Trabajo Práctico N° 3

Procesamiento de Imágenes I

Tecnicatura Universitaria en Inteligencia Artificial

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Universidad Nacional de Rosario

2023

Integrantes:

Augusto Farias,

Guido Lorenzetti,

Micaela Pozzo,

Patricio Vercesi

Objetivo:

El objetivo del trabajo fue desarrollar un algoritmo para detectar automáticamente cuando se detienen
los dados y leer el número obtenido en cada uno de un video en el cual una persona arroja dados a una mesa.

A su vez generar videos donde los dados, mientras estén en
reposo, aparezcan resaltados con un bounding box de color azul y además,
agregar sobre los mismos el número reconocido

Desarrollo:

Función create_video(tirada):

Esta función en practico.py es la función principal que llama a las otras dos y toma un número de tirada y procesa dicho video en cuestión, produciendo una versión con los dados y puntos detectados de los mismos.

Empieza almacenando los datos del video y usando la metadata del mismo para definir el formato del video que va resultar del procesamiento del mismo:

```
def create_video(tirada):
    # --- Leer y grabar un video -----------------------------------------------
    cap = cv2.VideoCapture(f'tirada_{tirada}.mp4')
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(cap.get(cv2.CAP_PROP_FPS))

    out = cv2.VideoWriter(f'Video-Output{tirada}.mp4', cv2.VideoWriter_fourcc(*'mp4v'), fps, (width,height))
    centroids1=[[0,0]]
    fr=0
```

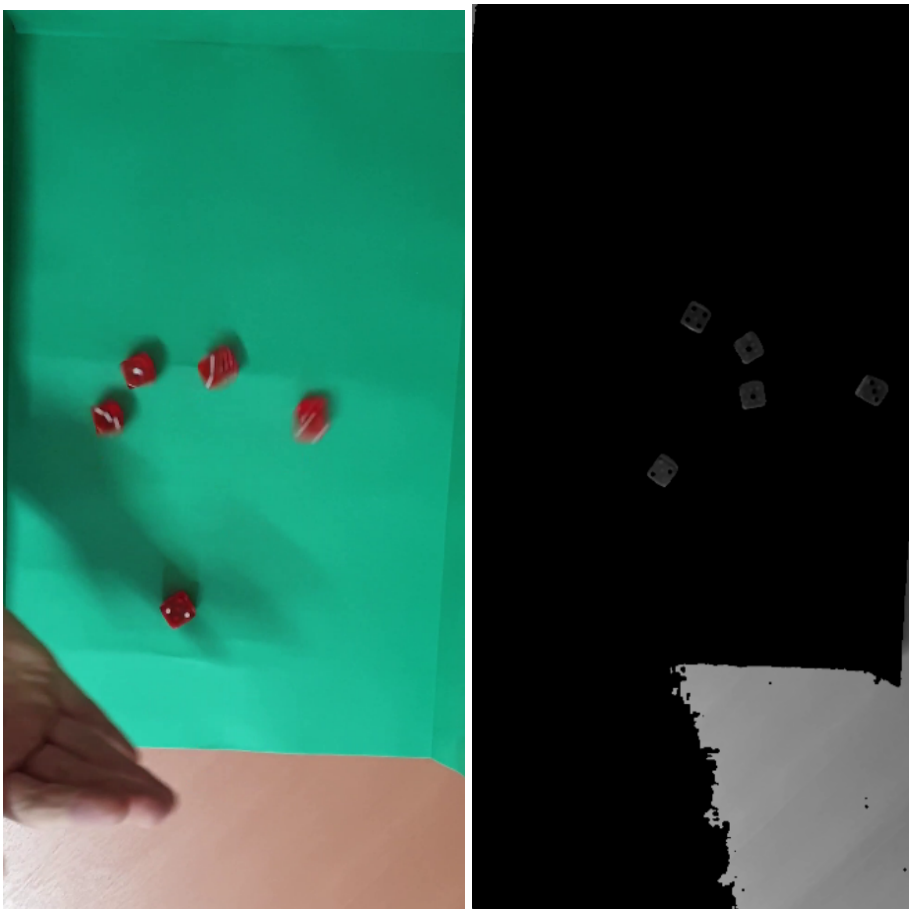
Luego recorremos frame por frame el video para hacer la edición analizándolos y en el caso de que sean los importantes, modificándolos, uno por uno:

```
while (cap.isOpened()):
    ret, frame = cap.read()
    if ret==True:
```

Luego se le aplica la función de filtrado que nos devolverá los fotogramas con los dados y los puntos bien separados del fondo (en el caso de que estén quietos):

```
        img_filtered = filtered(frame)
```

Observamos la imagen sin filtrar y filtrada:



Luego se aplica la función manhattan_distance() definida como:

```
def manhattan_distance(list1, list2):
    list1 = sorted(list1, key=len)
    list2 = sorted(list2, key=len)
    return sum(sum(abs(x - y) for x, y in zip(tuple1, tuple2)) for tuple1, tuple2 in zip(list1, list2))
```

la misma obtiene la distancia entre los centroides de dados.

Se aplica un condicional que la distancia sea menor a 1200, en caso positivo aplica la función detectar_dado, explicada proximamente en el informe:

```
        distancia = manhattan_distance(centroids1, centroids)
        if distancia < 1200:
            frame=detectar_dado(frame, num_labels, labels, stats, centroids)
        centroids1 = centroids
```

Por último, además de hacer un muestro por pantalla, almacena en el video salida “out” el fotograma final redimensionado:

```
        frame_show = cv2.resize(frame, dsize=(int(width/3), int(height/3)))
```

Como esta función es llamada dentro de un bucle for, se aplica sobre cada video original.

```
for i in range(1, 5):
    create_video(i)
```

Función detectar_dado(frame, num_labels, labels, stats, centroids)

Definimos una función para detectar los dados en la imagen original, la cual recibe como entrada los resultados de aplicar componentes conectados calculados anteriormente con la función 'cv2.connectedComponentsWithStats': 'frame', 'num_labels', 'labels' y 'stats'.

'labeled_shape' se inicializa como matriz de ceros con el mismo tamaño que la imagen de entrada.

Se establecen umbrales, 'RHO_TH' y 'AREA_TH'.

'labeled_image' se crea otra matriz usando 'cv2.merge' para combinar las tres matrices 'aux' en una sola matriz en formato de imagen a color.

Se inicializa una lista vacía de huecos de los dados.

```
def detectar_dado(frame, num_labels, labels, stats, centroids):
    labeled_shape = np.zeros_like(frame)
    RHO_TH = 0.8    # Factor de forma (rho)
    AREA_TH = 500   # Umbral de area
    aux = np.zeros_like(labels)
    labeled_image = cv2.merge([aux, aux, aux])
    # Clasifico en base al factor de forma
    holes=[]
```

Iteramos por los objetos etiquetados en la imagen 'num_labels' y descartamos los que son menores al umbral definido en 'AREA_TH'.

Calculamos el factor de forma 'rho' de cada objeto, utilizamos operaciones de contorno para calcular el área y el perímetro del objeto, y los objetos que cumplen cierto criterio del factor de forma se clasifican como cuadrados.

```python
for i in range(1, num_labels):

    # --- Remuevo objetos con area chica -------------------------------------
    if (stats[i, cv2.CC_STAT_AREA] < AREA_TH):
        continue

    # --- Selecciono el objeto actual ----------------------------------------
    obj = (labels == i).astype(np.uint8)

    # --- Calculo Rho ---------------------------------------------------------
    ext_contours, _ = cv2.findContours(obj, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    area = cv2.contourArea(ext_contours[0])
    perimeter = cv2.arcLength(ext_contours[0], True)
    rho = 4 * np.pi * area/(perimeter**2)
    sens = 0.35
    flag_cuadrado = (1-sens) * RHO_TH < rho < (1+sens) * RHO_TH
```

Calculamos la cantidad de puntos que tiene cada dado con ‘find_contours’ y para los objetos cuadrados se dibuja el bounding box, un rectángulo alrededor de ellos en la imagen original. Se imprimen los ‘agujeros’ que serían los números que tiene cada dado en sus respectivos rectángulos.

```python
    # --- Clasifico ----------------------------------------------------------

    if flag_cuadrado:
        # --- Calculo cantidad de puntos -------------------------------------
        all_contours, _ = cv2.findContours(obj, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
        hole = len(all_contours) - 1
        if hole != 0:
            # --- Dibujo el bounding box -------------------------------------
            x, y, w, h = stats[i, cv2.CC_STAT_LEFT], stats[i, cv2.CC_STAT_TOP], stats[i, cv2.CC_STAT_WIDTH], stats[i, cv2.CC_STAT_HEIGHT]
            cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 4)
            # --- Dibujo el label --------------------------------------------
            cv2.putText(frame, f"{hole}", (x, y), cv2.FONT_HERSHEY_SIMPLEX, 3, (255, 255, 255), 2, cv2.LINE_AA)

return frame
```

La función devuelve la imagen original ‘frame’ con los objetos detectados y etiquetados.

Función **filtered(frame):**

Esta función en filter_image.py toma cada frame y realiza un filtro para separar claramente los dados de todo el resto de la imagen.

Lo que hace al principio es crear 2 máscaras de la parte de las imágenes que están en rojo usando 2 rangos de rojos distintos, facilitados por el hecho de almacenar la imagen en formato HSV:

```python
    # Convertir el frame de BGR a HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

```
    # Definir el rango de colores rojos en HSV
    lower_red = np.array([0, 75, 20])
    upper_red = np.array([10, 255, 255])

    # Crear una máscara para los píxeles rojos
    mask1 = cv2.inRange(hsv, lower_red, upper_red)

    # Definir otro rango de colores rojos en HSV
    lower_red = np.array([160, 100, 20])
    upper_red = np.array([179, 255, 255])

    # Crear una máscara para los píxeles rojos en el rango 2
    mask2 = cv2.inRange(hsv, lower_red, upper_red)
```

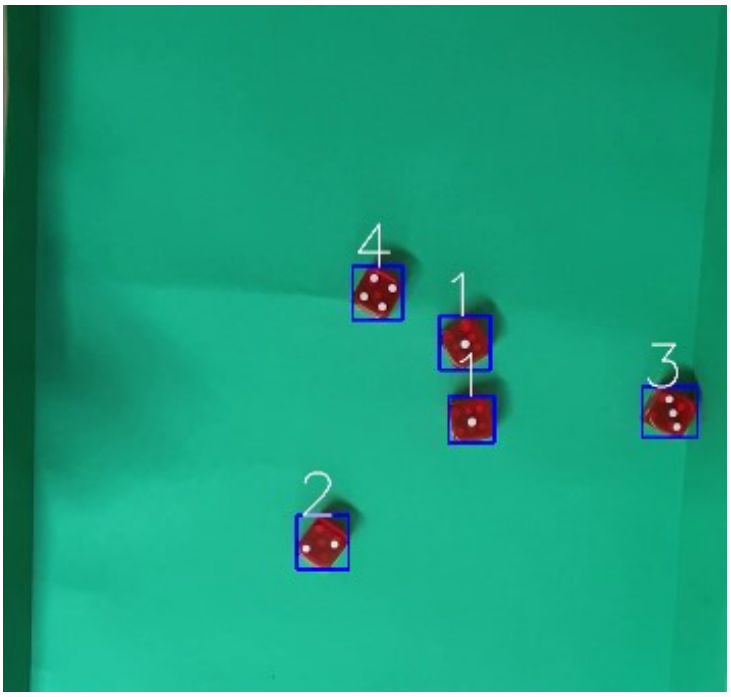
Luego une dichas máscaras y las usa para obtener una imagen filtrada en escala de gris:

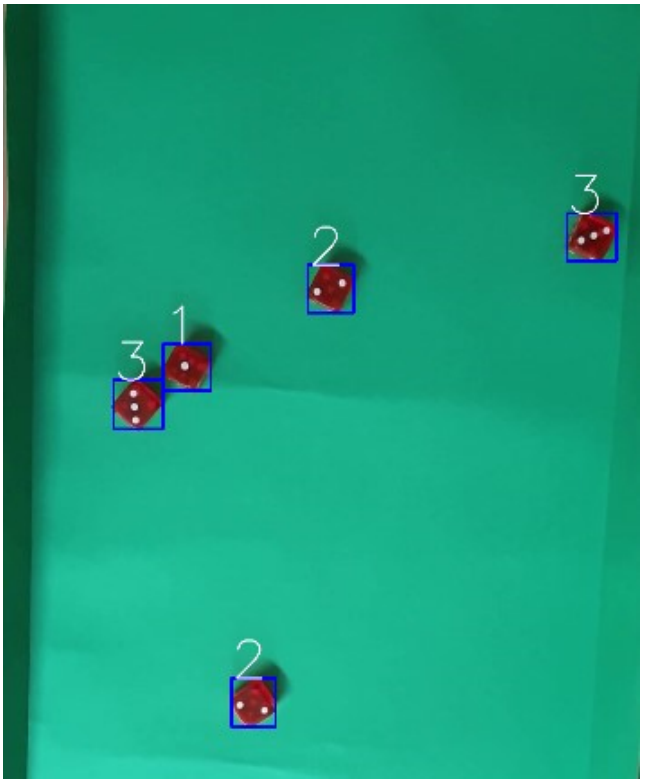
```
    # Combinar las máscaras para abarcar un rango más amplio de rojos
    mask = mask1 + mask2

    # Aplicar la máscara al frame original
    result = cv2.bitwise_and(frame, frame, mask=mask)

    img_gray = cv2.cvtColor(result, cv2.COLOR_RGB2GRAY)
```

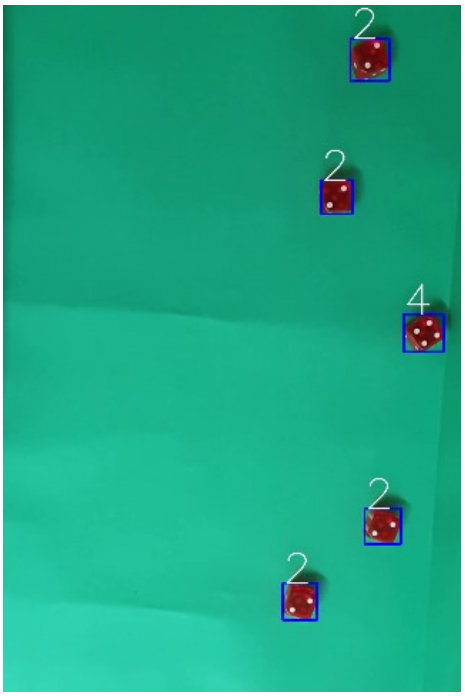
Y después le aplicamos una serie de transformaciones a la imagen obtenida para unir mejor los bordes de los dados y separar los puntos en sus caras del fondo así podemos contarlos bien y retornamos ese resultado final:

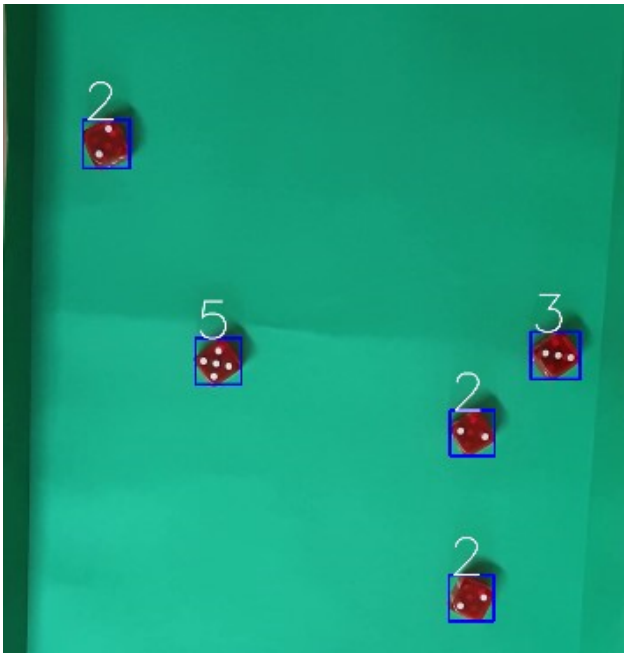
```
    img_filtered = cv2.medianBlur(img_gray, 7)
    se = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
    binary_img = cv2.morphologyEx(img_filtered, cv2.MORPH_OPEN, se)    #
Apertura para remover elementos pequeños
    binary_img = cv2.morphologyEx(binary_img, cv2.MORPH_CLOSE, se)   #
Clausura para rellenar huecos.

    return binary_img
```

Resultados de cada video:









Conclusiones:

Logramos obtener buen resultado en los 4 videos ofrecidos por los docentes. Estos pueden verse en la sección Resultados del informe. A lo largo del trabajo lidiamos con que no se reconocían de buena manera los numeros de los dados. Por ellos aplicamos técnicas de transformación y filtrado aprendidas en la materia, logrando obtener buenos resultados. A su vez nos fue importante trabajar con HSV para segmentar por colores y detectar con mayor importancia los rangos de valores rojos.
También fue importante trabajar con los RHO para comparar factores de forma y poder segmentar los cuadrados para poder analizar los holes de cada uno utilizando find contours.
Fueron importantes aplicar todas las técnicas aprendidas a lo largo de la materi, ya que sin todas ellas no llegábamos a buenos resultados.