TRABAJO PRÁCTICO N°2

Procesamiento de Imágenes I

Avecilla Tomás

Calcia Franco

25 de noviembre de 2024

TECNICATURA UNIVERSITARIA EN INTELIGENCIA ARTIFICIAL - UNR

Índice

Introducción

El presente informe tiene como objetivo presentar el desarrollo y los resultados obtenidos en el Trabajo Práctico N° 2, correspondiente al segundo semestre del año 2024. Este trabajo aborda dos problemas fundamentales relacionados con la detección y clasificación de objetos en imágenes utilizando técnicas de procesamiento digital de imágenes.

En el **Problema 1**, se propone diseñar un algoritmo capaz de segmentar, clasificar y realizar un conteo automático de monedas y dados presentes en una imagen con un fondo de intensidad no uniforme. Para las monedas, se identificaron diferentes tipos según su tamaño y valor; para los dados, se determinó el número representado en la cara superior de cada uno. Este proceso involucra etapas como el preprocesamiento de la imagen, la detección de bordes y la segmentación de objetos mediante análisis de contornos.

Por su parte, en el **Problema 2**, se busca automatizar la detección y segmentación de placas patentes en imágenes de vehículos, así como el reconocimiento individual de sus caracteres. Este problema se abordó mediante técnicas avanzadas de segmentación y análisis de regiones, con énfasis en la extracción precisa de los caracteres para su posterior reconocimiento.

Ambos problemas destacan la importancia del procesamiento de imágenes en aplicaciones prácticas, tales como la clasificación automática de objetos y la identificación vehicular. Las herramientas empleadas incluyeron bibliotecas como OpenCV, y los resultados de cada etapa de procesamiento se documentaron con el objetivo de evaluar la efectividad y la precisión de los algoritmos diseñados.

Materiales y Herramientas

Para el desarrollo del presente trabajo práctico, se utilizó lo siguiente:

Software y Librerías

1. **Python**: Lenguaje de programación empleado para implementar los algoritmos.
2. **Biblioteca OpenCV**: Para realizar operaciones de procesamiento de imágenes como:
    - Conversión a escala de grises.
    - Detección de bordes (Canny Edge Detection).
    - Segmentación y análisis de contornos.
3. **NumPy**: Para el manejo eficiente de matrices y operaciones numéricas.
4. **Matplotlib/Seaborn**: Para la visualización de imágenes y gráficos.
5. **Editor de código**: Para este caso se utilizó Visual Studio Code.

Procedimiento: Problema 1 - Detección y Clasificación de Monedas y Dados

1. Carga y Preprocesamiento de la Imagen

- **Carga de la imagen original:** Se utiliza `cv2.imread` para leer la imagen desde el archivo `monedas.jpg`.
- **Conversión a escala de grises:** Se convierte la imagen original a escala de grises usando `cv2.cvtColor`, facilitando el análisis de intensidad y bordes.
- **Aplicación de desenfoque Gaussiano:** Se aplica un filtro gaussiano con un kernel de 5x5 para reducir el ruido y suavizar la imagen.
- **Detección de bordes (Canny):** Utilizando el algoritmo Canny, se detectan bordes significativos con umbrales de 80 y 180.
- **Dilatación de bordes:** Se dilatan los bordes detectados para mejorar la conectividad de los objetos con iteraciones amplias (`iterations=15`), facilitando la detección de contornos más definidos.

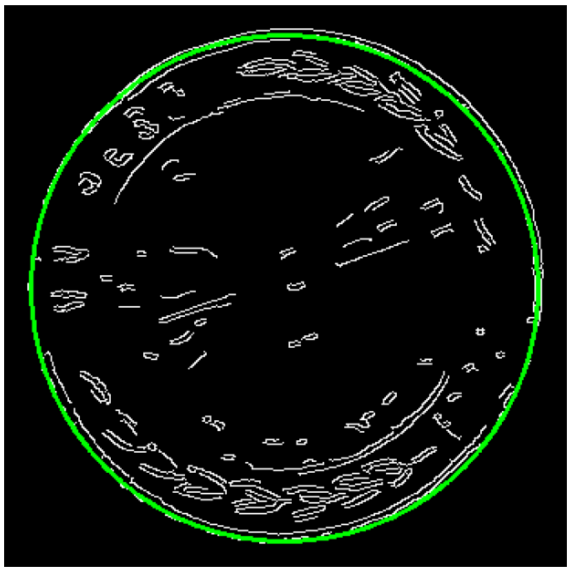
2. Detección de Contornos

- **Obtención de contornos externos:** Con `cv2.findContours` y la opción `cv2.RETR_EXTERNAL`, se detectan los contornos más externos, descartando detalles internos.
- **Dibujo de contornos:** Se copian los contornos detectados sobre la imagen original con `cv2.drawContours`, ayudando a visualizar las áreas detectadas.
- **Resultados:** Se muestran los contornos superpuestos y se imprime el número total de objetos detectados que para este caso es de 19.

3. Clasificación de Monedas

- **Iteración por contornos detectados:** Cada contorno se analiza para determinar si corresponde a una moneda o un dado.
- **Extracción de regiones de interés (ROI):** Con `cv2.boundingRect`, se extraen regiones delimitadas por cada contorno.
- **Detección de círculos:** Usando la Transformada de Hough Circular (`cv2.HoughCircles`), se buscan círculos en cada recorte para identificar monedas. Para esta ocasión fuimos corriendo el código y retocando los parámetros para llegar al resultado final.
- **Clasificación por área:** Una vez detectado lo que son monedas lo que hicimos fue basarnos en el área del círculo y nos dimos cuenta que cada grupo de moneda se relacionan demasiado las áreas por lo cual decidimos clasificarlas como:
    - **1 peso:** Área entre 65000 y 89000.
    - **50 centavos:** Área mayor a 90000.
    - **10 centavos:** Área entre 50000 y 62000.
- **Visualización:** Se dibujan y muestran los círculos detectados en los recortes correspondientes.
- **Recortes no clasificados:** Si no se detectan círculos, el índice del recorte se guarda como posible dado.
- **Suma total:** Para la parte final del código y como extra decidimos imprimir cuanto es la suma total de las monedas que encontramos.

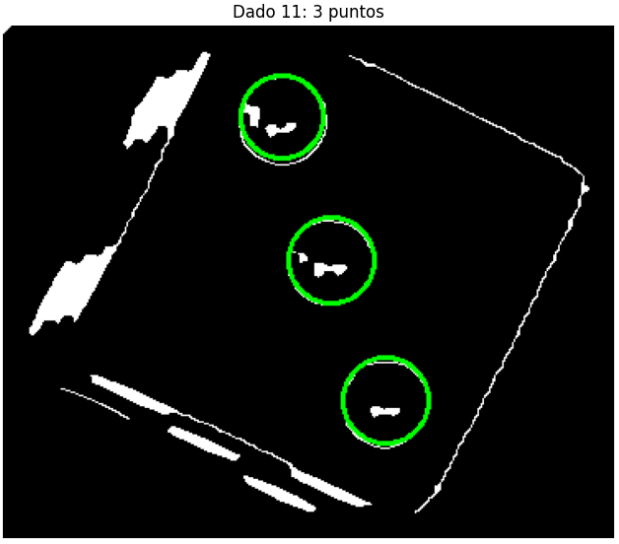
Recorte 1: Círculo Detectado

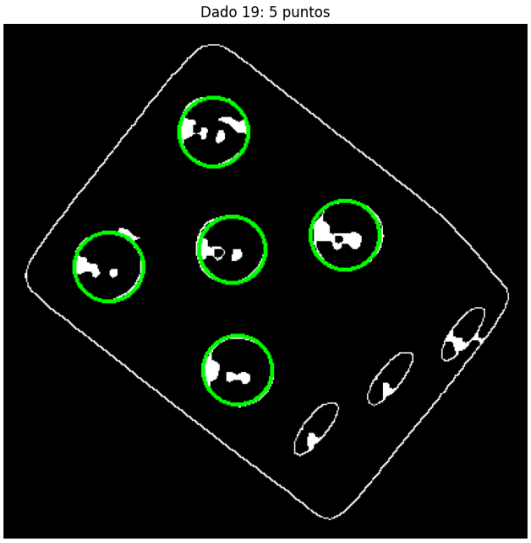


Recorte 12: Círculo Detectado

4. Procesamiento de Dados

- **Aplicación de morfología:** Los recortes descartados como monedas se procesan con operaciones morfológicas (`cv2.morphologyEx`) para reducir ruido y conectar puntos en las caras de los dados. Esto lo realizamos ya que ciertas formas dentro del dado eran detectados como círculos por ende para evitar errores decidimos aplicar lo siguiente:
    - **Apertura:** Elimina ruido mediante la erosión seguida de dilatación.
    - **Cierre:** Une regiones cercanas eliminando pequeños espacios entre puntos.
- **Detección de puntos:** Nuevamente, se usa la Transformada de Hough Circular para detectar los puntos en las caras de los dados pero con parámetros distintos ya que los círculos son más chicos.
- **Conteo de puntos:** Los puntos detectados representan el número en la cara superior del dado y luego se imprimen por pantalla.
- **Visualización:** Se dibujan los puntos sobre los recortes y se muestra el resultado.





5. Resultados Finales

- **Conteo de monedas:** Se imprime el número de monedas clasificadas por tipo y el valor total sumado en pesos.
- **Resultados de los dados:** Se muestra cuántos puntos tiene cada dado en su cara superior.
- **Visualización integral:** Se presentan las imágenes en cada etapa (original, preprocesada, con contornos, recortes de monedas y dados) para documentar

gráficamente el proceso.

Procedimiento: Problema 2 - Detección y segmentación de patentes de vehículos.

1. Visualización de Imágenes

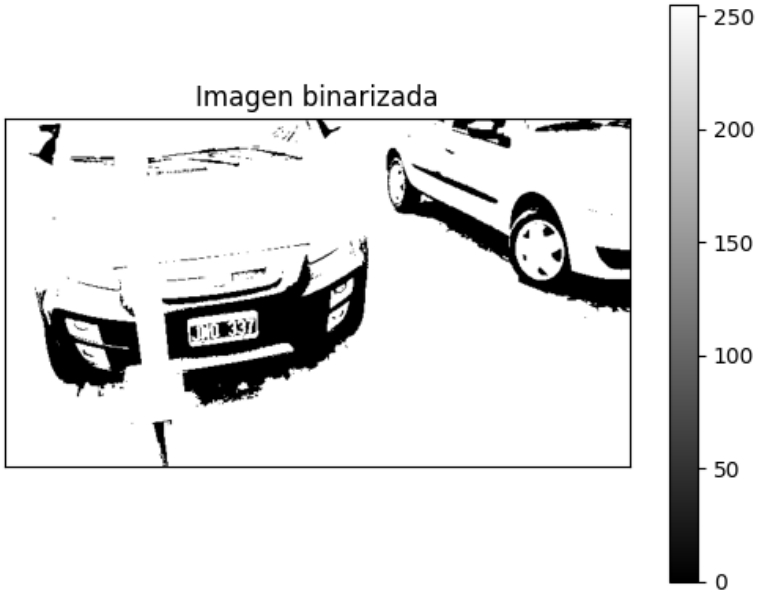
Definición de la función `imshow`

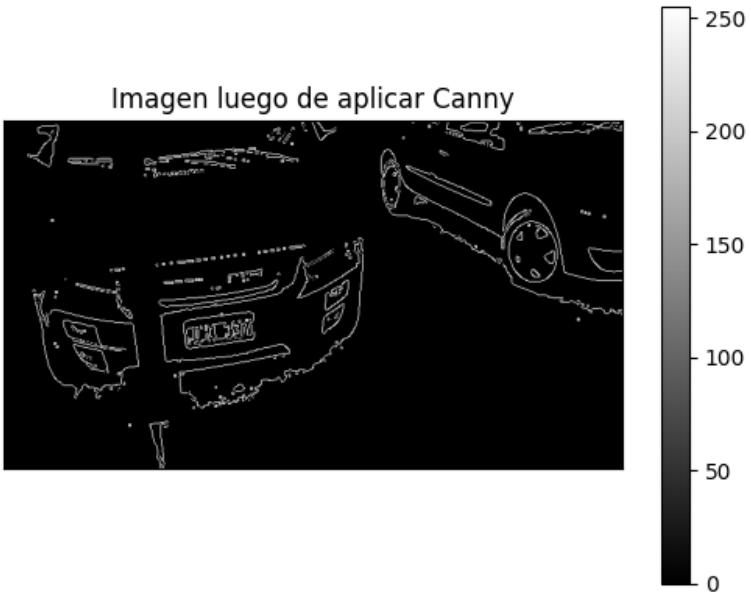
1. **Creación de una nueva figura**: Se utiliza `plt.figure()` para crear un nuevo gráfico donde se mostrará la imagen.
2. **Manejo del color de la imagen**:
    - Si `color_img` es `True`, la imagen se convierte de formato BGR (propio de OpenCV) a RGB (propio de Matplotlib) para que los colores se visualicen correctamente.
    - Si no, la imagen se muestra en escala de grises.
3. **Configuración del título**: Si se proporciona un título, este se muestra sobre la imagen.
4. **Opcional: Ocultar ejes**: Si `ticks=False`, se ocultan los valores numéricos de los ejes X e Y para una visualización más limpia.
5. **Agregar una barra de colores**: Si `colorbar=True`, se muestra una barra indicando la intensidad de los píxeles (escala de grises o colores).
6. **Mostrar la imagen**: Finalmente, la imagen se renderiza en pantalla usando `plt.show()`.

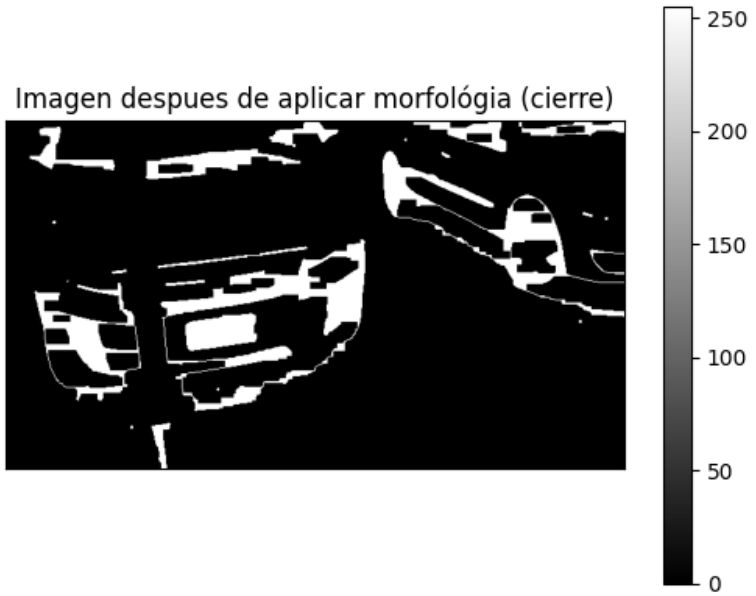
2. Preprocesamiento de la Imagen

Definición de la función `preprocesar_imagen`

1. **Conversión a escala de grises**: Se convierte la imagen original a escala de grises para reducir la información de color y trabajar solo con intensidades de píxel.
2. **Binarización**: Se aplica un umbral fijo para dividir los píxeles en dos categorías: blanco y negro, eliminando detalles innecesarios.
3. **Detección de bordes**: El algoritmo de Canny resalta los bordes significativos en la imagen binarizada. Esto facilita identificar contornos claros.
4. **Cierre morfológico**:
    - Se utiliza un kernel rectangular de tamaño **(9, 4)**. Este tamaño se selecciona en función de las características esperadas de los objetos que queremos identificar, en este caso, las patentes de vehículos.
    - La operación de cierre combina dilatación (expande los bordes de las regiones blancas) y erosión (reduce el ruido adicional introducido por la dilatación), rellenando huecos en los bordes detectados para mejorar la conectividad de las formas.





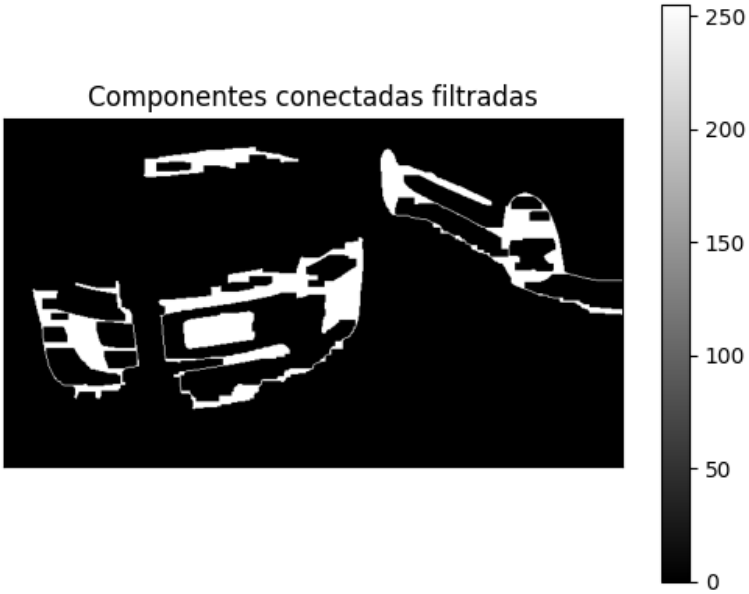


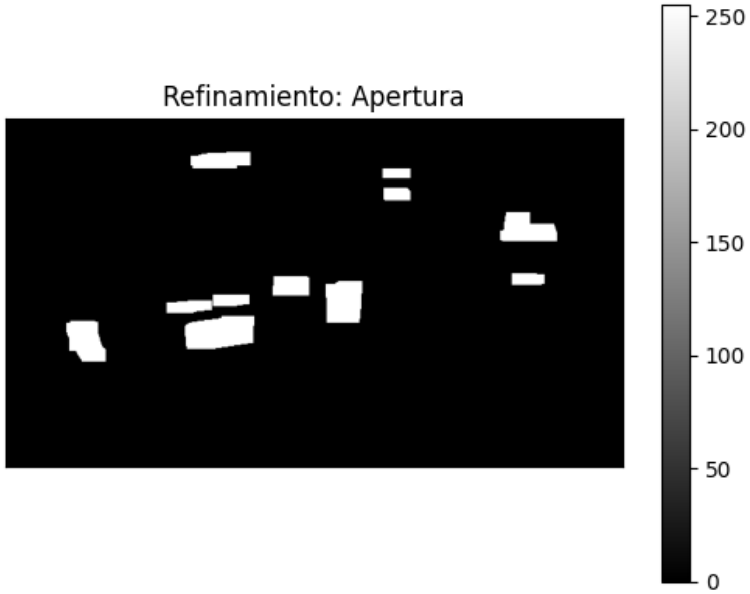
3. Filtrado de Componentes Conectadas

Definición de la función `filtrar_componentes_conectadas`

- **Identificación de componentes conectadas**: Con `cv2.connectedComponentsWithStats`, se detectan grupos de píxeles conectados que representan posibles objetos.
- **Creación de una imagen filtrada**: Se inicializa una imagen vacía (negra).
- **Filtrado por área**:
  - Se itera sobre cada componente conectada (excepto el fondo).

    - Si el área del objeto es mayor que el valor mínimo especificado (`min_area = 1500`), se conserva en la imagen filtrada. Con este criterio nos ayudamos a eliminar elementos pequeños o ruidos que no tienen relevancia en el análisis.
- **Visualización del resultado**: Se muestra la imagen con los objetos conectados que cumplieron el criterio de área.
- **Refinamiento (Apertura):**
    - Luego de ejecutar esta función, en la main se realiza un refinamiento adicional mediante la operación morfológica de **apertura**, que combina erosión y dilatación.
    - Este paso permite eliminar detalles irrelevantes y suavizar los bordes de los objetos detectados quedándose con rectángulos, y así generar una imagen aún más limpia para los pasos posteriores.



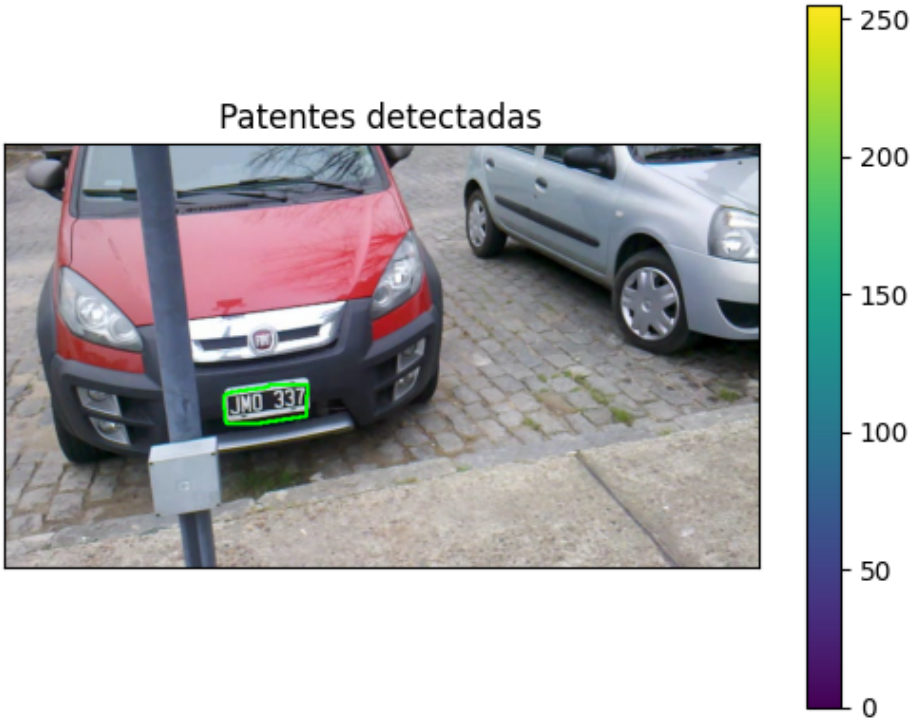


4. Detección de Patentes

Definición de la función `encontrar_patentes`

1. **Extracción de contornos externos**: Se utilizan los contornos de la imagen binaria filtrada para identificar posibles patentes.

2. **Iteración sobre los contornos**:
    - Cada contorno detectado se procesa con la función `cv2.approxPolyDP` para simplificar su forma, reduciendo el número de vértices del contorno.
    - La aproximación se controla mediante un parámetro llamado **epsilon**, que define la distancia máxima permitida entre el contorno original y el polígono aproximado. En este caso **epsilon** se calcula como un porcentaje de la longitud del contorno (usando `epsilon_factor * cv2.arcLength(contour, True)`), lo que adapta automáticamente el nivel de simplificación al tamaño del contorno.
    - Después de la aproximación, se verifica si el polígono tiene **4 lados**, lo que indica que podría ser una forma rectangular, como la de una patente de vehículo.
    - Se evalúan también las dimensiones del rectángulo (ancho y alto) y su relación de aspecto (**aspect ratio**) para asegurarse de que cumpla con los rangos típicos esperados:
        - El ancho y alto deben estar dentro de los valores mínimos y máximos definidos (`rect_min_size` y `rect_max_size`).
        - La relación de aspecto (ancho/alto) debe estar entre **2.1 y 3.4**, que corresponde a las proporciones comunes de una placa de patente Argentina.
3. **Recorte de regiones de interés**:
    - Para cada rectángulo válido, se extrae una región de la imagen original ampliada con márgenes.
    - Las regiones se almacenan en una lista como posibles patentes.
4. **Visualización y retorno**: Se muestra la imagen original con los contornos detectados y se retorna la lista de posibles patentes.



5. Recorte de la Primera Patente

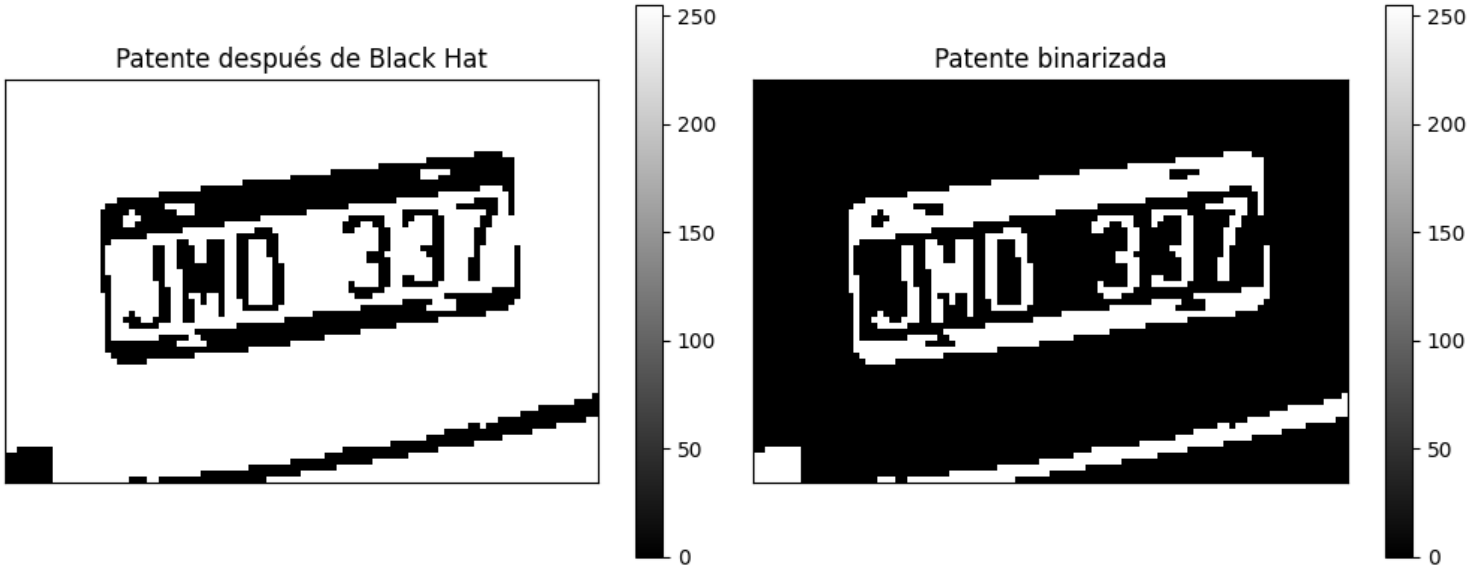
Definición de la función `recortar_primer_patente`

1. **Verificación de patentes detectadas**: Si no hay ninguna patente en la lista, se imprime un mensaje y la función termina.
2. **Conversión a escala de grises**: La primera patente detectada se convierte a escala de grises para simplificar su análisis.
3. **Visualización del recorte**: Se muestra la imagen de la patente recortada.

6. Preprocesamiento de Patentes

Definición de la función `preprocesar_patente`

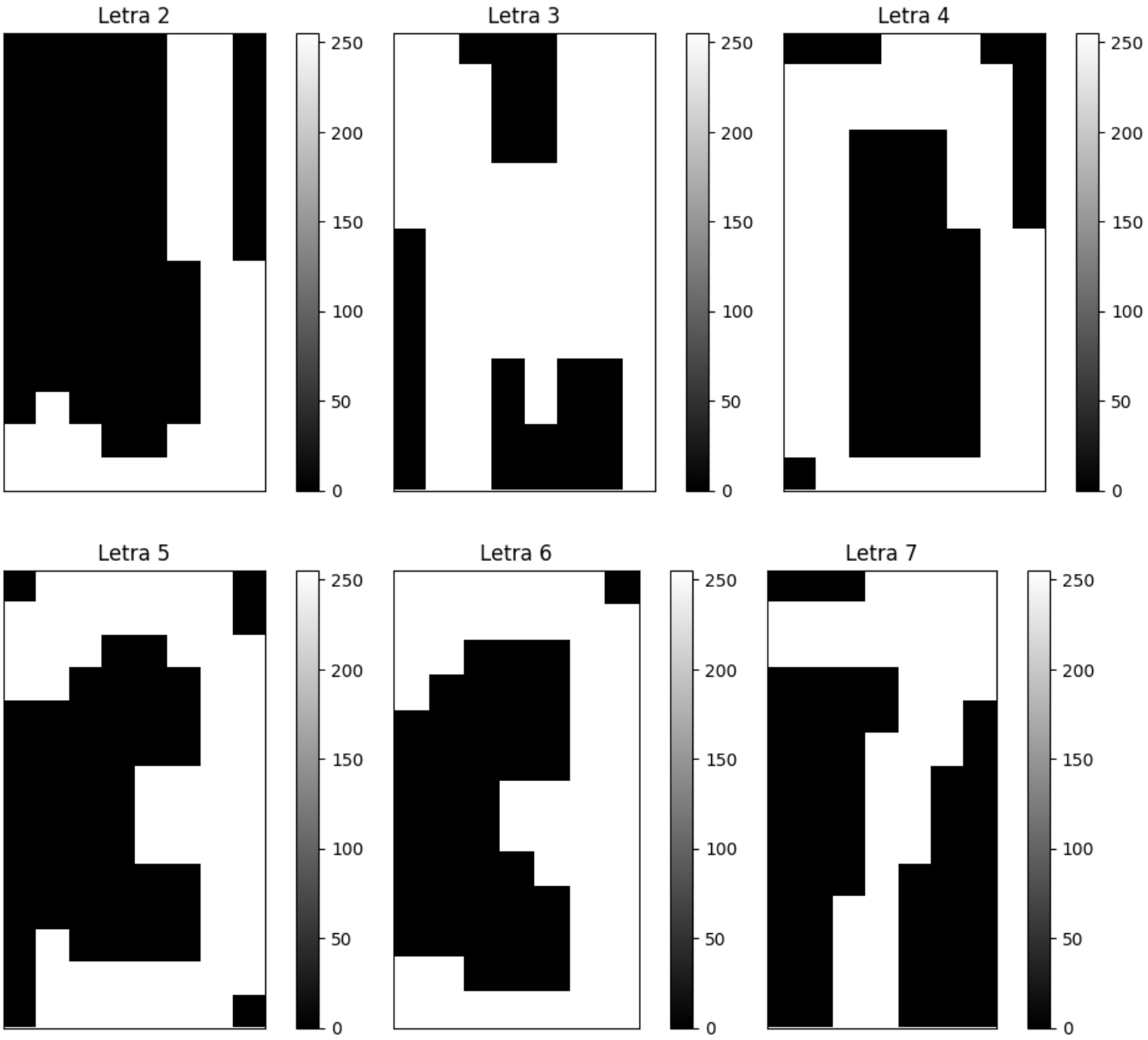
1. **Creación de un kernel estructurante grande**: Esto se usa para realizar una operación morfológica denominada **Black Hat**.
2. **Aplicación de Black Hat**:
    - Resalta las áreas oscuras (fondo) rodeadas por áreas claras (caracteres de la patente).
    - Esto facilita el análisis y segmentación de las letras.
3. **Binarización**: Se aplica un umbral para separar claramente las letras del fondo.



7. Segmentación de Letras

Definición de la función `procesar_patente`

1. **Preprocesamiento de la patente**: Se llama a `preprocesar_patente` para resaltar los caracteres de la patente.
2. **Binarización inversa**: Se invierten los colores de la imagen para que las letras sean blancas sobre fondo negro, facilitando la segmentación.
3. **Segmentación por componentes conectadas**: Se identifican las letras individuales como regiones conectadas.
4. **Filtrado por área**:
    - Se eliminan componentes demasiado pequeñas (ruido) o demasiado grandes (no son letras).
    - Las letras válidas se recortan y almacenan junto con su área.
5. **Orden de las letras**: Las letras se ordenan de izquierda a derecha según su posición en la patente.



8. Flujo Principal del Programa

Definición de la función `main`

1. **Carga de imágenes**: Se procesan imágenes secuenciales (`img01.png`, `img02.png`, etc.). Si alguna imagen no puede cargarse, se muestra un mensaje de error.
2. **Preprocesamiento de imágenes**:
    - Se llama a `preprocesar_imagen` para realizar las etapas de binarización,

detección de bordes y operaciones morfológicas.
    - Se filtran los componentes conectados para eliminar el ruido.
    - Se aplica una operación de **apertura morfológica** como refinamiento final.
3. **Detección de patentes**:
    - Se buscan regiones rectangulares que coincidan con las dimensiones esperadas de una patente.
    - Si se detectan, se muestra la cantidad encontrada.
4. **Procesamiento de la primera patente**:
    - Si hay al menos una patente, se recorta la primera y se segmentan sus letras.
    - Se muestra cada letra individual junto con su área.

9. Ejecución del Programa

Llamada a la función `main`

1. Si el archivo se ejecuta directamente (no como módulo), se invoca `main`, iniciando el procesamiento para todas las imágenes definidas.

Conclusión Final

En este proyecto logramos resolver ambos problemas planteados, aunque con distintos niveles de complejidad.

En el primer caso, el proceso fue relativamente sencillo debido a la nitidez de la imagen y a un fondo uniforme. Bastó con ajustar algunos parámetros básicos para que los objetos fueran correctamente detectados sin mayores inconvenientes. Esto demuestra cómo la calidad de la imagen inicial influye directamente en la facilidad del procesamiento y los resultados.

Por otro lado, el segundo problema presentó mayores desafíos. El recorte y detección de las patentes requirió múltiples intentos y ajustes, ya que las imágenes tenían una alta variabilidad en términos de fondo, iluminación, color y posición del vehículo. Para abordar estas dificultades, probamos diferentes métodos de procesamiento hasta encontrar una combinación que permitiera obtener relativamente buenos resultados:

1. Conversión a blanco y negro.
2. Binarización para destacar regiones de interés.
3. Aplicación del algoritmo de Canny para la detección de bordes.
4. Operaciones morfológicas como cierre y apertura, que fueron claves para reducir el ruido y aislar las regiones rectangulares correspondientes a posibles patentes.
5. Uso de componentes conectadas para filtrar elementos no deseados, como pequeñas imperfecciones del fondo.

A pesar de lograr identificar las patentes en la mayoría de las imágenes, aún enfrentamos ciertos inconvenientes. Por ejemplo, en algunas situaciones, el sistema detectó letras falsas debido a que los elementos de fondo coincidían en tamaño o forma con las letras reales. Esto resalta las limitaciones de los métodos utilizados cuando las imágenes no tienen un fondo uniforme o presentan objetos con características similares a las de las letras.

En cuanto a la segmentación y reconocimiento de las letras, este fue otro desafío importante. Aunque ajustamos los parámetros para que la mayoría de las patentes fueran correctamente procesadas, las variaciones en el fondo y las condiciones de las imágenes dificultaron obtener resultados consistentes. Sabemos que herramientas avanzadas como librerías de OCR (Reconocimiento Óptico de Caracteres) están diseñadas específicamente para esta tarea y habrían proporcionado resultados más robustos. Sin embargo, no contamos con estas herramientas en este caso, lo que nos obligó a explorar soluciones alternativas y adaptarnos a las limitaciones del enfoque basado en procesamiento de imágenes.

Este trabajo, aunque desafiante, nos permitió profundizar en el análisis, experimentar con distintos métodos y explorar soluciones para problemas reales. Nos enfrentamos a diversas dificultades, pero este proceso nos motivó a pensar críticamente, ajustar estrategias y seguir iterando hasta alcanzar resultados satisfactorios. Si bien los resultados no fueron perfectos, nos quedamos satisfechos con lo que conseguimos.