

Procesamiento de Imágenes I - TUIA - FCEIA, UNR

TRABAJO PRÁCTICO N° 2

Estudiantes:

- Alejandro Armas
- Facundo Ferreira daCamara
- Gabriel Soda

01/12/2025

Introducción

El presente trabajo práctico tiene como objetivo el desarrollo de algoritmos de visión por computadora para resolver problemas de segmentación, detección y clasificación automática de objetos.

A continuación se describen los dos problemas planteados para este entregable:

Problema 1: Detección y clasificación de monedas y datos

Este ejercicio propone el análisis de la imagen `monedas.jpg`, la cual presenta un escenario con un fondo de intensidad no uniforme donde se encuentran dispersos dados y monedas de distintos tamaños y valores.

Los objetivos específicos son:

1. **Segmentación:** Separar automáticamente las monedas y los dados del fondo.
2. **Clasificación de monedas:** Identificar los tipos de monedas y realizar su conteo.
3. **Lectura de datos:** Determinar el número de la cara superior y contabilizar.

Desarrollo del Problema 1

1. Carga y Análisis Inicial

Para comenzar, cargamos la imagen en escala de grises y analizamos su histograma e intensidades. Definimos también una función auxiliar `imshow` para visualizar los resultados a lo largo del informe.

```
# importacion de librerias
import cv2
import numpy as np
import matplotlib.pyplot as plt

# función para mostrar las imágenes
def imshow(img, new_fig=True, title=None, color_img=False, blocking=True, colorbar=True, ticks=True):
    if new_fig:
        plt.figure(figsize=figsize)
    if color_img:
        plt.imshow(img)
    else:
        plt.imshow(img, cmap='gray')
    plt.title(title)
    if not ticks:
        plt.xticks([]), plt.yticks([])
    if colorbar:
        plt.colorbar()
    if new_fig:
        plt.show(block=blocking)

mon = cv2.imread('monedas.jpg', cv2.IMREAD_GRAYSCALE)
imshow(mon, title="Imagen Original (Escala de Grises)")
```



Resultado:

- Se observa la imagen original cargada correctamente.
- *Observación:* La imagen presenta un fondo texturizado y no uniforme que podría dificultar la segmentación simple por umbralado global.

2. Pre-procesamiento y Selección de Detector de Bordes

Nuestro primer enfoque fue utilizar **HoughCircles**, pero requiere bordes muy bien definidos. Probamos inicialmente con **Sobel** y **Laplaciano de Gauss (LoG)**. Sin embargo, estos métodos resultaron muy sensibles al ruido del fondo o requerían un post-procesamiento (umbralado y filtros de mediana) complejo para limpiar la imagen.

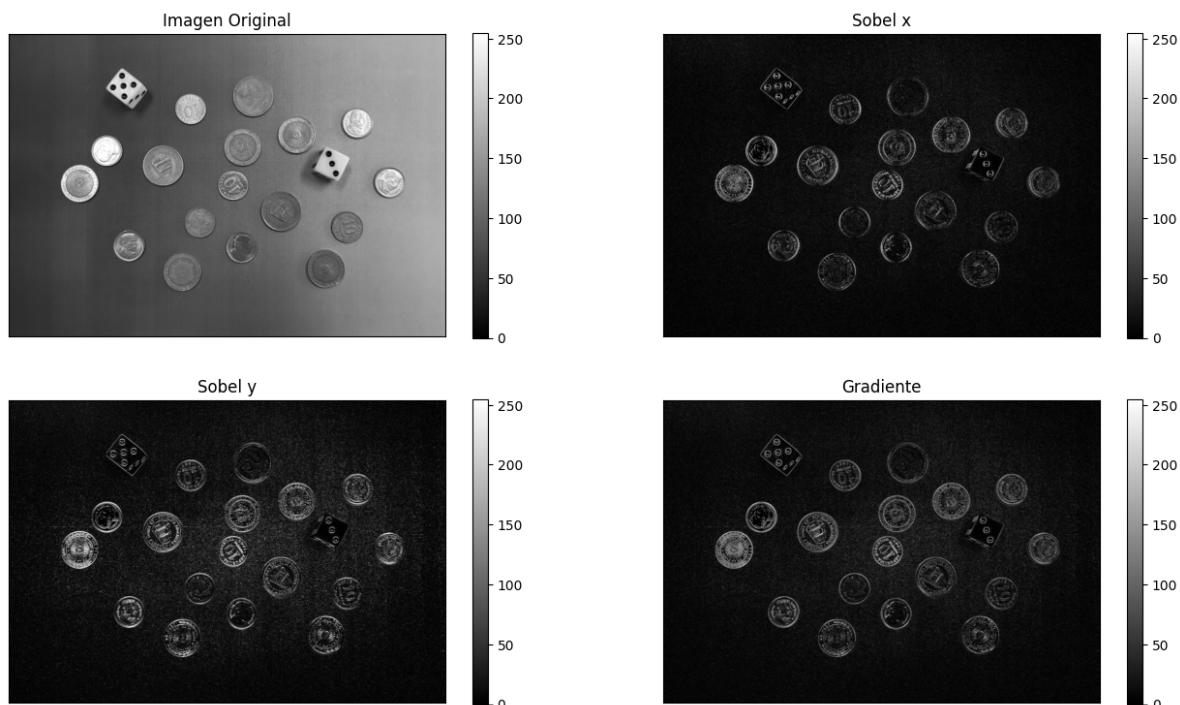
```
# detección de bordes con gradiente
ddepth = cv2.CV_16S
grad_x = cv2.Sobel(mon, ddepth, 1, 0, ksize=3)
grad_y = cv2.Sobel(mon, ddepth, 0, 1, ksize=3)
np.unique(grad_x)
np.unique(grad_y)
```

```

# Acondicionamiento
abs_grad_x = cv2.convertScaleAbs(grad_x)
abs_grad_y = cv2.convertScaleAbs(grad_y)
grad = cv2.addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0)

plt.figure(figsize=(16, 9))
ax = plt.subplot(221)
imshow(mon, new_fig=False, title="Imagen Original")
plt.subplot(222, sharex=ax, sharey=ax), imshow(abs_grad_x, new_fig=False, title="Sobel x", ticks=[0, 50, 100, 150, 200, 250])
plt.subplot(223, sharex=ax, sharey=ax), imshow(abs_grad_y, new_fig=False, title="Sobel y", ticks=[0, 50, 100, 150, 200, 250])
plt.subplot(224, sharex=ax, sharey=ax), imshow(grad, new_fig=False, title="Gradiente", ticks=[0, 50, 100, 150, 200, 250])
plt.show(block=False)

```



```

# Umbralizamos los gradientes
abs_grad_x_th = np.zeros_like(abs_grad_x)
abs_grad_x_th[abs_grad_x == abs_grad_x.max()] = 255

abs_grad_y_th = np.zeros_like(abs_grad_y)
abs_grad_y_th[abs_grad_y == abs_grad_y.max()] = 255

grad_th = np.zeros_like(grad)

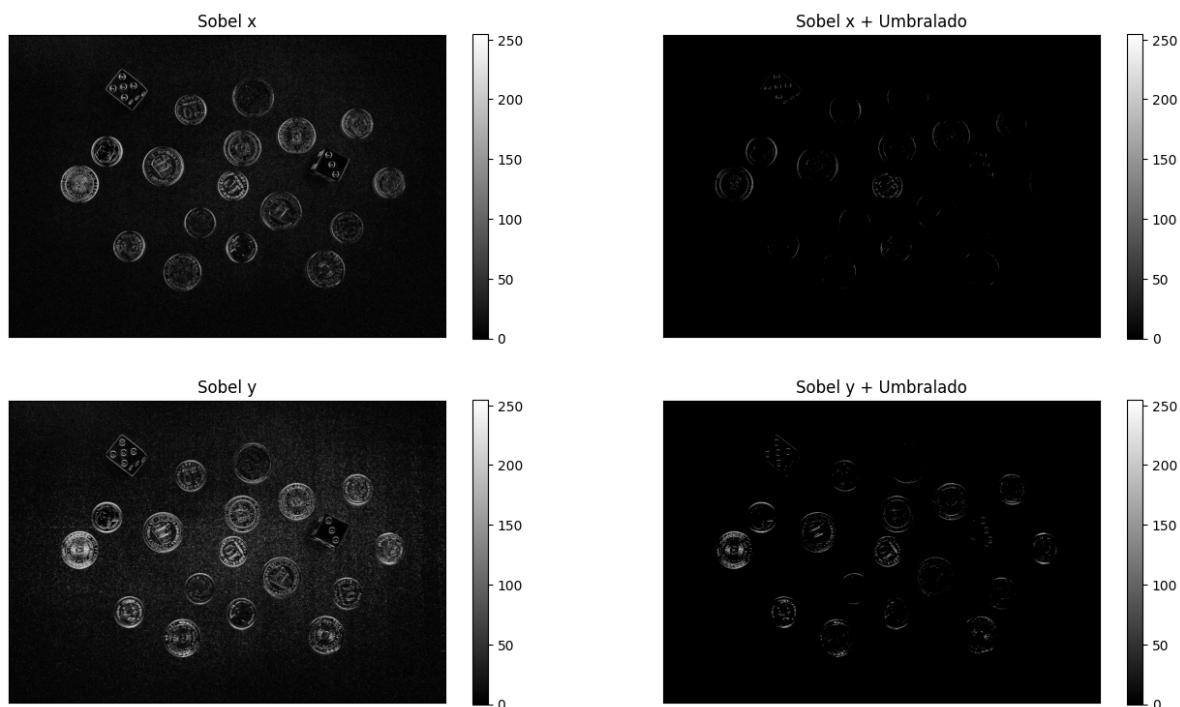
```

```

grad_th[grad >= 0.5*grad.max()] = 255

plt.figure(figsize=(16, 9))
ax = plt.subplot(221)
imshow(abs_grad_x, new_fig=False, title="Sobel x")
plt.subplot(222, sharex=ax, sharey=ax), imshow(abs_grad_x_th, new_fig=False, title="Sobel x + Umbralado")
plt.subplot(223, sharex=ax, sharey=ax), imshow(abs_grad_y, new_fig=False, title="Sobel y")
plt.subplot(224, sharex=ax, sharey=ax), imshow(abs_grad_y_th, new_fig=False, title="Sobel y + Umbralado")
plt.show(block=False)

```



Como se observa en los resultados, tanto sobel, como sobel + umbralado, no consiguen destacar los bordes de los objetos sin mucho ruido del fondo. Esto se debe principalmente a la uniformidad que presenta haciendo imposible encontrar un umbral que elimine todo el ruido.

Antes de aplicar la Transformada de Hough, realizamos un análisis de bordes mediante el algoritmo Canny para determinar la sensibilidad adecuada.

Previo a la detección, aplicamos un GaussianBlur para suavizar aún más la textura de la mesa.

```

# Canny
mon = cv2.imread('monedas.jpg', cv2.IMREAD_GRAYSCALE) # cargo de nuevo por si alguna función

```

```

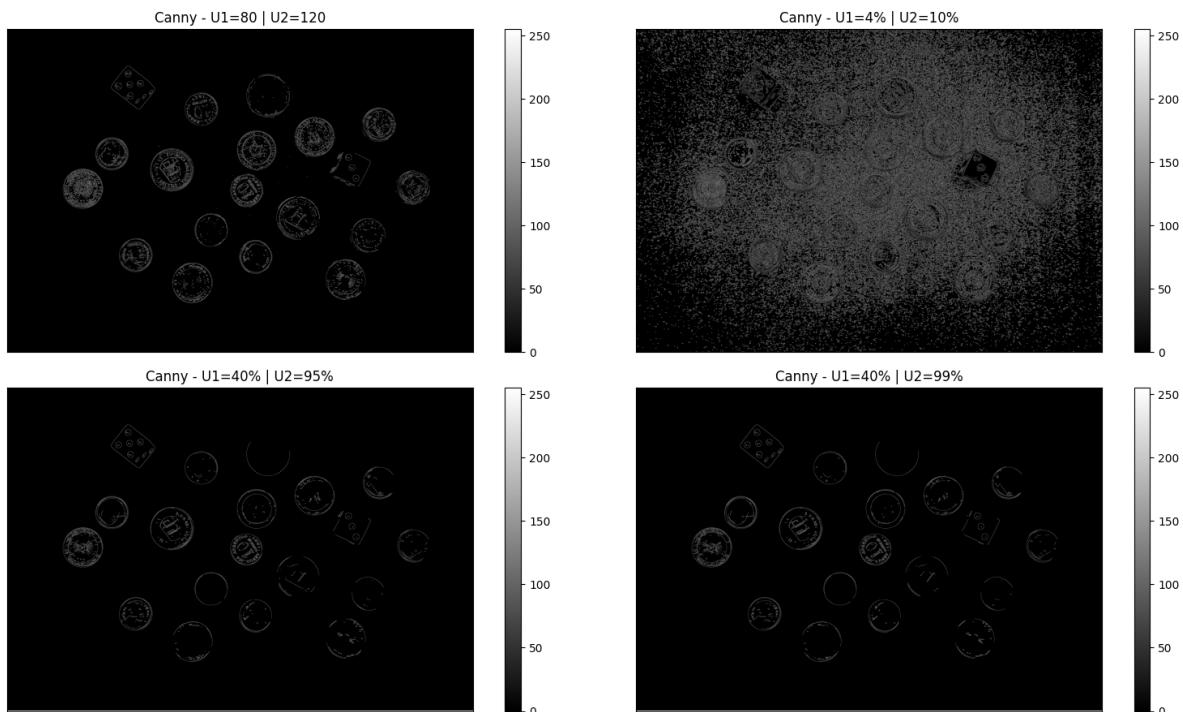
f.blur = cv2.GaussianBlur(mon, ksize=(5, 5), sigmaX=1.5)

gcan = cv2.Canny(f.blur, threshold1=80, threshold2=120)

gcan1 = cv2.Canny(f.blur, threshold1=0.04*255, threshold2=0.1*255)
gcan2 = cv2.Canny(f.blur, threshold1=0.4*255, threshold2=0.95*255)
gcan3 = cv2.Canny(f.blur, threshold1=0.4*255, threshold2=0.99*255)

plt.figure(figsize=(16, 9))
ax = plt.subplot(221)
imshow(gcan, new_fig=False, title="Canny - U1=80 | U2=120")
plt.subplot(222, sharex=ax, sharey=ax)
imshow(gcan1, new_fig=False, title="Canny - U1=4% | U2=10%")
plt.subplot(223, sharex=ax, sharey=ax)
imshow(gcan2, new_fig=False, title="Canny - U1=40% | U2=95%")
plt.subplot(224, sharex=ax, sharey=ax)
imshow(gcan3, new_fig=False, title="Canny - U1=40% | U2=99%")
plt.tight_layout()
plt.show(block=False)

```

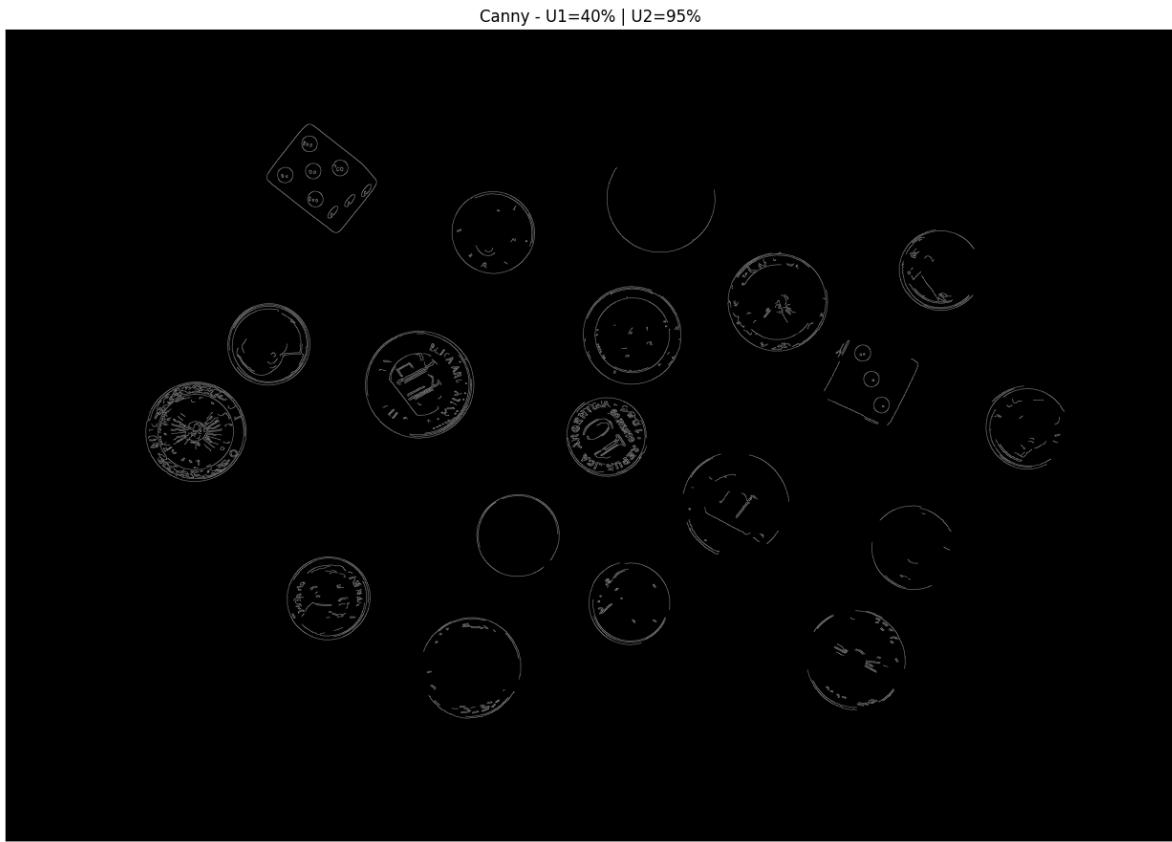


Ahora mostramos la imagen que mejor resultados presentó

```

plt.figure(figsize=(16, 9))
imshow(gcan2, new_fig=False, title="Canny - U1=40% | U2=95%", colorbar=False)
plt.tight_layout()
plt.show(block=False)

```



Análisis del Resultado:

- La imagen resultante muestra los contornos de las monedas suficientemente bien y los dados claramente definidos.
- Los umbrales altos ($0.99 * 255$) ayudaron a eliminar gran parte de la textura de la madera de la mesa, dejando principalmente los bordes fuertes de los objetos que sí interesan.

Para determinar el parámetro `param1` de la función `HoughCircles`, realizamos un estudio previo utilizando el detector de bordes Canny de forma aislada (ver figuras `gcan2` y `gcan3`).

Observamos que la combinación de umbrales [102, 242] (correspondientes al 40% y 95% de intensidad) lograba aislar los bordes de las monedas eliminando el ruido de la textura de madera.

Dado que HoughCircles utiliza `param1` como umbral superior y `param1/2` como inferior para su detección de bordes interna, configuramos `param1=200`. Esto establece implícitamente un umbral inferior de 100, replicando casi con exactitud los valores óptimos encontrados en nuestra experimentación manual (102), pero con un umbral superior ligeramente más relajado (200 vs 242) para asegurar la continuidad de los bordes circulares.

3. Detección y Clasificación de Monedas

Con los bordes definidos, utilizamos la **Transformada de Hough para Círculos**. Realizamos una primera pasada configurada específicamente para detectar los radios grandes correspondientes a las monedas.

Implementamos una lógica de clasificación basada en el radio detectado para distinguir entre monedas de 10 centavos, 1 peso y 50 centavos.

```
from collections import defaultdict

# Configuración para detectar MONEDAS
minDist = 350      # Distancia mínima entre centros
param2 = 25         # Umbral del acumulador (mientras más bajo, menos riguroso para detectar)
minRadius = 110     # Radio mínimo esperado
maxRadius = 200     # Radio máximo esperado
circles = cv2.HoughCircles(f.blur,
                           method=cv2.HOUGH_GRADIENT,
                           dp=1,
                           minDist=minDist,
                           param1=200, # Umbral alto de Canny (pasado a la función)
                           param2=param2,
                           minRadius=minRadius,
                           maxRadius=maxRadius)

Nc = circles.shape[1]
# Dibujo los círculos detectados
circles = np.uint16(np.around(circles))
fc = cv2.cvtColor(mon, cv2.COLOR_GRAY2RGB) # Imagen color para dibujar resultados

monedas = defaultdict(int)

def identificar_moneda(radio):
    if radio < 140:
        monedas[10] += 1
    return (0,0,255), "10 centavos" # Rojo
```

```

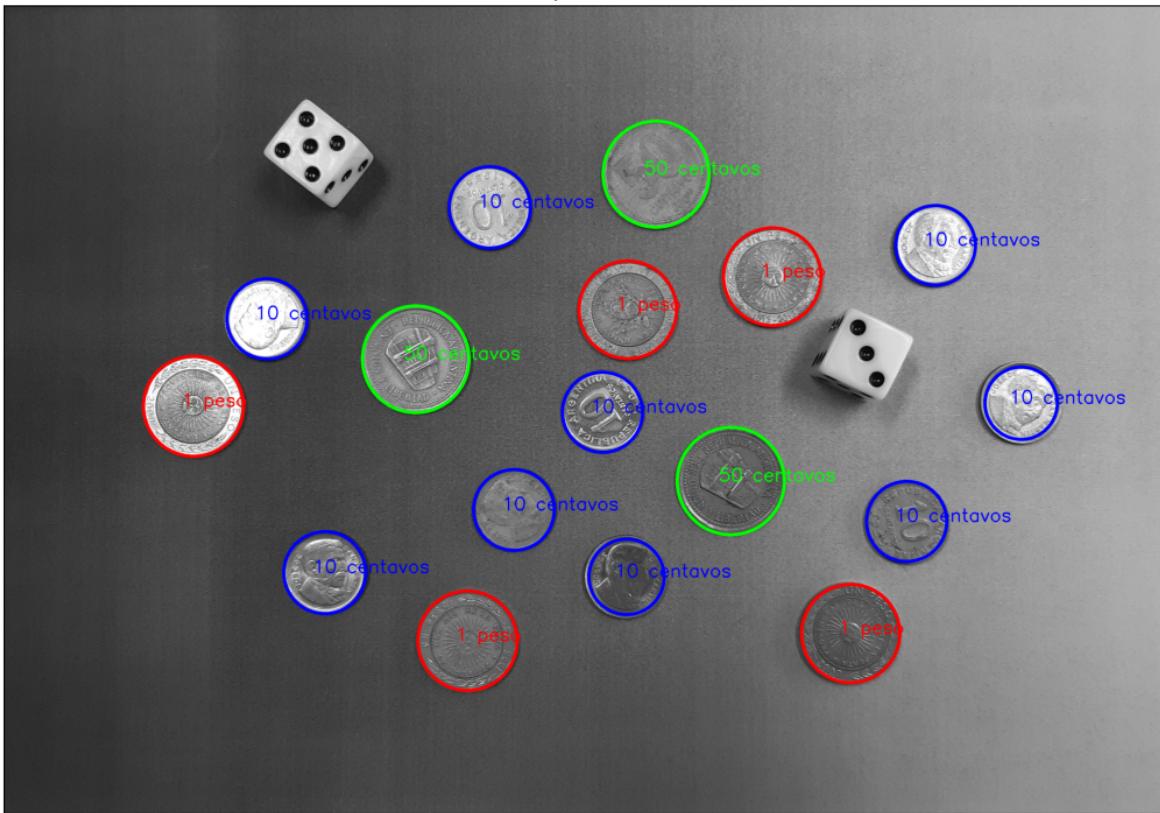
    elif radio < 170:
        monedas[100] += 1
        return (255,0,0), "1 peso"      # Azul
    else:
        monedas[50] += 1
        return (0,255,0), "50 centavos" # Verde

# Dibujado y clasificación
if circles is not None:
    for c in circles[0,:]:
        radio = c[2]
        color, tipo = identificar_moneda(radio)
        # Dibujar círculo y texto
        cv2.circle(fc, (c[0],c[1]), radio, color, 10)
        cv2.putText(fc, tipo, (c[0]-40, c[1]), cv2.FONT_HERSHEY_SIMPLEX, 2, color, 3)

imshow(fc, colorbar=False, title=f"Círculos detectados ({Nc}), minDist={minDist}, param2={pa

```

Círculos detectados (17), minDist=350, param2=25, minRadius=110, maxRadius=200



Análisis:

- Los parámetros `minRadius` y `maxRadius` permitieron filtrar los puntos de los dados (que son círculos pequeños), quedándonos solo con las monedas.

4. Detección y Lectura de Dados

Para los dados, se presenta un doble problema: detectar los puntos y agruparlos para saber a qué dado pertenecen.

1. **Detección:** Realizamos una segunda pasada de `HoughCircles` con radios muy pequeños (25-27 px).
2. **Agrupamiento:** Creamos una clase `Dado` para poder identificar cada dado de la imagen y contar sus valores. Usamos un algoritmo muy simple que itera sobre todos los puntos de dados, de izquierda a derecha, y asume que los puntos que están cerca pertenecen al mismo dado y que todos los dados cubren un rango distinto del eje horizontal, los dados encontrados son en realidad conjuntos de puntos pequeños encontrados. La imagen a

analizar en cuestión cumple con esos requisitos por lo que la detección funciona bien, aunque es un algoritmo muy poco robusto.

```
# Configuración para detectar círculos pequeños de datos
minDist = 40
param2 = 20
minRadius = 25
maxRadius = 27

circles = cv2.HoughCircles(f_blur,
                           method=cv2.HOUGH_GRADIENT,
                           dp=1,
                           minDist=minDist,
                           param2=param2,
                           minRadius=minRadius,
                           maxRadius=maxRadius)

Nc = circles.shape[1]
# Dibujo los círculos detectados
circles = np.uint16(np.around(circles))

class Dado():
    def __init__(self, x, y):
        self.min_x = x
        self.min_y = y
        self.max_x = x
        self.max_y = y
        self.valor = 1

    def actualizar(self, x, y):
        if x < self.min_x:
            self.min_x = x
        if x > self.max_x:
            self.max_x = x
        if y < self.min_y:
            self.min_y = y
        if y > self.max_y:
            self.max_y = y
        self.valor += 1

# Lógica de agrupación
circulos = sorted(circles[0,:], key=lambda c: c[0]) # ordeno por posición x
dados = []
```

```

distancia_anterior = 0

for c in circulos:
    x, y = c[0], c[1]
    distancia = np.hypot(x, y)
    cv2.circle(fc, (c[0],c[1]), c[2], (255,0,255), 2)
    # print(distancia, distancia_anterior)
    if abs(distancia - distancia_anterior) > 200: # si la distancia es suficientemente distante
        d = Dado(x, y)
        dados.append(d)
        distancia_anterior = distancia
    else:
        d.actualizar(x, y)

for d in dados:
    cx = (d.min_x + d.max_x) // 2
    cy = (d.min_y + d.max_y) // 2
    max_radio = max(d.max_x - d.min_x, d.max_y - d.min_y) // 2
    cv2.rectangle(fc, (d.min_x - max_radio, d.min_y - max_radio), (d.max_x + max_radio, d.max_y + max_radio), (0,255,0), 2)
    cv2.putText(fc, f"Valor {d.valor}", (d.min_x - max_radio, d.min_y - max_radio - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255))

```

5. Resultados Finales

Integramos ambas detecciones (monedas y datos) en la imagen final.

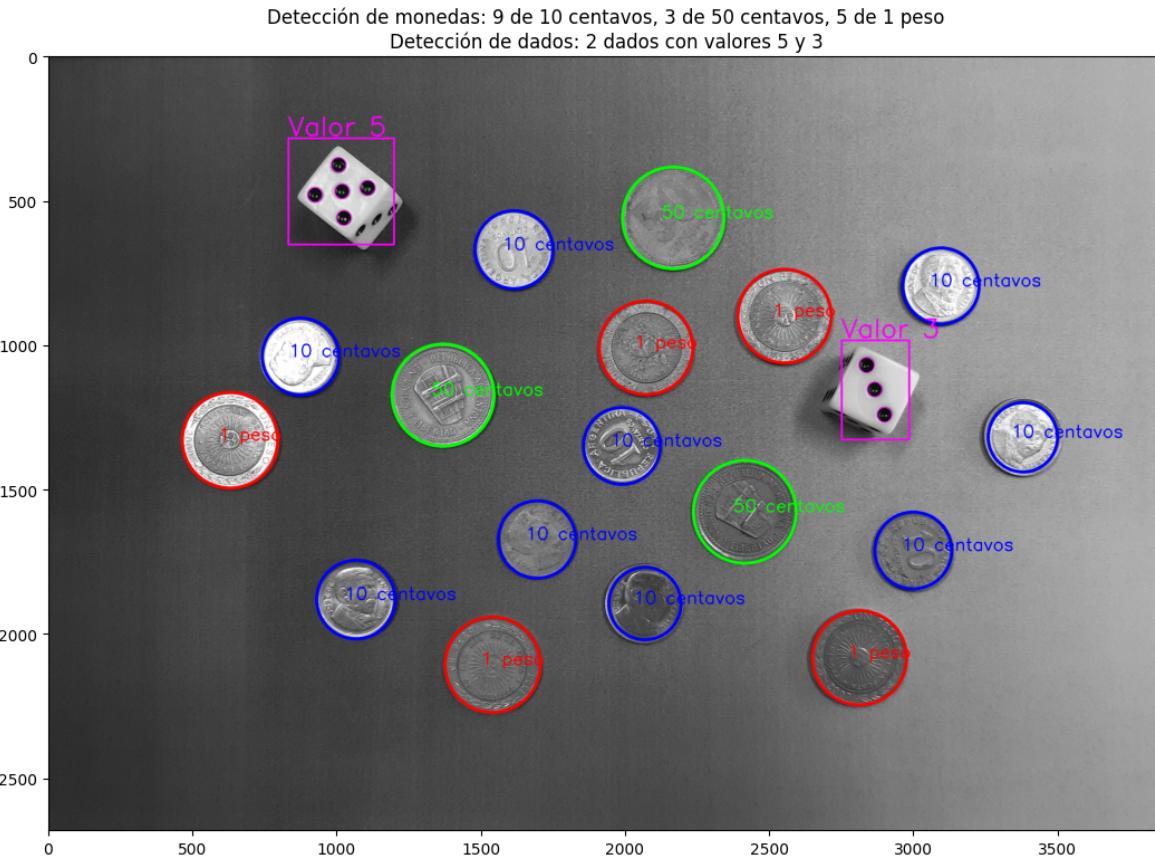
```

fig, ax = plt.subplots(figsize=(16, 9))

texto_titulo = f"Detección de monedas: {monedas[10]} de 10 centavos, {monedas[50]} de 50 centavos y {len(datos)} datos con valores"
texto_titulo += f"\nDetección de datos: {len(dados)} datos con valores {' y '.join(str(d.valor) for d in dados)}"

ax.imshow(fc)
plt.title(texto_titulo)
plt.tight_layout()
plt.show()

```



Conclusión del Problema 1:

El uso de la Transformada de Hough en dos etapas (una para radios grandes y otra para radios pequeños), combinado con un filtrado previo mediante Canny, demostró ser una estrategia efectiva.

- Se logró clasificar correctamente las monedas por su tamaño.
- Se logró agrupar los puntos e identificar los dados utilizando proximidad espacial, permitiendo calcular sus valores automáticamente.

Problema 2: Detección de patentes y segmentación de caracteres.

Este ejercicio propone detectar patentes y la posterior segmentación de sus caracteres. Los objetivos específicos son:

1. Detectar automáticamente la placa patente y segmentarla.

2. Implementar un algoritmo de procesamiento que segmente los caracteres de la placa patente detectada en el punto anterior.

Para dar por resuelto el problema, se cuenta con 12 imágenes con patentes tanto de frente como de la parte trasera de los vehículos. Aunque en general relativamente similares, presentan diferencias cualitativas: - Iluminación - Ángulo - Diferentes colores de los vehículos - Patentes viejas y dañadas, o con fundas de otro color.

Desarrollo del Problema

1. Primer enfrentamiento al problema

La primer idea que se nos ocurrió al intentar resolver este problema fue aplicando una lógica “desde afuera hacia adentro”, es decir, primero encontrar la patente y posteriormente los caracteres de la misma. Sin embargo rápidamente notamos que no era un buen camino para continuar dado que **para una sola patente** se necesitaba de muchas condiciones específicas para que el algoritmo pueda encontrarla. Nos llevaba por un camino en el que iba a ser difícil obtener robustez. Para validar esta idea, utilizamos la imagen img06.png y aplicamos el siguiente flujo de trabajo:

1. **Pre-procesamiento:** Suavizado y umbralizado fijo.
2. **Extracción de contornos:** Búsqueda de todos los objetos cerrados.
3. **Filtrado Geométrico:** Selección de candidatos basada en el **Aspect Ratio** (Relación de Aspecto) de las patentes argentinas/mercosur y su área en píxeles.

El *Aspect Ratio* teórico se calculó tomando las medidas estándar de una patente: 294 mm/129 mm ≈ 2.28.

A continuación se muestra el progreso que obtuvimos

```
img = cv2.imread('img06.png', cv2.IMREAD_GRAYSCALE)
img = cv2.GaussianBlur(img, (3,3), 0)

# umbralizado
thresh = cv2.threshold(img, 87, 255, cv2.THRESH_BINARY) [1]

# contornos
contours = cv2.findContours(thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE) [0]
canvas = np.zeros_like(img)
```

```

cv2.drawContours(canvas, contours, -1, 255, 2)

img_color = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)      # ... Paso a RGB
# aspect ratio de las patentes 294 mm de largo por 129
# es decir 2.28:1
ratio = 294 / 129
min_w = 60
max_w = 110
min_h = 25
max_h = 60

candidatos = []
for i, cnt in enumerate(contours):
    x, y, w, h = cv2.boundingRect(cnt)
    aspect_ratio = float(w) / h
    # Comparamos con el nuevo ratio (2.28)
    # atol=0.7 es la tolerancia (permite que el ratio varíe entre ~1.58 y ~2.98)
    if (np.isclose(aspect_ratio, ratio, atol=0.7) and
        (max_w > w > min_w) and
        (max_h > h > min_h)):
        candidatos.append(cnt)
cv2.drawContours(img_color, contours, i, (0,255,0), 1)
cv2.rectangle(img_color, (x, y), (x+w, y+h), (255,0,255), 1)
imshow(img_color, ticks=False, title="Contornos")

```



A continuación, con el objetivo de descartar contornos que no sean de interés, se agregan nuevas condiciones. En particular un área de 2300. A continuación se grafica el ROI.

```
# filtra tambien por área además de aspect ratio
candidatos_filtrados = []
for cnt in candidatos:
    area = cv2.contourArea(cnt)
    if area > 2300:
        candidatos_filtrados.append(cnt)

canvas = np.zeros_like(img)
cv2.drawContours(canvas, candidatos_filtrados, -1, 255, 1)

# roi de la patente
x, y, w, h = cv2.boundingRect(candidatos_filtrados[0])
crop = img[y:y+h, x:x+w]

# acá se grafica:
# 1- imagen original en escala de grises,
# 2- umbralada,
# 3- contornos detectados,
```

```

# 4- contornos que cumplen la condicion que le damos,
# 5- roi de ese contorno
fig = plt.figure(figsize=(15, 10))
ax1 = fig.add_subplot(2, 3, 1)
ax1.imshow(img, cmap='gray')
ax1.set_title("Imagen Original en escala de grises con blur")
ax1.axis('off')

ax2 = fig.add_subplot(2, 3, 2, sharex=ax1, sharey=ax1)
ax2.imshow(thresh, cmap='gray')
ax2.set_title("Imagen binarizada")
ax2.axis('off')

ax3 = fig.add_subplot(2, 3, 3, sharex=ax1, sharey=ax1)
ax3.imshow(canvas, cmap='gray')
ax3.set_title("Contornos")
ax3.axis('off')

ax4 = fig.add_subplot(2, 3, 4, sharex=ax1, sharey=ax1)
ax4.imshow(canvas, cmap='gray')
ax4.set_title("Contornos de candidatos")
ax4.axis('off')

ax5 = fig.add_subplot(2, 3, 5)
ax5.imshow(crop, cmap='gray')
ax5.set_title("Patente")
ax5.axis('off')

plt.tight_layout()
plt.show()

```



Análisis crítico de este enfoque

Si bien logramos aislar la patente en la imagen img06.png, concluimos que este método **no es generalizable** por las siguientes razones:

- 1. Dependencia de la Iluminación:** El umbral de binarización (87) fue elegido manualmente. Si la imagen fuera más oscura o tuviera sombras (como ocurre en otras imágenes del set), la patente no se formaría como un bloque sólido.
- 2. Dependencia de la Escala:** Los filtros de `min_w`, `max_w` y `area > 2300` asumen que el auto está a una distancia fija de la cámara. Si el auto está más lejos o más cerca, el algoritmo falla.
- 3. Deformación:** El filtrado por *Aspect Ratio* es seguramente más robusto pero continúa siendo sensible a la rotación y perspectiva del vehículo.

Debido a esta falta de robustez, decidimos descartar el enfoque de “buscar el rectángulo de la patente” y cambiar a una estrategia “de adentro hacia afuera”, buscando patrones de caracteres, que suelen ser más distintivos y consistentes por la estructura misma de las patentes.

2. Segundo enfoque: Estrategia “Desde adentro hacia afuera” (Detección de Caracteres)

Dada la falta de robustez del método geométrico global, cambiamos la estrategia. En lugar de buscar el contenedor (la patente), buscamos el contenido (los caracteres). La hipótesis es que los caracteres de una patente presentan patrones de repetición, alineación y consistencia

morfológica mucho más fuertes que el marco exterior, además de facilitar la diferenciación con binarización por el contraste que presentan los caracteres blancos con el fondo negro.

2.1 Análisis de Estadísticas y Calibración

Para no establecer reglas arbitrarias, realizamos primero un análisis cuantitativo sobre una imagen de control (`img06.png`). Se extrajeron las componentes conectadas y se identificaron manualmente los IDs correspondientes a los caracteres de la patente para estudiar sus propiedades.



Figure 1: Componentes conectadas en la patente



Figure 2: Identificación de IDs de las componentes conectadas en la patente

Análisis de las componentes: Observando los IDs del primer grupo (64, 63, 59) y del segundo grupo (58, 56, 57), obtuvimos las siguientes métricas:

```

ID 64: Ancho=12, Alto=19, AR=1.58, Area=59
    -> Distancia al siguiente (ID 63): 1 px
    -> Ratio Espacio/Ancho: 0.08 (El hueco es un 8.3% del ancho de la letra)
ID 63: Ancho=12, Alto=20, AR=1.67, Area=81
    -> Distancia al siguiente (ID 59): 1 px
    -> Ratio Espacio/Ancho: 0.08 (El hueco es un 8.3% del ancho de la letra)
ID 59: Ancho=13, Alto=21, AR=1.62, Area=110
    -> Distancia al siguiente (ID 58): 13 px
    -> Ratio Espacio/Ancho: 1.00 (El hueco es un 100.0% del ancho de la letra)
ID 58: Ancho=12, Alto=20, AR=1.67, Area=112
    -> Distancia al siguiente (ID 56): 1 px
    -> Ratio Espacio/Ancho: 0.08 (El hueco es un 8.3% del ancho de la letra)
ID 56: Ancho=11, Alto=20, AR=1.82, Area=105
    -> Distancia al siguiente (ID 57): 2 px
    -> Ratio Espacio/Ancho: 0.18 (El hueco es un 18.2% del ancho de la letra)
ID 57: Ancho=12, Alto=20, AR=1.67, Area=86

```

En resumen: * **Dimensiones:** Anchos entre 11-13 px y Altos entre 19-21 px. * **Relación de Aspecto (AR):** Varió entre **1.58** y **1.82**. Los caracteres son rectángulos verticales consistentes. * **Área:** El área de los caracteres osciló entre 59 px y 112 px. * **Espaciado:** * La distancia entre caracteres del mismo grupo es mínima (1-2 px). * La distancia de la separación central (hueco entre letras y números) es de aprox. 13 px (lo cual equivale a un 100% del ancho de un carácter).

2.2 Definición de Reglas de Filtrado

Basados en las stats extraídas de los componentes de los caracteres de la patente (ID 64 a ID 57) conformamos las siguientes reglas para el filtrado:

1. **Restricciones Geométricas (Filtro de Forma)** Los caracteres válidos presentan una Relación de Aspecto (AR) consistente entre **1.58 y 1.82**. Por tal motivo, se decidió establecer el rango de aceptación en $1.3 \leq AR \leq 2.8$. El límite inferior (1.3) descarta formas cuadradas o apaisadas. El límite superior (2.8) descarta ruido vertical excesivamente delgado. Se busca también que el rango sea un tanto permisivo ante irregularidades en las imágenes.
2. **Umbral de Ruido (Filtro de Área)** Las áreas de los caracteres varían entre **59 px** (mínimo) y **112 px** (máximo). Por esto se fijó el umbral mínimo en **Area > 30**. Este margen de seguridad por debajo del valor mínimo observado (59) permite eliminar artefactos pequeños resultantes de la binarización sin descartar caracteres válidos de menor tamaño.
3. **Criterio de Agrupamiento (Distancia Horizontal)**
 - Distancia entre caracteres adyacentes: 1-2 px (Ratio Espacio/Ancho $\approx 0.08 - 0.18$).
 - Distancia en la separación central (ID 59 a 58): 13 px (Ratio Espacio/Ancho = **1.00**). Implementar una búsqueda dinámica de vecinos con la fórmula: `umbral_distancia = ancho_previo * 2.5`. El umbral debe superar el Ratio de 1.00 observado en la separación central. Un multiplicador de **2.5** garantiza que el algoritmo cruce el espacio divisorio (que equivale al 100% del ancho de la letra) y mantenga unidos todos los caracteres en un único bloque lógico, evitando la segmentación de la patente.

2.3 Primera Prueba: Restricciones Estrictas

Intentamos una detección inicial aplicando umbralado de Otsu y exigiendo condiciones ideales:
* Encontrar **exactamente 6 caracteres**. * Tolerancia muy baja en variaciones de altura (20%) y alineación vertical (15%).



Figure 3: Resultados de la primera prueba con restricciones estrictas

Resultado: El enfoque estricto fue insuficiente. Solo se detectaron correctamente 3 de las 12 patentes. La mayoría falló (“NO DETECTADO”) debido a pequeñas variaciones en la segmentación que rompían la regla de “exactamente 6”.

2.4 Segunda Prueba: Flexibilización de Parámetros

Para mejorar la tasa de aciertos, relajamos las restricciones: * Se permite detectar entre **5 y 7 caracteres** (para tolerar ruidos extra o alguna fusión). * Se amplió la tolerancia de altura y alineación al 30-35%. * Se expandió el rango de Aspect Ratio permitido a **1.0 - 3.0**.



Figure 4: Resultados con restricciones más blandas



Figure 5: Detalle de los caracteres detectados (bounding boxes verdes)

Resultado: Se obtuvo una mejora notable detectando 9 de 12 patentes.

A continuación se muestra de cerca lo que sucede con las componentes de patentes problemáticas

img03.png



Figure 6: Detalle de la imagen 03

img09.png



Figure 7: Detalle de la imagen 09

img11.png



Figure 8: Detalle de la imagen 11

Como mencionamos, la mejora es sustancial. Sin embargo, al observar los detalles (imágenes superiores), notamos un problema recurrente: **la fusión de caracteres**. * En la img03 y img09, algunos caracteres se fusionan entre sí. * En la img11, no se detecta nada, y es posible que esto suceda porque los caracteres se fusionan con el marco blanco de la patente, creando una única “mancha” grande que es descartada por los filtros de tamaño.

2.5 Erosión

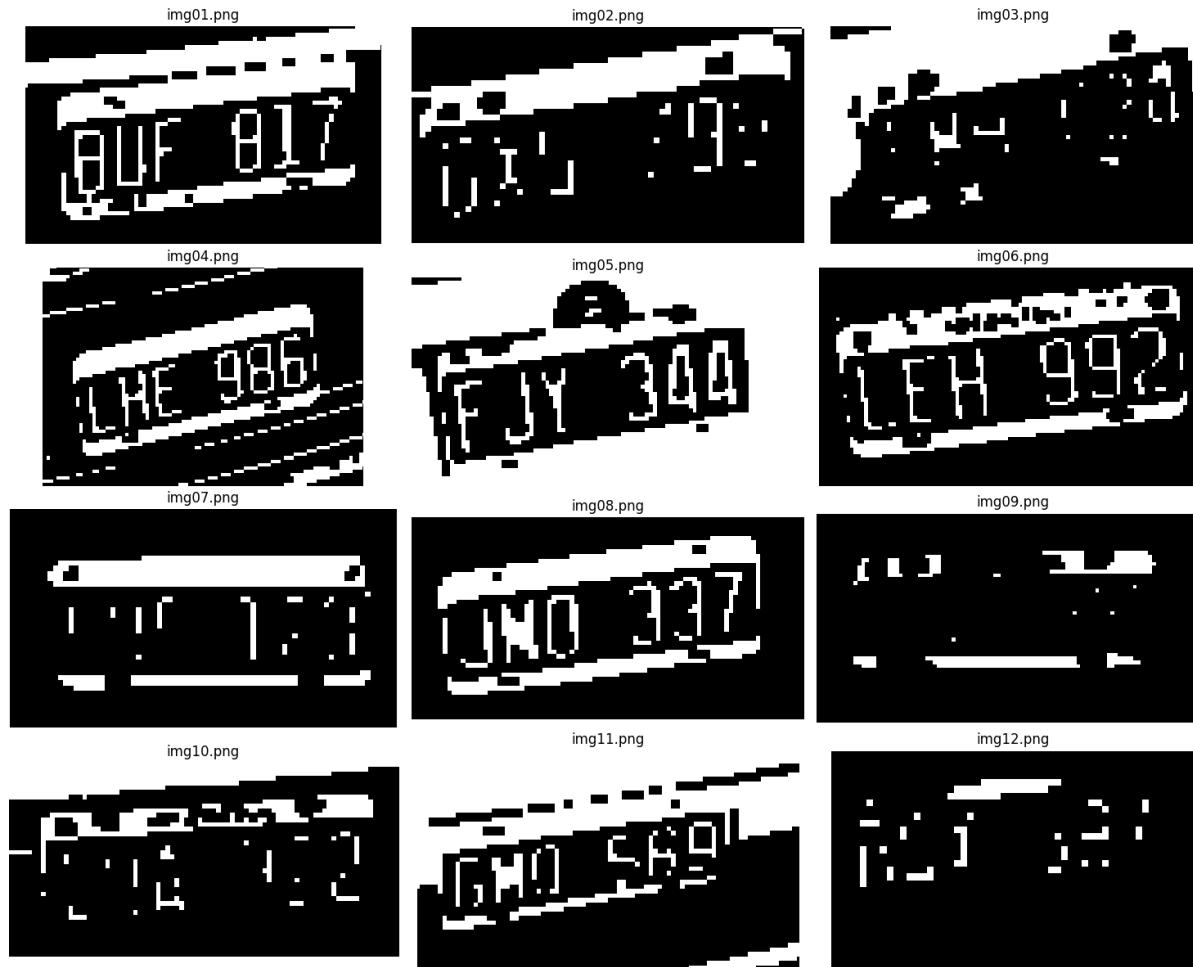
Intentamos solucionar la fusión aplicando operaciones morfológicas de **erosión** para separar los caracteres entre sí y del marco.



Figure 9: Componentes con erosión

Se rompen algunos caracteres. En la imagen 11 ahora empiezan a aparecer, antes no se encontraba ninguno. Vamos a observar a las patentes binarizadas para tener más información

de qué está haciendo la erosión en las mismas:



Continuando con la descripción de la imagen 11 vemos que la erosión rompe algunos caracteres. Sin embargo, en varias otras patentes como la 02, 03, 09, 10, 12 entre otras, los caracteres se rompen totalmente. Ahora comparamos la binarización pero sin erosión, únicamente con umbral otsu:



Figure 10: Comparativa de máscaras binarias: Sin Erosión

Análisis: La erosión resultó generar otros problemas. * **Pros:** En la img11, la erosión logró separar las letras del marco, permitiendo su detección parcial. * **Contras:** En patentes que ya eran delgadas o tenían poco contraste (como img09, img10, img12), la erosión “rompió” los caracteres, fragmentándolos en pedazos pequeños que fueron eliminados por el filtro de área mínima.

Conclusión parcial: Un único pre-procesamiento (Otsu con o sin erosión) no es suficiente para todas las condiciones de luz y desgaste de las patentes.

3. Solución Final Propuesta: Binarización Adaptativa

Dado que el problema central es encontrar el punto exacto de binarización donde los caracteres no se fusionen (umbral muy bajo) ni se rompan (umbral muy alto), decidimos implementar

una **búsqueda de umbral iterativa**.

Algoritmo Final: El sistema no se queda con un solo umbral (Otsu). En su lugar: 1. Itera por una lista de distintos niveles de umbral de binarización. 2. Para cada umbral, extrae contornos y aplica los filtros de **Forma, Área y Agrupamiento** definidos en la sección 2.2. 3. Si encuentra un grupo de candidatos que cumple con las condiciones de “ser una patente” (5 a 7 caracteres alineados), detiene la búsqueda y retorna el resultado.

Esto permite que una patente oscura se detecte con un umbral y una patente quemada por el sol se detecte con otro.



Figure 11: Resultados finales con umbral adaptativo sin reglas estrictas

Con los umbrales adaptados se obtuvo una mejora sustancial. Dado que los resultados fueron positivos, se volverá a aplicar las restricciones estrictas, pero permitiendo detectar con umbrales más permisivos en caso de que las estrictas no funcionen.

A continuación se muestran los resultados:



Figure 12: Resultados finales con umbral adaptativo con reglas estrictas

Por último, se procede a la segmentación y muestra de la patentes junto con sus caracteres encontrados.



Figure 13: Patentes y caracteres segmentados

Conclusión del Problema 2: La implementación del esquema adaptativo permitió resolver satisfactoriamente la detección en la gran mayoría de los casos, logrando robustez ante: * **Fusión con el marco:** Ajustando el umbral automáticamente hasta que se separan. * **Fragmentación:** Evitando la erosión agresiva en patentes delgadas. Resultando en una detección de 70 caracteres sobre 72. El algoritmo final demuestra que combinar reglas geométricas fuertes (conocimiento del dominio) con flexibilidad en el pre-procesamiento (búsqueda de umbrales) es superior a intentar fijar condiciones ideales de entrada.