Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Tecnicatura
Universitaria en Inteligencia Artificial
Procesamiento de Imágenes I - IA 4.4

TRABAJO PRÁCTICO N°3 - Año 2024 - 2° Semestre

Alumnos:
Antuña, Franco A-4637/1
Gallardo, Jonatan G-5970/6
Orazi, Roberto O-1815/5

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 esten en reposo, aparezcan resaltados en un bounding box de color azul y ademas, agregar sobre los mismos el numero reconocido.

Figura 1 – Dados luego de una tirada.

Descripción del problema

El objetivo principal de este proyecto es diseñar e implementar un algoritmo capaz de procesar secuencias de vídeo en formato MP4 para analizar el estado de movimiento de los dados. Específicamente, se busca determinar si los dados están en reposo o en movimiento, identificar sus ubicaciones dentro del cuadro y reconocer el número que presentan en la cara superior una vez que se encuentran en reposo.

Durante el desarrollo de la solución, se han identificado varios desafíos técnicos que complican el procesamiento preciso de los videos:

- Desenfoque de imagen: Cuando los dados están en movimiento, las imágenes capturadas presentan desenfoque, lo que dificulta la identificación precisa de los bordes y las características de los objetos.
- Interrupción por la mano: La mano de la persona que lanza los dados frecuentemente cruza el campo de visión, lo que genera oclusiones temporales que pueden interferir con la detección de los dados.
- Variaciones en la intensidad de luz: Cambios en las condiciones de iluminación, como sombras dinámicas o reflejos en los dados, pueden afectar la calidad de la imagen y dificultar tanto la segmentación como el reconocimiento de los números.

Implementación

Se decidió implementar una solución basada en el uso del espacio de color HSV (*Hue*, *Saturation*, *Value*). Este modelo de color tiene tres parámetros principales:

- ***Hue*** (Tono): Representa el matiz del color, expresado en grados en un rango de 0 a 360 (por ejemplo, $rojo \approx 0°$, $verde \approx 120°$, $azul \approx 240°$).
- ***Saturation*** (Saturación): Define la intensidad o pureza del color, variando de 0 (gris) a 100% (color completamente puro).
- ***Value*** (Brillo): Indica la luminosidad del color, donde 0 representa negro (sin brillo) y 100% representa el máximo brillo del color.

El uso del espacio de color HSV permite abordar una de las problemáticas principales planteadas: la variación en la intensidad de luz. A diferencia del espacio de color RGB, donde los canales combinan información de color y brillo, el espacio HSV separa de manera más eficiente estos dos aspectos. Esto permite procesar las imágenes para distinguir colores de manera más robusta frente a cambios en las condiciones de iluminación.

En este proyecto, se utilizó el espacio HSV para definir umbrales específicos que permitieron separar el color rojo de los dados del color verde del fondo. Esta segmentación eficiente fue clave para identificar y aislar los dados en cada cuadro del video.

Para explicar la implementación del algoritmo, se utilizará como referencia el video 1.

La función principal, llamada ***procesar_video***, organiza y gestiona el flujo de trabajo del procesamiento de los videos. Esta función realiza los llamados necesarios a las subfunciones responsables de cada etapa del procesamiento.

Inicialización: La función comienza capturando ciertas estadísticas clave del video de entrada, como:

- Resolución: Altura y ancho del cuadro en píxeles.
- FPS (Frames per Second): Número de cuadros por segundo.
- Número total de cuadros: Cantidad total de frames en el video.

Configuración del archivo de salida: Se establece un archivo de salida denominado *out*, donde se almacenarán las imágenes procesadas correspondientes a cada cuadro del vídeo. Este archivo contiene el video generado con los resultados del procesamiento, incluyendo la identificación de los dados y su resaltado en un cuadro delimitador azul, además de los números reconocidos.

Con las estadísticas del video ya obtenidas, el algoritmo procede a iterar sobre cada cuadro (frame) del video. Durante esta iteración, se llama a la función ***recortar_transformar_hsv***, que se encarga de procesar cada frame recibido como argumento.

Función ***recortar_transformar_hsv***

El objetivo principal de esta función es identificar la región del fondo verde dentro del frame utilizando el espacio de color HSV y recortar esa sección para facilitar la detección posterior de los dados.

Pasos principales de la función:

Definición de umbrales en el espacio HSV: La función establece un rango de valores para el tono (Hue), saturación (Saturation) y brillo (Value) que corresponde al color verde del fondo:

- Hue (H): De 0 a 100.
- Saturation (S): De 190 a 250.
- Value (V): De 50 a 230.

Estos umbrales permiten segmentar las áreas verdes del frame con mayor precisión, incluso ante ligeras variaciones en el color debido a cambios en la iluminación.

Conversión del frame al espacio HSV: El frame original, que está en formato BGR (formato por defecto de *OpenCV*), se convierte al espacio de color HSV usando la función *cv2.cvtColor*.

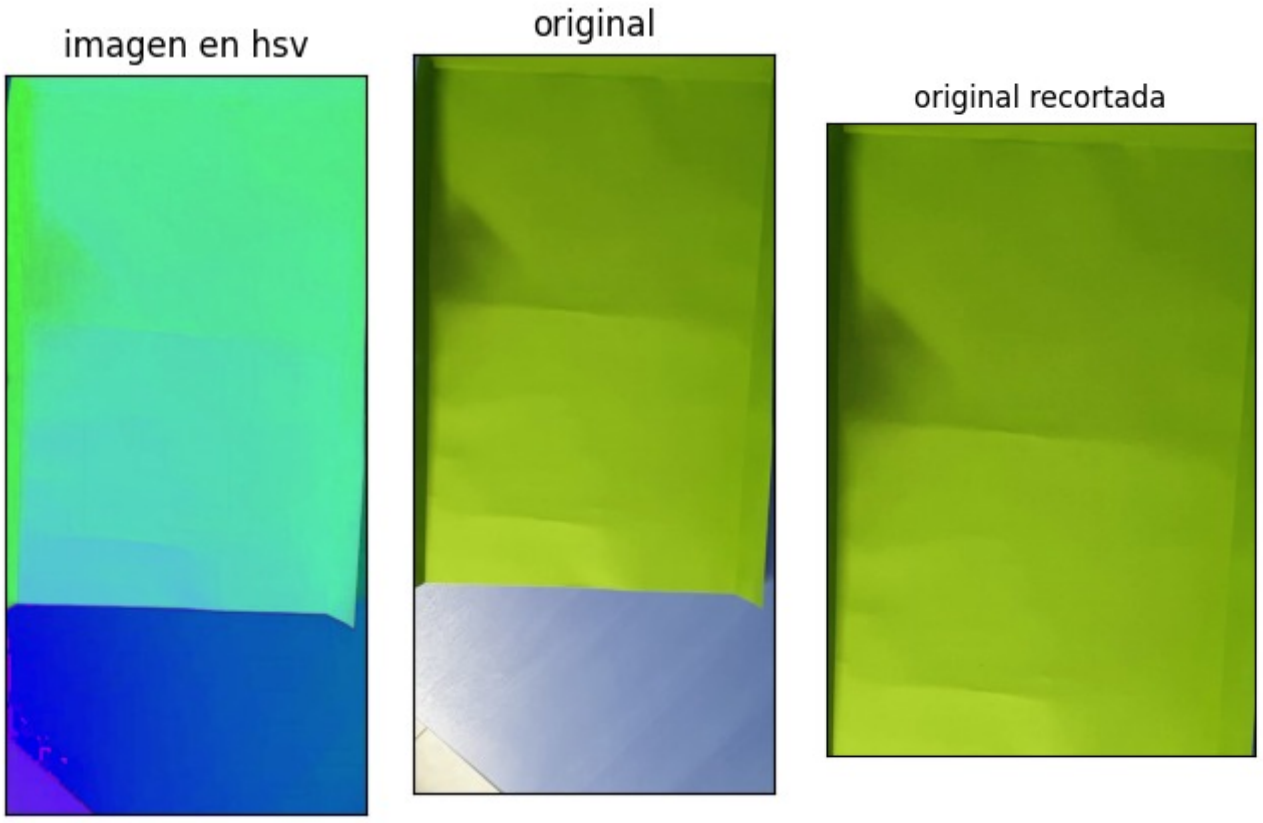
Aplicación de la máscara por umbral: Se crea una máscara binaria usando la función *cv2.inRange*, que marca en blanco (valor 255) los píxeles que caen dentro del rango especificado para el color verde, y en negro (valor 0) los que no cumplen con los criterios. Esto genera una imagen binaria donde el área verde del fondo es destacada.

Identificación de las coordenadas del área verde: Utilizando la matriz resultante, se identifican los píxeles blancos correspondientes al fondo verde mediante la función *np.column_stack(np.where(frame_threshold == 255))*. Esto devuelve las coordenadas (x, y) de todos los píxeles blancos.

A partir de estas coordenadas, se calcula:

- Las esquinas superiores izquierdas: min_x y min_y.
- Las esquinas inferiores derechas: max_x y max_y.
- Estas esquinas delimitan un rectángulo que contiene el fondo verde en el frame.

Resultado: La función devuelve las coordenadas $min\_x,\ min\_y,\ max\_x,\ max\_y$, que representan los límites del área verde. Estas coordenadas se utilizan posteriormente para recortar el frame y concentrar el procesamiento en el área relevante, ignorando el fondo y posibles elementos distractores.

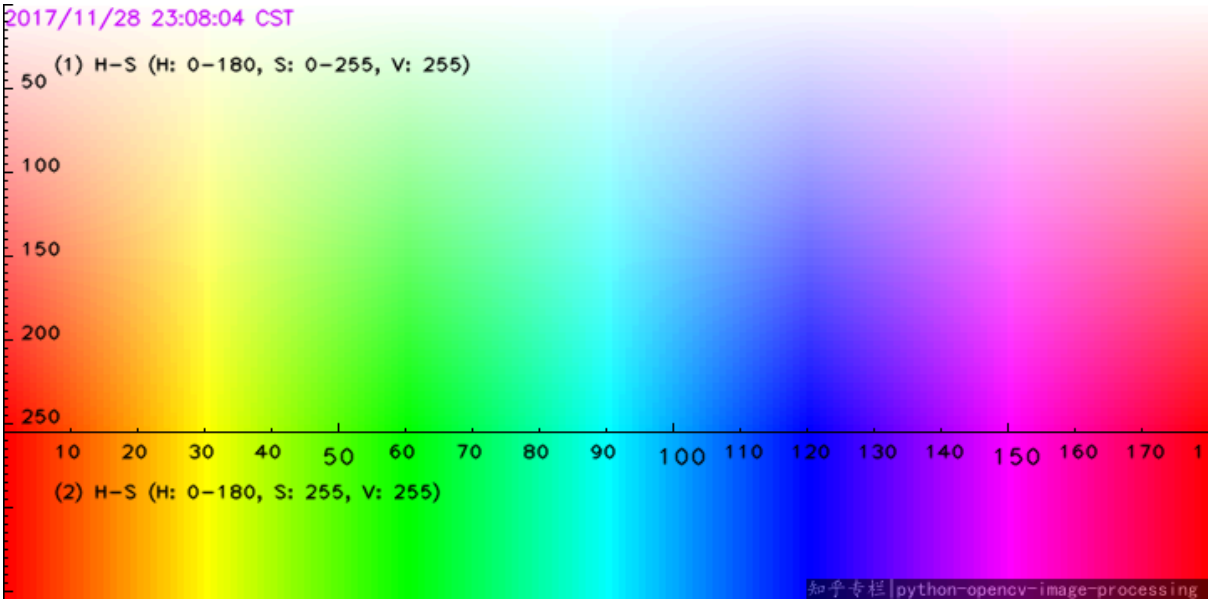


Función ***detectar_dados_rojos***

Una vez recortada la imagen para aislar el área de interés (fondo verde), se llama a la función *detectar_dados_rojos*, que recibe como entrada el frame previamente recortado. Esta función se encarga de identificar las áreas correspondientes a los dados rojos mediante el procesamiento en el espacio de color HSV.

Particularidad del color rojo en HSV: El color rojo tiene una representación especial en el espacio HSV, ya que está presente en ambos extremos del rango de tono (Hue):

- Un rango bajo, centrado en valores cercanos a 0°.
- Un rango alto, cerca de los valores de 360° (que, en el espacio HSV de OpenCV, se representan como valores cercanos a 180 debido a su escala interna).



Por esta razón, es necesario definir dos conjuntos de umbrales para abarcar ambos rangos.

A continuación se explica el procesado de la función

Conversión a HSV: Al igual que en la función anterior, el frame se convierte del espacio de color BGR al espacio HSV usando la función cv2.cvtColor.

Definición de umbrales: Se establecen dos conjuntos de umbrales para el color rojo:

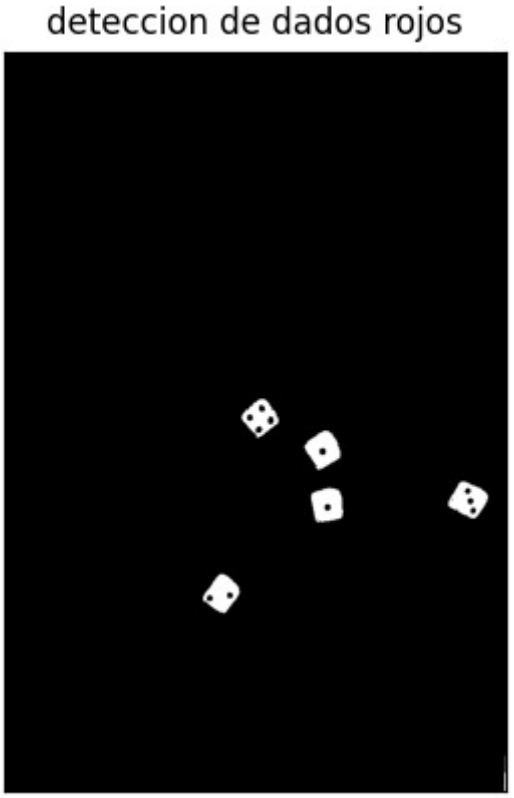
- Primer rango (rojo_bajo1 a rojo_alto1): Representa los tonos rojos en la parte baja del espectro HSV.
- Segundo rango (rojo_bajo2 a rojo_alto2): Corresponde a los tonos rojos en el extremo superior del espectro HSV.

Generación de máscaras binarias: La función *cv2.inRange* se aplica dos veces, una para cada rango de color rojo.

Esta función genera una máscara binaria donde los píxeles que se encuentran dentro del rango de color especificado son marcados con blanco (valor 255), y los demás píxeles son marcados con negro (valor 0).
Por ejemplo, si un píxel tiene un valor HSV dentro del rango *rojo_bajo1* y *rojo_alto1*, su valor en la máscara será 255.

Combinación de las máscaras: La función *cv2.bitwise_or* combina las dos máscaras binarias generadas para los rangos bajos y altos del color rojo. El resultado es una única máscara binaria que incluye todos los píxeles que corresponden a cualquier tonalidad de rojo en el frame.
En términos simples, es una operación lógica que asegura que un píxel esté marcado como blanco si pertenece a al menos uno de los dos rangos.

Resultado: La función retorna la máscara combinada, que es una imagen binaria donde los dados rojos aparecen destacados en blanco sobre un fondo negro.



Detección de la Ubicación de los Dados

Con el preprocesamiento del color rojo ya realizado, el siguiente paso en el algoritmo es identificar las ubicaciones exactas de los dados dentro del frame. Esto se logra utilizando la función ***detectar_ubicacion_dados***, que analiza la máscara binaria generada previamente.

Detección de componentes conectados: La función utiliza *cv2.connectedComponentsWithStats*, una técnica de análisis de imágenes que identifica regiones conectadas en una imagen binaria. Esta función segmenta las áreas conectadas de píxeles blancos (valor 255) en la máscara y devuelve estadísticas pertinentes para el procesado.

Filtrado por criterios geométricos y de área: Para identificar correctamente los dados, se aplican las siguientes condiciones de filtrado a cada región conectada:

Tamaño del área: Se consideran regiones cuya área esté entre 1500 y 5500 píxeles. Esto permite descartar áreas muy pequeñas (ruido) o muy grandes (objetos que no son dados).

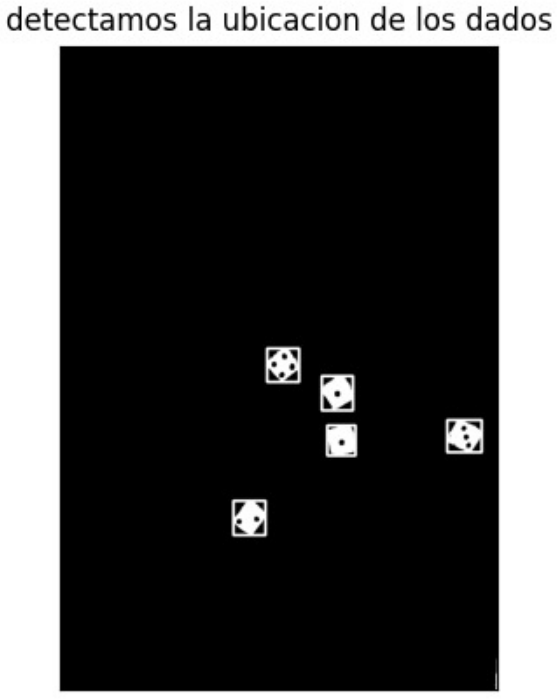
Proporción ancho/alto (w/h): También llamado relación de aspecto, los dados, al ser aproximadamente cuadrados, deben cumplir con una proporción de entre 0.5 y 1.5. Esto elimina regiones que son demasiado alargadas o estrechas.

Almacenamiento de información relevante: Para cada región que cumpla con los criterios anteriores, se guarda su información en la lista *caja_centroides*. Cada elemento de la lista contiene:

- x, y, w, h: Parámetros del rectángulo delimitador.
- *centroids[i]*: Coordenadas del centroide de la región.

Condición para garantizar detección confiable: Si el número de dados detectados (regiones válidas) es igual o superior a 3, se devuelve la lista *caja_centroides*.

Si no se alcanzan al menos 3 detecciones, se retorna una lista vacía como medida de precaución para evitar resultados incompletos o falsos positivos.



Último Procesamiento: Creación de Cajas y Etiquetas

El procesamiento final del algoritmo está encargado de identificar el valor de la cara visible de cada dado detenido y marcarlo visualmente en el video. Este paso se realiza mediante las funciones ***calcular_cara*** y ***crear_caja_etiqueta***, que se describen a continuación.

Función ***calcular_cara***

Esta función recibe como entrada el frame actual y las coordenadas de una bounding box que contiene un dado. Su objetivo es determinar cuántos puntos (o círculos) hay en la cara visible del dado, que representan el valor del dado.

Extracción de la región de interés: A partir de las coordenadas de la caja (x, y, w, h), se recorta el área del frame donde se encuentra el dado, añadiendo un pequeño margen para asegurar que la cara completa del dado esté incluida.

Conversión a escala de grises: La imagen recortada se convierte a escala de grises mediante *cv2.cvtColor*, lo que simplifica el procesamiento y reduce el ruido de color.

Binarización de la imagen: Se aplica un umbral binario (*cv2.threshold*) para resaltar las regiones blancas (potenciales puntos) en la cara del dado.

Los valores menores a 120 se asignan a negro (0) y los mayores a blanco (255).

Detección de componentes conectados: Se utiliza *cv2.connectedComponentsWithStats* para identificar regiones conectadas en la imagen binaria.

Solo se consideran componentes cuyo tamaño de área esté entre 80 y 250 píxeles, lo que corresponde al tamaño típico de los puntos en un dado.

Validación geométrica: Para garantizar que las regiones detectadas sean efectivamente puntos, se calculan las proporciones de área y perímetro mediante el "factor de forma":

$$factor\ forma = \frac{área}{perímetro^2}$$

Solo se aceptan componentes cuyos factores de forma estén entre 0.05 y 0.09, valores típicos de círculos pequeños.

Suma de puntos: Cada componente válido se contabiliza como un punto visible en la cara del dado. El total se devuelve como el valor de la cara del dado.

Función ***crear_caja_etiqueta***

Esta función es responsable de dibujar rectángulos alrededor de los dados detectados, etiquetarlos con su valor numérico, y mantener la correspondencia entre los dados detectados en diferentes frames del video.

Extracción de las cajas actuales: Se evalúa si en el frame actual (num_img) se detectaron dados *(box[num_img])*.

Si hay detecciones, se obtienen las cajas delimitadoras y sus centroides.

Dibujar las cajas delimitadoras: Para cada caja en box[num_img], se dibuja un rectángulo en el frame utilizando *cv2.rectangle* con un color azul y un grosor de línea especificado.

Seguimiento entre frames: Para garantizar la correspondencia entre los dados detectados en el frame actual y el anterior, se comparan las coordenadas de los centroides actuales con los centroides del frame anterior.

Se utiliza una distancia máxima de 10 píxeles en ambas coordenadas (x, y) como criterio de coincidencia.

Etiquetado de los dados: Para cada dado que coincide entre frames consecutivos, se llama a la función *calcular_cara* para determinar su valor visible.

Este valor se escribe sobre el rectángulo del dado utilizando *cv2.putText*, con un texto del formato:

$$Valor\ es\colon [número]$$

Retorno del frame procesado: Se devuelve el frame con las cajas y etiquetas añadidas.

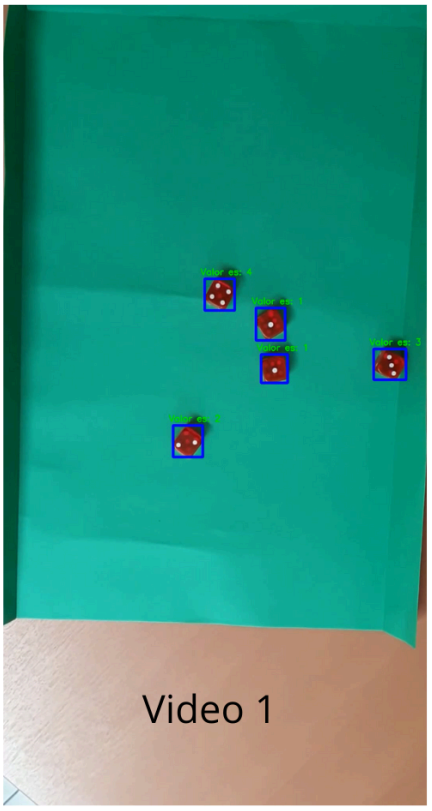
Salida

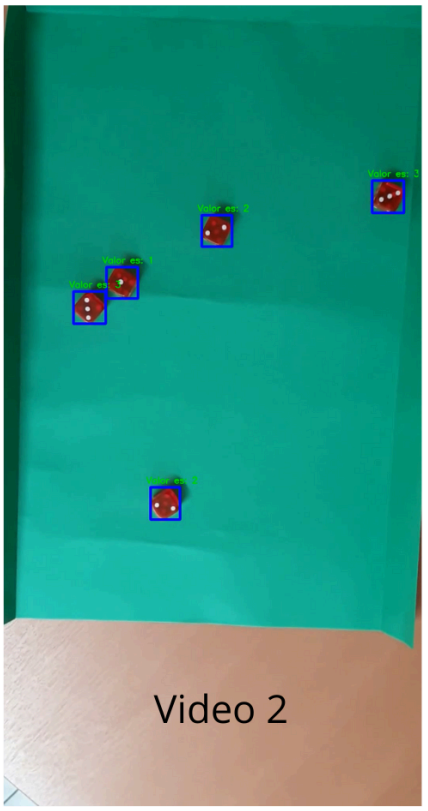
El frame procesado y anotado se inserta de vuelta en el frame original (en la región recortada). Luego el frame modificado se escribe en el archivo de vídeo de salida utilizando *out.write()*. Una vez procesados todos los frames, los objetos cap (video de entrada) y out (video de salida) se liberan para cerrar los archivos y evitar fugas de memoria.

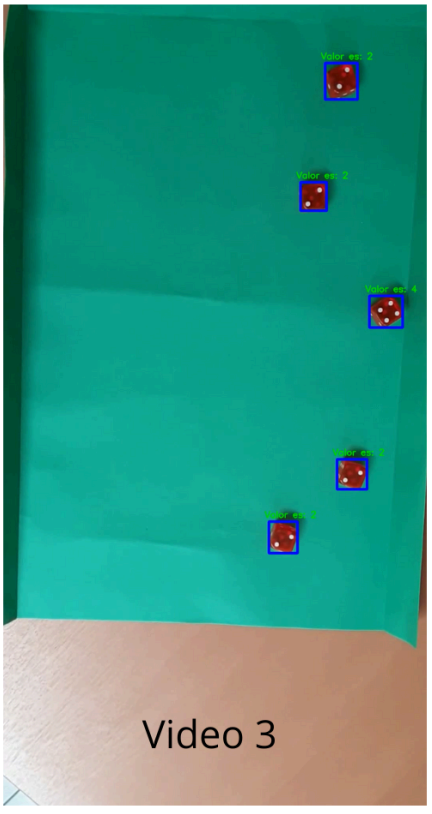
Resultado

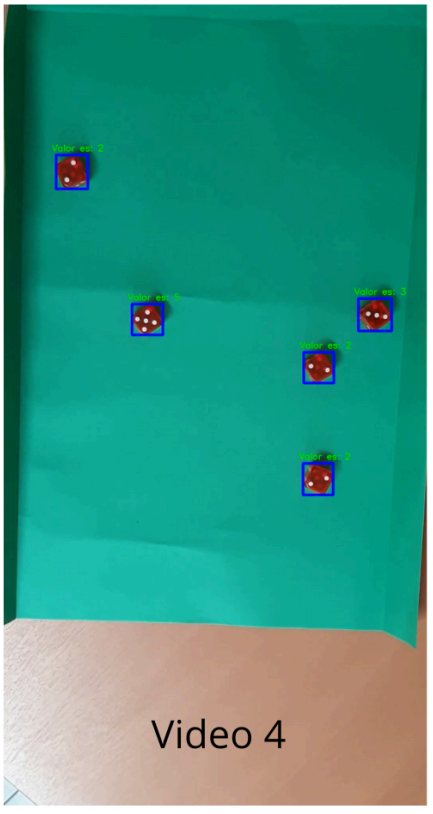
Los resultados obtenidos del procesamiento de los videos muestran una detección precisa de todos los dados y sus respectivos valores. El algoritmo ha demostrado ser efectivo incluso ante las dificultades previamente planteadas, como el desenfoque de imagen, la interferencia causada por la mano del usuario y cambios en la intensidad de la luz.

La detección de los dados es sólida, garantizando que cada dado sea identificado correctamente junto con su valor. El algoritmo es capaz de procesar cada video en aproximadamente 10 segundos. Este tiempo puede variar dependiendo del hardware utilizado, pero se considera razonable para el procesamiento de video en tiempo casi real.







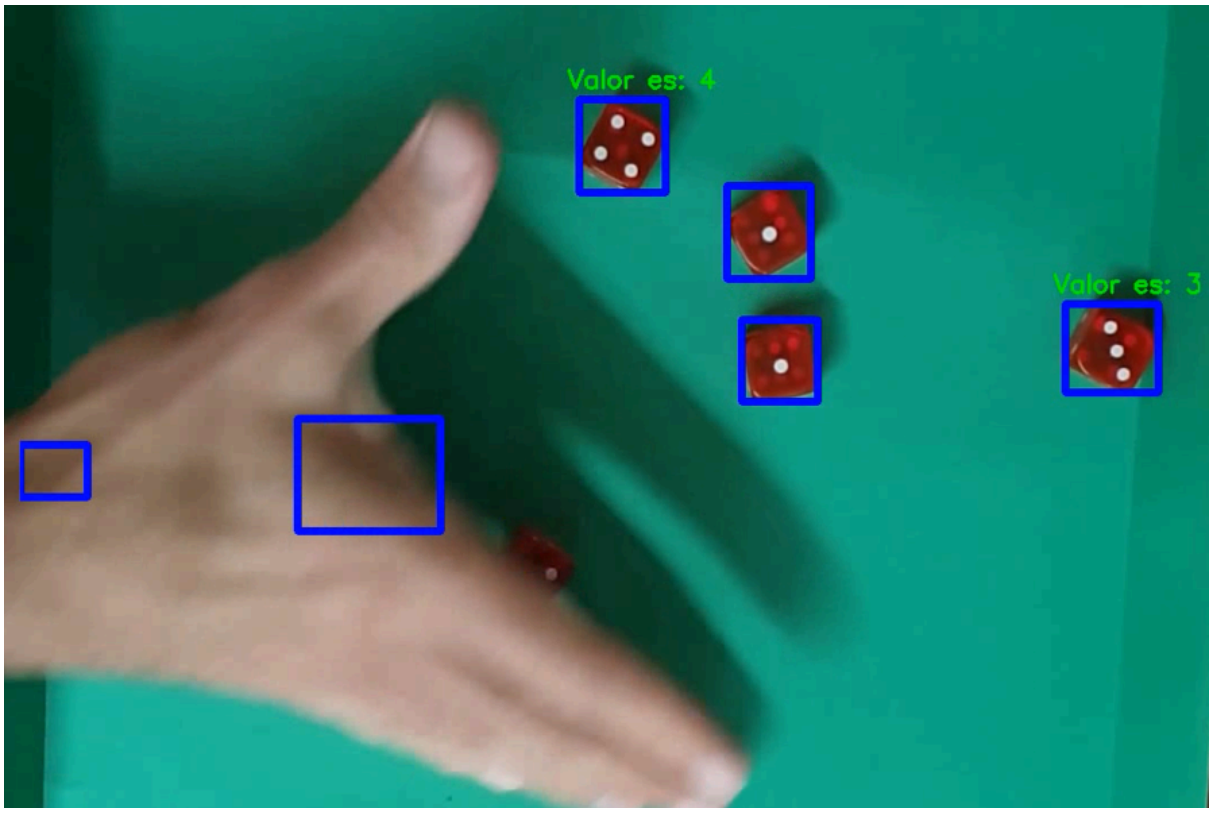


Posibles puntos de mejora

Uno de los problemas identificados es la detección inexacta del estado de movimiento de los dados. Aunque el algoritmo detecta bien los dados cuando están en reposo, en situaciones donde los dados están en movimiento o se solapan con otros objetos, como las manos, la precisión puede disminuir.

Este problema radica en falsos positivos en áreas que no corresponden a dados, como por ejemplo en el video 1, donde se observan puntos erróneos relacionados con la mano del usuario.

Se propone utilizar el seguimiento de objetos de *cv2* utilizando técnicas como *cv2.denseOpticalFlow* o *cv2.Tracker* para identificar y rastrear cada dado en movimiento. Sin embargo, esto podría requerir un mayor consumo de recursos y tiempo de procesamiento, especialmente en casos de alta movilidad o múltiples objetos.



Otro desafío es la eliminación de objetos que no corresponden al objetivo, como objetos que no son dados, especialmente en situaciones donde se presentan movimientos interrumpidos, como manos o elementos adicionales en la escena.

Esto se refleja en falsos positivos generados por movimientos o colores similares que no corresponden a los dados.

Se propone ajustar los rangos en el espacio HSV para diferenciar colores similares. También, considerar el uso de técnicas como Contornos de Objeto y clasificación de área, para descartar componentes que no cumplan con las características deseadas (como tamaño, forma o factor de forma).

Conclusión

El algoritmo actual es funcional y sólido para la detección inicial de dados en un video, pero su robustez puede mejorarse en términos de seguimiento de objetos y eliminación de falsos positivos, especialmente en escenarios complejos o con alta movilidad. La adición de técnicas avanzadas como seguimiento de objetos y optimización del consumo de recursos podría aumentar considerablemente la eficiencia y precisión del sistema.