TRABAJO PRÁCTICO N°3

Procesamiento de Imágenes I

Avecilla Tomás (A-4239/9)

Calcia Franco (C-7363/2)

9 de diciembre de 2024

TECNICATURA UNIVERSITARIA EN INTELIGENCIA ARTIFICIAL - UNR

Índice

Introducción

El presente informe tiene como objetivo presentar el desarrollo y los resultados obtenidos en el Trabajo Práctico N° 3, correspondiente al segundo semestre del año 2024. Este trabajo aborda un problema relacionado con la detección y clasificación de dados en vídeos utilizando técnicas de procesamiento de imágenes como el análisis de movimiento.

El problema consiste en diseñar un algoritmo capaz de detectar automáticamente cuándo cinco dados se detienen en videos de tiradas, para luego identificar y resaltar el número de la cara superior en cada dado.

Materiales y Herramientas

Para el desarrollo del presente trabajo práctico, se utilizó lo siguiente:

Software y Librerías

1. **Python**
2. **Biblioteca OpenCV**: Para realizar operaciones de procesamiento de imágenes como:
    - Conversión a escala de grises.
    - Aplicación de filtro gaussiano.
    - Detección de bordes (Canny Edge Detection).
    - Segmentación y análisis de contornos.
3. **NumPy**: Para el manejo eficiente de matrices y operaciones numéricas.
4. **Matplotlib**: Para la visualización de imágenes y gráficos.
5. **Editor de código**: Para este caso se utilizó Visual Studio Code.

Procedimiento: Problema 1 - Detección y Clasificación de Dados en Video.

1. Carga y Preprocesamiento del Video

- **Carga del video original:**
  Se utiliza `cv2.VideoCapture` para abrir los videos de las tiradas (e.g., `tirada_1.mp4`). Esto permite leer cuadro por cuadro cada frame para su procesamiento.

- **Configuración del escritor de video:**
  Se utiliza `cv2.VideoWriter` para crear un archivo de salida con las mismas dimensiones, FPS y formato del video original. Esto permite generar un video donde los dados quietos estén resaltados con un cuadro en azul y se les asigna el valor de la cara correspondiente.

2. Detección de Dados Quietos por Frames

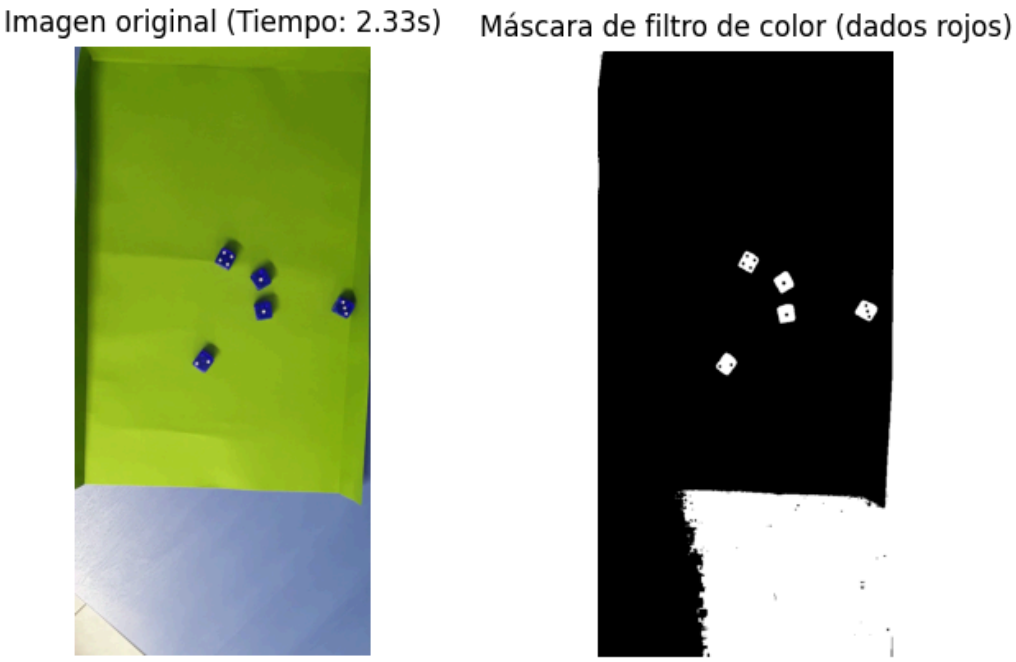
- **Ignorar los primeros segundos del video:**
  Se implementa un filtro temporal que omite el primer segundo del video (`tiempo_espera = 1.58`). Esto es útil porque en ese intervalo suele no haber movimientos significativos de los dados, evitando falsos positivos al analizar cuadros iniciales.
- **Comparación de contornos en frames consecutivos:**
  Una vez transcurrido el tiempo de espera, se detectan contornos en el frame actual y en el frame previo mediante la función `detectar_contornos`. Posteriormente, se utiliza la función `comparar_contornos` para evaluar si los dados permanecen en la misma posición entre ambos cuadros consecutivos. Esto permite identificar momentos en los que los dados están completamente quietos.
- **Registro de frames relevantes:**
  Cuando los dados quietos son detectados, se muestra el frame correspondiente en pantalla con el tiempo exacto en que ocurre el evento. Esto facilita la validación visual de los resultados y resalta los momentos de interés en el video.

3. Segmentación de Dados Rojos

- **Conversión a espacio de color HSV:**
  Cada frame es convertido de BGR a HSV mediante `cv2.cvtColor`. Este espacio de color es más adecuado para segmentar colores, ya que separa la intensidad (valor) del tono (hue).

- **Definición de máscaras para tonos rojos:**
  Se definen dos rangos de color en HSV para cubrir tonos rojos (e.g., `[0, 50, 50]` a `[10, 255, 255]` y `[170, 50, 50]` a `[180, 255, 255]`). Las máscaras correspondientes son generadas con `cv2.inRange`, y luego combinadas con `cv2.bitwise_or`.

- **Generación de una máscara binaria:**
  El resultado es una máscara donde los píxeles correspondientes a los dados son blancos y el resto negros. Esta máscara es importante ya que es la base para detectar los contornos que necesitamos.



4. Detección de Movimiento de los Dados

- **Detección de bordes con Canny:**
  Sobre la máscara binaria, se aplica `cv2.Canny` con umbrales altos (e.g., `1000` y `1500`) para detectar bordes significativos y evitar el ruido.

- **Dilatación de bordes:**
  Se utiliza `cv2.dilate` para engrosar los bordes, asegurando que los contornos de los dados sean cerrados y más fáciles de detectar.

- **Extracción de contornos:**
  Con `cv2.findContours` se obtienen los contornos externos (opción `cv2.RETR_EXTERNAL`) de los dados. Esto descarta cualquier contorno interno o irrelevante.

- **Comparación de contornos entre frames:**
  Se calcula el área total de los contornos en el frame actual y el previo. Si la diferencia es menor a un umbral (e.g., `45`), se considera que los dados están en reposo.

5. Reconocimiento de caras visibles de los dados

- **Recorte de contornos válidos:**

  Para identificar los dados en la imagen, se recorren los contornos detectados y se descartan aquellos cuya área esté fuera del rango esperado (mínima: 4300, máxima: 6400). Este rango fue determinado iterativamente al observar las áreas de los dados detectados, confirmando que todos ellos se encuentran dentro de estos valores.

  Los dados que cumplen con este criterio son considerados válidos y se recortan utilizando la función `cv2.boundingRect`.

  Luego se llama a otra función que hace lo siguiente:

- **Conversión a escala de grises:**
  Cada recorte es convertido a escala de grises con `cv2.cvtColor` para facilitar la detección de características (caras).

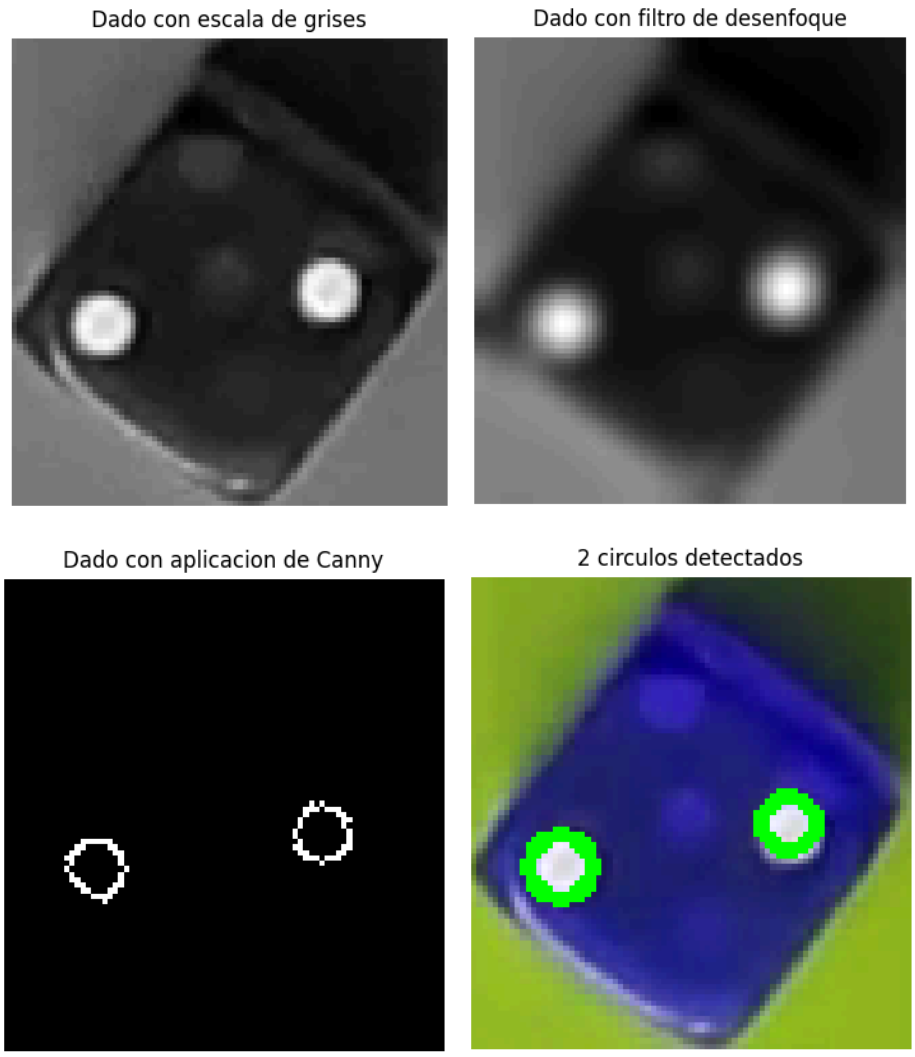
- **Aplicación de filtro Gaussiano:**
  Se aplica `cv2.GaussianBlur` para reducir el ruido, con un kernel de tamaño 9x9 y sigma 3. Este valor de sigma ayuda a resaltar los puntos presentes en los dados, facilitando así la identificación de sus caras.

- **Detección de círculos (caras) con Transformada de Hough:**
  Se utiliza `cv2.HoughCircles` para detectar los puntos en la cara del dado.

  - Parámetros importantes:
    - `dp=1.2`: Resolución inversa del acumulador.
    - `minDist=8`: Distancia mínima entre centros de círculos detectados.
    - `param1=25` y `param2=10`: Umbrales para Canny y el acumulador.
    - `minRadius=5` y `maxRadius=8`: Radio esperado de los círculos.

- **Conteo de caras:**
  El número de círculos detectados corresponde al número de caras visibles en el

dado.



6. Dibujo de Resultados

- **Dibujo de bounding boxes:**
  Se dibuja un rectángulo azul (`cv2.rectangle`) alrededor de cada dado detectado para cada uno de los videos procesados.

- **Agregado de texto con el número de caras:**

Sobre cada dado, se imprime el número de caras detectadas mediante `cv2.putText`.

7. Generación del Video Procesado

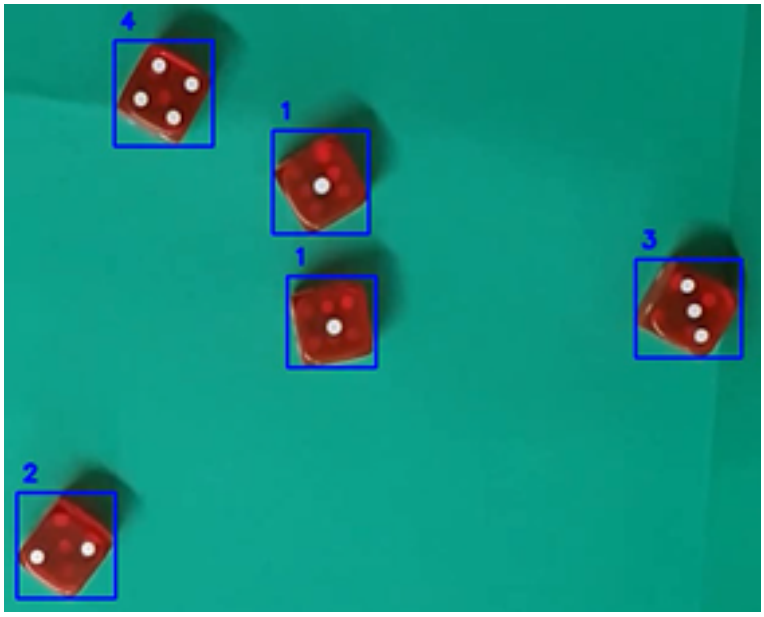
- **Escritura de cuadros en el video de salida:**
  Cada cuadro procesado es escrito en el archivo de salida con `out.write`.

- **Liberación de recursos:**
  Al finalizar, se liberan los recursos del vídeo original y del escritor mediante `cap.release` y `out.release`.

8. Resultados Finales

- **Visualización de máscaras y contornos:**
  Durante el proceso, se muestran imágenes intermedias para verificar la segmentación por color, procesamiento de imagen, bordes detectados, y contornos dibujados.

- **Video de salida:**
  Los videos generados resaltan con bounding boxes azules los dados en reposo, y muestran sobre ellos el número de caras reconocidas.



Conclusión

El procedimiento desarrollado para la **detección y reconocimiento de caras visibles en dados** a partir de videos demostró ser efectivo al combinar técnicas avanzadas de procesamiento de imágenes y visión por computadora. Al utilizar la segmentación por color, detección de bordes, análisis de movimiento, y la Transformada de Hough, se logró identificar los dados en reposo y contar las caras visibles con precisión.

Los principales logros incluyen:

1. **Precisión en la detección de dados:** La segmentación en el espacio de color HSV permitió identificar de manera precisa los dados rojos con una precisión del 100%.
2. **Reducción de ruido:** El uso de filtros como el desenfoque gaussiano y la dilatación de bordes garantizó contornos definidos para la detección, minimizando falsos positivos.
3. **Reconocimiento confiable de caras:** Con los parámetros que asignamos, la detección de círculos mediante la Transformada de Hough mostró alta precisión para identificar los puntos en las caras de los dados.
4. **Procesamiento optimizado:** La combinación de detección basada en frames y análisis del área total de contornos permitió identificar los momentos en los que los dados estaban en reposo. Este enfoque evitó errores causados por el movimiento de los dados y aseguró que los resultados fueran buenos.