

UNIVERSIDAD NACIONAL DE ROSARIO

FACULTAD DE CS. EXACTAS, INGENIERÍA Y AGRIMENSURA

Procesamiento de Imágenes Digitales

Trabajo Práctico N°2

*López Ceratto, Julieta : L-3311/1,
Slepoy, David : S-5782/7*

Noviembre, 2024



Figure 1: Logo Facultad Cs. Exactas, Ingeniería y Agrimensura

Introducción

En este informe se explicará de manera detallada los procedimientos realizados y resultados obtenidos para los ejercicios planteados por la cátedra. Se abordarán tanto el planteo de la solución como el código realizado y la lógica detrás del mismo.

Contents

1 Ejercicio 1: Contador de Monedas y Dados	3
1.1 Planteo del problema	3
1.2 Abordaje	3
1.3 Punto 1 y 2: Homogeneizar la influencia de luz en la imagen y umbralizar.	3
1.4 Punto 3: detectar formas y clasificar entre dado y moneda	5
1.5 Punto 4: Contar monto de las monedas	8
1.6 Punto 5: Contar puntaje de los dados.	8
1.7 Punto 6: Realizar una función usuario.	9
1.8 Instrucciones de uso.	10
2 Problema 2: Detección y Recorte de Patentes.	11
2.1 Planteo del problema	11
2.2 Abordaje	11
2.3 Estructura	12
2.4 Punto 1: Detectar Rectángulos con Aspecto Similar a Patente	12
2.5 Punto 2: Diferenciar los Rectángulos que son Patentes de Aquellos que no.	15
2.5.1 Encontrar caracteres / componentes similares.	16
2.5.2 Filtrar por componentes aquellos rectángulos que son patentes de aquellos que no.	18
2.5.3 Instrucciones de Uso	19
2.5.4 Resultados Obtenidos.	21
2.6 Funciones utilizadas.	26
2.6.1 Ejercicio 1	26
2.6.2 Ejercicio 2.	27
2.6.3 Requerimientos	28

Chapter 1

Ejercicio 1: Contador de Monedas y Dados

1.1 Planteo del problema

Se plantea una imagen con dados y monedas. De dicha imagen se quiere obtener, por un lado las figuras mencionadas diferenciadas y, por otro, sus valores: es decir, si es dado, el puntaje y, si es moneda, su valor.

1.2 Abordaje

Para este problema se abordaron los siguientes pasos:

1. Homogeneizar la influencia de luz en la imagen.
2. Umbralizar.
3. Detectar formas y clasificar entre dado y moneda.
4. Contar monto de monedas.
5. Contar puntaje de dados.
6. Realizar una función usuario.

1.3 Punto 1 y 2: Homogeneizar la influencia de luz en la imagen y umbralizar.

Un paso importante a la hora de encarar el problema es lograr que la influencia de luz en la imagen no afecte a la posterior detección de los objetos. Para conseguirlo, se plantea de forma experimentar la variación de las distintas componentes: HSVL (Hue, Saturation, Value y Light). Lo que se buscaba es una especie de "mapa de calor" donde las figuras deseadas quedasen más resaltadas que el fondo o el ruido (entendido este último como aquello que no es ni dado ni moneda). Se aplica una función que permite variar dichas componentes de la imagen mediante una barra deslizante, mostrando la imagen HSV y también su umbralizada; a esta última también se le permite variar el threshold.

Mediante este proceso se llega a los valores óptimos de:

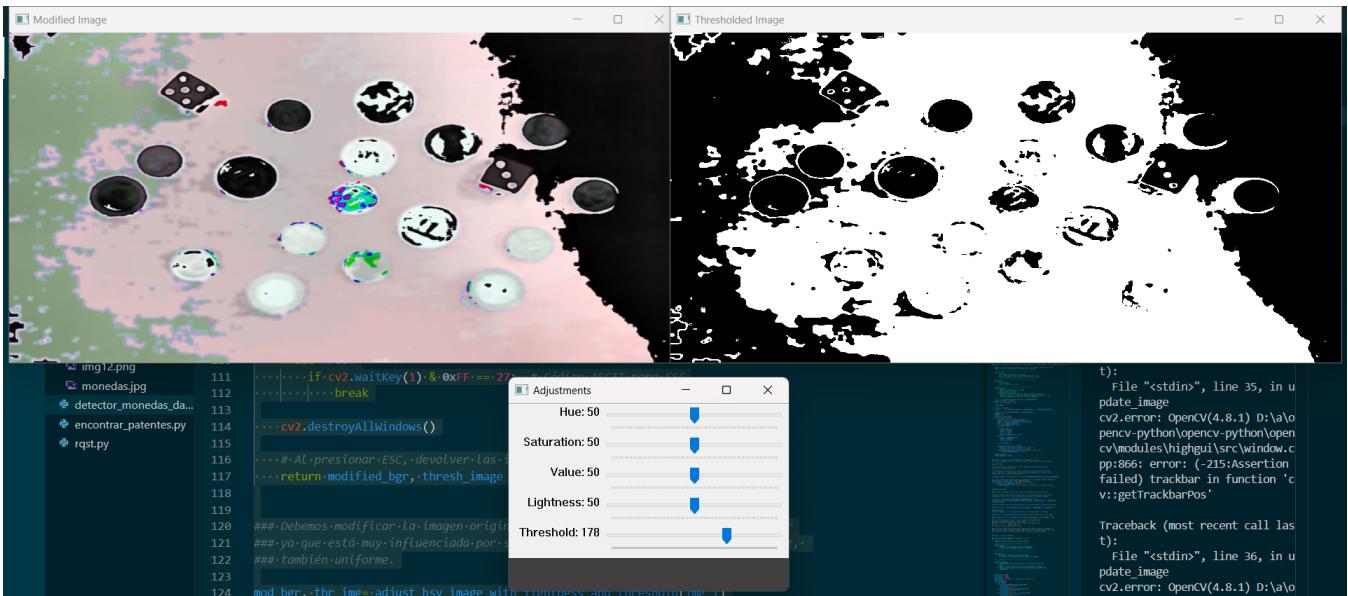


Figure 1.1: Editor HSVL y Treshold con valores por defecto

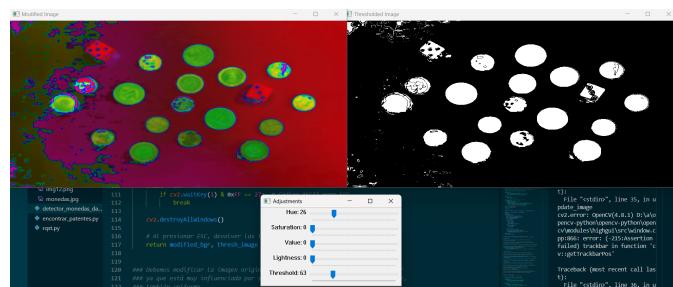


Figure 1.2: Editor HSVL y Treshold con valores óptimos,

- H = 26.
- Sat = 0.
- Val = 0.
- L = 0.
- Treshold = 63.

Obteniendo así las siguientes imágenes:

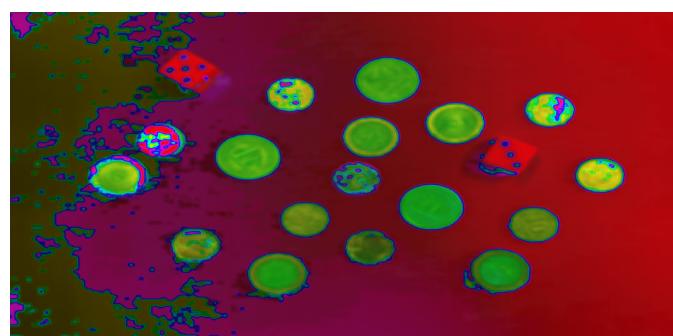


Figure 1.3: Imagen HSVL con valores óptimos.

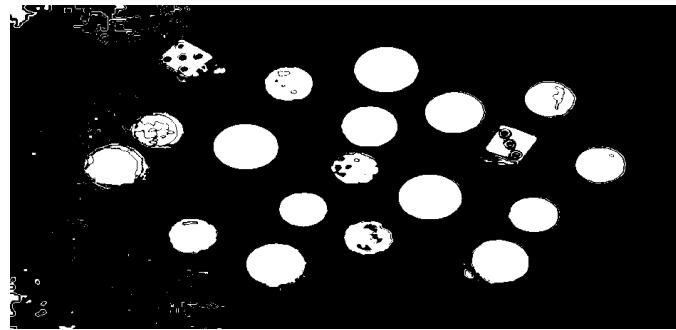


Figure 1.4: Imagen umbralada con Treshold óptimo.

1.4 Punto 3: detectar formas y clasificar entre dado y moneda

Una vez umbralada la imagen se deben detectar las formas de la misma. Sin embargo, para ello antes debemos aplicar algunas transformaciones y filtrados:

Primeramente, se aplica un filtro por área a aquellas componentes conectadas dentro de la imagen cuya área sea menor a 1300 píxeles. De allí se obtiene la siguiente imagen binaria:

```
● ● ●

num_labels, labels, stats, centroids =
cv2.connectedComponentsWithStats(thr_img, connectivity=8)

# Crea una máscara para conservar solo los componentes con área
# mayor o igual a 1300 píxeles
filtered_img = np.zeros_like(thr_img, dtype=np.uint8)
for i in range(1, num_labels): # Omite el fondo (etiqueta 0)
    if stats[i, cv2.CC_STAT_AREA] >= 1300:
        filtered_img[labels == i] = 255
    }
```

Figure 1.5: Imagen de código que filtra por área

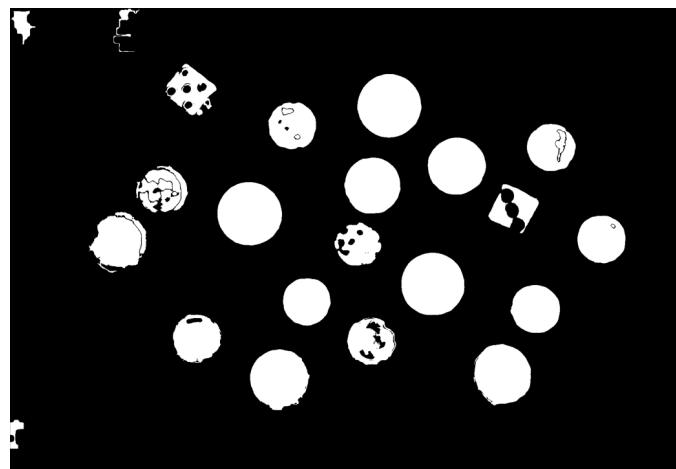


Figure 1.6: Imagen binaria con filtro por área aplicado.

Luego, aplicamos clausura para lograr así círculos y cuadrados cerrados. Para esto, aplicamos un kernel circular de (5,5) y realizamos 9 iteraciones de clausura. La imagen obtenida es la siguiente:

```
● ● ●  
s: np.array = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,  
(5,5))  
close_img: np.array =cv2.morphologyEx(filtered_img.copy(),  
cv2.MORPH_CLOSE, s, iterations=9)
```

Figure 1.7: Código que aplica clausura.

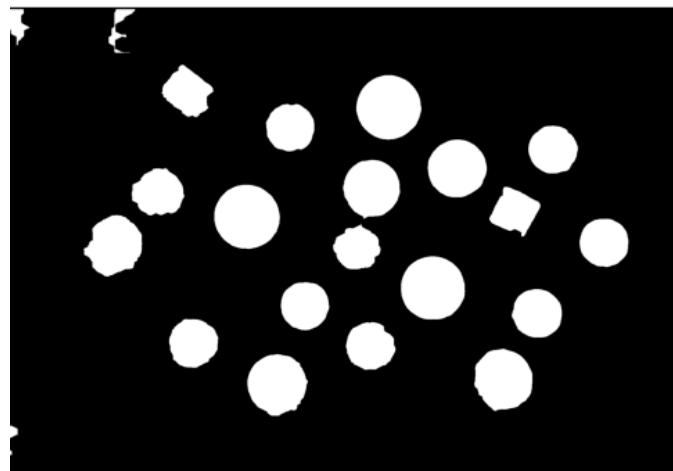


Figure 1.8: Imagen binaria con clausura aplicada.

Como tercer paso, aplicamos apertura para limpiar ciertas irregularidades producto de la ejecución morfológica anterior. Utilizamos el mismo kernel y la misma cantidad de iteraciones, obteniendo finalmente una imagen como esta:

```
● ● ●  
open_img: np.array = cv2.morphologyEx(close_img.copy(),  
cv2.MORPH_OPEN, s, iterations=9)
```

Figure 1.9: Código que aplica apertura.

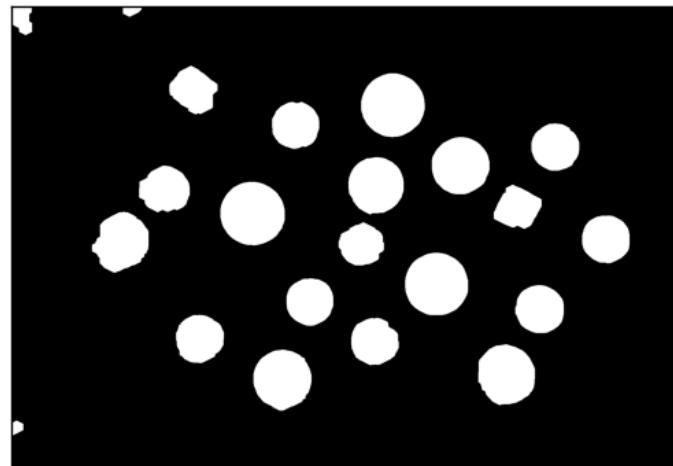


Figure 1.10: Imagen binaria con apertura aplicada.

Una vez obtenida la imagen anterior, donde se tiene figuras cerradas y bien diferenciadas, podemos proceder a detectar los contornos de los objetos. Para ello realizamos una función que realiza lo siguiente:

1. Obtiene la binarización de la imagen.
2. Detecta bordes con Canny.
3. Encuentra los contornos.
4. Mediante el factor de forma del círculo clasifica los contornos, dibuja en distintos colores cada figura diferenciada junto con su área y arma las respectivas listas de círculos y cuadrados.

De esta forma, dividimos los objetos de la siguiente manera: Sin embargo, aún tenemos el problema

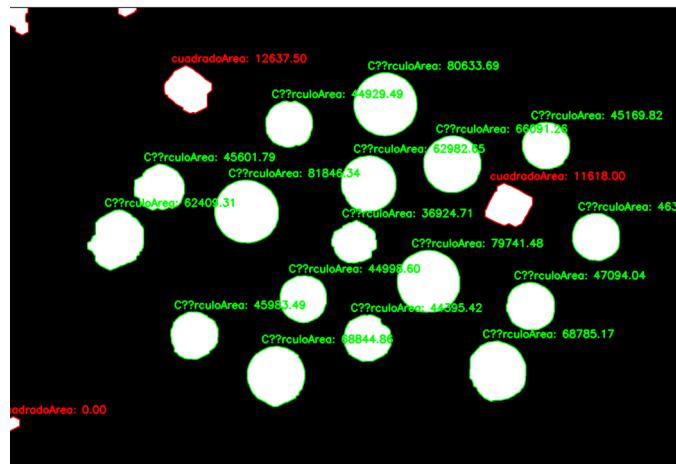


Figure 1.11: Imagen con cuadrados y círculos diferenciados junto al tamaño de su área

de que detecta como cuadrados objetos que en realidad son ruido (es decir, no son ni cuadrados ni círculos), esto se soluciona en el siguiente paso.

1.5 Punto 4: Contar monto de las monedas

Para este punto, primero utilizamos la imagen anterior para observar las áreas de las monedas y qué rango de área le corresponde a cada moneda. De allí podemos ver que:

- Las de 1 peso tienen áreas desde 62000 a 69000 aprox, siendo su punto medio 65500.
- Las de 10 centavos tienen áreas desde 36000 a 47100 aprox, siendo su punto medio 41500.
- Las de 50 centavos, van desde 79000 a 82000, siendo su punto medio 80500.

Por lo tanto, para clasificar las monedas vemos si se encuentran en dichos rangos de área. Realizamos una función que hace lo siguiente:

1. A cada moneda se la clasifica según su área en 10 centavos, 50 centavos y 1 peso.
2. Según su clasificación se suma en 1 al diccionario la moneda.
3. Según su clasificación se suma el monto en dinero que dicha moneda representa.

De la ejecución de esa función obtenemos:



```
{'1 peso': 5, '50 cent': 3, '10 cent': 9} 7.4
```

Figure 1.12: Imagen de salida del programa que indice la moneda y su cantidad, y el total en pesos.

Es decir, un diccionario que indica la cantidad de monedas para cada clase y luego el monto total en pesos obtenido de la suma de todas las monedas. Se puede ver que se clasifican correctamente las monedas siendo para la clase '1 peso' = 5 monedas, '50 centavos' = 3 y '10 centavos' = 9 monedas. Asimismo, el total es 7.40 pesos.

1.6 Punto 5: Contar puntaje de los dados.

En este apartado se realiza el conteo de los puntos de cada dado y el puntaje total. Para esto, creamos una función que hace lo siguiente:

1. Se filtra por área los dados mal clasificados en la etapa anterior.
2. Se recorta la imagen binaria del paso 1 mediante el contorno del dado (ya que allí el dado presenta los círculos internos).
3. Para cada dado se busca sus círculos (es decir sus puntos) utilizando HoughCircles.
4. Se cuenta la cantidad de puntos y se los suma en el diccionario al dado i.
5. Se suma al total de puntos cada punto detectado en dicho dado.

De esta forma detecta la cantidad de círculos internos dentro del dado:

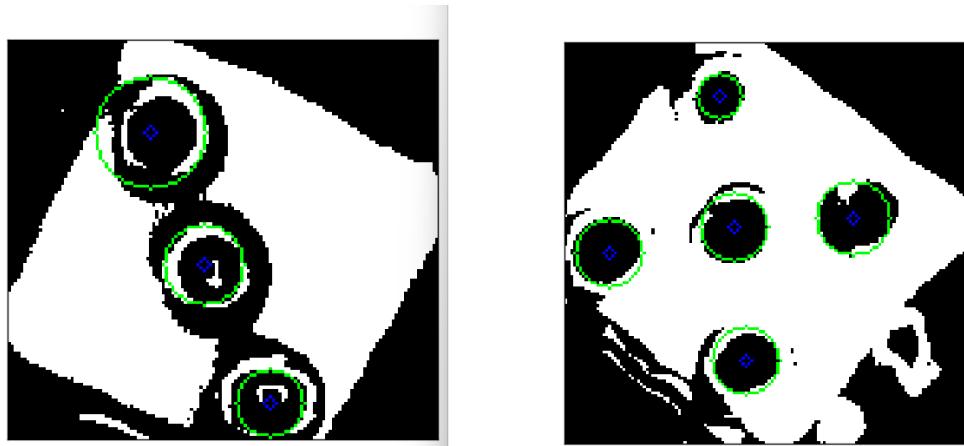


Figure 1.13: Dados con puntos detectados.

La ejecución de la función devuelve:

```
● ● ●  
({'dado 1': 3, 'dado 2': 5}, 8)
```

Figure 1.14: Salida de la ejecución de la función para contar puntaje. Devuelve puntos por dado y puntaje total.

Es decir, devuelve los puntos que hay en cada dado y luego el puntaje total.

1.7 Punto 6: Realizar una función usuario.

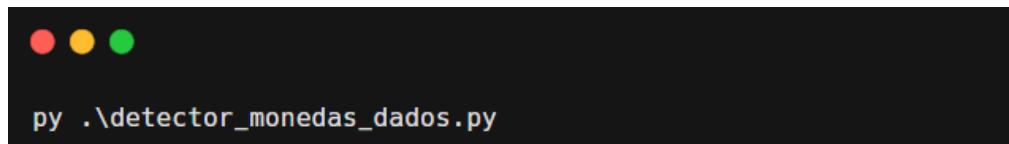
Como tarea final, se realiza una función usuario de manera tal que ejecuta las tareas anteriores de la imagen que este le ingresa y devuelve de manera más amigable los resultados. Al ejecutarla entonces devuelve:

```
● ● ●  
  
ingrese la dirección donde se encuentra guardada la  
imagedata\monedas.jpg  
Las monedas encontradas fueron  
Tipo de moneda: 1 peso, total encontradas: 5  
Tipo de moneda: 50 cent, total encontradas: 3  
Tipo de moneda: 10 cent, total encontradas: 9  
El monto total es de $7.4  
Los puntos por dado fueron  
dado 1, total puntos: 3  
dado 2, total puntos: 5  
El puntaje total es 8
```

Figure 1.15: Salida ejecución programa

1.8 Instrucciones de uso.

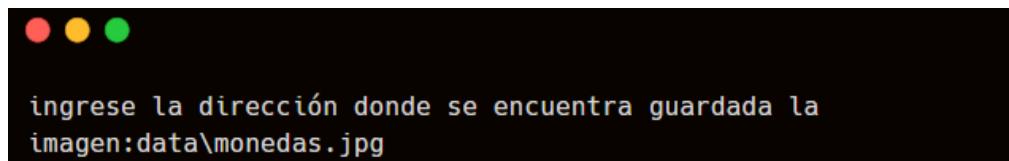
Ejecutar en consola el programa con:



```
py .\detector_monedas_dados.py
```

Figure 1.16: Enter Caption

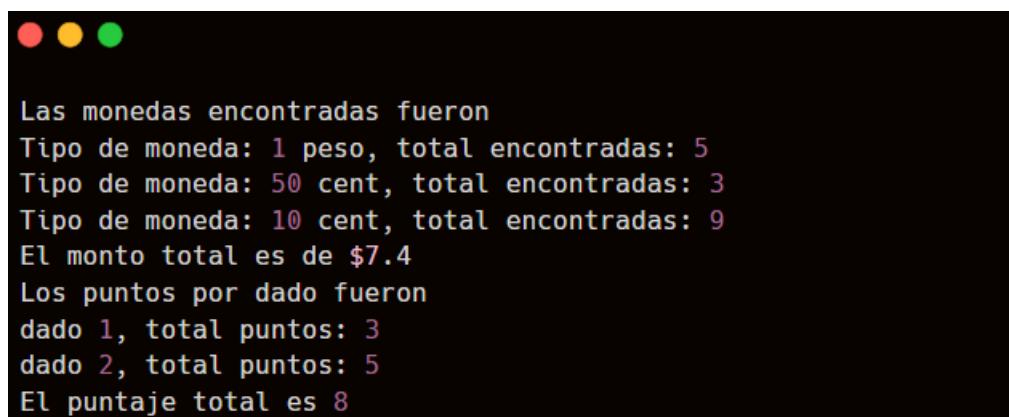
Luego, el programa le solicitará que indique la dirección (path) de la imagen a analizar; debe introducirla:



```
ingrese la dirección donde se encuentra guardada la
imagen:data\monedas.jpg
```

Figure 1.17: Solicitud programa

Finalmente, le devolverá lo encontrado:



```
Las monedas encontradas fueron
Tipo de moneda: 1 peso, total encontradas: 5
Tipo de moneda: 50 cent, total encontradas: 3
Tipo de moneda: 10 cent, total encontradas: 9
El monto total es de $7.4
Los puntos por dado fueron
dado 1, total puntos: 3
dado 2, total puntos: 5
El puntaje total es 8
```

Figure 1.18: Salida programa

Chapter 2

Problema 2: Detección y Recorte de Patentes.

2.1 Planteo del problema

Se plantean 12 imágenes de distintos autos estacionados de los cuales se desea recortar de manera automática la patente y los caracteres en cada una de ellas.

2.2 Abordaje

Para resolver el problema, se plantean los siguientes pasos:

1. Detectar rectángulos con aspecto similar a patente.
2. Diferenciar los rectángulos que son patentes de aquellos que no.
3. Visualizar patente con los caracteres recortados.

A lo largo del informe vamos a trabajar con la siguiente imagen de ejemplo:



Figure 2.1: Imagen de auto original

2.3 Estructura

La solución consta de 4 funciones:

1. user(): se comunica con el usuario.
2. detectar_patentes(): aplica transformaciones morfológicas, llama a la función encontrar_rectangulos() y luego a la función encontrar_patente(). Según la devolución de esta ultima función imprime o no "no se encontró patente".
3. encontrar_rectangulos(): encuentra rectángulos dentro de la imagen que pueden ser patentes.
4. encontrar_patente(): encuentra las componentes conectadas dentro del rectángulo y, según si encuentra 6 componentes similares imprime una imagen al usuario y retorna True o bien, solo retorna False.

2.4 Punto 1: Detectar Rectángulos con Aspecto Similar a Patente

Para este punto, se comienza con la función detectar_patentes la cual: primero se aplica Canny y transformaciones morfológicas a la imagen, entre otros. El orden es el siguiente: Primero, se pasa a escala de grises la imagen recibida:



Figure 2.2: Imagen de auto en escala de grises

Como segundo paso, se binariza la imagen en escala de grises:



Figure 2.3: Imagen binarizada de auto en escala de grises.

En tercer lugar, se obtiene la imagen Canny de la anterior:

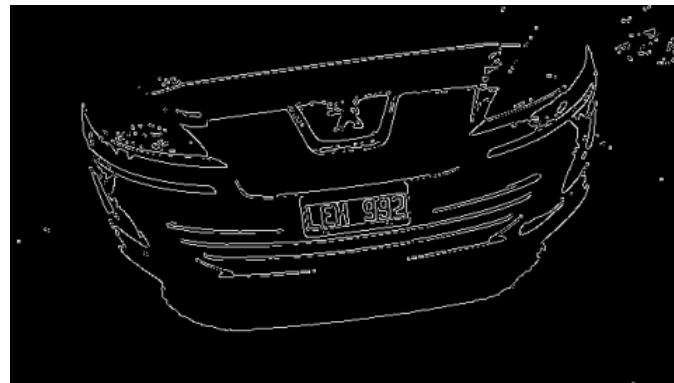


Figure 2.4: Canny de imagen binarizada de auto.

Como cuarto paso, se aplica clausura con un kernel (4,9)para cerrar bien las formas:

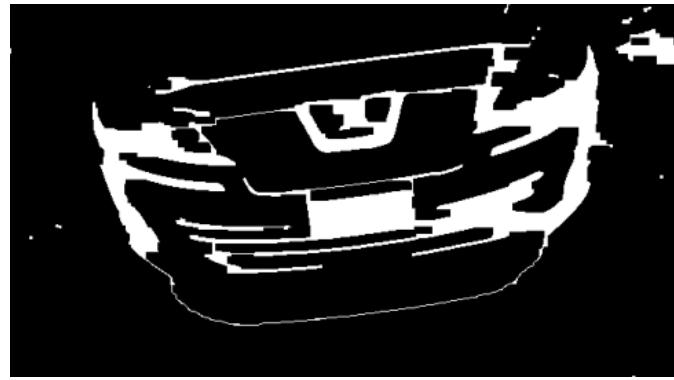


Figure 2.5: Canny de imagen del auto con clausura.

Luego, se obtienen sus componentes conectadas y se filtra la imagen eliminando aquellas que tienen un área menor a 1500px, obteniendo así la siguiente imagen filtrada:



Figure 2.6: Filtro de canny con clausura de imagen del auto.

Finalmente, antes de continuar se le aplica apertura a la imagen con el mismo kernel anterior, obteniendo así una imagen con las figuras más nítidas y diferenciadas:

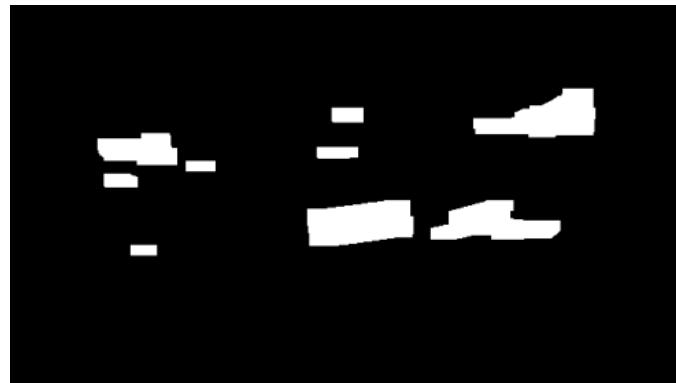


Figure 2.7: Apertura aplicada a la imagen filtrada.

Ahora, dentro de la función detectar_patentes, se llama a la función encontrar_rectangulos que hace lo siguiente:

- Extrae los contornos de la imagen binaria.



Figure 2.8: Contornos encontrados en imagen binaria

O bien, se pueden dibujar los contornos detectados en la imagen binaria sobre la imagen original:



Figure 2.9: Contornos encontrados en imagen binaria sobre imagen original.

- Aproxima los rectángulos en la imagen.

- Filtra los rectángulos por ancho, alto ($42 < W < 103$ and $11 < h < 46$) y cantidad de vértices (4):



Figure 2.10: Rectángulos sobre imagen original, en azul aquellos que son posibles patentes.

- Los recorta de la imagen original y los agrega a la lista ”posibles_patentes”.
- Retorna la lista ”posibles_patentes”.

2.5 Punto 2: Diferenciar los Rectángulos que son Patentes de Aquellos que no.

Esta tarea se subdivide en dos tareas: por un lado, encontrar las componentes conectadas y detectar si hay 6 que sean similares y, por otro lado, filtrar mediante componentes aquellos rectángulos que son patentes de aquellos que no.

Tanto para el primero como para el segundo subproblema, se utiliza la misma función llamada ”encontrar_patente”, la cual toma como parámetro una lista ”posibles_patentes” y devuelve True o False según se haya o no encontrado patente; además, en caso de que se encuentre una patente, dibuja la misma con los 6 caracteres recortados.

Esta función se llama dentro de la función detectar_patentes y se le pasa como parámetro la lista de posibles patentes que devuelve la función encontrar_rectangulos que fue llamada en el paso previo y explicada en el punto anterior.

2.5.1 Encontrar caracteres / componentes similares.

La función mencionada anteriormente, en relación a este punto, y para cada posible patente dentro de la lista recibida, realiza:

- Se la pasas a escala de grises y se la ecualiza localmente para resaltar mejor las letras.



Figure 2.11: Patente en escala de grises ecualizada localmente

- Se realiza blackhat con kernel (40,40) para diferenciar el fondo de las letras de forma más clara.



Figure 2.12: Patente con blackhat aplicado

- Se obtiene imagen binaria.



Figure 2.13: Patente binarizada

- Se filtra aquellas áreas pequeñas o muy grandes de la imagen obteniendo una imagen filtrada.



Figure 2.14: Patente con filtro por área.

- Se detectan componentes conectadas y se dibujan aquellas que sean similares tanto en anchura, altura como en posición a partir de una distancia hacia la mediana de dichos valores.



Figure 2.15: Patente con componentes detectadas.

2.5.2 Filtrar por componentes aquellos rectángulos que son patentes de aquellos que no.

En este punto, para diferenciar aquellos rectángulos que son patentes de aquellos que no, la función simplemente cuenta la cantidad de componentes conectadas similares que encontró dentro del mismo. Si son 6, la función muestra:

- La imagen recibida inicialmente.
- Una imagen con la patente arriba y debajo sus 6 caracteres.

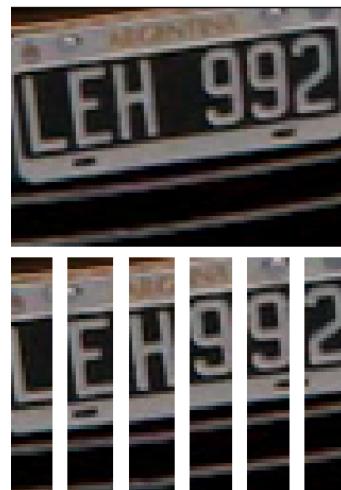


Figure 2.16: Patente con sus caracteres recortados, imagen que ve el usuario al ejecutar el programa.

Y además, retorna True; caso contrario, retorna False.

Finalmente, en la función principal detectar_patentes, una vez se retorna True o False de la función anterior, si el valor returnedo es False, se imprime en pantalla "No se encontró patente", caso contrario, no hace nada.



Figure 2.17: Salida por consola al no encontrar patente.

2.5.3 Instrucciones de Uso

Para utilizar el programa:

1. Llamar al programa con py .\encontrar_patentes.py.

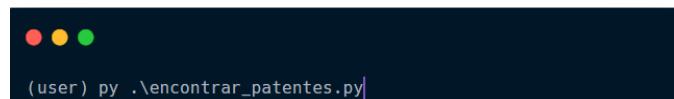


Figure 2.18: Llamada programa encontrar patentes

2. El programa le solicitará la ubicación de la imagen donde buscar la patente. En este caso ingresamos la dirección data\img06.png:

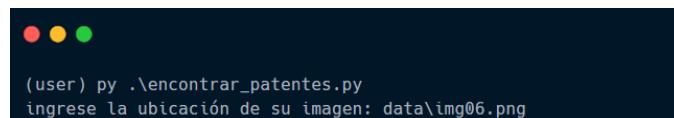


Figure 2.19: Solicitud del programa

El programa le devolverá, en caso de encontrar patente:

- La imagen recibida:



Figure 2.20: Imagen de auto original

- La patente recortada con sus caracteres:



Figure 2.21: Patente con sus caracteres recortados, imagen que ve el usuario al ejecutar el programa.

Y de no encontrarse:



Figure 2.22: Salida por consola al no encontrar patente.

2.5.4 Resultados Obtenidos.

A continuación, se muestran los resultados obtenidos al probar con las 12 imágenes:

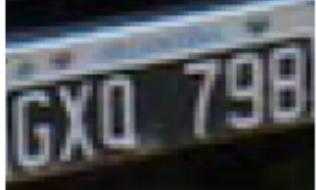
Nº Imagen	Imagen	Salida
1		 
2		 
3		 

Figure 2.23: Tabla resultados obtenidos fotos 1-3.

4		Patente no encontrada
5		
6		

Figure 2.24: Tabla resultados obtenidos fotos 4-6.

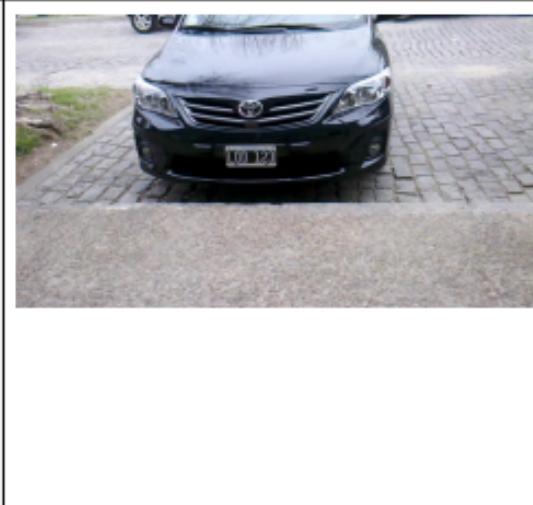
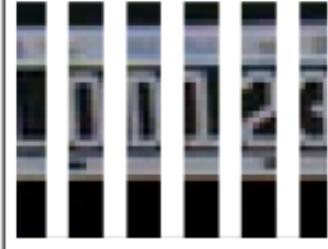
7		 
8		 

Figure 2.25: Tabla resultados obtenidos fotos 7-8.

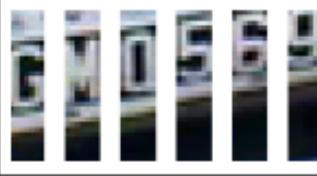
9		 
10		<p>Patente no encontrada</p>
11		 

Figure 2.26: Tabla resultados obtenidos fotos 9-11.



Figure 2.27: Tabla resultados obtenidos foto 12.

2.6 Funciones utilizadas.

A continuación se muestran las funciones utilizadas, sus parámetros y función propiamente dicha.

2.6.1 Ejercicio 1

EJERCICIO 1: ENCONTRAR MONEDAS Y DADOS.		
FUNCIÓN.	EXPLICACIÓN.	PARÁMETROS
adjust_hsv_image_with_lighness_and_threshold(img: np.array, scale: float = 0.5)	Ajusta los valores HSVL de una imagen en tiempo real utilizando OpenCV. También devuelve los valores finales de H, S, V, L y Threshold.	- img: imagen en hsv.
encontrar_cuadrados_y_circulos(img: np.array, min_fp : int = 0.06, max_fp : int = 0.08)-> tuple[list, list]	Esta función encuentra cuadrados y círculos en una imagen, utilizando el factor de forma para diferenciar los segundos de los primeros.	- img: imagen donde detectar las figuras. - min_fp: tolerancia mínima de factor de forma para círculo. - max_fp: tolerancia máxima de factor de forma para círculo.
clasificar_monedas(monedas: list[tuple]) -> dict	Esta función clasifica los tipos de monedas.	- monedas: lista de monedas, cada una representada como una tupla (contorno, área de la moneda).
contar_dados(dados, thr_img) -> tuple[dict, int]	Esta función devuelve para cada dado su puntaje y el puntaje total obtenido por todos los dados.	- dados: lista de dados, cada uno representado como un contorno.
user()	Interactúa con el usuario solicitándole la ubicación del archivo, aplica a la imagen los valores HSVL y TR óptimos y llama a las funciones anteriores (a excepción de la primera, que sirvió para encontrar los valores óptimos).	Ninguno

Figure 2.28: Tabla de funciones utilizadas en el ej. 1.

2.6.2 Ejercicio 2.

EJERCICIO 2: ENCONTRAR PATENTES.		
FUNCIÓN.	EXPLICACIÓN.	PARÁMETROS
encontrar_rectangulos(img_bin: np.array, img:np.array)-> list[np.array]	Recibe una imagen binaria y detecta en ella rectángulos que cumplan con especificaciones de ancho y alto, de forma tal que sean rectángulos horizontales similares a patentes.	- img_bin: imagen binaria donde buscar rectángulos. - img: imagen original de la cual se recortará el rectángulo obtenido.
encontrar_patente(posibles_pat : list[np.array], img_original: np.array)->bool	Recibe una lista de imágenes que son consideradas "posibles patentes" de allí detecta aquellas que lo son y recorta sus caracteres.	- posibles_pat [list[np.array]]: lista con imágenes que son consideradas. - posibles patentes. - img_original: imagen original para fines meramente ilustrativos.
detectar_patentes(img:np.array)-> None	Esta función recibe una imagen, le aplica transformaciones morfológicas y ejecuta las funciones 'encontrar_rectangulos' y 'encontrar_patente'; en caso de que esta ultima retorne False, imprime 'patente no encontrada'	- img: imagen en formato rgb.
user()	se comunica con el usuario solicitandole la dirección de la misma, carga la imagen en bgr y llama a la función detectar_patentes pasándola como argumento.	Ninguno

Figure 2.29: Tabla de funciones utilizadas en el ej. 2.

2.6.3 Requerimientos

Los requerimientos se encuentran en el archivo .rqst, allí hay importaciones a librerías y también la función imshow utilizada.



```
#Librerías
import cv2
import numpy as np
import matplotlib.pyplot as plt
from ipywidgets import interact, IntSlider
from PIL import Image
```

Figure 2.30: Requerimientos



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

Figure 2.31: Función Imshow