Facultad de
Ciencias Exactas,
Ingeniería y Agrimensura

Trabajo Practico Nº3
Procesamiento de Imágenes 1

Procesamiento de Imágenes 1

Integrantes:

- Garcia, Timoteo
- Moresco, Brisa
- Britos, Julian
- Rondini, Sofia

Ejercicio: Cinco dados

Las secuencias de video tirada_1.mp4, tirada_2.mp4, tirada_3.mp4 y tirada_4.mp4 corresponden a tiradas de 5 dados. En la Figura 1 se muestran los dados luego de una tirada. Se debe realizar lo siguiente:
a) Desarrollar un algoritmo para detectar automáticamente cuando se detienen los dados y leer el número obtenido en cada uno. Informar todos los pasos de procesamiento.
b) Generar videos (uno para cada archivo) 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.

Resolución:

- **Transformación a Espacio de Color HSV (Matiz, Saturación, Valor):** Se realiza una conversión de cada frame del video al espacio de color HSV para facilitar la identificación y filtrado de colores.
- **Filtrado por Tonalidades de Color Rojo:** Se aplican dos máscaras para filtrar las tonalidades de color rojo en el espacio de color HSV. Estas máscaras se combinan para obtener una máscara final que resalta las áreas rojas en el video.
- **Binarización de Frames:** El resultado de la filtración se convierte en una imagen binaria utilizando un umbral, donde los píxeles que cumplen con ciertas condiciones se establecen en blanco y los demás en negro.
- **Dilatación de Frames:** Se aplica una operación de dilatación a la imagen binaria utilizando un kernel de 5x5. Esto ayuda a unir regiones cercanas y a mejorar la detección de contornos.
- **Detección de Contornos:** Se encuentran los contornos en la imagen dilatada utilizando el algoritmo de detección de contornos de OpenCV.
- **Filtrado de Contornos por Relación de Aspecto y Área:** Se filtran los contornos identificados según su relación de aspecto (aproximadamente cuadrados) y su área. Solo aquellos contornos que cumplen con ciertos criterios (rel_Aspect aprox. = 1 , y area > 200) se consideran dados potenciales.
- **Seguimiento de Dados:** Se utiliza un mecanismo de seguimiento que compara las coordenadas de los dados en el frame actual con las del frame anterior. Si se cumplen ciertas condiciones (mismas coordenadas (+- 3 frames) durante 5 frames consecutivos), se considera que los dados están quietos.
- **Análisis de Números en Caras de los Dados:** Cuando se detecta que los dados están quietos durante al menos 5 frames, se captura un frame para su análisis. Se identifican los números en las caras de los dados mediante la detección de componentes conectadas y se dibujan bounding boxes alrededor de ellos.

Procedimiento

Cargamos el video. Obtenemos sus features, y definimos el video que vamos a devolver post procesamiento.

```
cap = cv2.VideoCapture(f'tirada_2.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-2.mp4', cv2.VideoWriter_fourcc(*'mp4v'), fps,
(width,height))
```

Definimos algunas variables globales que vamos a usar en el resto del codigo para su funcionamiento.

```
coordenadas_actuales = {} #--> Guardaremos las coordenadas de los dados en el frame act.
coordenadas_anteriores={} #--> Guardaremos las coordenadas de los dados en el frame ant.
imagen_flag = False # --> Flag que usamos para sacar una sola captura para analizar los da
dos
cont = 0 # Contador que cuenta je, cuantos frames estan quietos los dados
```

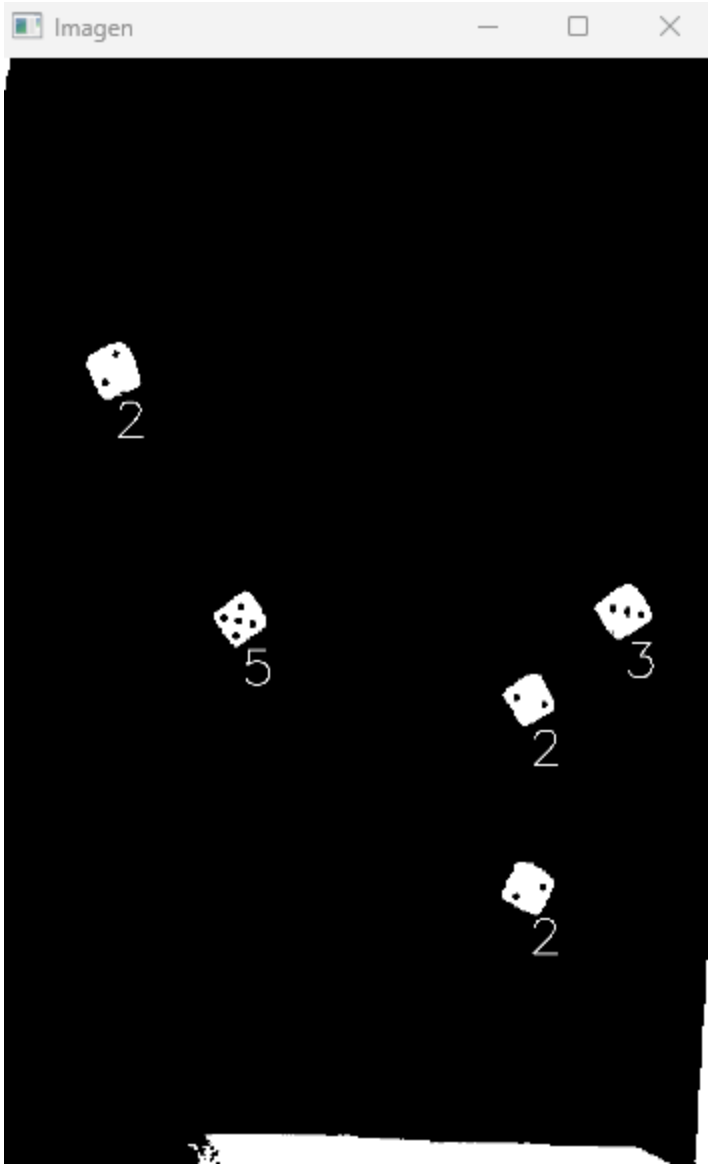
Creamos la función que calcula el numero del dado. Dicha Función recibe el primer frame (una imagen dilatada y binarizada) de cuando los dados están 'detenidos',calcula las componentes conectadas de esa imagen, filtra por área para quedarse solo con los dados, y vuelve a analizar las componentes conectadas pero en este caso del ‘interior de los dados', obtenemos todos los contornos 'hijos' que tiene cada dado, y nos quedamos con los que tienen área mayor a 20. La cantidad de esos contornos será el numero del dado.

Tambien devuelve una imagen, que es como finalmente detecta los dados, anexo ejemplo

```
def dados(frame, imagen):
    img = imagen.copy()
    num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(img)
    img3 = np.zeros_like(img)
    for ii in range(1, num_labels):
        obj = (labels == ii).astype(np.uint8)
        area = stats[ii,cv2.CC_STAT_AREA]
        ratio_aspecto = stats[ii,cv2.CC_STAT_HEIGHT] / stats[ii,cv2.CC_STAT_WIDTH]
        if 7000 > area > 3500:
            img3+= obj
    num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(img3)
    for jj in range(1, num_labels):
        obj = (labels == jj).astype(np.uint8)
        if (num_labels - 1) == 5:
            centroid_x, centroid_y = centroids[jj]
            contours, _ = cv2.findContours(obj, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
            contours_filtrados = [contour for contour in contours if cv2.contourArea(conto
```

```
ur) > 20]
            numero_dado = len(contours_filtrados) - 1
            #Reemplazar por frame
            cv2.putText(frame, f'{numero_dado}', (int(centroid_x), int(centroid_y)+ 100),
cv2.FONT_HERSHEY_SIMPLEX, 2.5, (255, 0, 0), 2)
            cv2.putText(img, f'{numero_dado}', (int(centroid_x), int(centroid_y)+ 100), cv
2.FONT_HERSHEY_SIMPLEX, 2.5, (255, 0, 0), 2)

    return frame, img
```



Analicemos Frame a Frame

Leemos los frames del video

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

Filtramos por color

Convertimos el frame de BGR a HSV para poder filtrar por el tono, la saturación y el valor. Creamos 2 mascaras, con 2 umbrales cada una, ya que el rojo va de 0 a 10 aprox, y de 165 a 180, en el espacio HSV.

Luego unimos las mascaras y creamos nuestro frame resultante ’resultado’, y lo pasamos a escala de grise.

```
hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
  #Filtramos para quedarnos con los colores rojos (Dados son rojos)
  rojo_bajo = np.array([0, 50, 50], dtype=np.uint8)
  rojo_alto = np.array([15, 255, 255], dtype=np.uint8)

  mascara_rojo_baja = cv2.inRange(hsv_frame, rojo_bajo, rojo_alto)

  rojo_bajo = np.array([160, 50, 50], dtype=np.uint8)
  rojo_alto = np.array([180, 255, 255], dtype=np.uint8)

  mascara_rojo_alta = cv2.inRange(hsv_frame, rojo_bajo, rojo_alto)

  mascara_rojo = cv2.bitwise_or(mascara_rojo_baja, mascara_rojo_alta)

  resultado = cv2.bitwise_and(frame, frame, mask=mascara_rojo)
  resultado = cv2.cvtColor(resultado, cv2.COLOR_BGR2GRAY)
```

Procedemos a binarizar y dilatar nuestro frame

```
th, binary = cv2.threshold(resultado, 20, 255, cv2.THRESH_BINARY)

kernel = np.ones((5,5), np.uint8)
dilatado = cv2.dilate(binary, kernel, iterations=1)
```

Buscamos y analizamos los contornos que se van detectando Frame a Frame

```
contornos, jerarquia = cv2.findContours(dilatado, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMP
LE)
```

Nuestro criterio de 'detencion' sera: encontrar los contornos, y sus coordenadas. Quedarnos con aquellos que sean 'cuadrados', pues los dados tienen ese aspecto. Y filtrando aquellos mayores a cierto umbral, para eliminar ruido. La idea es analizar las coordenadas de los contornos, si un contorno por 5 frames (nuestro criterio) tiene las misma coordenadas x,y consideramos que el dado esta quieto.

```
frame_contornos = frame2.copy()
coordenadas_actuales = {}
for idx, cnt in enumerate(contornos):
    x, y, w, h = cv2.boundingRect(cnt)
    aspect_ratio = float(w) / h
    area = w * h
    if np.isclose(aspect_ratio, 1, atol=0.1) and area > 200:
        dado_actual = f'dado{idx + 1}'
        coordenadas_actuales[dado_actual] = [x, y, cnt]
```

```
if coordenadas_anteriores and coordenadas_actuales:
  for dado_act, coords_act in coordenadas_actuales.items():
    x_act, y_act, cnt_act = coords_act
# Busca el dado correspondiente en coordenadas_anteriores basándose en la distancia euclid
iana
    dado_ant, coords_ant = min(coordenadas_anteriores.items(), key=lambda item: np.linalg.
norm(np.array(item[1][:2]) - np.array([x_act, y_act])))
    x_ant, y_ant, cnt_ant = coords_ant
    resta_x = abs(x_act - x_ant)
    resta_y = abs(y_act - y_ant)
    margen = 3 #pixels
    cantidad_contornos = len(coordenadas_actuales)
"A la condicion de 'dados quietos' le agregamos que los contornos detectados tienen que se
r 5, pues hay 5 dados."
    if resta_x <= margen and resta_y <= margen and cantidad_contornos == 5: #COndicion de
dados quietos
      cont +=1 #Contamos 1 frame quieto
    else:
      cont = 0 #Si se mueve vuelve a 0
```

Como dijimos arriba, nuestro criterio de detención van a ser 5 frames, es decir que los dados deben cumplir la condición de estacionalidad por 5 frames. Una vez que

cumplan esta condición, se toma una captura (el primer frame quieto) y se analizan dado a dado para saber su numero.

A su vez si están quietos, se dibujan los contornos, o bounding boxes, alrededor de los dados.

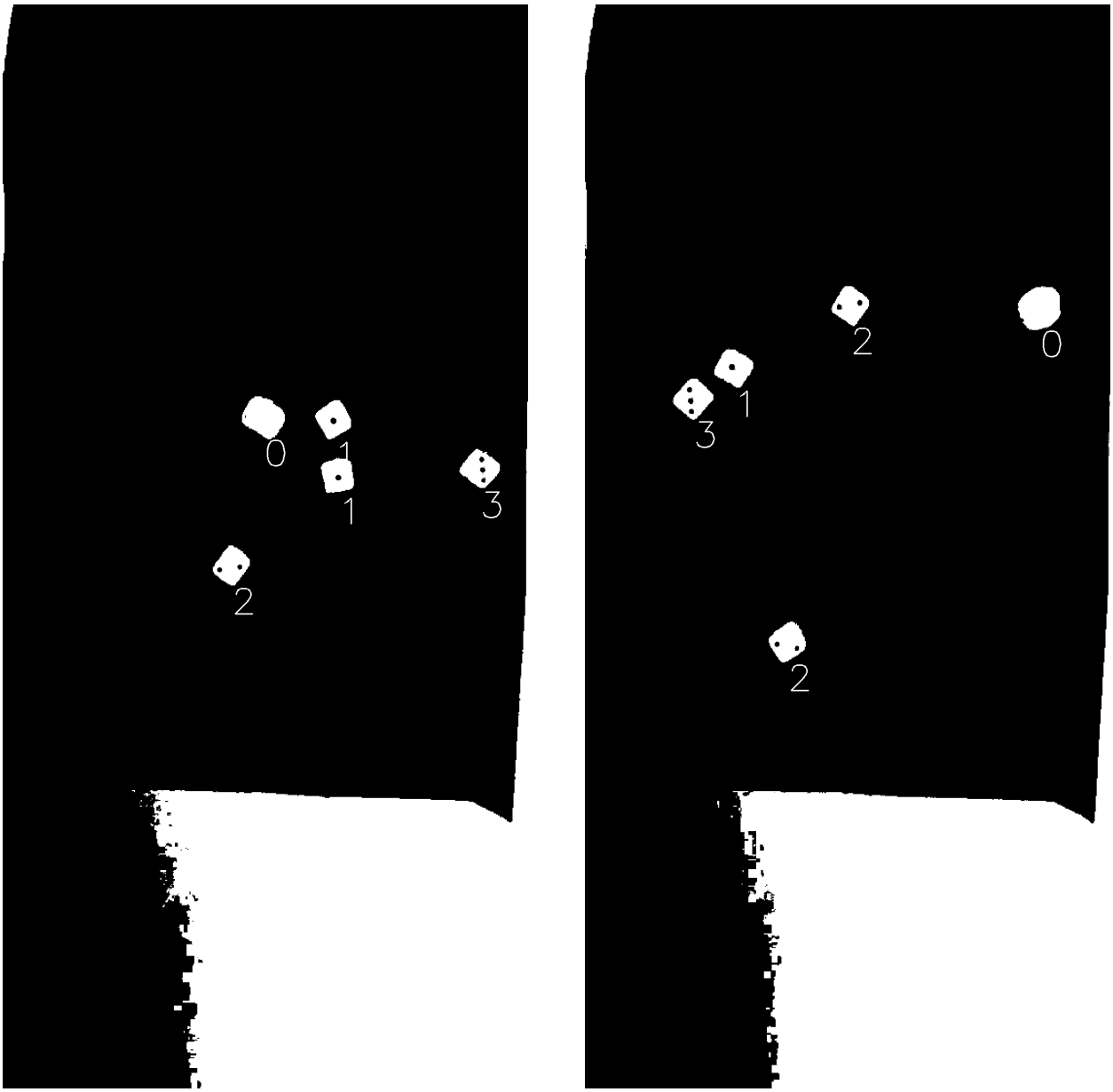
```
if not(imagen_flag):
  imagen = binary
  imagen_flag = True
frame_contornos, imagen_numeros = dados(frame_contornos, imagen)
cv2.imshow('Imagen', cv2.resize(imagen_numeros, dsize=(int(width/3), int(height/3) )))
cv2.drawContours(frame_contornos, [cnt_ant], -1, (255, 0, 0), 2)
```

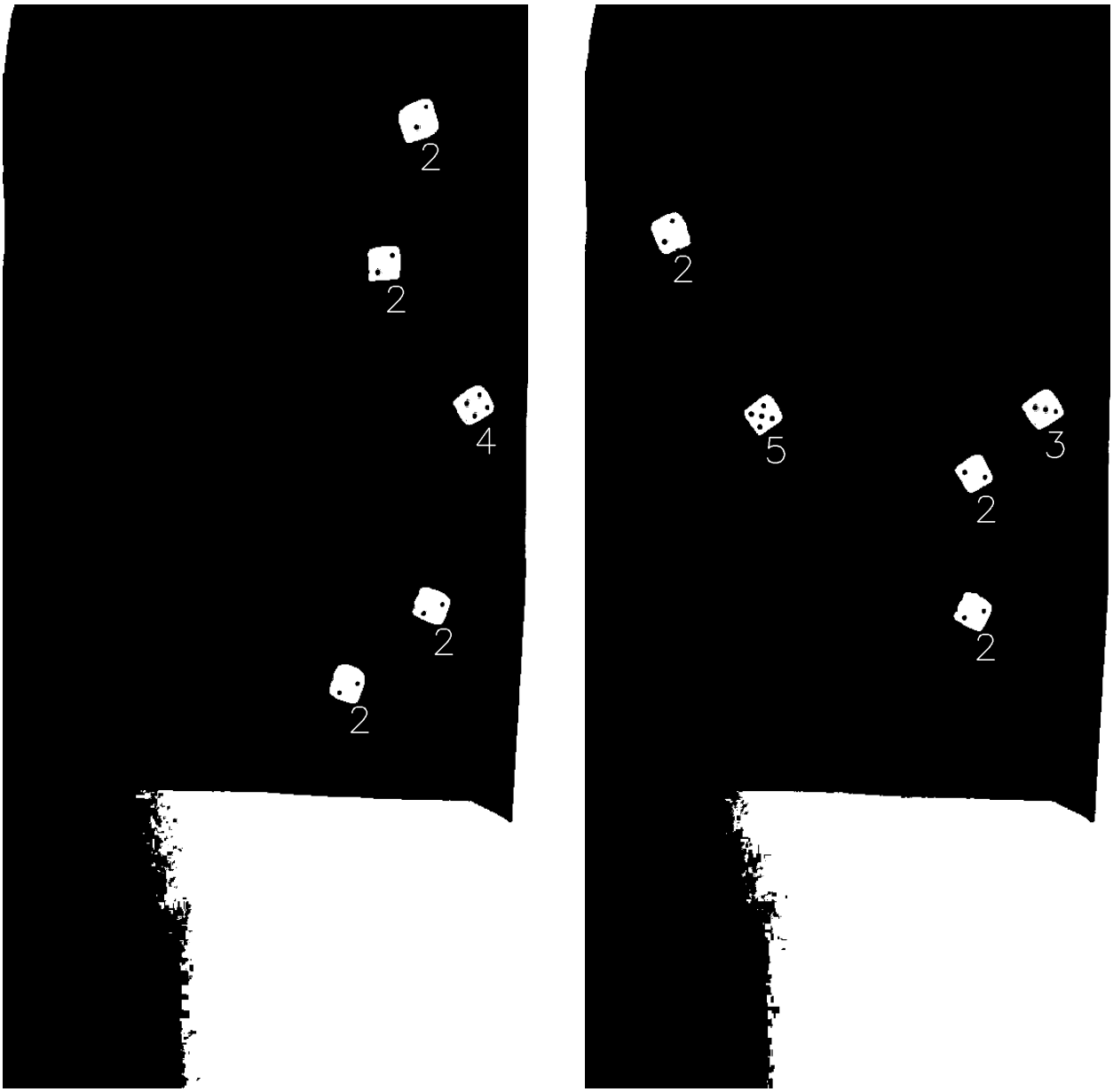
Mostramos nuestro “frame” modificado, y exportamos el video final.

```
frame_show = cv2.resize(frame_contornos, dsize=(int(width/3), int(height/3)))
cv2.imshow('Frame', frame_show)

        # -----------------------------------------------------------------
out.write(frame_contornos)  # grabo frame --> IMPORTANTE: frame debe tener el mismo tamaño
que se definio al crear out.
if cv2.waitKey(25) & 0xFF == ord('q'):
  break
else:
  break
```

Nota: En las 2 primeras tiradas tenemos el problema de que no detecta cuando los 5 dados estan quietos, y por eso saca mal la captura.


0
1
1
3
2
2
1
3
0
2



https://clipchamp.com/watch/Qb9GTfKDMyM