Tecnicatura Universitaria en Inteligencia Artificial

Procesamiento de Imágenes 1 (IA4.4)

Trabajo Práctico N° 3

Docentes:

- Álvarez, Julián
- Reyes, Facundo
- Sad, Gonzalo

Grupo: 3

Integrantes:

- Aguirre, Fabian A-4516/1
- Fontela, Facundo F-3724/9
- Wagner, Juan W-0557/6

Año: 2023

PROBLEMA 1 – Cinco dados

Las secuencias de vídeo 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.

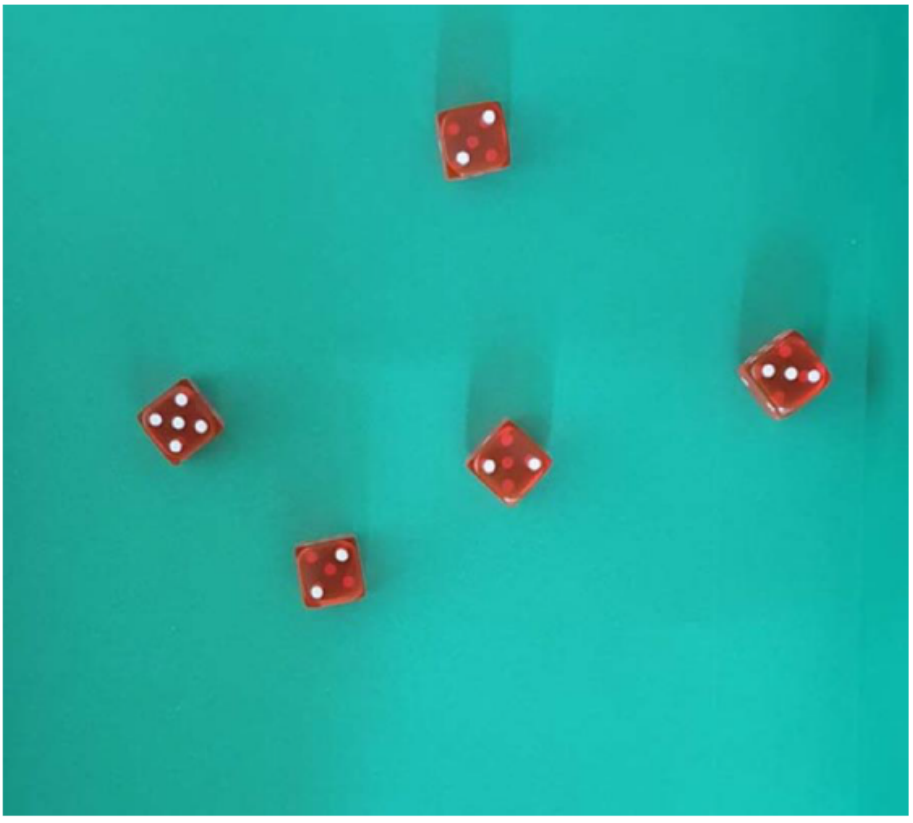


Figura 1 – Dados luego de una tirada.

Observaciones Generales:

Se observaron los diferentes videos para identificar estrategias para abordar la problemática. Se observó que todos los videos cuentan con un fondo verde y que además todos los dados son identicos, transparentes de color rojo y marcas blancas, lo cual nos facilitará la tarea de segmentación de los dados.

A su vez observamos que la cámara de todo los videos es estática lo cual nos facilita la tarea de detección del término del movimiento de los dados.

A continuación procedemos a la resolución utilizando Python y apalancándonos de la librería OpenCV para el manejo de Imagenes/Videos.

Paso 1: Inicio

```
if __name__ == "__main__":
    assert len(sys.argv) > 1, "INGRESE EL ARCHIVO!"
    main(sys.argv[-1])
```

Al ejecutar el programa se espera que por parámetro de se la pase el path del video a analizar. El programa se ejecuta al final del script cuando se cumple la condición `if __name__ == "__main__"`, y se asegura de que se proporcione un archivo de video como argumento de línea de comando, si no devuelve el mensaje **`"INGRESE EL ARCHIVO!"`**

Paso 2: Ejecución del Programa

```python
# Main: leer y grabar video
def main(path):
  cap = cv2.VideoCapture(f'{path}.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'{path}_out.mp4', cv2.VideoWriter_fourcc(*'mp4v'), fps, (int(width/3),int(height/3)))

  ret, prev_frame = cap.read()
  ret, current_frame = cap.read()

  while ret:
      diff = calculate_frame_diff(prev_frame, current_frame)

      frame = cv2.resize(current_frame, dsize=(int(width/3), int(height/3)))
      if cv2.countNonZero(diff) < 2000:
          draw_dados(frame)

      out.write(frame)
      prev_frame = current_frame
      ret, current_frame = cap.read()

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

- La función principal (`main`) se encarga de abrir el archivo de vídeo especificado como argumento de línea de comandos.
- Luego obtiene las dimensiones y la velocidad de cuadros (fps) del video.
- Se leen los 2 primeros frames del video.
- Después en el bucle se calcula la diferencia entre el frame actual y el frame anterior usando *calculate_frame_diff* (El funcionamiento de esta función se explica con más detalles más adelante).
- Se redimensiona el frame actual para el video resultante.
- Experimentalmente se considera que si la cantidad de píxeles diferentes es menor que 2000, los dados están en reposo.
- Se llama a la función para dibujar y reconocer los dados en reposo *draw_dados*.
- Escribe el frame procesado en el video resultante y vuelve a hacer el bucle.
- El video resultante se guarda con el nombre de archivo *path_out.mp4*.

Paso 3: Detección de Dados

```python
# Buscar Dados, dibujar rectangulo. Buscar numero de puntos
def draw_dados(frame):
    mask = filtrar_rojos(frame)
    mask = dilate_img(mask)
    img = cv2.bitwise_and(frame, frame,  mask=mask)
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    (
        componentes_conectadas_dados,
        etiquetas_dados,
        stats,
        _,
    ) = cv2.connectedComponentsWithStats(mask, cv2.CV_32S, connectivity=8)
    print(componentes_conectadas_dados)
    if(componentes_conectadas_dados != 6 ):
      return;

    for i in range(1, componentes_conectadas_dados):
      mascara = np.uint8(etiquetas_dados == i)
      dado = cv2.bitwise_and(img_gray, img_gray, mask=mascara)
      _, puntos_blancos = cv2.threshold(dado, 190, 255, cv2.THRESH_BINARY)
      puntos_blancos = dilate_img(puntos_blancos, soft=True)

      count_puntos = 0
      ( componentes_puntos, etiquetas_puntos, estadisticas, _, ) = cv2.connectedComponentsWithStats(puntos_blancos, cv2.CV_32S, connectivity=8)
      for j in range(1, componentes_puntos):
        mascara_punto = np.uint8(etiquetas_puntos == j)
        area = estadisticas[j, cv2.CC_STAT_AREA]
        if area > 10:
          count_puntos +=1
      if (count_puntos == 0):
        return;

      x = stats[i, cv2.CC_STAT_LEFT]
      y = stats[i, cv2.CC_STAT_TOP]
      w = stats[i, cv2.CC_STAT_WIDTH]
      h = stats[i, cv2.CC_STAT_HEIGHT]
      cv2.rectangle(frame, (x, y), (x+w,y+h), 150, 1)
      draw_number(frame, (x+w, y+h), count_puntos)
```

Dentro de la función *draw_dados* se realiza lo siguiente:

- Se comienza filtrando los dados sabiendo que son rojos. Se utiliza la función *filtrar_rojos* para identificar los dados rojos en el frame mediante el espacio de color HSV. (El funcionamiento de esta función se explica con más detalles más adelante)
- Se aplica dilatación a la máscara resultante para mejorar la detección y se realiza una operación de bitwise para obtener la región de interés de los dados en el frame original (El funcionamiento de la función de dilatación se explica con más detalles más adelante)
- Se convierte la imagen a escala de grises y luego se utiliza la función *connectedComponentsWithStats* para encontrar componentes conectadas en la máscara de dados.
- Son 5 dados por lo que si hay un número de componentes distinto de 6 se sale de la función.
- Se itera sobre cada componente conectada, que representa un dado. Se dibuja un rectángulo alrededor de cada dado en el frame original.
- Se realiza un procesamiento adicional para contar los puntos en cada dado. Se crea una máscara para el dado actual, se extraen los puntos blancos y se aplica umbral y dilatación suave para mejorar la detección.
- Se cuenta la cantidad de puntos blancos en el dado. Si la cuenta da cero se sale de la función, si no se dibuja el rectángulo y el número sobre el dado en

el frame original utilizando la función *draw_number*. (El funcionamiento de esta función se explica con más detalles más adelante)

Función ***culculate_frame_diff***:

```python
# Funcion para detectar cambios en 2 frames de un video
def calculate_frame_diff(prev_frame, current_frame):
    diff_frame = cv2.absdiff(prev_frame, current_frame)
    gray_diff_frame = cv2.cvtColor(diff_frame, cv2.COLOR_BGR2GRAY)
    _, threshold_diff = cv2.threshold(gray_diff_frame, 30, 255, cv2.THRESH_BINARY)
    return threshold_diff
```

La función *calculate_frame_diff* compara dos frames consecutivos para detectar cambios significativos. Si la cantidad de píxeles diferentes es menor que un umbral, se asume que los dados están en reposo y se llama a la función *draw_dados* para resaltar los dados en el video.

- Primero se calcula la diferencia absoluta entre el frame actual y el frame anterior
- Luego se convierte la diferencia a escala de grises
- Y luego se aplica un umbral para obtener una imagen binaria que destaque las áreas donde hay cambios

Función ***filtrar_rojos***:

```python
def filtrar_rojos(img):
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    return cv2.inRange(hsv, lower_rojo, upper_rojo)
```

Se define un rango de colores en el espacio de color HSV (*lower_rojo* y *upper_rojo*) para filtrar los dados rojos en la imagen. Se utiliza la función filtrar_rojos para obtener una máscara que destaque los dados rojos.

- Se definen los valores lower_rojo y upper_rojo para filtrar los píxeles de color rojo en el espacio de color HSV. Estos valores corresponden a un rango de tonalidades de rojo.

- La función filtrar_rojos toma una imagen (img) y realiza las siguientes operaciones:
    - Convierte la imagen de BGR a HSV.
    - Utiliza cv2.inRange para crear una máscara que retiene solo los píxeles dentro del rango de color definido para el rojo.

Función ***dilate_img***:

```python
import cv2
import numpy as np
import sys

# Dilatacion
kernel_dilatacion = np.ones((9, 9), np.uint8)
kernel_dilatacion_soft = np.ones((3, 3), np.uint8)

def dilate_img(img, soft=False):
  kernel = kernel_dilatacion_soft if soft else kernel_dilatacion
  return cv2.dilate(img, kernel)
```

Se proporcionan dos kernels diferentes (*kernel_dilatacion* y *kernel_dilatacion_soft*) para diferentes propósitos, dependiendo del parametro *soft*. La dilatación se aplica a la máscara de la imagen para mejorar la detección de los dados.

- Se definen dos kernels, *kernel_dilatacion* y kernel_dilatacion_soft, para operaciones de dilatación. *kernel_dilatacion* se utiliza para dilatación estándar, y *kernel_dilatacion_soft* se utiliza para dilatación suave.
- La función *dilate_img* toma una imagen (img) y un parámetro opcional soft. Si soft es True, se utiliza el kernel suave; de lo contrario, se utiliza el kernel estándar. La operación de dilatación se aplica a la imagen con el kernel seleccionado.

Función ***draw_number***:

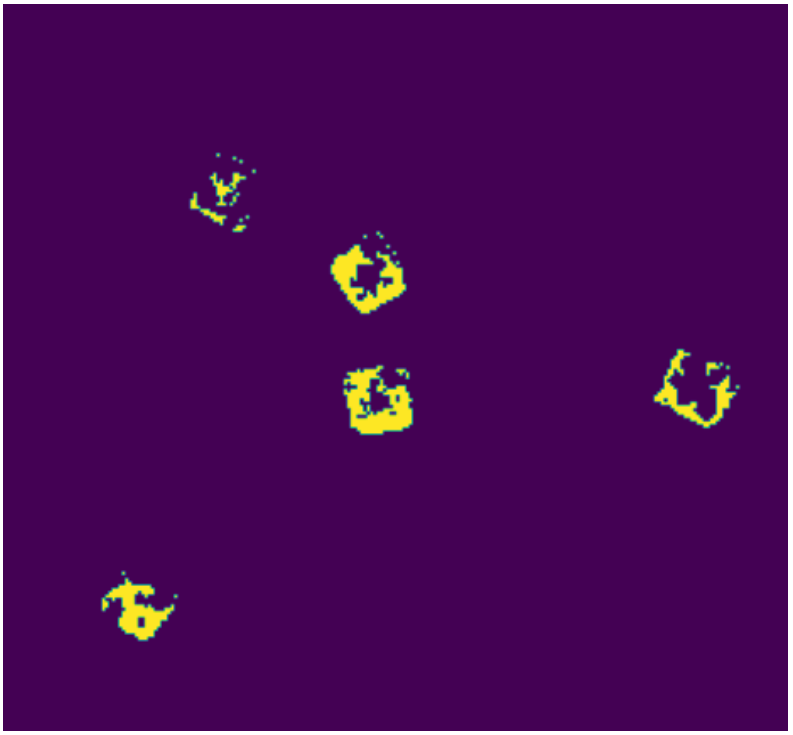
```python
# Dibujar numeros en frame
font                   = cv2.FONT_HERSHEY_SIMPLEX
fontColor              = (255,255,255)
fontScale              = 0.5
thickness              = 1
lineType               = 2

def draw_number(frame, position, number):
  cv2.putText(frame, f'{number}',
        position,
        font,
        fontScale,
        fontColor,
        thickness,
        lineType
  )
```

Se filtran los puntos pequeños y se utiliza la función *draw_number* para agregar el número de puntos al video.

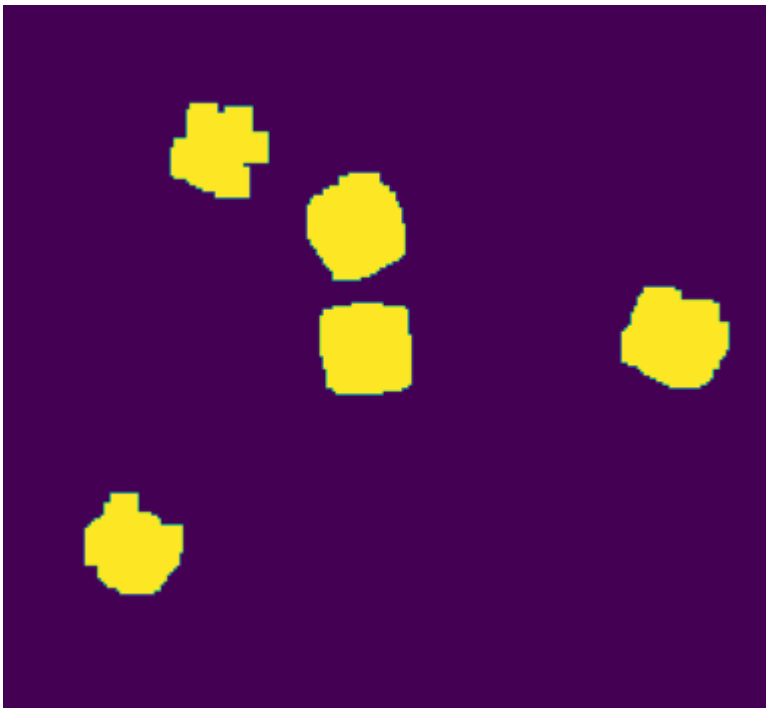
- Se configura la Fuente y Estilo de Texto que luego se mostrará en el frame.
- La función *draw_number* toma el frame, la posición (x, y) donde se dibujará el número, y el número a dibujar.
- Utiliza cv2.putText para agregar el número al frame.
- cv2.putText recibe varios parámetros, incluyendo la imagen de destino (frame), el texto a dibujar (f'{number}'), la posición (x, y), la fuente (font), la escala del texto (fontScale), el color (fontColor), el grosor de las líneas (thickness), y el tipo de línea (lineType).
- De esta manera, la función agrega el número reconocido sobre los dados en el video resultante.

Procesamiento de las imágenes:

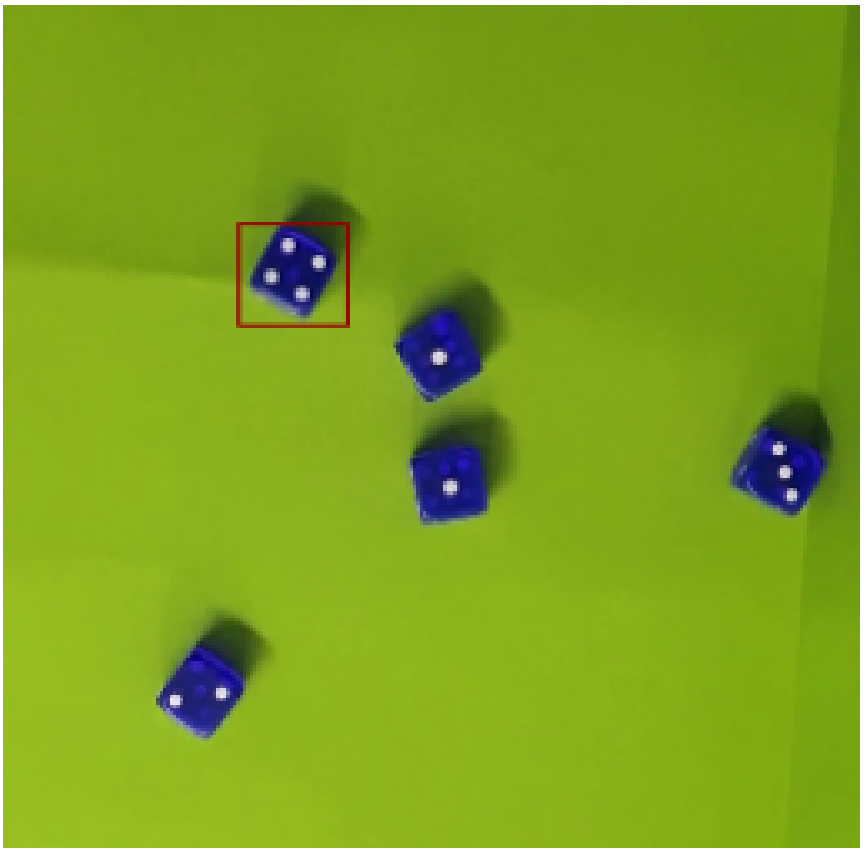
Después de aplicar el filtro de rojos:



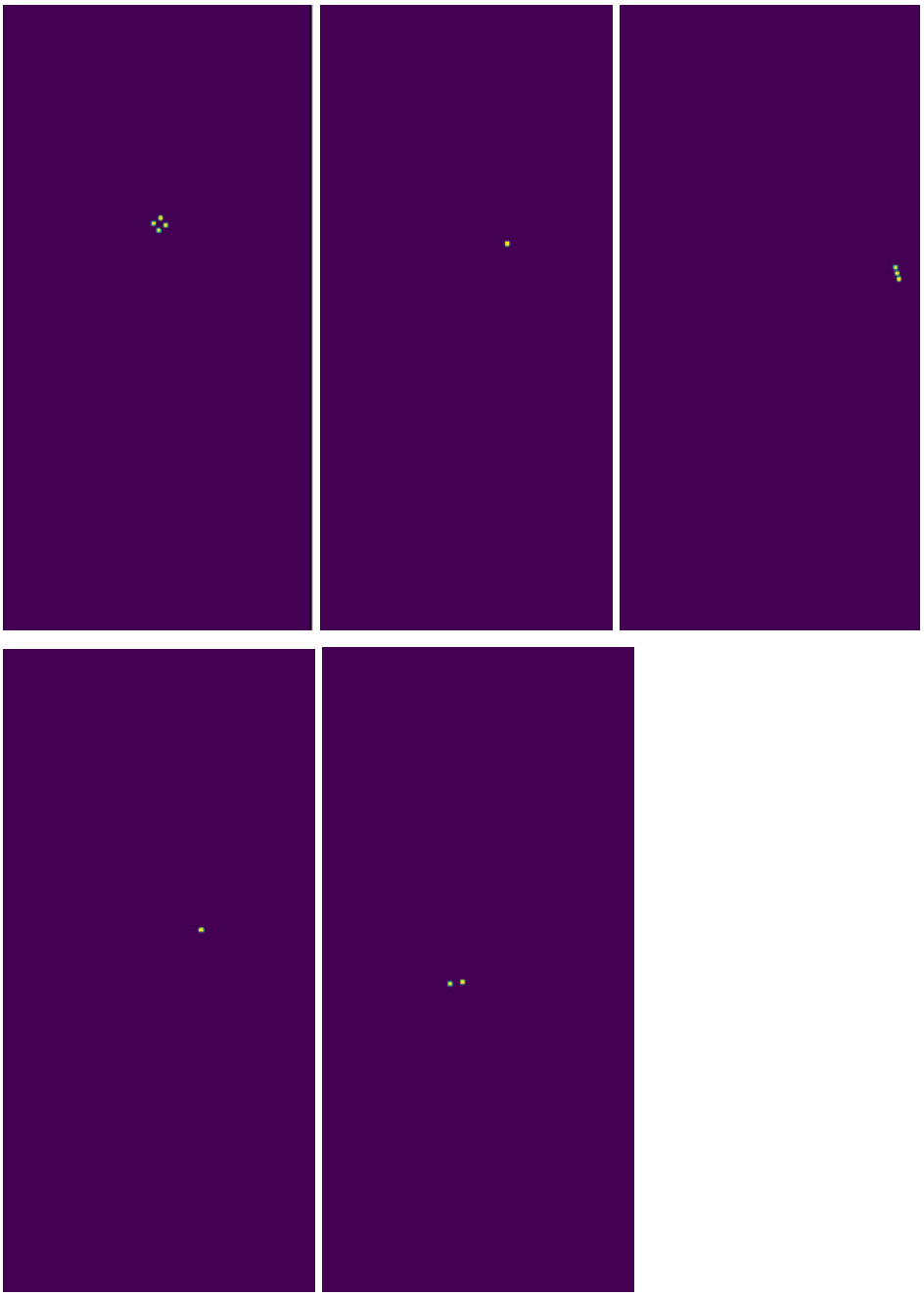
Después de aplicar dilatación:



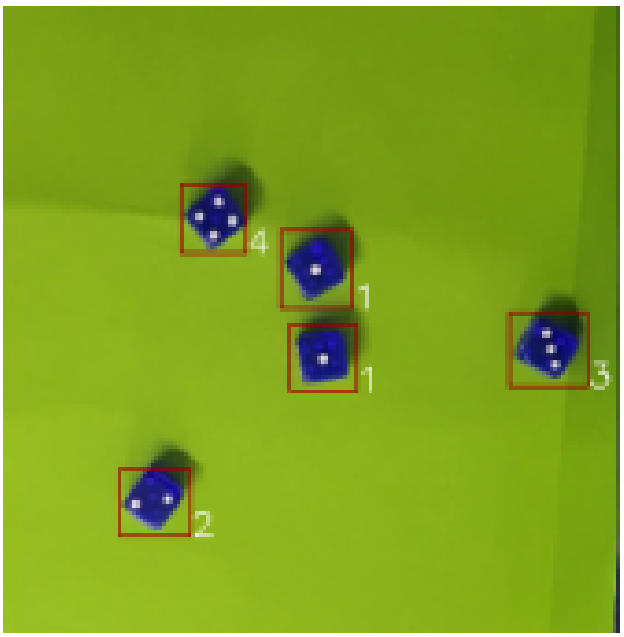
Después de detectar las componentes conectadas y colocar el rectángulo:



Después de detectar los puntos blancos y dilatarlos:

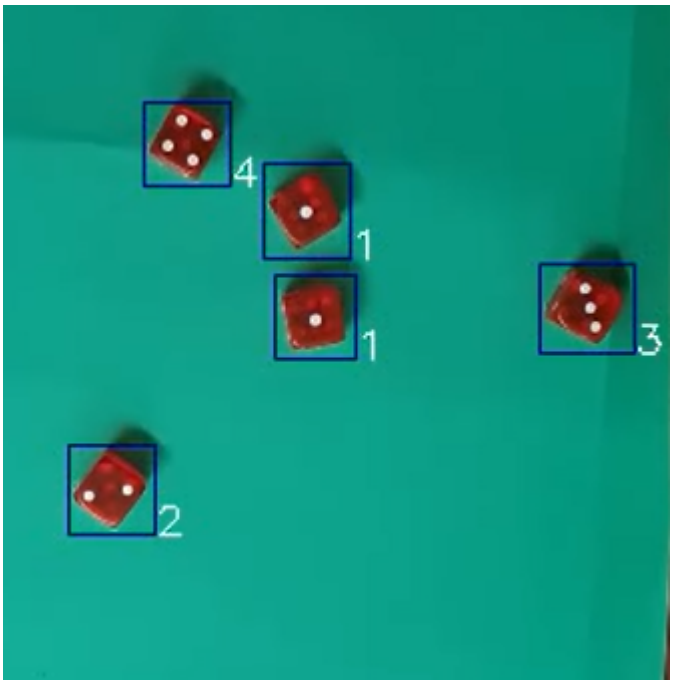


Después de haber contado la cantidad de componentes conectadas y agregarle el número:

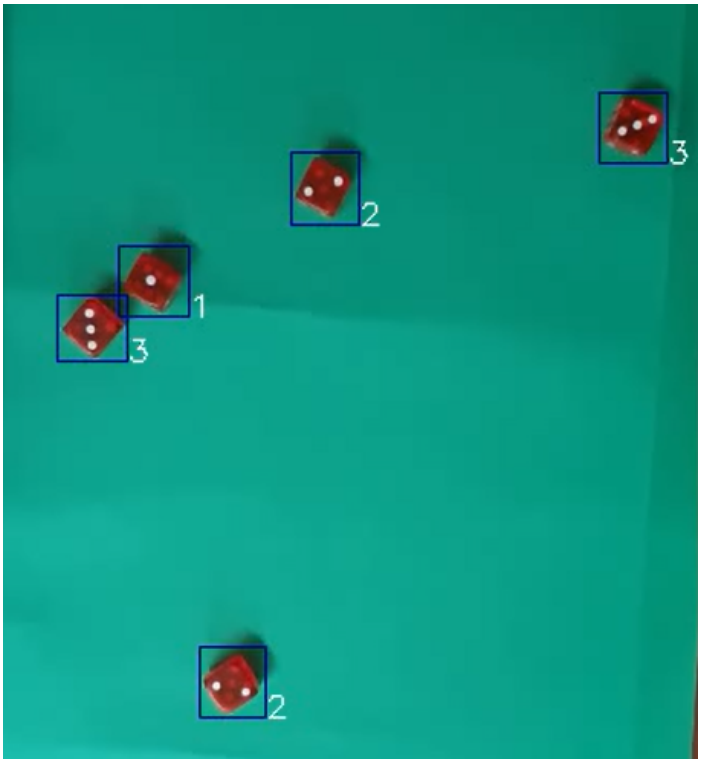


Resultado final de la detección de todos los dados en todas las tiradas

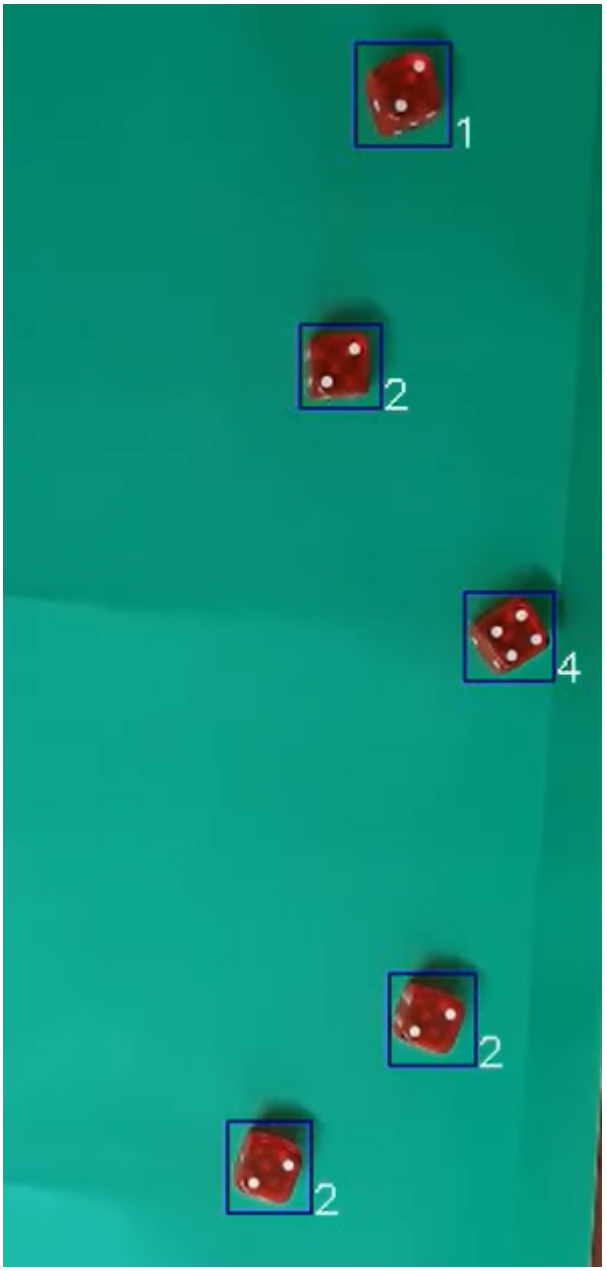
Tirada 1:



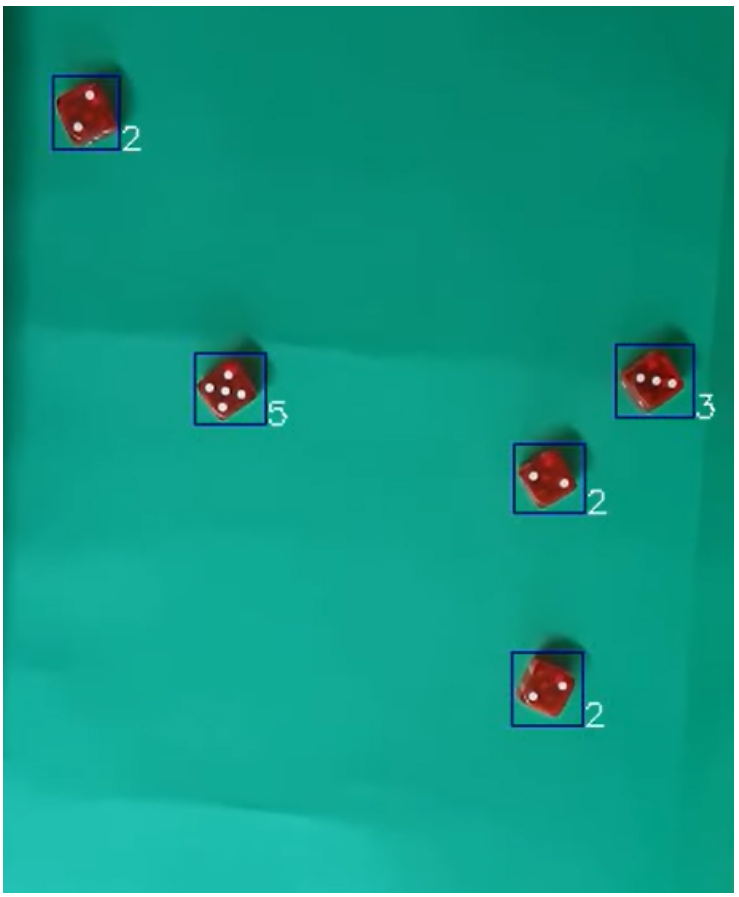
Tirada 2:



Tirada 3:



Tirada 4:



Conclusión General:

El proyecto aborda la tarea de procesar 4 videos de tiradas de dados, detectando automáticamente cuando los dados están en reposo y reconociendo el número obtenido en cada dado. Se ha utilizado la biblioteca OpenCV en Python para implementar algoritmos de procesamiento de imágenes y vídeo. A continuación, se destacan los aspectos clave del proyecto:

Detección de Dados:

- Se implementó un algoritmo que utiliza filtrado de color y operaciones morfológicas para detectar dados rojos en cada frame del video.

Conteo de Puntos en Dados:

- Se aplicó un análisis más detallado en cada dado para contar los puntos blancos, simulando el reconocimiento de los números en los dados.

Detección de Cambios:

- Se implementó una función para detectar cambios significativos entre frames consecutivos, indicando cuando los dados están en reposo.

Generación de Video Resultante:

- Se generaron videos de salida resaltando los dados en reposo con un bounding box azul y mostrando los números reconocidos sobre los dados.

Ajustes y Parametrización:

- El código proporciona parámetros ajustables, como umbrales y tamaños de kernel, que permiten la adaptación del algoritmo a diferentes condiciones de iluminación y resolución de video.

Automatización y Eficiencia:

- El código se estructura de manera que puede procesar automáticamente un video de entrada proporcionado como argumento de línea de comandos.

Reflexión final:

La detección se logró hacer casi a la perfección excepto por el video 2 que hay un frame que no dibuja un recuadro mientras está en movimiento y el video 4 que hay un movimiento del foco de la cámara por lo que se desaparecen los recuadros y vuelven a aparecer.

Desafíos:

- En el proceso de detectar cuando se detienen los datos, la dificultad estuvo en que la imagen no permanece estática. Hay cambios de foco, de iluminación o movimiento.

Optimización y Rendimiento:

- Dependiendo de la calidad y resolución del video, así como de la complejidad de la escena, podría ser necesario optimizar y ajustar los parámetros para lograr un rendimiento óptimo.

Calidad de la imagen:

- Destacamos la importancia de una buena grabación para que la detección sea óptima.