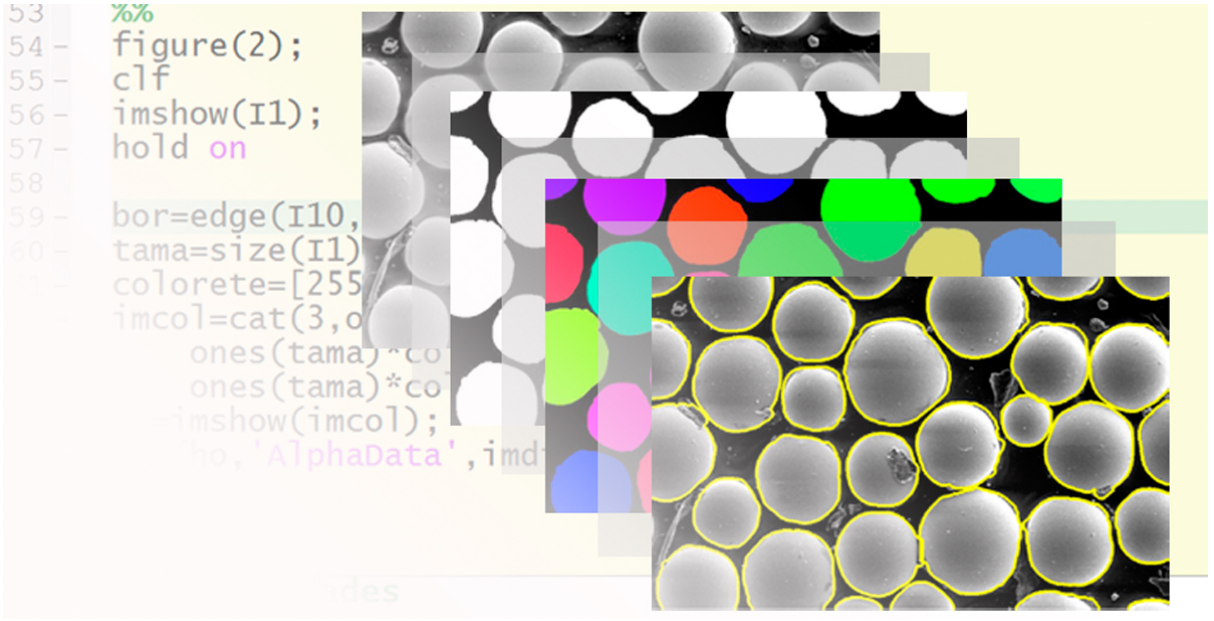
# Trabajo Práctico Nº3
# Procesamiento de Imágenes 1
## TUIA





**GRUPO 7**

**Integrantes:**

- **Ferrari, Enzo**
- **Rodriguez, Abril Nazarena**
- **Ferrero, Santiago**
- **Loza, Santiago**

ÍNDICE

## INTRODUCCIÓN

En el presente informe se detallan los pasos y los conceptos aplicados para la resolución del Trabajo Práctico N°3 de la asignatura, el cual consta de dos ejercicios.

El primero de ellos tiene como objetivo desarrollar un algoritmo capaz de detectar automáticamente cuando los dados de un video se detienen, proporcionando la información sobre los valores obtenidos en cada uno. En el segundo ejercicio, la meta es generar videos (uno para cada tirada) que muestren los dados en reposo, resaltando cada uno con un bounding box azul que incluya el número asociado.

Ambos ejercicios requieren la aplicación de técnicas de procesamiento de imágenes y la manipulación de videos, utilizando bibliotecas como **OpenCV** y **NumPy** en el entorno de programación **Python**.

El informe está organizado en secciones, cada una correspondiente a la solución detallada de un ejercicio. Al final del documento, se presenta una conclusión general derivada de los resultados obtenidos en ambos ejercicios.

Dependencias:

Los entornos que utilizamos para resolver el trabajo son Visual Studio Code y Google Colab. En cuanto a las librerías, son las siguientes.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import sys
```

Se cuenta con:

- **cv2**: OpenCV, para manipular las imágenes.
- **numpy:** ofrece funciones y métodos para manipular arrays.
- **matplotlib**: para realizar gráficas y visualizar imágenes.
- **sys**: para pasar un archivo (en este caso un vide) y ejecutar el código por consola

## EJERCICIO 1

Comenzamos definiendo la función **'imshow'** la cual utilizamos en ambos ejercicios. Esta función recibe una imagen y la muestra por pantalla. Nos resulta muy útil para visualizar qué está pasando en cada etapa de procesamiento de la imagen y facilita la toma de próximas decisiones (Imagen 1).

```python
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')
    plt.title(title)
    if not ticks:
        plt.xticks([]), plt.yticks([])
    if colorbar:
        plt.colorbar()
    if new_fig:
        plt.show(block=blocking)
```

Imagen 1. Función 'imshow'

Definimos 4 funciones que se encargarán de todo el procesamiento: seleccionar un frame con los dados en reposo, detectar los dados y evaluar el número que muestra cada uno.

La primera función la llamamos '**detectar_frame**'. Esta recibe como entrada un vídeo y, tras identificar los frames en los que los dados están en reposo, devuelve una lista que contiene dichos frames. (Imagen 2)

En el proceso, se aplica la técnica de componentes conectadas a cada frame, seguida de un filtro por área para descartar objetos demasiado grandes o pequeños. Si el número de componentes conectados en un determinado frame es igual a 6 (representando los cinco dados y el fondo), se considera que dicho frame muestra los dados en reposo, y se añade a la lista almacenada en la variable '**frames_con_dados**'.

```python
def detectar_frame(cap):
    """Recibe un video y devuelve un frame con los datos quietos """
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    frames_con_dados = []
    n_frame = 0
    while (cap.isOpened()):
        ret, frame = cap.read()
        if ret == True:
            if n_frame % 10 == 0:
                frame_example = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                # Detectar dados
                red_channel = frame_example[:,:,0]  # Seleccionamos el canal rojo
                green_channel = frame_example[:,:,1]  # Seleccionamos el canal verde
                frame_binary = ((red_channel > 80) & (green_channel < 70)).astype(np.uint8)  # imagen binaria
                # Aplicar componentes conectadas
                num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(frame_binary, connectivity=8)
                dice_centroids = []
                image_area = stats[0][4]
                for n in range(1, num_labels):  # Filtramos el fondo
                    if image_area * 0.001 < stats[n][4] < image_area * 0.01:
                        dice_centroids.append(centroids[n])

                #Frame con color
                frame = cv2.resize(frame, dsize=(int(width/3), int(height/3)))
                if len(dice_centroids) == 6:
                    frames_con_dados.append(frame)

            n_frame += 1
            if cv2.waitKey(25) & 0xFF == ord('q'):
                break

        else:
            break

    return frames_con_dados[-1] #Elegimos el último frame así nos aseguramos que los dados no estén en movimiento
```

Imagen 2. Función ‘detectar_frame’

La función denominada '**detectar_dados**' recibe como entrada un frame que contiene los dados en reposo. Utilizando un umbral determinado, obtenido mediante pruebas (en este caso, 70), binariza la imagen y aplica el concepto de componentes conectadas. A través de un bucle '**for**', se recorre cada componente, y si cumple con los criterios de filtrado especificados, se identifica como un dado, y su información se almacena en un diccionario. Estos diccionarios, correspondientes a cada dado, se agregan a una lista llamada '**lista_objetos**'. Finalmente, la función devuelve tanto la imagen '**labeled_image**', resaltando los componentes conectados, como la lista '**lista_objetos**' que contiene la información de cada dado detectado. (Imagen 3)

```python
def detectar_dados(frame):
    '''Crea una nueva imagen con los dados y tambien devuelve una lista de diccionarios
    con caracteristicas de los dados'''

    # Proceso de detección de dados
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Detectar dados
    red_channel = frame[:,:,0]  # Seleccionamos el canal rojo
    frame_binary = (red_channel > 70).astype(np.uint8)  # imagen binaria

    lista_objetos = []
    # Aplicar componentes conectadas
    num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(frame_binary, connectivity=8)
    dice_centroids = []

    labeled_image = np.zeros_like(frame_binary)  # Creamos una imagen para volcar los dados
    image_area = stats[0][4]  # Área de la imagen entera
    for n in range(1, num_labels):  # Filtramos el fondo

        # Creamos diccionario para almacenar el objeto y sus datos
        objeto_dicc = {}

        obj = (labels == n).astype(np.uint8)
        if image_area * 0.002 < stats[n][4] < image_area * 0.01:

            # Guardamos el centroide
            objeto_dicc['labels'] = obj
            objeto_dicc['centroid'] = centroids[n]

            # Recortamos la region del dado y la guardamos
            x, y, w, h = stats[n][:4]  # Coordenadas del cuadro delimitador
            objeto_dicc['recorte'] = frame[y:y + h, x:x + w]

            labeled_image[obj == 1] = 255
            dice_centroids.append(centroids[n])
            lista_objetos.append(objeto_dicc)

    return labeled_image, lista_objetos
```

Imagen 3. Función ‘detectar_dados’

La tercera función, '**valor**', acepta una imagen representativa de un dado y tiene como tarea principal la detección del valor asociado al mismo. En primera instancia, convierte la imagen a escala de grises, seguido de la aplicación de un umbral para su binarización (en este caso, 128). A continuación, emplea el concepto de componentes conectadas, y al iterar sobre ellas, se aplica un filtro basado en la relación área/perímetro al cuadrado ('fp'). Este proceso permite identificar los círculos asociados a cada dado, incrementando el contador '**numero_dado**' por cada detección. Finalmente, la función devuelve un string que representa el número detectado en el dado. (Imagen 4)

```python
def valor(image):
  '''Al ingresarle un dado, te devuelve cual es la puntuacion del mismo'''
  gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  _, thresh_img = cv2.threshold(gray_image, thresh=128, maxval=255, type=cv2.THRESH_BINARY)  # Umbralamos

  # Hacemos componentes conectadas para separar cada punto del dado
  num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(thresh_img, connectivity = 8, ltype=cv2.CV_32S)

  labeled_image = np.zeros_like(thresh_img)
  numero_dado = 0
  for i in range(1, num_labels):

      # --- 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)

      if perimeter == 0 :
        continue
      fp = area / perimeter**2

      # Removemos el caso donde fp = 0, es decir, una componente conectada de un solo punto
      if fp == 0:
        continue

      # Identificamos círculos y nos quedamos con aquellos más grandes (filtramos según área)
      #print(stats[i][4])
      if 1/fp > 8 and 1/fp < 17 and stats[i][4] > 4 and stats[i][4] < 15:
        labeled_image[obj == 1] = 255
        # Contamos los puntos
        numero_dado += 1
  return(str(numero_dado))
```

Imagen 4. Función ‘valor’

La función final, denominada '**numero_dados**', acepta como entrada un video y devuelve una lista que contiene los dados junto con el frame en el que estos están en reposo. (Imagen 5)

Su funcionamiento comienza invocando a '**detectar_frame**', proporcionándole el video recibido como parámetro, y guarda el resultado, que es el frame con los dados en reposo, en la variable '**frame**'. Luego, procede a llamar a la función '**detectar_dados**', pasándole la variable '**frame**' obtenida previamente, y recibe como resultado una imagen que destaca los dados y una lista de diccionarios que contienen diversas características asociadas a cada dado.

```python
def numero_dados(image):
    '''Le asigna la puntuacion correspondiente a cada dado'''
    #devuelve el ulitmo frame y el ultimo frame con gcan aplicado
    frame = detectar_frame(image)

    #devuelve una imagen con los dados, y una lista de diccionarios con caracteritisticas de los mismos
    dados,lista_dados = detectar_dados(frame)


    for dado in lista_dados:
      dado['valor'] = valor(dado['recorte'])

    return lista_dados,frame
```

Imagen 5. Función ‘numero_dados’

A continuación, se muestra el código principal. Se parte almacenando en una variable el nombre del video que ingresa el usuario (en caso de que no se ejecute el código desde la consola, este paso se puede hacer a mano cargando el video en el entorno) y leyéndolo con cv2. (Imagen 6)

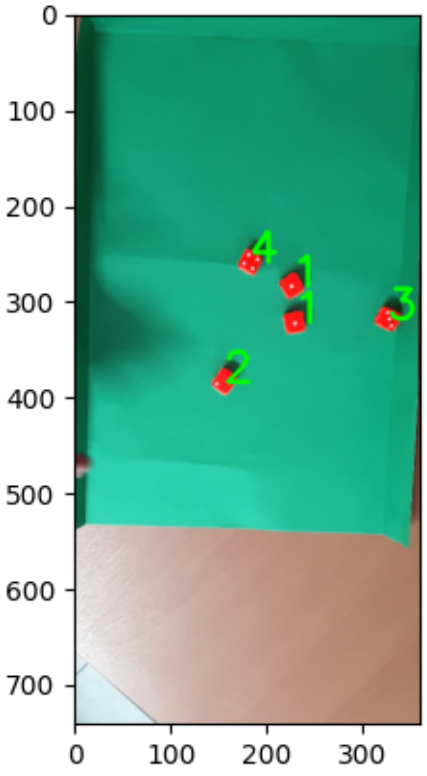
Luego, mediante un while se ejecutan las siguientes tareas:

- Se llama a la función '**numero_dados**' pasándole el video y almacenando los resultados en dos variables '**dados_list**' (lista con los dados obtenidos) y '**frame**' (frame donde se detectaron los dados)
- Se setean los parámetros para el texto.
- Se recorre la lista '**dados_list**', en donde a cada dado se le aplica un color y se le escribe el número que muestra.
- Se muestra la imagen resultante, con cada componente conectada (cada dado) coloreada y su valor.

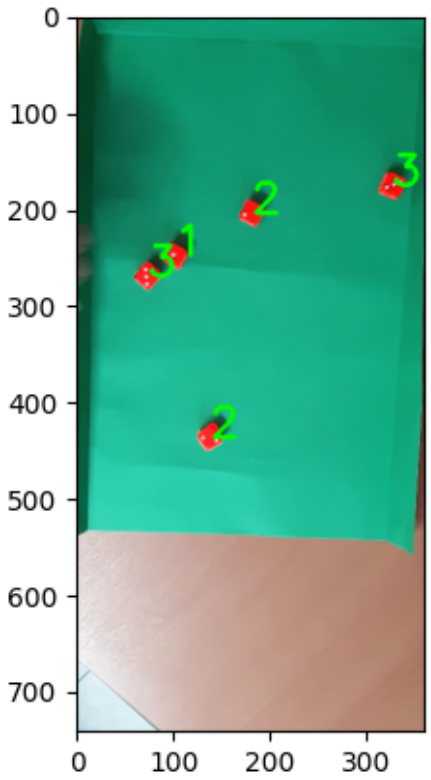
```
''' Ejecucion del codigo, donde pedimos al usuario que ingrese el nombre del video '''

# Video path... Para ejecutar desde la terminal
video_path = sys.argv[1] +'.mp4'

#Sino ingresando a mano
#video_path = 'tirada_1.mp4'

cap = cv2.VideoCapture(video_path)

width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

while cap.isOpened():
    dados_list, frame = numero_dados(cap)

    # Parámetros para el texto
    font = cv2.FONT_HERSHEY_SIMPLEX
    font_scale = 1.5
    font_thickness = 4
    font_color = (0, 255, 0)

    for objeto in dados_list:
        obj = (objeto['labels']).astype(np.uint8)
        if objeto['valor'] != '0' :
            frame[obj == 1, 2] = 255
            cv2.putText(frame, objeto['valor'], (int(objeto['centroid'][0]), int(objeto['centroid'][1])),
                        font, font_scale, font_color, font_thickness)

    # Muestra la imagen resultante
    plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))  # Convertimos de BGR A RGB
    plt.show()

    # Espera la tecla 'q' durante 1 milisegundo
    key = cv2.waitKey(1) & 0xFF
    if key == ord('q'):
        break

# Libera el objeto VideoCapture
cap.release()
cv2.destroyAllWindows()
```

Imagen 6. Código principal

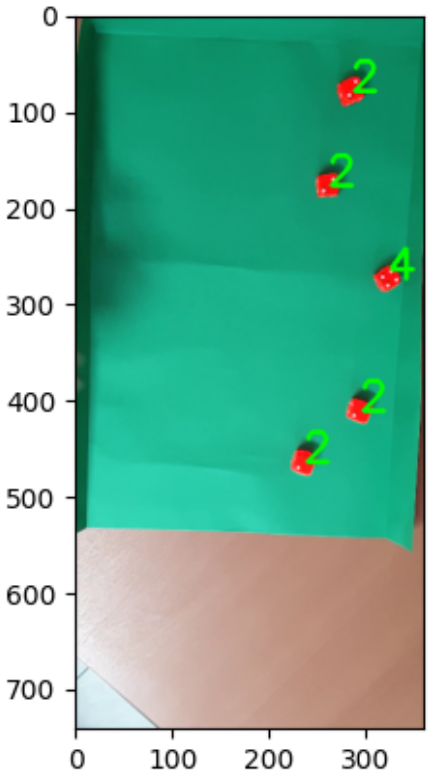
Finalmente, mostramos los resultados que planteamos para este ejercicio:



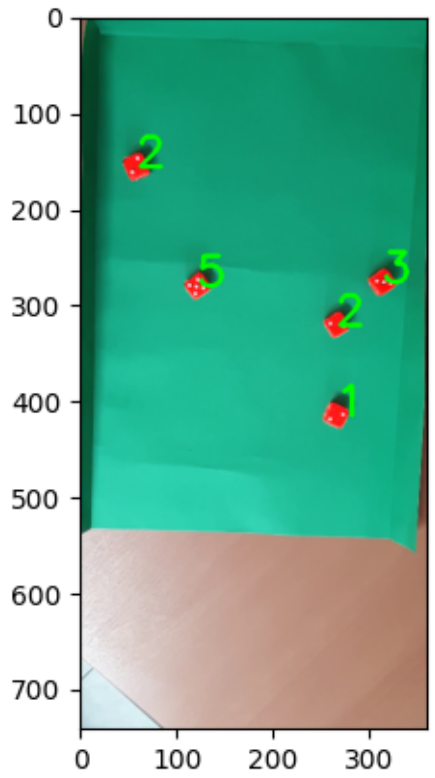

Tirada 1




Tirada 2




Tirada 3




Tirada 4

## EJERCICIO 2

La solución para este ejercicio sigue una metodología similar a la del ejercicio anterior. Sin embargo, se distingue mayormente por la generación de cuatro videos, uno para cada tirada. En estos videos, se implementa la condición de que cuando los dados están en reposo, se visualice un bounding box alrededor de cada uno, acompañado del valor correspondiente.

Definimos 4 funciones, similares a las trabajadas en el ejercicio 1.

La primera de ellas, la llamamos '**detectar_dados**'. Esta función recibe un frame y una de sus tareas es crear una imagen con los dados y sus centroides detectados, la cual nos fue de utilidad para corroborar el funcionamiento paso a paso de la función.

Específicamente, la función proporciona una lista de centroides, acompañada de stats y labels. Esta lista de centroides es fundamental para el código principal, ya que representa las ubicaciones de los dados detectados en el frame y son de utilidad para determinar si los mismos se encuentran en reposo. (Imagen 7)

```python
def detectar_dados(frame):
    '''Crea una nueva imagen con los dados y sus centroides detectados'''
    # Proceso de detección de dados
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    # Detectar dados
    red_channel = frame[:,:,0]  # Seleccionamos el canal rojo
    frame_binary = (red_channel > 70).astype(np.uint8)  # imagen binaria

    # Aplicar componentes conectadas
    num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(frame_binary, connectivity=8)
    dice_centroids = []
    statss = []
    labelss= []

    labeled_image = np.zeros_like(frame_binary)  # Creamos una imagen para volcar los dados
    image_area = stats[0][4]  # Area de la imagen entera
    for n in range(1, num_labels):  # Filtramos el fondo
        obj = (labels == n).astype(np.uint8)
        height = stats[n][cv2.CC_STAT_HEIGHT]
        width = stats[n][cv2.CC_STAT_WIDTH]
        ratio = width / height
        area = stats[n][cv2.CC_STAT_AREA]
        delta = 0.2 # Margen de error para el ratio
        ratio_condition = 1 - delta < ratio < 1 + delta # Buscamos aquellas componentes conectadas cuyo bounding box sea un cuadrado perfecto,
        area_condition = image_area * 0.001 < area < image_area * 0.01 # Filtramos areas muy chicas o muy grandes
        if ratio_condition and area_condition:
            labeled_image[obj == 1] = 255
            labeled_image = cv2.circle(labeled_image, center=(int(centroids[n][0]),int(centroids[n][1])),radius=0,thickness=20,color=0)
            dice_centroids.append(centroids[n])
            statss.append(stats[n])
            labelss.append(labels[n])
    return labeled_image, dice_centroids, statss, labelss
```

Imagen 7. Función 'detectar_dados'

La segunda función, 'distanciaEuclideana' recibe dos pares ordenados de coordenadas, y devuelve la distancia euclidiana entre ambos pares. En el contexto del código principal, esta función se emplea para comparar las posiciones de los centroides en frames sucesivos y determinar si los dados han permanecido en reposo. (Imagen 8)

```python
def distanciaEuclideana(p1,p2):
    '''Siendo X1 y X2 dos pares ordenados de coordenadas, calcula la distancia euclideana entre ellos dos'''
    x1, y1 = p1
    x2, y2 = p2
    return ((x2-x1)**2 + (y2-y1)**2)**(1/2)
```

Imagen 8. Función ‘distanciaEuclideana’

La siguiente, ‘**dadosQuietos**’, recibe dos conjuntos de posiciones de centroides, uno correspondiente al frame anterior y otro al frame actual, junto con un umbral que representa el movimiento máximo permitido entre frames. (Imagen 9)

El propósito de esta función es determinar si entre dos frames consecutivos, hubo movimiento o no. La función compara las posiciones de los centroides entre los frames anterior y actual, calculando la distancia euclidiana entre pares correspondientes. La distancia calculada se compara con el umbral establecido para determinar si el movimiento entre los centroides es inferior al límite permitido. Si se cumple esta condición para todos los centroides, la función devuelve '**True**', indicando que los dados están en reposo. Por el contrario, si al menos un par de centroides muestra un desplazamiento que supera el umbral, la función devuelve '**False**', indicando que ha habido movimiento entre frames.

```python
def dadosQuietos(previousFrameCentroids, currentFrameCentroids, threshold):
    '''Detecta si los dados ya están quietos dados dos conjunto de centroides y un umbral.
    El umbral representa el movimiento máximo que pueden tener los centroides más proximos
    al próximo frame para que se tome como que no hay movimiento'''
    # Primero, chequeamos que en cada frame hay 5 centroides
    if len(previousFrameCentroids) == 5 and len(currentFrameCentroids) == 5:
        # Comparar distancia de los centroides
        # Para esto, iteramos sobre los centroides actuales y buscamos la mínima distancia entre cada centroide anterior
        # Si se cumple la condición de que la distancia es menor que el umbral para alguno de los puntos, se toma como válida
        # la distancia. En el caso de que haya un centroide muy lejos que el resto se termina el programa.
        # Este no es el método más óptimo pero sí el más fácil de entender.
        for currentCentroid in currentFrameCentroids:
            # Hallamos el centroide anterior más cercano usando la función min usando como clave la distancia euclideana
            nearestCentroid = min(previousFrameCentroids, key= lambda c: distanciaEuclideana(currentCentroid, c))
            distance = distanciaEuclideana(nearestCentroid, currentCentroid)
            if distance > threshold:
                # En el caso de que haya un punto muy alejado, todavía los dados no están quietos
                #print(distance)
                return False
        # En el caso de que se cumpla la condición del umbral para todos los centroides, entonces se toma como que se está quieto
        return True
    else:
        # Si no hay 5 centroides en alguno de los dos frames, devolver falso
        return False
```

Imagen 9. Función ‘dadosQuietos’

La última función que definimos es la misma utilizada en el ejercicio 1, **‘buscovalordados’**, la cual determina el valor de la imagen del dado pasado por parámetro mediante un análisis de componentes conectadas. (Imagen 10)

```python
def buscovalordados(image):
  '''Recibe la imagen de un dado y determina su valor'''
  gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
  _, thresh_img = cv2.threshold(gray_image, thresh=128, maxval=255, type=cv2.THRESH_BINARY)  # Umbralamos

  # Hacemos componentes conectadas para separar cada punto del dado
  num_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(thresh_img, connectivity = 8, ltype=cv2.CV_32S)

  labeled_image = np.zeros_like(thresh_img)
  numero_dado = 0
  for i in range(1, num_labels):

      # --- 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)

      if perimeter == 0 :
        continue
      fp = area / perimeter**2

      # Removemos el caso donde fp = 0, es decir, una componente conectada de un solo punto
      if fp == 0:
        continue

      # Identificamos círculos y nos quedamos con aquellos más grandes (filtramos según área)
      if 1/fp > 8 and 1/fp < 17 and stats[i][4] > 15:
        labeled_image[obj == 1] = 255
        # Contamos los puntos
        numero_dado += 1
  return(str(numero_dado))
```

Imagen 10. Función ‘buscovalordados’

En cuanto al código principal, se parte almacenando en una variable el nombre del video que ingresa el usuario (en caso de que no se ejecute el código desde la consola, este paso se puede hacer a mano cargando el video en el entorno) y leyéndolo con cv2. (Imagen 11)

Además, se inicializa el contador ‘**n_frame**’ en 0 y se define una lista vacía ‘**previous_centroids**’ que se utilizará para comparar las distancias entre los centroides.

Luego, a través de un bucle while, se ejecutan las siguientes tareas:

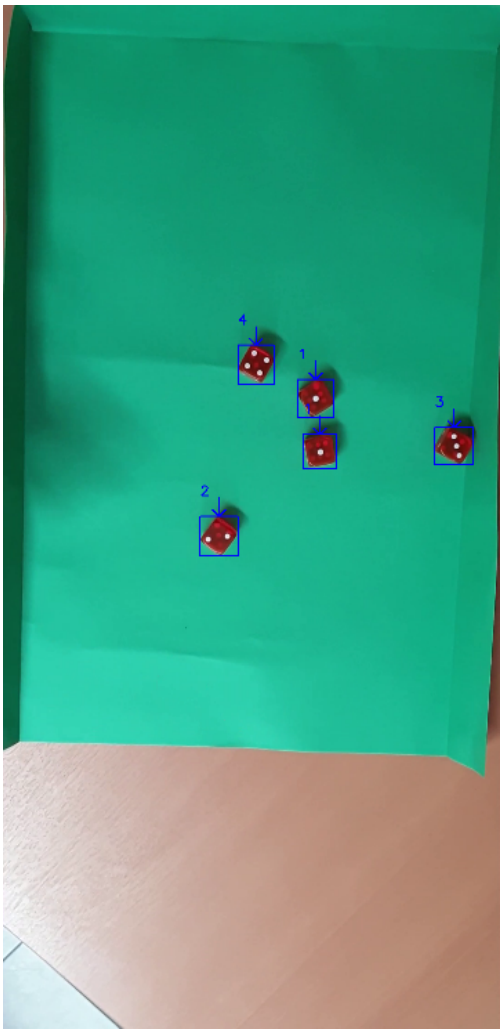
- Se llama a ‘**detectar_dados**’ en el frame actual, almacenando sus resultados en las variables ‘example’, ‘current_centroids’, ‘statss’ y ‘labels’.
- Se llama a la función ‘**dadosQuietos**’ pasándole las listas ‘previous_centroids’ (posiciones de los centroides del frame anterior) y ‘current_centroids’ (posiciones de los centroides del frame actual) junto con un umbral, en este caso, de valor 3. Su resultado se almacena en la variable booleana ‘are_dices_steady’.
- Se declara que ‘**previous_centroids**’ ahora toma el valor de ‘**current_centroids**’ luego de realizar la comparación para la próxima iteración del bucle.

- Si el valor de '**are_dices_stady**' es 'True', entonces se considera que los dados están en reposo y se incrementa el contador 'frames_consecutivos'. De lo contrario, toma el valor 0.
- En el caso de que la cantidad de frames consecutivos sea mayor o igual a 2 (frames_consecutivos >= 2), mediante un for se recorta la imagen de cada componente conectada (cada dado) y se le pasa por parámetro a la función 'buscovalordados' para determinar el valor del mismo. Una vez obtenido el valor, se escribe sobre el frame un bounding box, una flecha y el número obtenido por el dado de color azul.
- Se graba el frame.

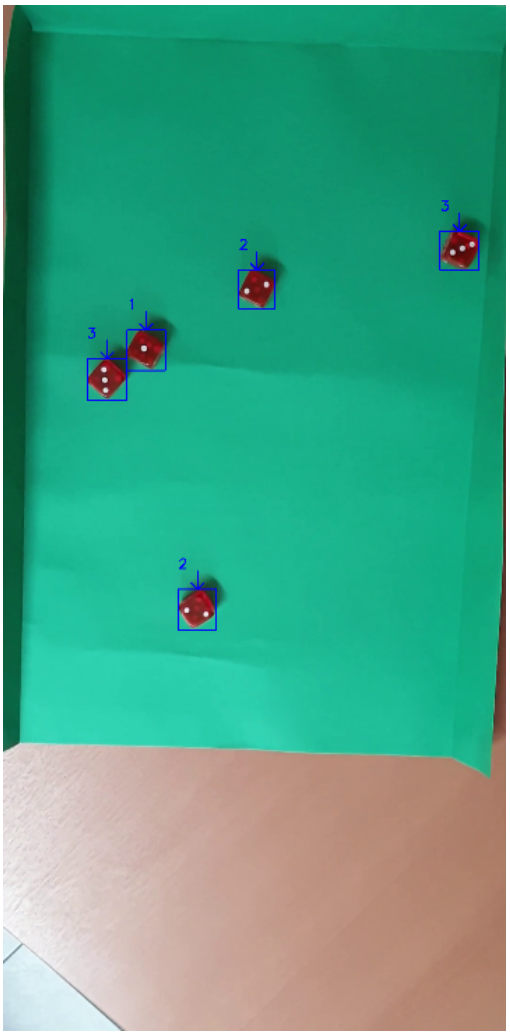
```
video_path = sys.argv[1] +'.mp4' # Video path... Para ejecutar desde la terminal
#Sino ingresando a mano: video_path = 'tirada_1.mp4'
cap = cv2.VideoCapture(video_path)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))
n_frame = 0
out = cv2.VideoWriter('Video-Output.mp4', cv2.VideoWriter_fourcc(*'mp4v'), fps, (width,height))
previous_centroids=[]
while cap.isOpened():
    ret, frame = cap.read()

    if ret == True:
        if n_frame:
            example = frame  # Capturamos un frame de ejemplo
            example, current_centroids, statss, labels = detectar_dados(example)
            #print(len(current_centroids))
            example = cv2.resize(example, dsize=(int(width/3), int(height/3)))

            if n_frame % 1 == 0:
                are_dices_steady = dadosQuietos(previous_centroids, current_centroids, 3)

                # Luego de comparar, declaramos los centroides del frame actual como centroides anteriores para el próximo frame
                previous_centroids = current_centroids

                if are_dices_steady:
                  frames_consecutivos +=1
                else:
                  frames_consecutivos = 0

                if frames_consecutivos >=2:
                    for st in statss:
                      x, y, w, h = st[:4]  # Coordenadas del cuadro delimitador
                      dado = frame[y:y + h, x:x + w]
                      valor = buscovalordados(dado)
                      # print('valor= ',valor)
                      cv2.rectangle(frame, (st[0], st[1]), (st[0]+st[2], st[1]+st[3]),(255,0,0), 2)  # Agrego un rectángulo en cada dado
                      cv2.arrowedLine(frame, (round(st[0]+st[2]/2), round(st[1]-40)), (round(st[0]+st[2]/2), st[1]), (255,0,0), 2, tipLength=0.5)
                      cv2.putText(frame, valor, (st[0], round(st[1]-45)), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0),2)

            # --- Muestro por pantalla ------------
            frame_show = cv2.resize(frame, dsize=(int(width/3), int(height/3)))
            #imshow(frame_show)
            out.write(frame)  # grabo frame --> IMPORTANTE: frame debe tener el mismo tamaño que se definio al crear out.

        n_frame += 1

        if cv2.waitKey(25) & 0xFF == ord('q'):
            break
    else:
        break

cap.release()
out.release()
cv2.destroyAllWindows()
```

Imagen 11. Código principal

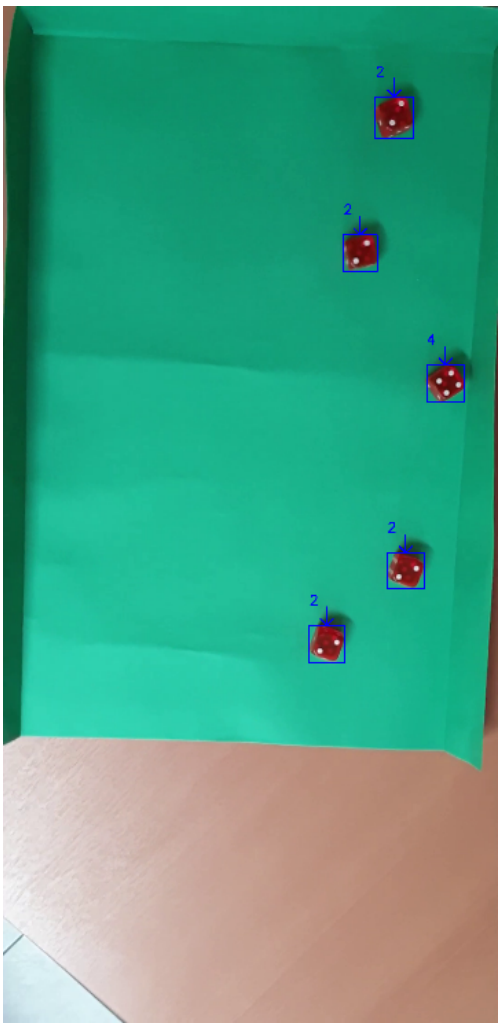
Para concluir, presentamos los resultados obtenidos para uno de los frames de cada una de las cuatro tiradas. Estos resultados incluyen la visualización de bounding boxes alrededor de cada dado, junto con la indicación del valor correspondiente.



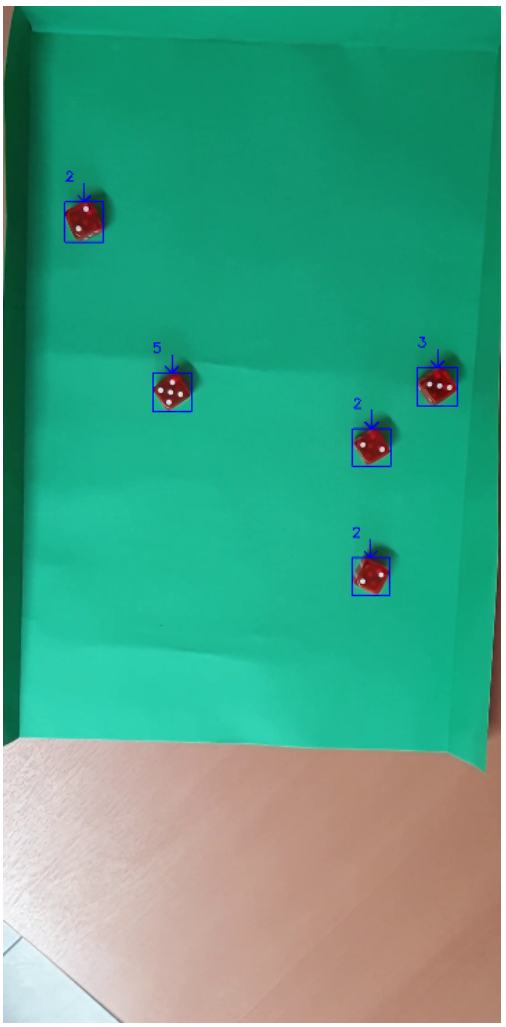

Tirada 1




Tirada 2




Tirada 3




Tirada 4

Aquí finaliza la resolución del ejercicio 2. En la siguiente y última sección, se resaltan las conclusiones derivadas del trabajo.

## CONCLUSIONES FINALES

El presente trabajo práctico consistió en la resolución de dos ejercicios integradores que nos han llevado a aplicar varias técnicas de procesamiento de imágenes y visión por computadora aprendidas a lo largo de toda la asignatura.

La implementación de ambos algoritmos para la detección automática de dados en reposo se basó en un conjunto de estrategias, incluyendo la lectura y el grabado de videos, la conversión de color, umbralización, detección de componentes conectadas y filtros por área y aspecto.

Ambos procesos iniciaron con la lectura de un video en particular (pasado por consola o manualmente) y con la detección de frames con 5 componentes conectadas (los cinco dados), que nos aseguraron que los objetos estén en reposo. La umbralización se utilizó para destacar las regiones de interés, en este caso, los dados en reposo y, posteriormente, la detección de componentes conectadas permitió segmentar los dados individuales y calcular sus características, como centroides y áreas.

El filtrado por área y aspecto se convirtió en un paso crítico para eliminar componentes no deseadas y garantizar que solo se consideraran los dados. Este enfoque proporcionó robustez al algoritmo, asegurando que la detección se enfocara únicamente en los elementos de interés y reduciendo el impacto de obtener otros objetos innecesarios.

En resumen, este trabajo nos ayudó a reforzar algunas de las herramientas más importantes aprendidas a lo largo del cuatrimestre, ofreciéndonos una experiencia valiosa en la resolución de problemas en el ámbito del procesamiento de imágenes.