



Tecnicatura Universitaria en Inteligencia Artificial

Procesamiento de Imágenes 1 (IA4.4)

Trabajo Práctico N° 2

Docentes:

- Álvarez, Julián
- Reyes, Facundo
- Sad, Gonzalo

Grupo: 3

Integrantes:

Aguirre, Fabian | A-4516/1
Fontela, Facundo | F-3724/9
Wagner, Juan | W-0557/6

Año: 2023

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.



Figura 1 – Imagen con monedas y dados.

Resolución:

El primer paso fue hacer una inspección visual de la imagen para determinar de que manera abordar el problema. Observamos que tiene un fondo con ruido por lo que habrá que aplicar algún filtrado. También se observa que el fondo no es liso, sino que presenta un degradado en escala de grises. Esto puede dificultar la elección de algún umbral ya que en algunos lugares el fondo tiene un color similar a las monedas.

A continuación procedemos a la resolución. El código proporcionado es una implementación en Python utilizando la biblioteca OpenCV para el procesamiento de imágenes.

1. Lectura y preprocesamiento de la imagen:

- Primero se comienza leyendo la imagen "monedas.jpg" y convirtiéndola a escala de grises para facilitar el procesamiento.
- Se aplica un filtro de mediana para reducir el ruido en la imagen.
- El operador Canny se utiliza para detectar bordes en la imagen. Los parámetros (50, 150) determinan los umbrales mínimo y máximo para la detección de bordes que encontramos convenientes.
- Se aplica una operación de dilatación para cerrar los bordes y mejorar la detección de contornos.

```
imagen = cv2.imread("img/monedas.jpg")
imagen_gris = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)

# Aplica operación de dilatación para cerrar los bordes
kernel_dilatacion = np.ones((10, 10), np.uint8)

def dilate_image(img, iterations=1):
    return cv2.dilate(img, kernel_dilatacion, iterations=iterations)

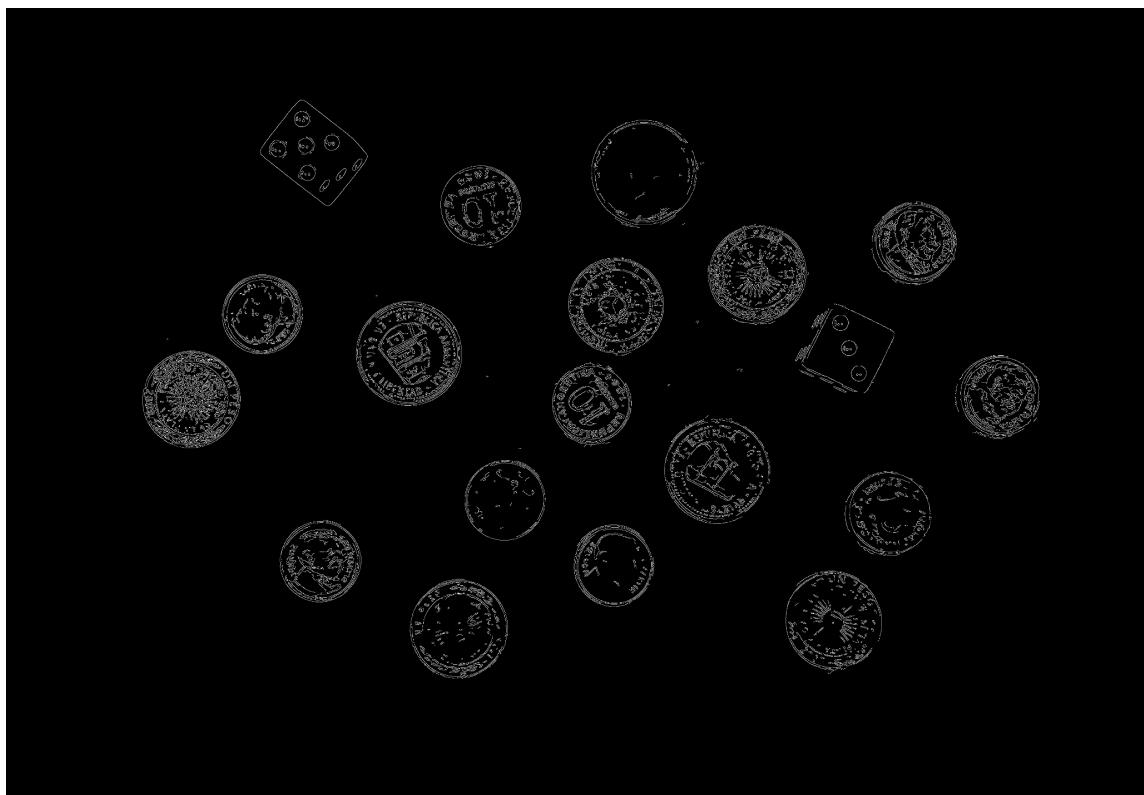
# Canny
imagen_suavizada = cv2.medianBlur(imagen_gris, 5)
canny = cv2.Canny(imagen_suavizada, 50, 150)

bordes_dilatados = dilate_image(canny, iterations=3)
```

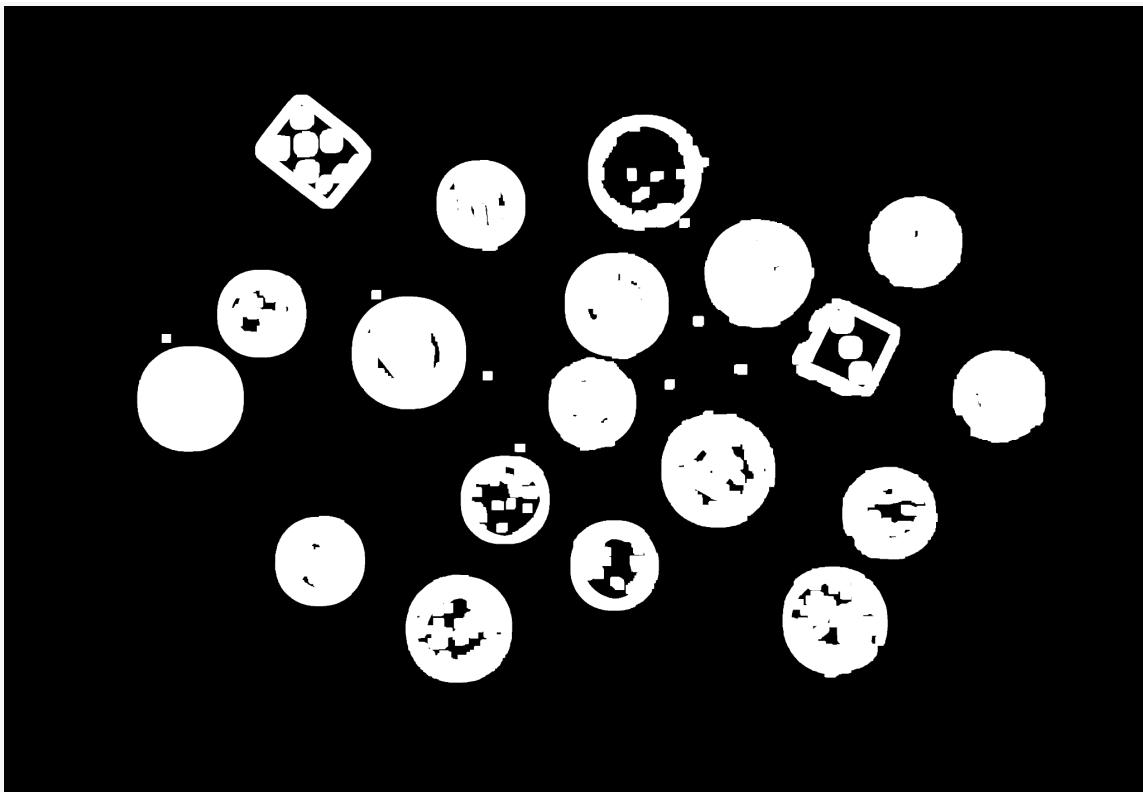
Observamos la imagen después de convertirla a escala de grises:



Después de aplicar Canny, obtenemos esta imagen:



Este es el resultado de al final de aplicar la dilatación:

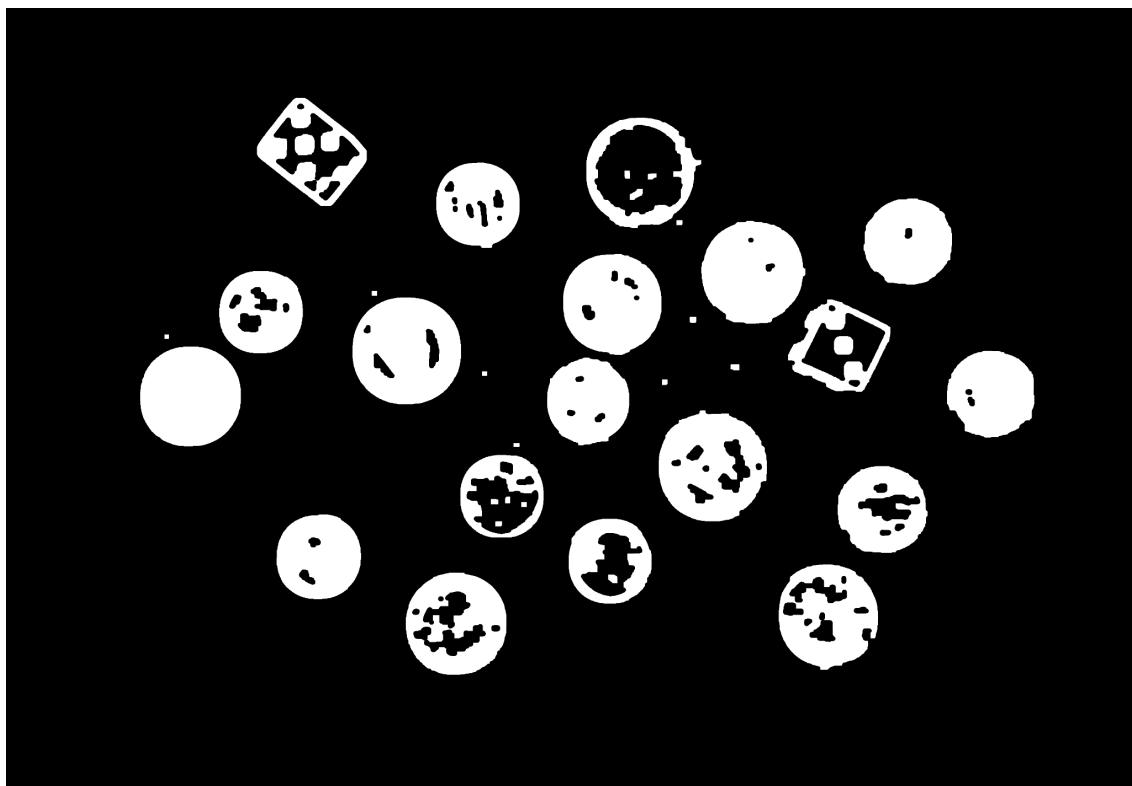


2. Segmentación de monedas y dados:

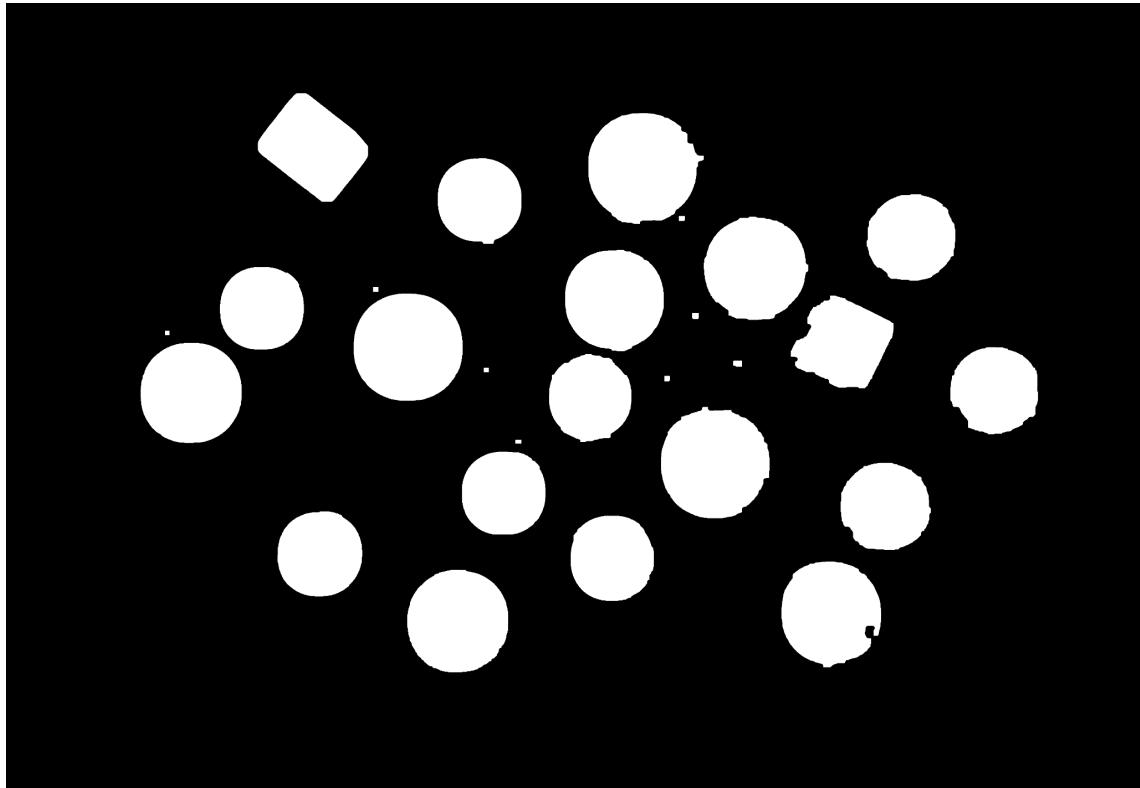
- Se realiza una erosión para eliminar pequeñas imperfecciones en los contornos.
- Luego, se encuentran los contornos, con la función *findContours*, en la imagen con bordes definidos.
- Luego se crea una máscara en blanco del mismo tamaño que la imagen original para resaltar las regiones de interés (monedas y dados).

```
test = cv2.erode(  
    bordes_dilatados,  
    cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (17, 17)),  
    iterations=1,  
)  
  
# Se encuentran los contornos  
contornos, _ = cv2.findContours(test, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)  
# Creación de máscara  
mascara = np.zeros_like(imagen_gris)  
cv2.drawContours(mascara, contornos, -1, (255), thickness=cv2.FILLED)
```

Imagen luego de aplicar la erosión:



Después de aplicar la máscara y dibujar los contornos:

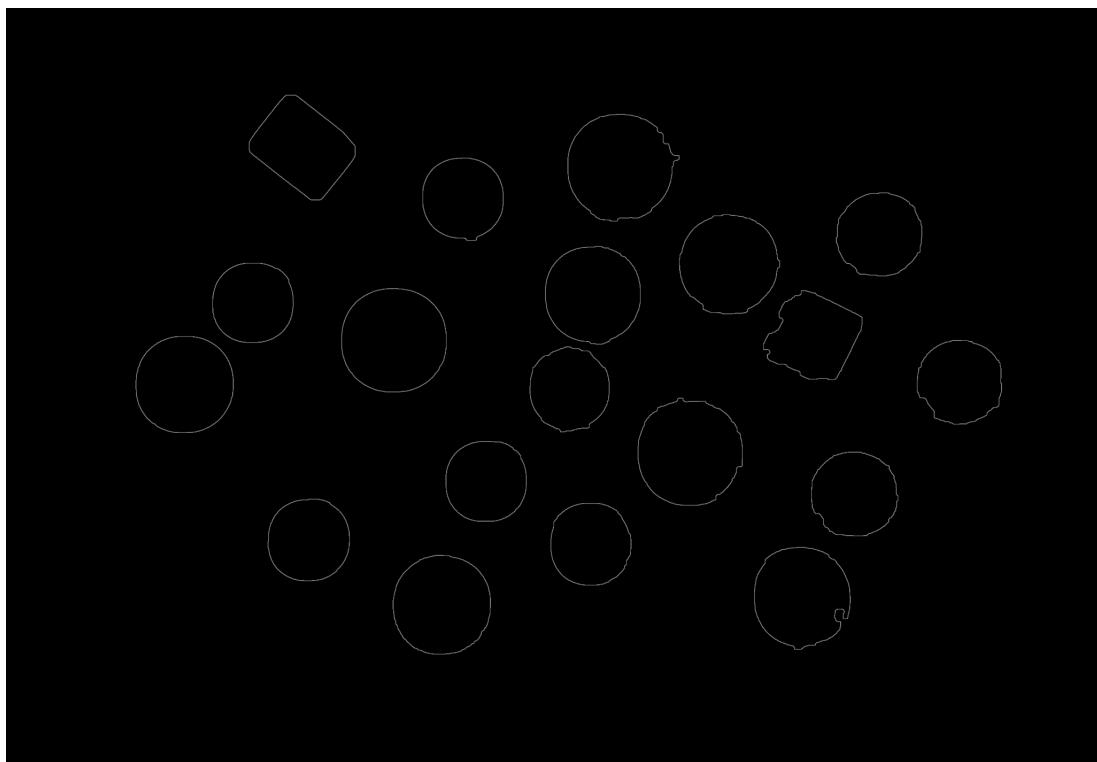


3. Filtrado de regiones de interés:

- Se realiza una segmentación basada en el tamaño de las áreas de los contornos encontrados.
- Luego se eliminan las regiones pequeñas y se retienen aquellas que corresponden a monedas y dados.
- Al final se dibujan los contornos de las monedas en la imagen resultante.

```
(  
    componentes_conectadas,  
    etiquetas,  
    estadisticas,  
    centroides,  
) = cv2.connectedComponentsWithStats(mascara, cv2.CV_32S, connectivity=8)  
  
imagen_resultado = np.zeros_like(test)  
  
for i in range(1, componentes_conectadas):  
    area = estadisticas[i, cv2.CC_STAT_AREA]  
  
    if area > 600:  
        mascara = np.uint8(etiquetas == i)  
        contorno, _ = cv2.findContours(  
            mascara, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE  
)  
        cv2.drawContours(imagen_resultado, contorno, -1, 120, 2)
```

Resultado de aplicar el filtrado por área:



4. Operaciones morfológicas:

- Se realiza el relleno de los contornos de las monedas y dados en la imagen procesada. Se utiliza la función `fill_contours` a través la operación `cv2.floodFill`
- Posteriormente, se aplica una operación de dilatación a la imagen resultante para asegurar la integridad de las regiones de interés.

```
# Funcion para llenar los contornos
def fill_contours(img):
    seed_point = (0, 0)
    blanco = (255, 255, 255)

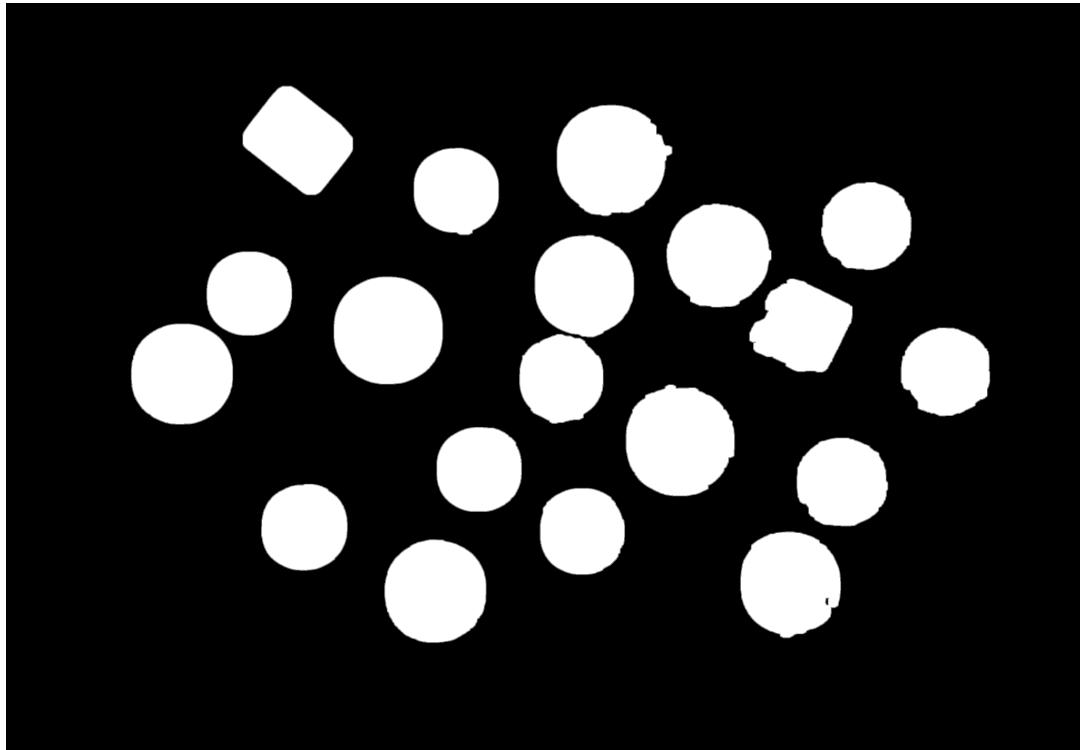
    flags = 4
    lo_diff = (10, 10, 10)
    up_diff = (10, 10, 10)

    cv2.floodFill(img, None, seed_point, blanco, lo_diff, up_diff, flags)
    return ~img

imagen_resultado = fill_contours(imagen_resultado)

result = cv2.dilate(
    imagen_resultado, cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5)), iterations=5
)
```

Imagen después de realizar las operaciones morfológicas:



5. Clasificación y etiquetado de regiones:

- Primero se usa la función `cv2.connectedComponentsWithStats` para etiquetar y analizar las regiones conectadas en la imagen binaria resultante.
- Luego se itera sobre cada componente conectada para evaluar si tiene características que sugieren que podría ser un cuadrado.
- Para cada región, se calcula la relación de área y perímetro (ρ) y se verifica si esta región tiene la forma de un cuadrado.
- Según la circularidad (`flag_circ`) y el área de la región, se asignan etiquetas de colores a la región en la imagen final (`labeled_image`).
- También, se crea una máscara cuadrada (`square_mask`) para la sección siguiente del conteo en los datos..

```

(
    componentes_conectadas,
    etiquetas,
    estadisticas,
    centroides,
) = cv2.connectedComponentsWithStats(result, cv2.CV_32S, connectivity=8)

squares_masks = []
aux = np.zeros_like(result)
labeled_image = cv2.merge([aux, aux, aux])

RHO_TH = 0.83

```

```

RHO_TH = 0.83
for i in range(1, componentes_conectadas):
    mascara = np.uint8(etiquetas == i)
    contorno, _ = cv2.findContours(mascara, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    area = cv2.contourArea(contorno[0])
    perimetro = cv2.arcLength(contorno[0], True)
    rho = 4 * np.pi * area / (perimetro**2)
    flag_circ = rho > RHO_TH
    if flag_circ:
        if area > 123000:
            labeled_image[mascara == 1, 0] = 255
        elif area > 85000:
            labeled_image[mascara == 1, 1] = 255
        else:
            labeled_image[mascara == 1, 2] = 255
    else:
        labeled_image[mascara == 1, 2] = 120
        labeled_image[mascara == 1, 1] = 120

    square_mask = np.zeros_like(result)
    cv2.drawContours(square_mask, contorno, -1, 120, 2)
    squares_masks.append(square_mask)

dst = cv2.addWeighted(imagen, 0.7, labeled_image, 0.3, 0)

```

Resultado de hacer la operación de clasificación y etiquetado:



6. Detección y el conteo de en los dados:

- Se itera sobre cada máscara de cuadrado previamente identificada en la sección anterior.
- Se rellenan los contornos de las máscaras y se aplica una operación de AND bit a bit con la imagen Canny para obtener los bordes en la mascara cuadrada..
- Se aplica una operación de dilatación a la imagen resultante de la operación AND para identificar mejor las componentes dentro del dado.
- Se utiliza nuevamente la función `cv2.connectedComponentsWithStats` para identificar y analizar las regiones conectadas en la imagen dilatada de los dados.
- Luego, se itera sobre estas regiones.
- Se cuentan los dados que cumplen con criterios específicos de área y circularidad, para determinar cuál es el número de la cara de arriba. Para esto se tiene en cuenta que esta cara tiene círculos con una forma más circular que la de las caras de los costados.
- Finalmente se muestra el número de la cara superior de cada dado.

```

RHO_TH = 0.78

for id, square_mask in enumerate(squares_masks):
    square_mask = fill_contours(square_mask)
    squares = cv2.bitwise_and(canny, canny, mask=square_mask)

    squares_dilatados = dilate_image(squares)

    (
        componentes_conectadas,
        etiquetas,
        estadisticas,
        centroides
    ) = cv2.connectedComponentsWithStats(squares_dilatados, cv2.CV_32S, connectivity=8)
    count = 0

```

```

for i in range(1, componentes_conectadas):
    area = estadisticas[i, cv2.CC_STAT_AREA]
    mascara = np.uint8(etiquetas == i)
    contorno, _ = cv2.findContours(mascara, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    if area < 10000 and area > 500:
        area = cv2.contourArea(contorno[0])
        perimetro = cv2.arcLength(contorno[0], True)
        rho = 4 * np.pi * area / (perimetro**2)
        flag_circ = rho > RHO_TH
        if flag_circ:
            count += 1
print(f"Dado {id + 1} tiene: {count}")

```

Imagen de la máscara del dado 1:

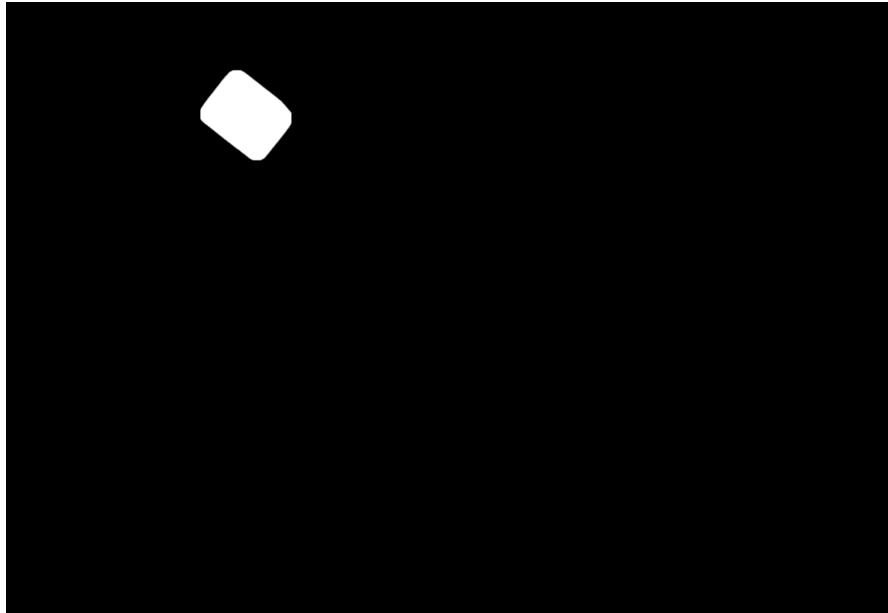
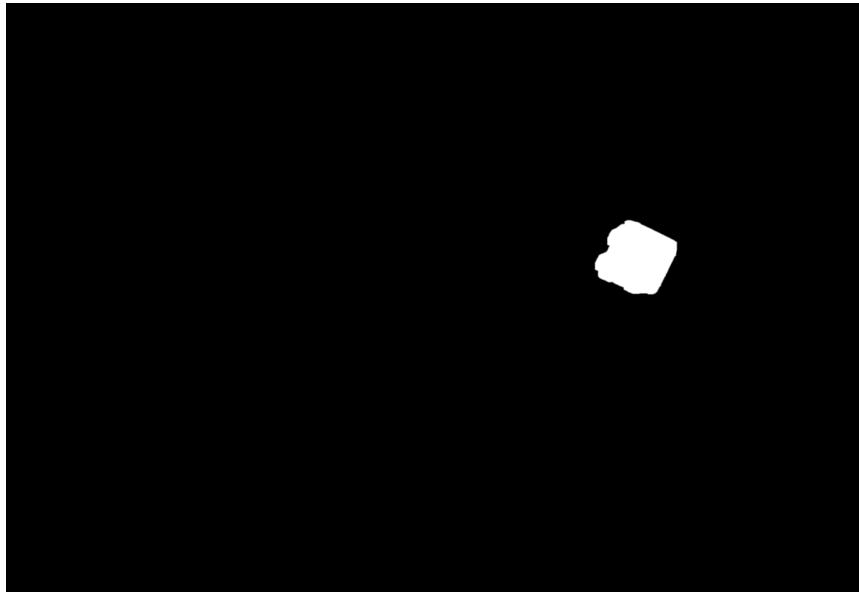
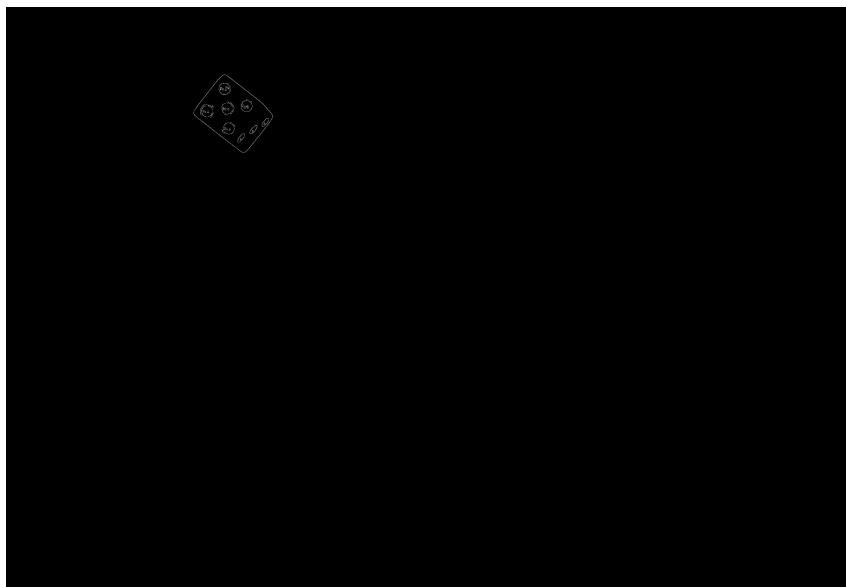


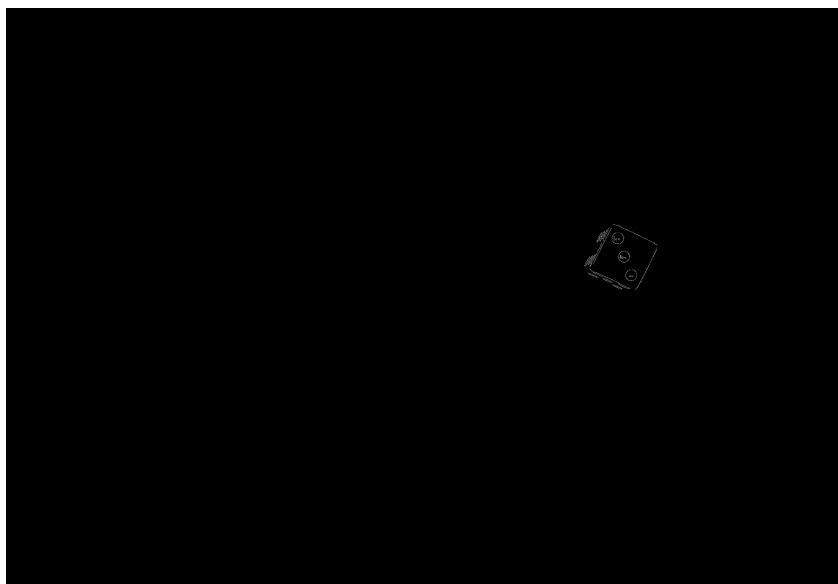
Imagen de la máscara del dado 2



Resultado de aplicar Canny al dado 1:



Resultado de aplicar Canny al dado 2:



Resultado del conteo de las componentes que cumplen la condición de círculo:

```
Dado 1 tiene: 5  
Dado 2 tiene: 3
```

Conclusiones:

Elementos de interés detectados:

- Se logró la detección de los elementos pedidos en la imagen, incluyendo las monedas de distintos tamaños y los dados, a pesar del fondo con ruido y difuminado en escala de grises.

Segmentación automática:

- La aplicación de técnicas de procesamiento de imágenes, como la dilatación, erosión y detección de contornos, nos permitió la segmentación automática de monedas y dados en la imagen.

Clasificación de monedas:

- Se implementó una clasificación automática de monedas basada en el área de los contornos detectados, con umbrales específicos que determinamos mediante prueba y error, para diferenciar entre monedas de distintos valores.

Detección del número en los dados:

- Se logró determinar el número presentado por cada dado mediante un análisis automático de los contornos y la aplicación de umbrales adecuados para tal fin..

Visualización clara de los resultados:

- La representación visual de los resultados, con colores diferenciados para distintos tipos de monedas y datos, facilita la interpretación de los resultados.

Reflexión Final:

Eficiencia del Proceso:

- La implementación modular y sistemática del proceso facilitó la comprensión y depuración del código, permitiendo abordar con eficiencia la detección y clasificación de objetos en la imagen.

Ajustes:

- Tuvimos que hacer muchos cambios de parámetros y ajustes en el código para dar con el resultado final.

Desafíos Superados:

- Se superaron desafíos relacionados con el ruido en la imagen, la segmentación de elementos y la clasificación precisa. Para esto segmentamos y aplicamos morfología varias veces.

Optimización de Umbrales:

- La selección cuidadosa de umbrales para la clasificación de monedas y datos fue crucial para obtener resultados precisos, demostrando la importancia de ajustar parámetros según las características de la imagen.

Visualización:

- La representación visual final, con colores diferenciados para monedas y datos, contribuye significativamente a la claridad y comprensión de los resultados obtenidos.

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.



Figura 2 – Imagen con monedas y datos.

Resolución:

El primer paso fue hacer una inspección visual de la imagen para determinar de que manera abordar el problema. Observamos que las patentes tienen distintos ángulos y además hay una que no se ve claramente el recuadro blanco de la patente. Esto podría ser un problema a la hora de poder encontrar la patente en la imagen. Otra cosa importante que vimos es que detrás de la patente, osea, el color del auto, en algunos casos es gris y se dificulta encontrar un contraste con la patente.

Este problema lo abordamos de distintas maneras con resultados variados. En ninguno de los casos logramos detectar las 12 patentes con ninguno de los métodos probados. Dos de las soluciones que más patentes encontramos fue:

- 1) Usar primero un *threshold*, despues *connectedComponentsWithStats*, de esas componentes filtrar por áreas y por relación de aspecto. Luego nos quedamos con las imágenes que se reconocieron 5 o 6 letras de la patente. Seguidamente obtenemos un recuadro que las contiene y ahí detectamos las patentes. Luego con esa imagen separada segmentamos los caracteres. Pero con esta solución detectamos pocas patentes.
- 2) La segunda es la optamos por elegir debido a que es la que detectó más patentes. A continuación se procede a explicar el proceso.

El código proporcionado es una implementación en Python utilizando la biblioteca OpenCV para el procesamiento de imágenes.

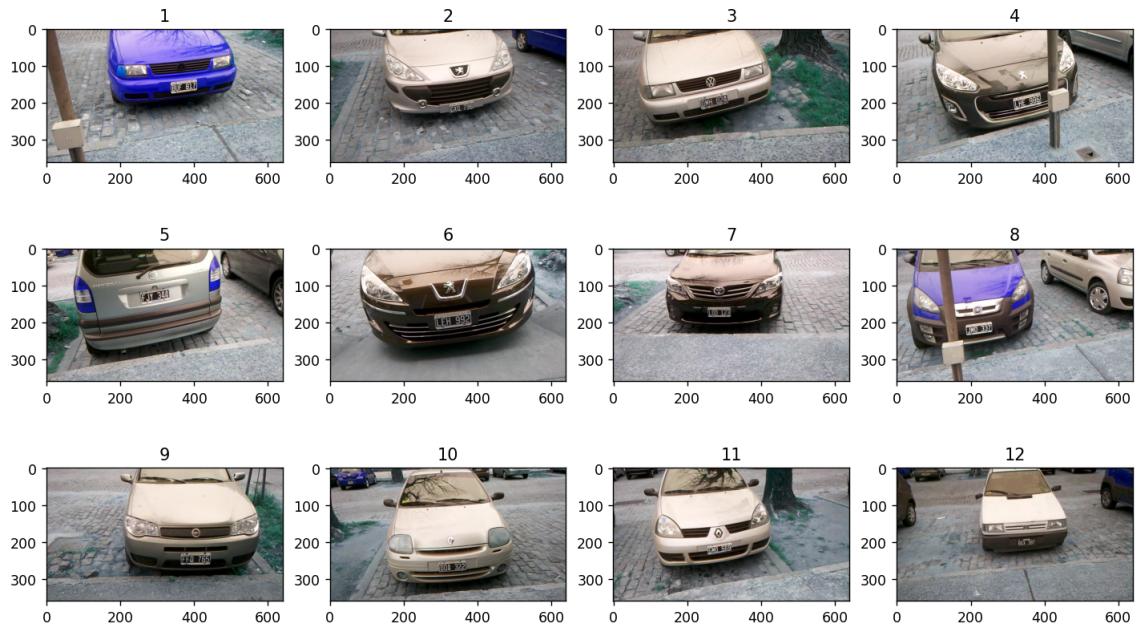
PARTE A

1. Carga de imágenes:

```
imgs = [cv2.imread(f"img{i:02n}.png") for i in range(1, 13)]  
rois = E2_A(imgs)
```

Se cargan todas las imágenes, para luego ser procesadas todas juntas. A continuación se llama a la función que realiza todo el proceso.

Visualización de todas las imágenes:



2. Umbralado y morfología:

```
# UMBRALADO
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (37, 8))
img = cv2.morphologyEx(img, cv2.MORPH_TOPHAT, kernel)
img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
```

En este paso, la imagen se convierte a escala de grises y se aplica una operación morfológica de tophat, que resalta las regiones más claras (en este caso, las patentes) en la imagen.

Resultado de esta parte para cada imagen:





2. Filtrado por color HSV:

```
# FILTRAMOS POR COLOR HSV
img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
img_mask = cv2.inRange(img, (0, 0, 101), (180,10,182))
img = cv2.bitwise_and(img, img, mask=img_mask)
img = cv2.cvtColor(img, cv2.COLOR_HSV2BGR)
```

En este paso, la imagen se convierte al espacio de color HSV y se aplica un umbral para poder filtrar un poco más y quedarnos con la patente que está en escala de grises.

Resultado de esta parte para cada imagen:





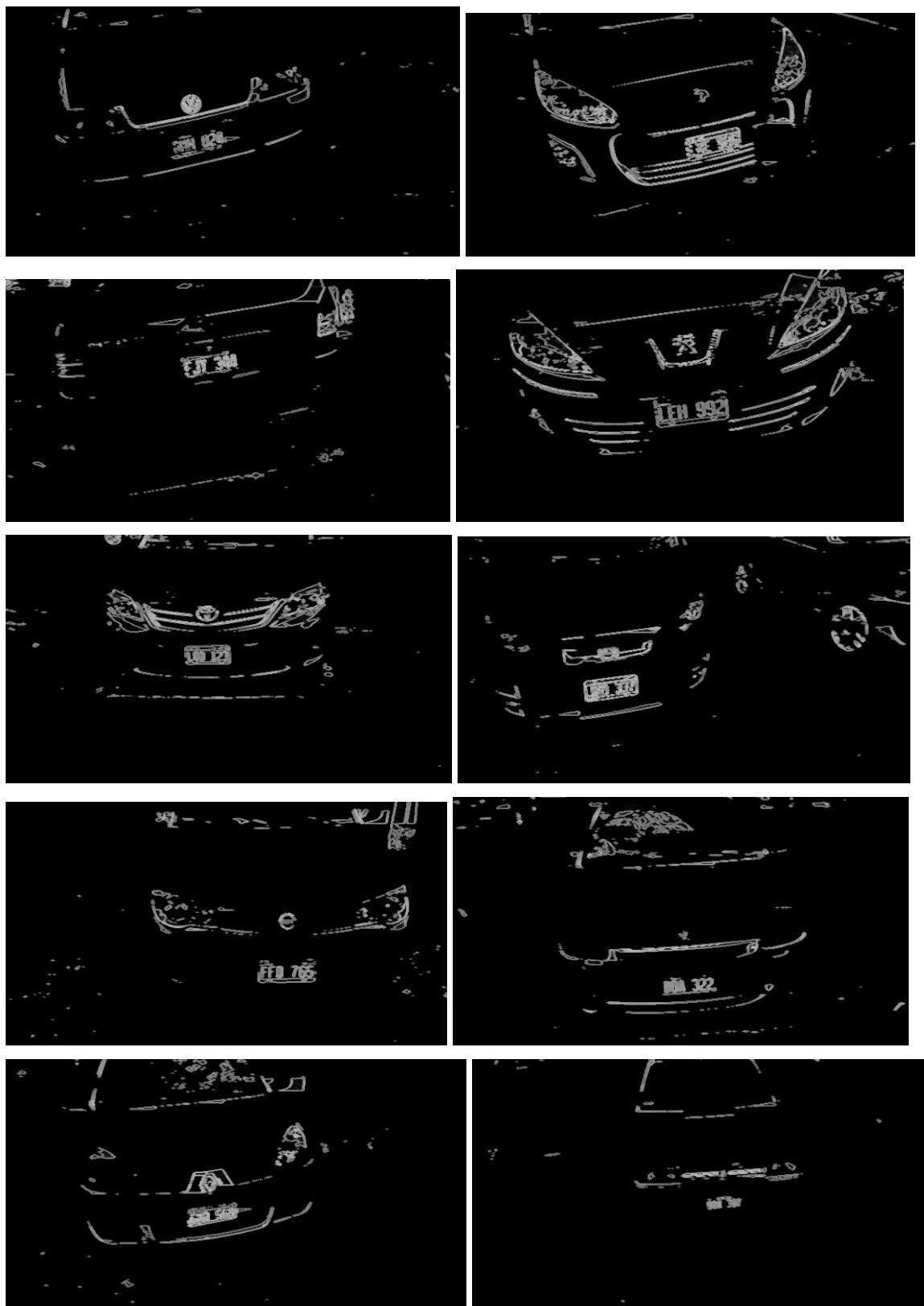
3. Morfología y gradiente:

```
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
clau_ker = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5,3))
img = cv2.morphologyEx(img, cv2.MORPH_GRADIENT, clau_ker)
```

Se realiza nuevamente una operación morfológica, esta vez un gradiente morfológico, que destaca los bordes de las regiones.

Resultado de esta parte para cada imagen:





4. Etiquetado y selección de regiones:

```

nlabels, labels, stats, centroids = cv2.connectedComponentsWithStats(img)
aux_rois=[]
for idx in range(1, nlabels):
    x = stats[idx, cv2.CC_STAT_LEFT]
    y = stats[idx, cv2.CC_STAT_TOP]
    w = stats[idx, cv2.CC_STAT_WIDTH]
    h = stats[idx, cv2.CC_STAT_HEIGHT]
    area = stats[idx, cv2.CC_STAT_AREA]
    if 537 < area < 2349 and 1.739 < w/h < 3.7:
        aux_rois.append((x, y), (x + w, y + h))

assert aux_rois!=[], "NO SE ENCONTRO NINGUNA PATENTE"

```

Se etiquetan las regiones conectadas en la imagen y se seleccionan aquellas que cumplen con ciertos criterios de área y relación de aspecto, lo que se espera de las patentes.

5. Etiquetado y selección de regiones:

```

assert aux_rois!=[], "NO SE ENCONTRO NINGUNA PATENTE"

if len(aux_rois)==1:
    rois.append(aux_rois[0])
else:
    results=[]
    for (X,Y),(dX,dY) in aux_rois:
        img_aux=imgs[i].copy()
        img_aux=img_aux[Y:dY,X:dX]
        img_aux=cv2.cvtColor(img_aux, cv2.COLOR_BGR2GRAY)
        area=img_aux.shape[0]*img_aux.shape[1]
        n_white_pixels = np.sum(img_aux > 200)/area
        results.append(n_white_pixels)

    patente_idx = np.argmin(results)
    rois.append(aux_rois[patente_idx])

```

Se asegura de que al menos una patente sea encontrada. Si hay exactamente una patente, se agrega a la lista de regiones de interés (rois). Si hay más de una región candidata, se selecciona la patente final basándose en la proporción de píxeles blancos en la región.

6. Visualización de resultados:

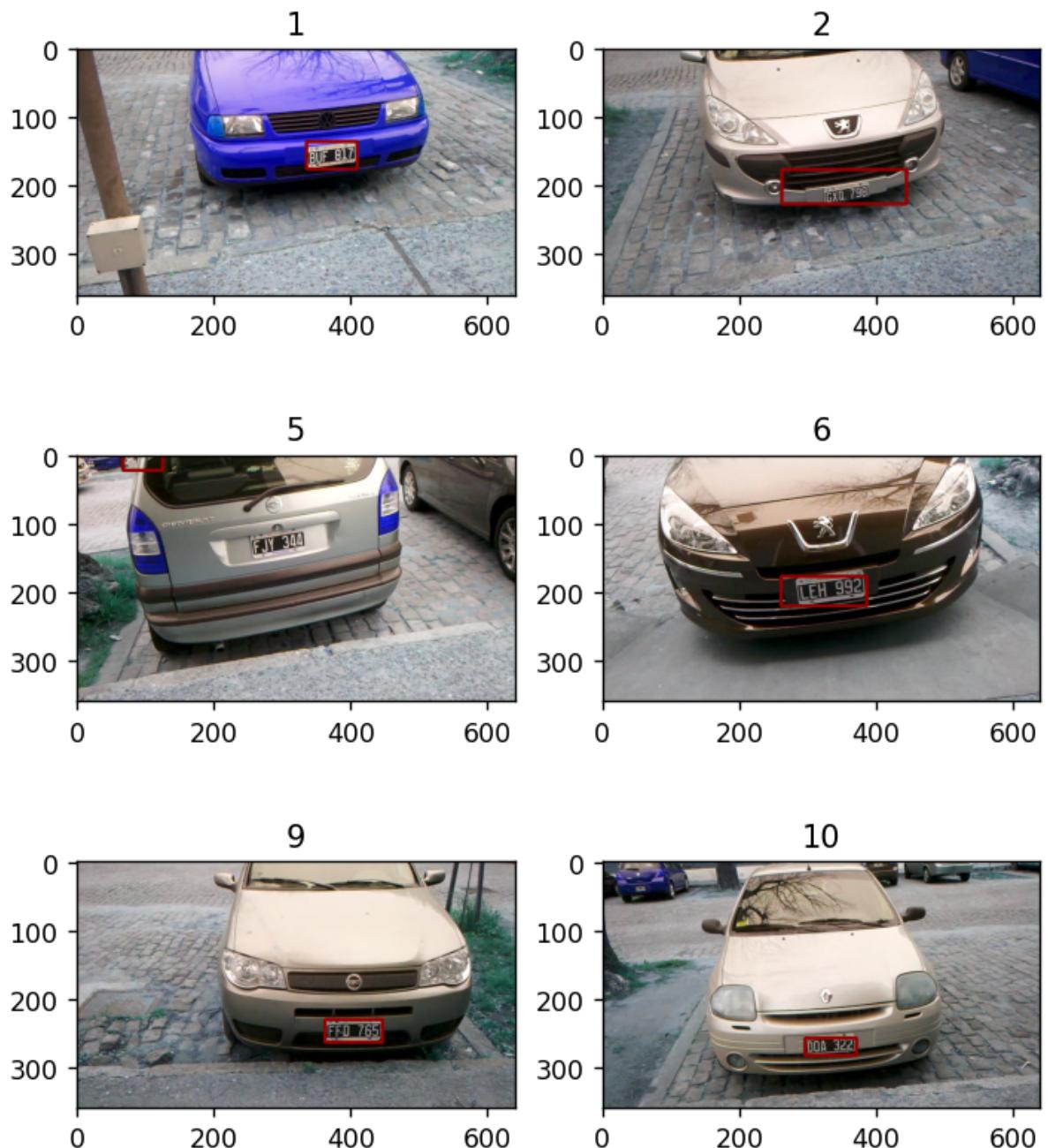
```
fig, axs = plt.subplots(3, 4, figsize=(14, 14))

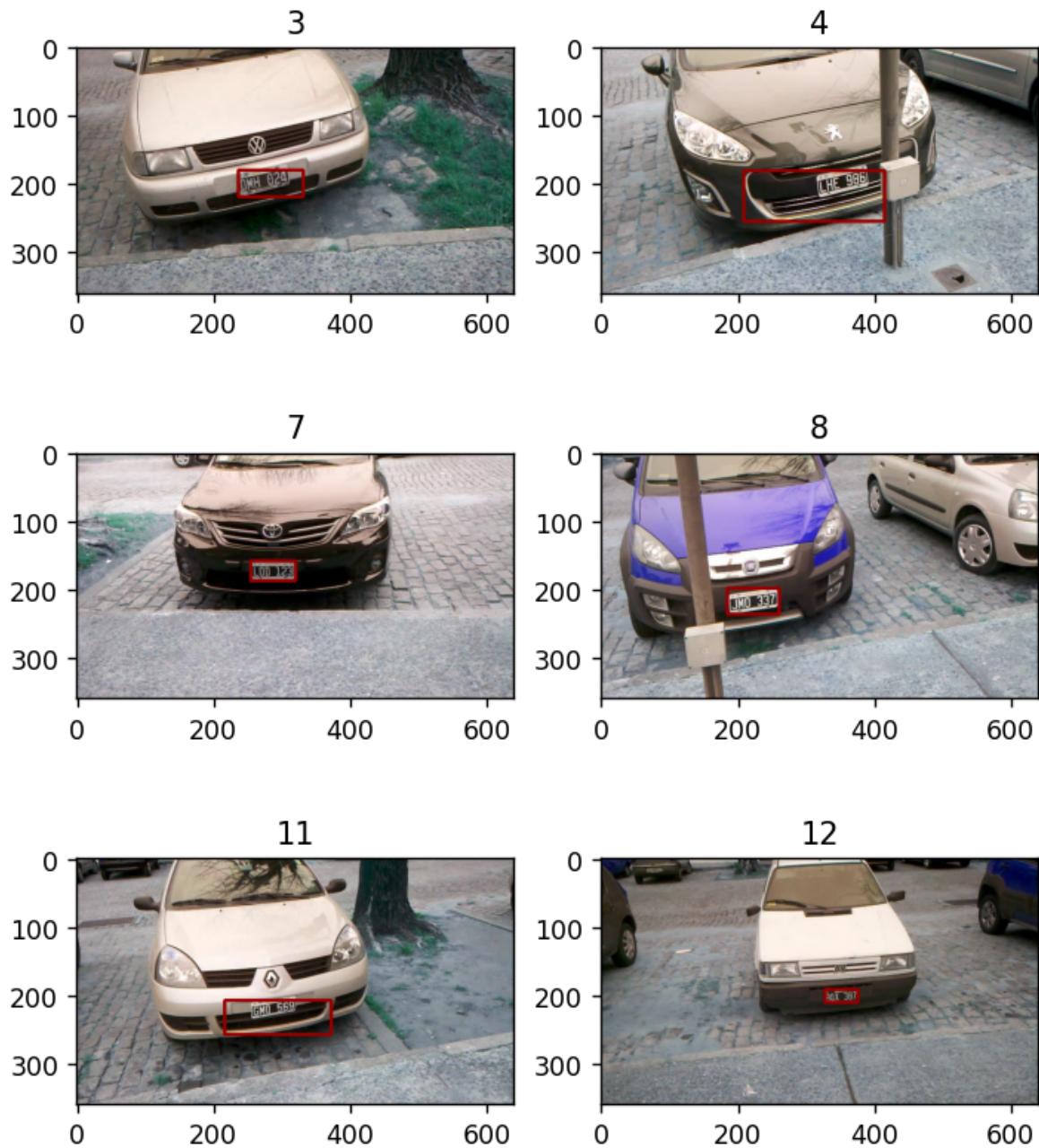
for i, img in enumerate(imgs):
    (X,Y),(dX,dY) = rois[i]
    cv2.rectangle(imgs[i], (X, Y), (dX,dY), 150, 1)
    axs[i // 4, i % 4].cla()
    axs[i // 4, i % 4].imshow(imgs[i], cmap="gray")
    axs[i // 4, i % 4].set_title(i + 1)

plt.show()
```

Finalmente, se visualizan las imágenes originales con las regiones de interés (patentes) destacadas mediante rectángulos.

Resultado de la detección de patentes:





PARTE B

La parte B del código se encarga de segmentar los caracteres de la patente previamente detectada en la Parte A.

1. Conversión a escala de grises y morfología:

```
img = cv2.cvtColor(img_patente, cv2.COLOR_BGR2GRAY)
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (45, 12))
img = cv2.morphologyEx(img, cv2.MORPH_TOPHAT, kernel)
```

En este paso, la imagen de la patente se convierte a escala de grises y se aplica una operación morfológica de tophat, similar al paso realizado en la Parte A. Esta operación resalta las regiones más claras de la imagen.

2. Umbralado:

```
img_mask = cv2.inRange(img, 104, 255)
img = cv2.bitwise_and(img, img, mask=img_mask)
```

Se aplica un umbral para segmentar los caracteres en la imagen. Los píxeles cuyo valor está en el rango de 104 a 255 se mantienen, mientras que los demás se descartan.

3. Filtrado por color HSV:

```
img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
img_mask = cv2.inRange(img, (0, 0, 73), (180, 255, 231))
img = cv2.bitwise_and(img, img, mask=img_mask)
img = cv2.cvtColor(img, cv2.COLOR_HSV2BGR)
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

La imagen se convierte nuevamente a color, esta vez a HSV, y se realiza un filtrado por color para resaltar los caracteres de la patente.

4. Etiquetado y selección de regiones:

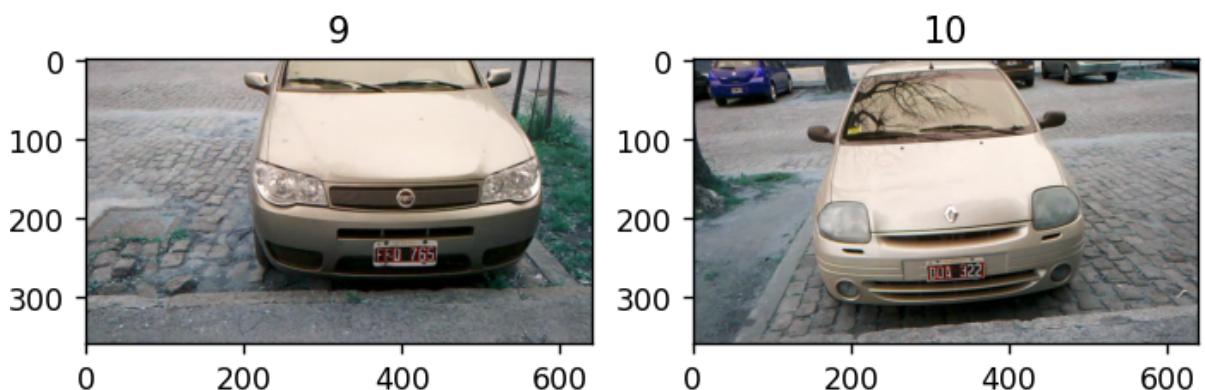
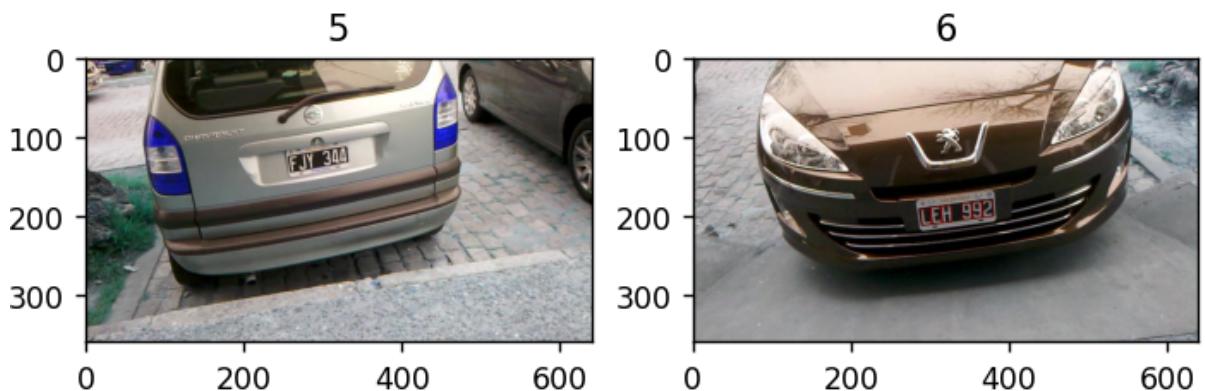
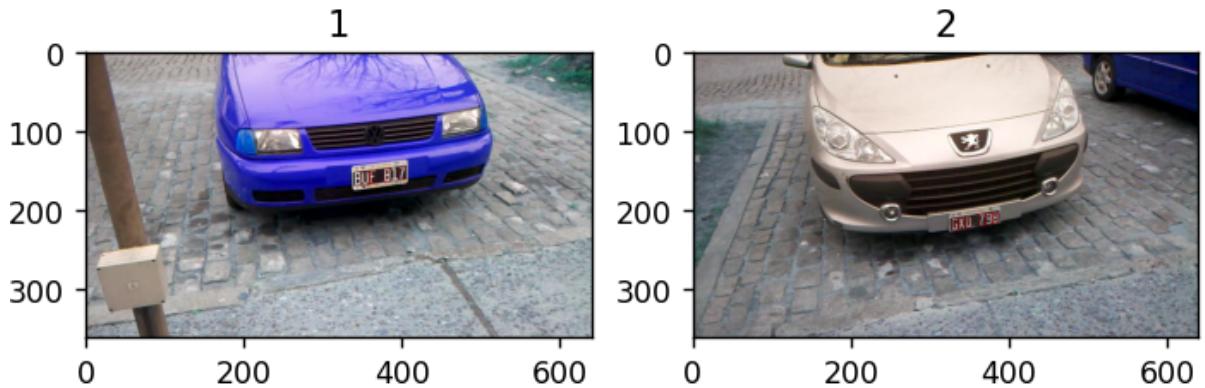
```
for idx in range(1, nlabels):
    x = stats[idx, cv2.CC_STAT_LEFT]
    y = stats[idx, cv2.CC_STAT_TOP]
    w = stats[idx, cv2.CC_STAT_WIDTH]
    h = stats[idx, cv2.CC_STAT_HEIGHT]
    area = stats[idx, cv2.CC_STAT_AREA]
    if 12 < area < 200 and h/w > 1:
        rois.append((x, y), (x + w, y + h))
```

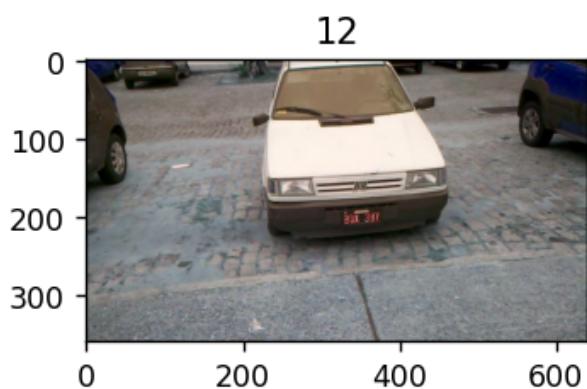
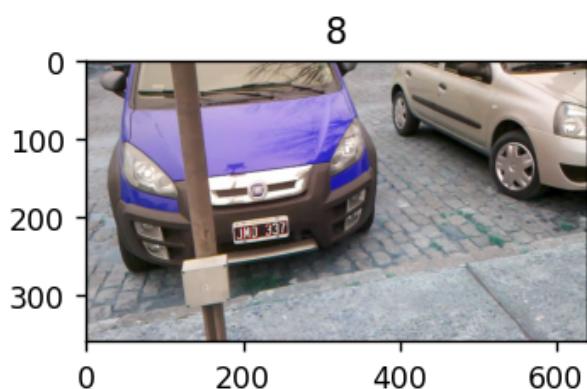
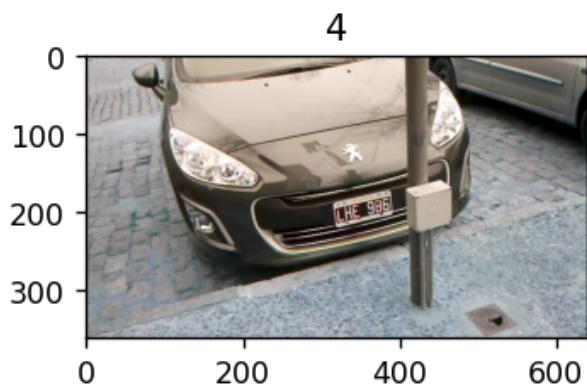
Se etiquetan las regiones conectadas en la imagen de los caracteres de la patente y se seleccionan aquellas que cumplen con ciertos criterios de área y relación de aspecto.

5. Visualización de resultados:

```
for i, img in enumerate(imgs):
    patente_roi = Problema_2_A(img)
    (X,Y),(dX,dY) = patente_roi
    rois = Problema_2_B(imgs[i][Y:dY,X:dX])
    axs[i // 4, i % 4].cla()
    for letra_roi in rois:
        (x,y),(dx,dy) = letra_roi
        cv2.rectangle(imgs[i], (X+x, Y+y), (X+dx,Y+dy), 150, 1)
    axs[i // 4, i % 4].imshow(img, cmap="gray")
    axs[i // 4, i % 4].set_title(i + 1)
plt.show()
```

Finalmente, se visualizan las imágenes originales con las regiones de interés (caracteres de la patente) destacadas mediante rectángulos. La región de la patente se obtiene previamente utilizando la función de la Parte A (`Problema_2_A`).





Las patentes que pudimos reconocer todos los caracteres:





Conclusiones:

Tratamos de resolver el problema de varias formas diferentes sin encontrar una solución 100% efectiva. De todas las maneras que probamos nos quedamos con la que mejor resultado nos dió. Sin embargo no en la detección de patentes pudimos reconocer 11 patentes. Para el caso de reconocer los caracteres se pudo detectar los 6 en sólo 5 de las imágenes. Nos resultó difícil poder hacer que los mismos parámetros y metodologías funcionen para todas las imágenes, porque solo un pequeño cambio alteraba la detección en las otras imágenes.