Trabajo Práctico N°2- Informe



Rodríguez Griño, Tomás: R-4643/4

Rosito, Valentín: R-4662/1

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Tecnicatura Universitaria en Inteligencia Artificial.

Problema 1

Convertimos la imagen a hls y aplicamos sobre el canal de saturación un filtro de desenfoque.

```
hls = cv2.cvtColor(image, cv2.COLOR_BGR2HLS)
h, l, s = cv2.split(hls)
image_blureada = cv2.blur(s, (3, 3))
```

Esto lo hacemos con el fin de:

- Reducir las variaciones rápidas de saturación
- Mejorar la segmentación

Probamos otros enfoques como convertirlo a hsv o directamente desde escala de grises, pero no podíamos llegar al resultado deseado

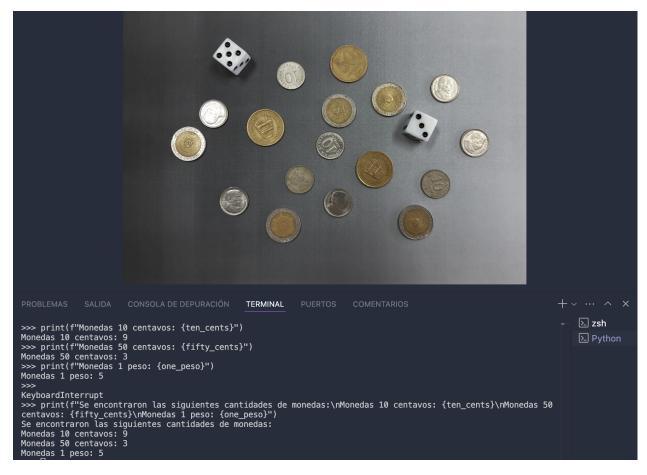
```
_, imagen_binaria = cv2.threshold(image_blureada, 12, 255, cv2.THRESH_BINARY)
```

Luego, binarizamos la imagen para poder aplicar operaciones morfológicas

Función que muestra la cantidad de monedas detectadas por cada distinta de moneda

```
def conteo_monedas(monedas):
    # Ordenamos la lista de monedas en base a su area para facilitar la visualziacion
    sorted_monedas = sorted(monedas , key=lambda x: x[1])
    ten_cents = 0
    fifty_cents = 0
    one_peso = 0

for moneda in sorted_monedas:
    if moneda[1] > 95000:
        fifty_cents += 1
    elif moneda[1] < 65000:
        ten_cents += 1
    elif moneda[1] < 85000 and moneda[1] > 65000:
        one_peso += 1
return print(f"Se encontraron las siguientes cantidades de monedas:\nMonedas 10 centavos: {ten_ce
```



Messi

Salida de la cantidad de monedas detectadas

```
>>> sorted_monedas = sorted(monedas , key=lambda x: x[1])
>>> sorted_monedas[0][1]
51280.0
>>> sorted_monedas[-1][1]
1011<u>5</u>5.0
```

Detección de número en dados, primero comenzamos guardando los contornos en una lista, pero al querer aplicar operaciones morfológicas no logramos que detecte bien los puntos, por lo que decidimos dibujar el contorno de cada dado en una máscara de la imagen original (Con valores binarios), y aplicando operaciones morfológicas, logramos que detecte bien los números

Salida final de la resolución del problema 1

```
>>> conteo_monedas(segmentacion_monedas_dados("TP2/monedas.jpg")[0])
Se encontraron las siguientes cantidades de monedas:
Monedas 10 centavos: 9
Monedas 50 centavos: 3
Monedas 1 peso: 5
>>>
KeyboardInterrupt
>>> conteo_dados((segmentacion_monedas_dados("TP2/monedas.jpg")[1]))
Dado 1 tiene 3 puntos
Dado 2 tiene 5 puntos
>>>
```

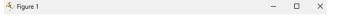
Problema 2

Lo primero que realizamos fue cargar las imágenes, para luego procesarlas a partir de una función:

```
def procesar_imagen(ruta_imagen):
   # Cargar la imagen
   imagen = cv2.imread(ruta_imagen)
   # Desenfocar la imagen
   imagen_blur = cv2.GaussianBlur(imagen, (5, 5), 0)
   # Convertir a escala de grises
   imagen_gray = cv2.cvtColor(imagen_blur, cv2.COLOR_BGR2GRAY)
   # Detectar bordes
   bordes = cv2.Canny(imagen_gray, 50, 250)
   contornos, _ = cv2.findContours(bordes, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
   for contorno in contornos:
       area = cv2.contourArea(contorno)
       if area > 1500: # Ajustar según el tamaño esperado de la placa
           x, y, w, h = cv2.boundingRect(contorno)
           placa_segmentada = imagen[y:y+h, x:x+w]
           mostrar_imagen(placa_segmentada, "Placa Segmentada")
           detectar_caracteres(placa_segmentada)
           break # Salir después de encontrar la primera placa
```

Lo que hace esta función es, primero cargar la imagen, luego la desenfoca, la convierte a escala de grises y a partir de esta misma, detecta los bordes. Luego filtra los contornos para encontrar la placa. Al ir ajustando el tamaño del área, nos dimos cuenta de que el valor que más patentes detectaba era 1500, con 6/12 patentes detectadas

Ejemplo de salida de una de las imágenes.



Placa Segmentada

BUF BIT

```
☆ ← → | ⊕ Q ៑ = | 🖺
```

```
def detectar_caracteres(placa):
   placa_gray = cv2.cvtColor(placa, cv2.COLOR_BGR2GRAY)
   # Umbral con Otsu
   _, imagen_binaria = cv2.threshold(placa_gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
   # Encontrar contornos
   contornos, _ = cv2.findContours(imagen_binaria, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
   caracteres_segmentados = []
   for contorno in contornos:
       x, y, w, h = cv2.boundingRect(contorno)
       relacion_aspecto = h / w if w != 0 else 0
       if w > 10 and h > 20 and 1.5 <= relacion_aspecto <= 3.0:
          caracteres_segmentados.append((x, y, w, h))
           cv2.rectangle(placa, (x, y), (x + w, y + h), (0, 255, 0), 2)
     Agrupar caracteres en grupos de 3
   for i in range(0, len(caracteres_segmentados), 3):
       grupo = caracteres_segmentados[i:i + 3]
       if len(grupo) == 3:
           x1, y1, w1, h1 = grupo[0]
           x2, y2, w2, h2 = grupo[2]
           cv2.rectangle(placa, (x1, y1), (x2 + w2, y1 + h1), (255, 0, 0), 2)
    mostrar_imagen(placa, "Placa con Caracteres Recuadrados")
```

Esta función se encarga de la detección de caracteres en las patentes. Tuvimos dificultades al detectar las patentes, sobre todo problemas de segmentación. Intentamos usando blackhat y diferentes operaciones morfológicas como OPEN Y CLOSE y no pudimos lograrlo.

Esta función se encarga de procesar todas las imágenes en un directorio específico, así la persona que ejecuta no tiene que estar cambiando el nombre del archivo, si no que recorre todos los archivos que hay en ese directorio.