.copy(), ii, axis=0)
    # Eliminamos el centroide del fondo
    objs = np.delete(objs, 0, axis=0)

    # --- Calculo distancias -------------------------------------------------------
    # Calculamos la diferencia entre los centroides de los otros componentes y el centroide actual
    aux = objs - ch
    # Calculamos la distancia euclidiana entre los centroides
    dists = np.sqrt(aux[:,0]**2 + aux[:,1]**2)

    # Si no hay ningún componente a una distancia menor que el umbral
    if not any(dists < DIST_TH):
        # Eliminamos el componente de la imagen (lo hacemos negro)
        Ipatente_cercania[labels==ii] = 0

# Mostramos la imagen después de filtrar por cercanía
imshow(Ipatente_cercania, title="Filtrado por cercanía")

# --- Resultado final ----------------------------------------------
# Establecemos la conectividad a 8
connectivity = 8
# Utilizamos la función connectedComponentsWithStats para obtener los componentes conectados en
la imagen
# Esta función también devuelve estadísticas (como el área) y los centroides de los componentes
num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(Ipatente_cercania,
connectivity, cv2.CV_32S)

# Creamos una imagen de 3 canales a partir de la imagen filtrada por cercanía
Ifinal = cv2.merge((Ipatente_cercania, Ipatente_cercania, Ipatente_cercania))

# Iteramos sobre cada componente conectado (ignorando el fondo)
for ii in range(1,num_labels):
```

```
    # Dibujamos un rectángulo alrededor del componente en la imagen
    cv2.rectangle(Ifinal, tuple(stats[ii,0:2]), tuple(stats[ii,0:2]+stats[ii,2:4]), (255,0,0), 1)

# Mostramos la imagen final con los caracteres detectados
imshow(Ifinal, title="Resultado Final: Caraceteres")
```

Segundo Codigo

```
# --- Cargo Imagen ----------------------------------------
# Cargando la imagen.
Img = cv2.imread(f"patentes/img07.png")

# Convertimos la imagen de BGR (Blue, Green, Red) a RGB (Red, Green, Blue)
Img = cv2.cvtColor(Img, cv2.COLOR_BGR2RGB)

# Mostramos la imagen original
imshow(Img, title="Imagen Original")

# Convertimos la imagen RGB a escala de grises
gray = cv2.cvtColor(Img, cv2.COLOR_RGB2GRAY)

# Aquí se muestra la imagen en escala de grises.
imshow(gray, title="Imagen en escala de grises")

# Aplicamos un umbral adaptativo a la imagen en escala de grises para obtener una imagen binaria
# El umbral se calcula como una suma ponderada de los valores de píxeles del vecindario
Binary = cv2.adaptiveThreshold(gray, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 7, 13)

# Aquí se muestra la imagen binaria.
imshow(Binary)

# Buscamos contornos en la imagen binaria. RETR_LIST recupera todos los contornos, pero no crea ninguna relación de padre a hijo.
contornos = cv2.findContours(Binary, cv2.RETR_LIST, cv2.CHAIN_APPROX_NONE )[0]

# Creamos un lienzo en blanco del mismo tamaño que la imagen original
canvas = np.zeros_like(Img)

# Dibujamos los contornos encontrados en el lienzo en blanco
cv2.drawContours(canvas , contornos, -1, (0, 255, 0), 1)

Aquí se muestra el lienzo con los contornos dibujados.
```

```
 imshow(canvas)
—-
# Se define una función llamada filter_candidates que toma como argumento los contornos
def filter_candidates(contours):
# Inicializamos una lista vacía para almacenar los contornos candidatos
candidates = []

# Definimos el ratio de aspecto de una placa de matrícula estándar
ratio = 2.27906977

# Iteramos sobre cada contorno en la lista de contornos
for cnt in contours:
    # Obtenemos el rectángulo delimitador para el contorno actual
    x, y, w, h = cv2.boundingRect(cnt)

    # Calculamos el ratio de aspecto para el rectángulo delimitador
    aspect_ratio = float(w) / h

    # Calculamos el área del rectángulo delimitador
    area = w * h

    # Si el ratio de aspecto está cerca del ratio de una placa de matrícula y el área es mayor
que 1000,
    # añadimos el contorno a la lista de candidatos
    if np.isclose(aspect_ratio, ratio, atol=0.7) and area > 1000:
        candidates.append(cnt)

# Llamamos a la función filter_candidates para obtener los contornos candidatos
candidates = filter_candidates(contornos)

# Creamos un lienzo en blanco del mismo tamaño que la imagen original
canvas = np.zeros_like(Img)

# Dibujamos los contornos candidatos en el lienzo
cv2.drawContours(canvas , candidates, -1, (0, 255, 0), 1)

# Mostramos el lienzo con los contornos candidatos dibujados
imshow(canvas)

# Inicializamos una lista vacía para almacenar las coordenadas y de los contornos
candidatos
ys = []

# Iteramos sobre cada contorno candidato
for cnt in candidates:
    # Obtenemos el rectángulo delimitador para el contorno candidato
    x, y, w, h = cv2.boundingRect(cnt)
```

```
    # Añadimos la coordenada y del rectángulo delimitador a la lista ys
    ys.append(y)


# Iteramos sobre cada contorno candidato
for cnt in candidates:
    # Obtenemos el rectángulo delimitador para el contorno candidato
    x, y, w, h = cv2.boundingRect(cnt)

    # Añadimos la coordenada y del rectángulo delimitador a la lista ys
    ys.append(y)

# Seleccionamos el contorno candidato con la coordenada y más alta (es decir, el que está más abajo en la imagen)
license = candidates[np.argmax(ys)]

# Creamos un nuevo lienzo en blanco del mismo tamaño que la imagen original
canvas = np.zeros_like(Img)

# Dibujamos el contorno seleccionado en el lienzo
cv2.drawContours(canvas , [license], -1, (0, 255, 0),  thickness=cv2.FILLED)

# Mostramos el lienzo con el contorno seleccionado dibujado
imshow(canvas)

# Obtenemos el rectángulo delimitador para el contorno seleccionado
x, y, w, h = cv2.boundingRect(license)

# Recortamos la imagen original usando las coordenadas del rectángulo delimitador
cropped = Img[y:y+h, x:x+w]

# Mostramos la imagen recortada
imshow(cropped)

# Convertimos la imagen recortada a escala de grises
Crop_Binary = cv2.cvtColor(cropped, cv2.COLOR_RGB2GRAY)

# Se muestra la imagen en escala de grises.
imshow(Crop_Binary)

# Se aplica un umbral a la imagen en escala de grises.
# Los píxeles con un valor mayor a 120 se convierten a blanco (255), y los demás a negro (0).
th , hola = cv2.threshold(Crop_Binary, 120, 255, cv2.THRESH_BINARY)

# Mostrar la imagen después de la umbralización
```

```
imshow(hola)

# Definir la conectividad para la función connectedComponentsWithStats. En este caso, se utiliza una conectividad de 8, lo que significa que un píxel se considera conectado a otro si comparten al menos un borde o un vértice.
connectivity = 8
# Aplicar la función connectedComponentsWithStats para etiquetar componentes conectados en la imagen, obtener estadísticas para cada componente y encontrar sus centroides.
num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(hola, connectivity, cv2.CV_32S)

# Mostrar la imagen con los componentes conectados etiquetados
imshow(labels, title="Componentes Conectados")
# Inicializar una matriz de ceros del mismo tamaño que la imagen umbralizada
Ibw_filtAspect = np.zeros_like(hola)

# Iterar sobre cada etiqueta/componente conectado (ignorando el fondo que es la etiqueta 0)
for jj in range(1, num_labels):
    # Obtener la altura del componente conectado actual
    altura = stats[jj, cv2.CC_STAT_HEIGHT]
    # Calcular la relación de aspecto del componente conectado actual (altura / ancho)
    rel_asp = stats[jj, cv2.CC_STAT_HEIGHT] / stats[jj, cv2.CC_STAT_WIDTH]

    # Si la altura está cerca de 16 (con una tolerancia de 3) o la relación de aspecto es menor que 1.5 o mayor que 3.0
    if np.isclose(altura, 16, atol = 3) or (rel_asp<1.5) or (rel_asp >3.0) :
        # Mantener los píxeles del componente conectado actual en la imagen filtrada
        Ibw_filtAspect[labels == jj] = hola[labels == jj]

# Mostrar la imagen filtrada
imshow(Ibw_filtAspect)
```