UNIVERSIDAD NACIONAL DE ROSARIO
FACULTAD DE CIENCIAS EXACTAS, INGENIERÍA Y AGRIMENSURA

Trabajo Práctico N° 3

PROCESAMIENTO DIGITAL DE IMÁGENES I

INFORME

Procesamiento de video

Detección y determinación de dados sobre un paño

Carrera: Tecnicatura Universitaria en Inteligencia Artificial

Alumnos: *Demarré, Lucas*

Donnarumma, Cesar Julian

Menescaldi, Brisa

Mussi, Miguel

Ciclo: *2023*

TABLA DE CONTENIDOS

PARTE I: CONSIGNAS

Problema 1 – 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.

Figura 1 – Dados luego de una tirada.

PARTE II: RESOLUCIÓN

Abtract

Este script Python realiza el procesamiento de videos que capturan lanzamientos de dados sobre una superficie. Utiliza la biblioteca OpenCV para llevar a cabo tareas avanzadas de visión por computadora. El código comienza importando las bibliotecas necesarias y definiendo funciones clave para el procesamiento de imágenes, como la reconstrucción de imágenes y la eliminación de elementos en los bordes.

El proceso se estructura en un bucle principal que itera sobre varios videos predefinidos. Para cada video, el script configura la lectura del video original y la escritura del video procesado. Luego, realiza el procesamiento de cada frame del video, segmentando el paño y los dados mediante técnicas de procesamiento de imágenes y analizando la relación de aspecto y área de los componentes conectados. Se implementa un seguimiento de los dados y se detecta si están en movimiento o en reposo. El resultado final se presenta en cada frame con la superposición de los dados y sus respectivos números.

Funcionamiento del código

Detección del paño.

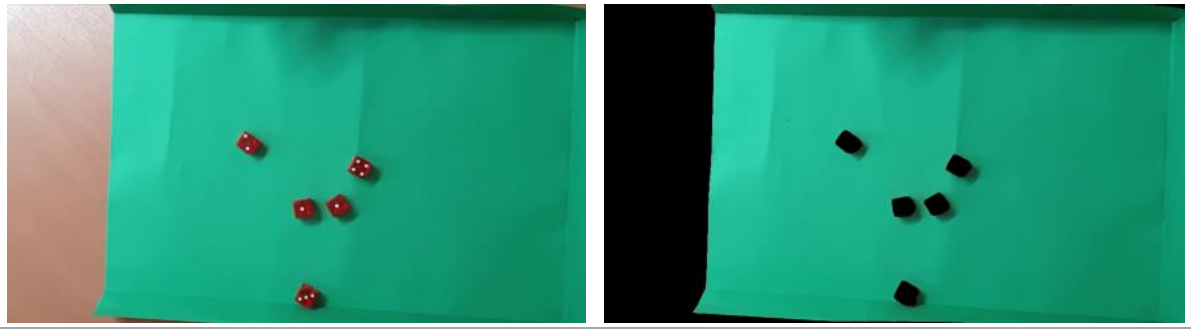
El primer paso para resolver el problema fue detectar donde estaba el paño. El fundamento de este paso es que una vez que detectemos los dados vamos a fijarnos si están sobre el paño. Este paso puede parecer, en principio, innecesario dado que en los videos provistos para el análisis todos los dados siempre caen ahí. De todos modos, se realiza este paso ya que hace no representa una dificultad importante y hace un poco más robusto al algoritmo.

```
# Segmentacion del paño con HSV
frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
# plt.imshow(frame_rgb), plt.show()
img_hsv = cv2.cvtColor(frame_rgb, cv2.COLOR_RGB2HSV)
h, s, v = cv2.split(img_hsv)
ix_h1 = np.logical_and(h > 65, h < 90)
ix_s = np.logical_and(s > 75, s < 255)
ix = np.logical_and(ix_h1, ix_s)

r, g, b = cv2.split(frame)
r[ix != True] = 0
g[ix != True] = 0
b[ix != True] = 0
paño_img = cv2.merge((r, g, b))
# plt.imshow(cv2.cvtColor(paño_img, cv2.COLOR_BGR2RGB), cmap='gray'), plt.show()
```

Lo hicimos a través del espacio de colores HSV estableciendo umbrales para h y s a partir de la observación de las escalas de cada canal que constan en los apuntes ofrecidos por la cátedra.

Partimos de una imagen como la izquierda y luego del procesamiento llegamos a la imagen de la derecha.



```python
paño_gris = cv2.cvtColor(paño_img, cv2.COLOR_BGR2GRAY)
# plt.imshow(frame_rgb, cmap='gray'), plt.show()

# Umbralamos para obtener componentes conectadas, ponemos el valor minimo porque todo lo que
# quedo segmentado en HSV es paño.
paño_umbralada = cv2.threshold(paño_gris, paño_gris.min(), 255, cv2.THRESH_BINARY)[1]
# plt.imshow(paño_gris, cmap='gray'), plt.show()

# Buscamos componentes conectadas para quedarnos solo con los pixels del bounding box del paño
num_labels_paño, labels_paño, stats_paño, centroids_paño = cv2.connectedComponentsWithStats(paño_umbral

# Filtramos (el de mayor area es el paño)
filtro_paño = np.argmax(stats_paño[:, -1])
paño = stats_paño[filtro_paño]

# Calculamos los limites del paño
izquierda_paño = paño[cv2.CC_STAT_LEFT]
derecha_paño = paño[cv2.CC_STAT_LEFT] + paño[cv2.CC_STAT_WIDTH]
arriba_paño = paño[cv2.CC_STAT_TOP]
abajo_paño = paño[cv2.CC_STAT_TOP] + paño[cv2.CC_STAT_HEIGHT]
```

Esto lo hicimos para obtener los valores máximos y mínimos en “x” e “y” donde se ubicaba el paño. Lógicamente, primero se aplicó el preprocesamiento habitual: convertir la imagen a escala de grises, umbralar (el valor del umbral es nivel mínimo de gris resultante de la imagen convertida porque queremos quedarnos con todo lo que hay en la imagen) y buscar las componentes conectadas.

En las estadísticas que se adjunta está la información que nos interesa de cada componente conectada.

```
>>> stats_paño
array([[     0,      0,   1080,   2224,  690805],
       [     0,      0,   1080,   1676, 1711010],
       [   515,    779,      1,      1,       1],
       [   513,    783,      1,      1,       1],
       [   566,    814,      1,      1,       1],
       [   714,    868,      1,      1,       1],
       [   708,    874,      1,      1,       1],
       [   694,    882,      1,      1,       1],
       [   684,    888,      1,      1,       1],
       [  1018,    962,      1,      1,       1],
       [   665,   1002,      1,      1,       1],
       [  1075,   1052,      1,     14,      14],
       [   463,   1111,      1,      1,       1],
       [  1075,   1120,      1,      7,       7],
       [  1075,   1130,      1,      1,       1],
       [  1075,   1133,      1,      2,       2],
       [  1075,   1140,      1,     56,      56],
       [   433,   1151,      1,      1,       1],
       [   504,   1156,      1,      1,       1],
       [   435,   1168,      1,      1,       1],
       [   480,   1188,      1,      1,       1],
       [   467,   1192,      1,      1,       1],
       [   476,   1192,      1,      1,       1],
       [   598,   1619,      1,      1,       1],
       [   932,   1631,      8,      1,       8]], dtype=int32)
```

De todo ese array, en cada frame, lo único que nos interesa es el elemento de mayor área, que es paño en cuestión, así que filtramos buscando el índice que mayor área tenga.

```
>>> paño
array([     0,      0,   1080,   1676, 1711010], dtype=int32)
```

Bien, ahora que tenemos los valores para el bounding box del paño es fácil encontrar los valores mínimos y máximos en “x” e “y”. En “x” el mínimo es la columna 0, en “y” el mínimo es la columna 1, y los máximos se calculan sumándole a los mínimos los valores del ancho y alto del bounding box según corresponda.

Detección de los dados.

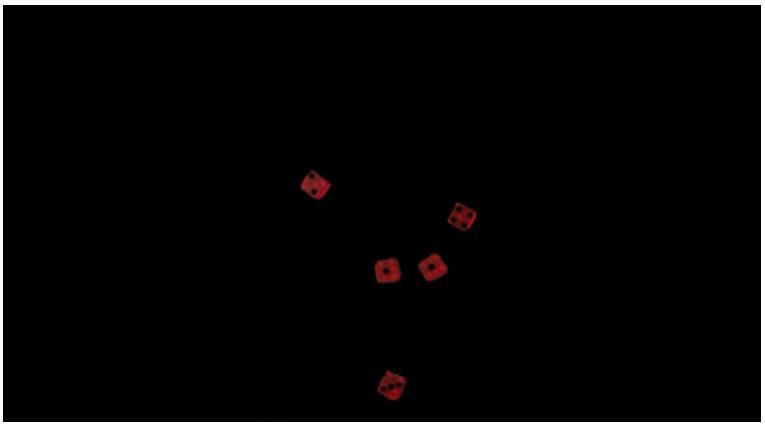
Ahora que ya sabemos cuáles son los límites del paño podemos empezar a buscar los dados.

Aprovechando que los dados son rojos (y prácticamente son lo único rojo que hay cada frame de los videos) los buscamos filtrando por HSV exactamente igual que con el paño.

```
# Filtramos por rojo
ix_h1 = np.logical_and(h > 175, h < 180)
ix_h2 = h < 10
ix_s = np.logical_and(s > 150, s <= 255)
ix = np.logical_and(np.logical_or(ix_h1, ix_h2), ix_s)

r, g, b = cv2.split(frame)
r[ix != True] = 0
g[ix != True] = 0
b[ix != True] = 0
dados = cv2.merge((r, g, b))
# plt.imshow(cv2.cvtColor(dados, cv2.COLOR_BGR2RGB)), plt.show()
```

Pasamos de algo como la primera imagen del informe a este resultado.



Los siguientes pasos son exactamente igual que con el paño ya que nuestro objetivo no es la imagen en si sino segmentar los objetos y obtener determinados datos de los mismos. Se aplica el mismo preprocesamiento anterior: conversión a escala de grises, umbralado y búsqueda de componentes conectadas.

```
# Pasamos la imagen del paño a escala de grises
dados_gris = cv2.cvtColor(dados, cv2.COLOR_BGR2GRAY)
# plt.imshow(dados_gris, cmap='gray'), plt.show()

# Umbralamos para obtener componentes conectadas, nuevamente como filtramos por color rojo
# queremos que nos quede todo lo fitrado (los dados) por eso el umbral es del minimo valor
# de gris en adelante
dados_umbralada = cv2.threshold(dados_gris, dados_gris.min(), 255, cv2.THRESH_BINARY)[1]
# plt.imshow(dados_umbralada, cmap='gray'), plt.show()

# Buscamos componentes conectadas para despues quedarnos con los dados
num_labels_dados, labels_dados, stats_dados, centroids_dados = cv2.connectedComponentsWithStats(dados_u
```

Las estadísticas de las componentes conectadas de los dados también traen muchos elementos que no nos interesan.

```
>>> stats_dados
array([[     0,      0,   1080,   2224, 2385794],
       [   514,    740,     74,     74,    2906],
       [   642,    814,     74,     74,    3317],
       [   662,    826,      1,      1,       1],
       [   685,    885,      1,      1,       1],
       [   668,    889,      1,      1,       1],
       [   670,    889,      1,      1,       1],
       [   671,    891,      5,      1,       5],
       [   943,    918,     77,     74,    3226],
       [   656,    932,     64,     68,    3326],
       [   694,    932,      2,      1,       2],
       [   697,    932,      1,      1,       1],
       [   719,    966,      1,      1,       1],
       [   954,    975,      1,      1,       1],
       [   664,    995,      1,      1,       1],
       [  1078,    999,      2,      7,      12],
       [  1076,   1020,      1,      1,       1],
       [  1078,   1027,      1,      4,       4],
       [  1076,   1028,      1,      4,       4],
       [  1078,   1032,      2,      3,       5],
       [  1076,   1035,      1,      3,       3],
       [  1078,   1049,      1,     20,      20],
       [  1078,   1080,      2,     21,      28],
       [  1078,   1104,      2,     12,      16],
       [   432,   1112,     74,     76,    3196],
       [  1078,   1136,      1,      4,       4],
       [   432,   1158,      1,      1,       1],
       [  1078,   1198,      1,     12,      12],
```

Por eso vamos a quedarnos con los elementos cuyas áreas estén dentro de un intervalo específico, ajustado con cotas determinadas empíricamente. A cada elemento también le vamos a calcular valores mínimos y máximos en “x” e “y” para poder comparar con los límites del paño luego y determinar si están dentro o fuera del mismo. Además vamos calcular el ancho y alto del elemento para filtrar por relación de aspecto en base a determinado intervalo de valores posibles.

```python
indice_dados_sobre_paño = []
for j in range(1, len(stats_dados)):

    # Si cumplen con el intervalo de area aceptable
    if (stats_dados[j, -1] > 2300) & (stats_dados[j, -1] < 4450):

        # Calculamos los valores en 'x' e 'y' donde estan los dados
        izquierda_dado = stats_dados[j, cv2.CC_STAT_LEFT]
        derecha_dado = stats_dados[j, cv2.CC_STAT_LEFT] + stats_dados[j, cv2.CC_STAT_WIDTH]
        arriba_dado = stats_dados[j, cv2.CC_STAT_TOP]
        abajo_dado = stats_dados[j, cv2.CC_STAT_TOP] + stats_dados[j, cv2.CC_STAT_HEIGHT]

        # Calculamos el ancho y alto para relacion de aspecto
        ancho_dado = derecha_dado-izquierda_dado
        alto_dado = abajo_dado-arriba_dado

        # Chequeamos que efectivamente este sobre el paño, si esta sobre el paño:
        if (izquierda_dado  >= izquierda_paño and izquierda_dado  <= derecha_paño) and \
            (derecha_dado >= izquierda_paño and derecha_dado <= derecha_paño) and \
            (arriba_dado >= arriba_paño and arriba_dado <= abajo_paño) and \
            (abajo_dado >= arriba_paño and abajo_dado <= abajo_paño):

            # Y ademas chequeamos que tenga cierta relacion de aspecto
            if (alto_dado/ancho_dado > 0) and (alto_dado/ancho_dado < 10):

                # Añadimos a la lista que de indices de elementos con los que nos vamos a quedar:
                indice_dados_sobre_paño.append(j)
```

La secuencia de verificación es la siguiente: recorrer elemento a elemento y comprobar si el elemento tiene el área que determinamos, ver si está adentro del paño y verificar si la relación de aspecto está dentro de lo que esperamos. Si todo esto se confirma, en ese caso es un dado y se agrega a la lista.

Detección del movimiento de los dados.

Ahora que ya tenemos detectados los dados que están sobre el paño, entraríamos en la etapa siguiente que es detectar cuando se detuvieron. Para hacer esto vamos a tener que comparar los centroides de los dados del frame actual con los del frame anterior.

```
# Esto cobra sentido a partir del segundo frame (de c=1 en adelante)
if c != 0:
    dados_anterior = dados_actual
    centroides_anterior = centroides_actual

# Filtramos los stats con los elementos que nos quedamos
dados_actual = stats_dados[indice_dados_sobre_paño]

# Y tambien filtramos los centroides con los elementos que nos quedamos
centroides_actual = centroids_dados[indice_dados_sobre_paño]
```

Claramente en el primer frame no se realizará comparación alguna ya que no existen los dados del frame anterior.

```
    # Esto tiene sentido a partir de la segunda iteracion (c=1)
    if (c!= 0) and (len(dados_actual) == len(dados_anterior)) and (len(dados_actual)==5):

        # Entonces calculamos la distancia euclidea
        diferencia = centroides_anterior - centroides_actual
        distancia = np.sqrt(np.sum(diferencia**2, axis=1)).reshape(-1,1)

        # Si esta distancia es menor a determinado umbral en todo el array significa que los dados
        # estuvieron quietos
        if all(distancia < 5):
            # Entonces llevamos un frame con los dados en ese estado (quietos)
            contador_frames_quietos+=1
        # Caso contrario re-incializamos el contador para empezar de 0 la proxima
        # que se cumplan las condiciones
        else:
            contador_frames_quietos=0
            numeros_dados = []
```

Entonces si estamos en un frame del video distinto del primero, tenemos 5 dados detectados en este frame y 5 dados detectados en el anterior. Calculamos la distancia euclídea entre ellos. Lógicamente solo podremos calcularla en el caso de tengamos la misma cantidad de dados que en el anterior frame.

Si nos mantenemos dentro de un rango de distancia en todos los dados consideramos que este frame tiene los dados quietos. En el caso que esto no pase, reiniciamos el contador de frames quietos para ver qué pasa el próximo.

Localización de los bounding box.

Una vez asegurado el reposo de los dados, si llevamos un número determinado de frames sin que se muevan, marcamos en cada uno de ellos un rectángulo utilizando los datos que teníamos guardados de los bounding box.

```
    # Si los dados estuvieron quietos mas de determinada cantidad de frames
    if contador_frames_quietos > 5:

        # Recorremos las estadisticas de los dados detectados
        for i in range(len(dados_actual)):
            # Dibujamos el rectangulo en cada uno
            resultado_final = cv2.rectangle(resultado_final, \
                                (dados_actual[i, cv2.CC_STAT_LEFT], dados_actual[i, cv2.CC_STAT_TOP]),
                                (dados_actual[i, cv2.CC_STAT_LEFT]+dados_actual[i, cv2.CC_STAT_WIDTH] ,
                                dados_actual[i, cv2.CC_STAT_TOP]+dados_actual[i, cv2.CC_STAT_HEIGHT]), \
                                (0, 0, 255), 4)
```

Determinación del número de los dados.

Solo resta un paso que es contar el número que tiene cada dado en su cara superior. Esto lo vamos a hacer con la imagen del primer frame que tenga los rectángulos ya marcados.

```
if contador_frames_quietos == 6:

    # Nos quedamos con el boundig-box del dado en cuestion
    recorte_dado = frame[dados_actual[i, cv2.CC_STAT_TOP]: \
                         dados_actual[i, cv2.CC_STAT_TOP]+dados_actual[i, cv2.CC_STAT_HEIGHT], \
                         dados_actual[i, cv2.CC_STAT_LEFT]:dados_actual[i, \
                         cv2.CC_STAT_LEFT]+dados_actual[i, cv2.CC_STAT_WIDTH]]
    # plt.imshow(recorte_dado), plt.show()
    # Pasamos a RGB
    recorte_dado_rgb = cv2.cvtColor(recorte_dado, cv2.COLOR_BGR2RGB)
    # plt.imshow(recorte_dado_rgb), plt.show()
    # Pasamos a escala de grises
    recorte_dado_gris = cv2.cvtColor(recorte_dado_rgb, cv2.COLOR_BGR2GRAY)
    # plt.imshow(recorte_dado_gris, cmap='gray'), plt.show()

    # Umbralamos para quedarnos con los circulos
    recorte_dado_umbralada = cv2.threshold(recorte_dado_gris, 85, 255, cv2.THRESH_BINARY)[1]
    # plt.imshow(recorte_dado_umbralada, cmap='gray'), plt.show()

    # Eliminamos todo lo que toque los bordes
    recorte_dado_bordes = imclearborder(recorte_dado_umbralada)
    # plt.imshow(recorte_dado_bordes, cmap='gray'), plt.show()

    # Componentes conectadas
    num_labels_recorte_dado, labels_recorte_dado, stats_recorte_dado, \
        centroids_recorte_dado = cv2.connectedComponentsWithStats(recorte_dado_bordes)
    # print(stats_recorte_dado)

    # Filtro por area
    filtro_recorte_dado = (stats_recorte_dado[:, -1] > 55) & (stats_recorte_dado[:, -1] < 180)

    # Calculamos el numero con la longitud del array filtrado con los stats
    numero_dado = len(stats_recorte_dado[filtro_recorte_dado])

    # Guardamos los numeros por unica vez en la lista
    numeros_dados.append(numero_dado)
```

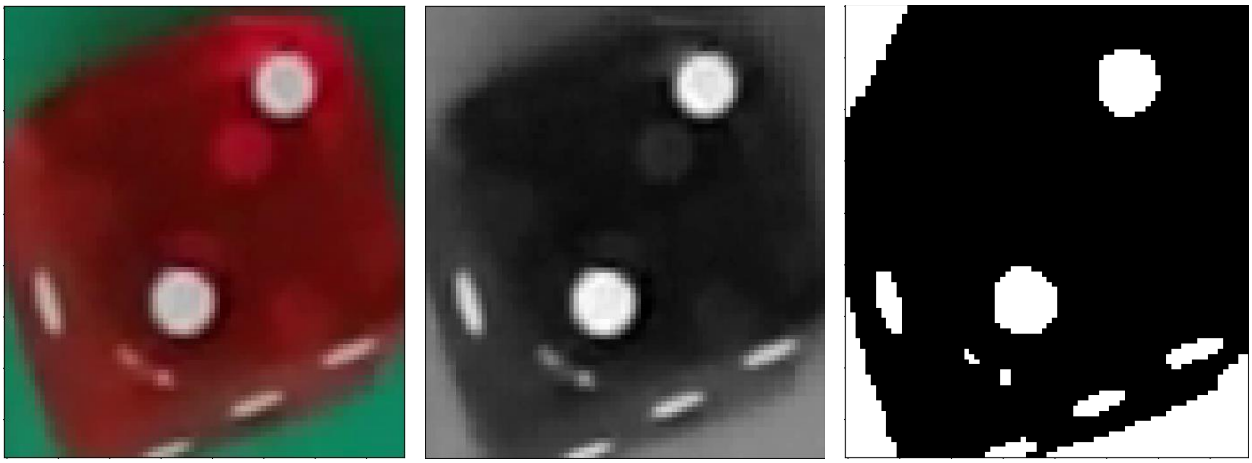
A este proceso de detección intentamos hacerlo dinámico y que se calcule en cada imagen pero nos encontramos con dificultades, dado que aunque en muchas oportunidades los dados están quietos y la imagen se ve igual, en las matrices hay cambios que afectan en el resultado.

Además, otro inconveniente surge cuando la mano se acerca a juntar los dados del paño ya que la sombra también cambia las imágenes. Esto hace muy difícil encontrar umbrales adecuados, que funcionen de manera universal para todos los pasos y en cada uno de los cuatro videos.

Podemos justificar, también, la decisión de aplicarlo del modo en que lo hicimos en el primer frame, en el hecho de que los dados, cuando hacemos los chequeos de que estén en reposo, mientras estén en ese estado, la cara superior siempre va a tener el mismo número.

En el primer paso hacemos cropping sobre el frame con los datos aportados por las estadísticas del dado. Luego pasamos a escala de grises y umbralamos.

Se muestra la evolución de la imagen original y cada uno de los resultados de esos procesos.



Nos quedó una imagen con bastante más elementos que los números en sí: el paño, los números laterales y algún que otro brillo.

En primer lugar nos deshicimos de todos los elementos que toquen el borde. A la imagen resultante le buscamos los componentes conectados y filtramos por área.

```
>>> stats_recorte_dado[filtro_recorte_dado]
array([[ 48,  32,  12,  12, 120],
       [ 10,  50,  13,  13, 126]], dtype=int32)
```

Luego de eso la longitud de ese array es el número de nuestro dado y lo guardamos en una lista. Esta lista tendrá los números de nuestros dados mientras los dados continúen quietos, una vez que se muevan la lista se inicializará vacía nuevamente junto con el contador de frames quietos.

Escritura del número de los dados.

Una vez determinado el número que se encuentra en la cara superior de cada dado, sólo resta escribir arriba del bounding box de los dados el número del dado en cuestión.

```
    # A partir de ahora en cada iteracion escribimos el numero correspondiente al dado
    # i que esta guardada en la posicion i de la lista anterior
    resultado_final = cv2.putText(resultado_final, f'{str(numeros_dados[i])}', \
            (dados_actual[i, cv2.CC_STAT_LEFT]+15,dados_actual[i, cv2.CC_STAT_TOP]-15), \
            cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 0, 0), 3)
    # plt.imshow(resultado_final), plt.show()
```

Ejemplo de ejecución y salida

El algoritmo puede correrse desde Visual Studio (o similar), o directamente desde un terminal para procesar los cuatro videos y obtener los resultados solicitados por la consigna.

Dependencias: para la correcta ejecución del script se debe tener instaladas las librerías necesarias. Pueden instalarse ejecutando:

pip install opencv-python matplotlib numpy.

Script: el archivo .py debe estar en la misma carpeta que los videos provistos para el análisis.

Ejecución desde Visual Studio Code: se puede abrir y ejecutar el script desde VS Code asegurando tener la extensión de Python instalada y configurado correctamente el intérprete.

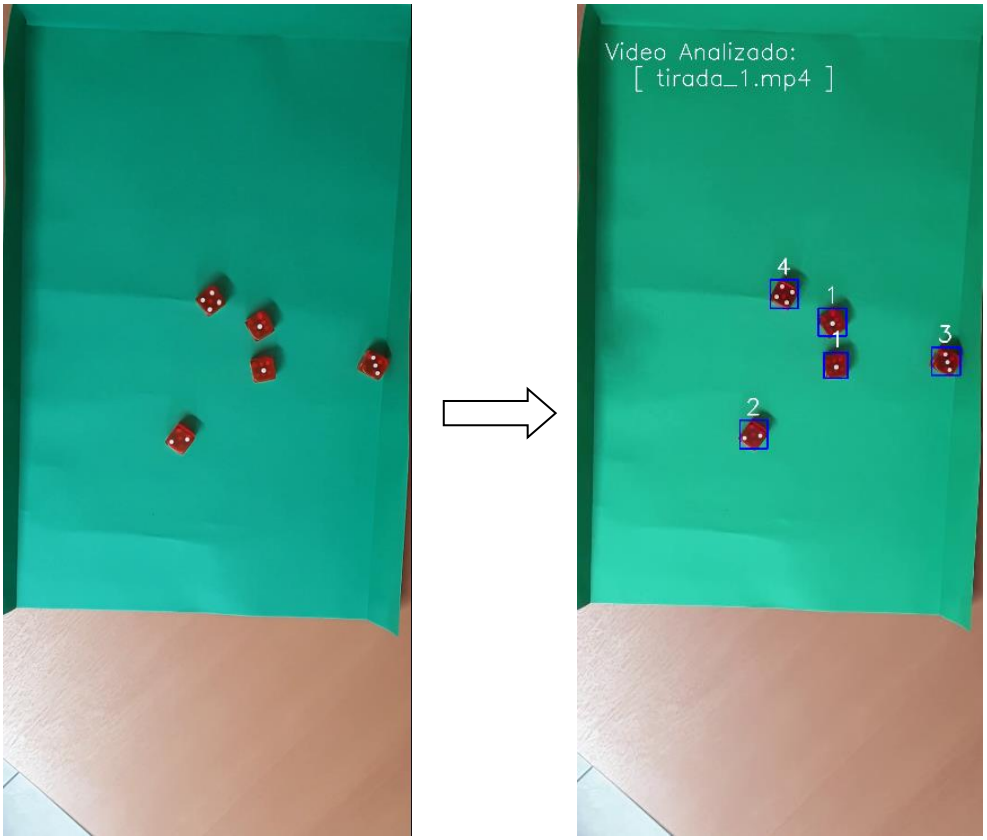
Ejecución desde un terminal: se puede ejecutar el script directamente abriendo un terminal en la carpeta donde se encuentra el script y ejecutando el siguiente comando:

python ejercicio_1.py

La ejecución correcta del script generará la visualización de los cuatro videos (*tirada_1.mp4, tirada_2.mp4, tirada_3.mp4* y *tirada_4.mp4)* y creará en el mismo directorio los videos con el procesamiento solicitado (*resolucion_tirada_1.mp4, resolucion_tirada_2.mp4, resolucion_tirada_3.mp4* y *resolucion_tirada_4.mp4).*

Observación: dentro del script se encuentra declarada una variable que hace referencia al directorio local en el que se encuentran los videos a procesar y en el que se guardarán los videos procesados con la extensión .mp4.

Ejemplo de ejecución:



PARTE III: REPOSITORIO

GitHub

Todos los archivos correspondientes a la resolución de este Trabajo Práctico se encuentran alojados en un repositorio público en GitHub.

https://github.com/MiguelMussi/PDI_TP3