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:

Desarrollar un algoritmo para detectar automáticamente cuando se detienen los dados y leer el número obtenido en cada uno. Informar todos los pasos del procesamiento.

Generar videos (uno para cada archivo) donde los dados, mientras esten en reposo, aparezcan resaltados con un bounding box de color azul y ademas, agregar sobre los mismos el número reconocido.

Figura 1 – Dados luego de una tirada.

Descripción del problema

El presente problema consiste en desarrollar un sistema automatizado para el análisis de videos que contienen tiradas de cinco dados, con el objetivo de identificar los números obtenidos en cada dado una vez que estos se detienen. Este análisis incluye varias etapas de procesamiento digital de imágenes y visión por computadora, abarcando desde la detección del estado de reposo de los dados hasta el reconocimiento del número en su cara superior.

El sistema debe ser capaz de:

- Detectar automáticamente el momento en que los dados dejan de moverse.
- Identificar y leer el número visible en cada dado en su estado de reposo.
- Generar un video de salida para cada archivo procesado, en el cual:
- Los dados que estén en reposo sean resaltados con un bounding box de color azul.
- El número reconocido en cada dado se muestra superpuesto en la imagen.

Este problema plantea desafíos relacionados con el procesamiento de imágenes dinámicas y el reconocimiento de patrones, incluyendo la detección del movimiento, segmentación de objetos (dados) en la escena, reconocimiento numérico, y la generación de contenido visual para representar los resultados.

Funciones

imreconstruct

La función ***imreconstruct*** implementa un procedimiento de reconstrucción morfológica, un proceso iterativo utilizado para propagar regiones de interés dentro de una imagen bajo ciertas restricciones. Este método es especialmente útil en el procesamiento de imágenes para realizar operaciones como la eliminación de ruido, la segmentación y la detección de objetos enmascarados.

Funcionamiento

La reconstrucción morfológica parte de dos imágenes:

- marker: una imagen que contiene la semilla inicial de la región que se desea propagar.

- mask: una imagen que actúa como restricción, limitando hasta dónde puede crecer la región definida por el marcador.

El objetivo de la función es modificar iterativamente el marcador mediante operaciones de dilatación restringida, hasta que su propagación converge, es decir, hasta que no haya más cambios.

Implementación

Verificación de consistencia entre marker y mask: Se asegura que ambas imágenes tengan las mismas dimensiones y tipo de datos, ya que estas condiciones son necesarias para las operaciones morfológicas posteriores. Si hay discrepancias en el tipo de datos, se fuerza la conversión del marker al tipo de la mask.

Definición del kernel estructural: Si no se especifica un kernel, se utiliza por defecto una matriz de 3x3 de unos. Este kernel define el vecindario que influye en las operaciones de dilatación.

Iteración del proceso de reconstrucción:

- En cada iteración, se realiza una dilatación sobre el marker. La dilatación expande los píxeles blancos (o valores más altos en imágenes en escala de grises) en torno a sus vecinos definidos por el kernel.
- La imagen dilatada se intersecta con la mask utilizando una operación lógica bit a bit (cv2.bitwise_and). Esto asegura que la expansión del marker no exceda los límites establecidos por la mask.
- Se verifica si la imagen resultante es igual al marker de la iteración anterior. Si no hay diferencias, el proceso ha convergido y se rompe el ciclo.

Resultado final: Cuando el proceso converge, se devuelve el marker reconstruido, que ahora representa la región de interés completamente propagada pero restringida por la mask.

imfillhole

La función ***imfillhole*** está diseñada para rellenar los huecos en una imagen binaria, donde un hueco se define como una región de píxeles negros (valor 0) completamente rodeada por píxeles blancos (valor 255). Este tipo de operación es común en procesamiento de imágenes, especialmente en tareas de preprocesamiento y segmentación, para obtener formas sólidas y eliminar discontinuidades internas.

Funcionamiento

1. Entrada esperada: La función toma como entrada una imagen binaria, donde los píxeles tienen valores de 0 (negro) o 255 (blanco).
2. Generación de la máscara de bordes:
    - Se crea una máscara vacía del mismo tamaño que la imagen de entrada, inicializada con ceros.

    - Luego, se utiliza ***cv2.copyMakeBorder*** para agregar un borde de un píxel alrededor de esta máscara con un valor constante de 255 (blanco). Esto asegura que el fondo de la imagen quede conectado al borde externo y facilita identificar regiones externas al área de interés.
3. Definición del marcador (**marker**):
    - El marcador se define como el complemento (inversión de color) de la imagen de entrada, restringido a la máscara de bordes. Esto selecciona las regiones externas negras en la imagen, que potencialmente conectan con los huecos.
4. Definición de la máscara (**mask**):
    - La máscara, que restringirá el crecimiento del marcador, se define como el complemento de la imagen de entrada. Esta inversión transforma las regiones negras (0) en blancas (255) y viceversa, permitiendo trabajar con los huecos como regiones internas.
5. Reconstrucción morfológica:
    - Se utiliza la función ***imreconstruct*** para calcular la reconstrucción del marcador bajo la máscara. Esto permite propagar el marcador a través de los píxeles negros que están conectados al borde, rellenando los huecos internos al encontrarse limitados por el fondo blanco.
6. Obtención de la imagen final:
    - Una vez calculada la reconstrucción, se toma el complemento de esta imagen reconstruida para devolver la versión rellenada de la imagen original. Los huecos internos que estaban rodeados de píxeles blancos ahora se convierten en regiones blancas sólidas.

se_mueve_no_se_mueve

La función se_mueve_no_se_mueve determina qué objetos o elementos en una imagen (representados por sus centroides) han permanecido en reposo entre dos cuadros consecutivos de un conjunto de imágenes o un video. Esto es particularmente útil en análisis de movimiento, como la detección de objetos estacionarios o la determinación de estabilidad.

Funcionamiento

- Entrada esperada:
    - foto: Un número entero que identifica el cuadro o imagen actual en la secuencia.
    - centroides: Una lista que contiene información de los centroides detectados en cada cuadro, con el siguiente formato:

      $[(foto\_número,\ [(id\_objeto,\ (x\_centro,\ y\_centro)),\ \ldots]),\ \ldots]$

      Donde cada elemento describe el número de la imagen $(foto\_número)$ y una lista de objetos detectados, cada uno representado por su identificador (id_objeto) y las coordenadas de su centroide.

- Búsqueda de los datos relevantes:

    - La función busca los centroides correspondientes al cuadro anterior $(foto - 1)$ y al cuadro actual $(foto)$ dentro de la lista centroides.
    - Si no encuentra datos para cualquiera de los dos cuadros, la función retorna una lista vacía, indicando que no se puede determinar el movimiento en esos casos.
- Comparación de posiciones:
    - Si se encuentran los centroides para ambos cuadros, se comparan las coordenadas de los centroides correspondientes al cuadro actual (act_cen) con los del cuadro anterior (ant_cen).
    - Para cada par de centroides, se verifica si el desplazamiento en las coordenadas x e y está dentro de un margen de tolerancia de ±20 píxeles. Este umbral considera que un objeto no se ha movido si su posición permanece dentro de esta ventana de tolerancia.
- Detección de objetos estacionarios:
    - Si las coordenadas de un objeto en el cuadro anterior caen dentro del rango permitido en relación al cuadro actual, el identificador (id_objeto) del objeto es añadido a la lista lista.
- Resultado:
    - La función devuelve una lista con los identificadores de los objetos que no han cambiado de posición de manera significativa entre los dos cuadros comparados.

Explicación de implementación

- Iteraciones para búsqueda:
  Se recorre la lista centroides para extraer los datos relevantes del cuadro anterior y actual. Esto asegura que solo se procesan los datos de los cuadros especificados, reduciendo el impacto del tamaño total de la lista.
- Margen de tolerancia:
  La comparación de posiciones utiliza un rango de ±20 píxeles, lo cual permite cierta flexibilidad para tolerar errores menores en la detección de centroides o ligeras variaciones debido a ruido.
- Eficiencia:
  La función opera en un ciclo doble sobre los centroides, con un costo computacional que depende del número de objetos detectados en cada cuadro.

procesar_imagenes

La función ***procesar_imagenes*** procesa un cuadro individual de una secuencia para detectar dados, determinar su estado de movimiento o reposo y calcular el valor de la cara visible en aquellos que estén estacionarios. Además, resalta visualmente los dados en el cuadro procesado, dibujando un bounding box y anotando el valor de la cara detectada. Es útil en tareas como análisis de imágenes en secuencias de video para sistemas de detección automática.

Descripción del funcionamiento

El proceso comienza con la preprocesamiento de la imagen para preparar la detección de dados y puntos en sus caras.

1. Preprocesamiento para detección de dados:
   Se convierte la imagen a escala de grises y se aplica un umbral binario para simplificar la información. Luego, se utiliza el operador de Canny para detectar bordes. Estos bordes se dilatan para engrosarlos y luego se rellenan los posibles huecos internos utilizando la función ***imfillhole***. Posteriormente, se aplica una erosión para eliminar artefactos pequeños y mejorar la definición de los contornos de los dados.
2. Preprocesamiento para detección de puntos en las caras de los dados:
   Se aplica nuevamente el operador de Canny con parámetros más agresivos, seguido por una dilatación y un rellenado de huecos similar al de los dados. Este procesamiento identifica los contornos que corresponden a puntos en las caras visibles de los dados.
3. Detección de los bounding boxes de los dados:
   Usando ***cv2.connectedComponentsWithStats***, se identifican componentes conectados en la imagen procesada. Para cada componente, se evalúan características como ancho, alto, relación de aspecto y área para determinar si corresponde a un dado. Los centroides y estadísticas de estos componentes se almacenan para análisis posterior.
4. Detección de movimiento:
   Si hay al menos cinco dados detectados en dos cuadros consecutivos, se utiliza la función ***se_mueve_no_se_mueve*** para identificar cuáles permanecen estacionarios comparando los centroides entre ambos cuadros.
5. Cálculo del valor de la cara visible:
   Para cada dado estacionario, se selecciona la región correspondiente al bounding box y se aísla en una máscara binaria que resalta los puntos en la cara. Se aplica nuevamente ***cv2.connectedComponentsWithStats*** para detectar cada punto. Para cada componente, se evalúan su área y relación de forma para determinar si corresponde a un punto válido. El número total de puntos detectados representa el valor de la cara visible del dado.
6. Anotación de la imagen:
   Se dibujan bounding boxes alrededor de los dados. Para aquellos estacionarios y con un valor de cara calculado, también se añade el valor como texto sobre el bounding box. Los dados que no cumplen con los criterios permanecen sin anotaciones específicas.
7. Salida:
   La función retorna la imagen procesada con las anotaciones visuales y una lista actualizada con los centroides de los dados detectados, útil para análisis en cuadros posteriores.

Implementación

El código principal implementa un sistema para procesar un video y analizar las tiradas de dados presentes en él. El objetivo es identificar los dados, determinar si están en movimiento o en reposo, calcular el valor de las caras visibles y generar un nuevo video con anotaciones visuales sobre los dados estacionarios. Este análisis se realiza cuadro por cuadro, garantizando que los resultados se actualicen dinámicamente según la información obtenida en cada etapa del procesamiento.

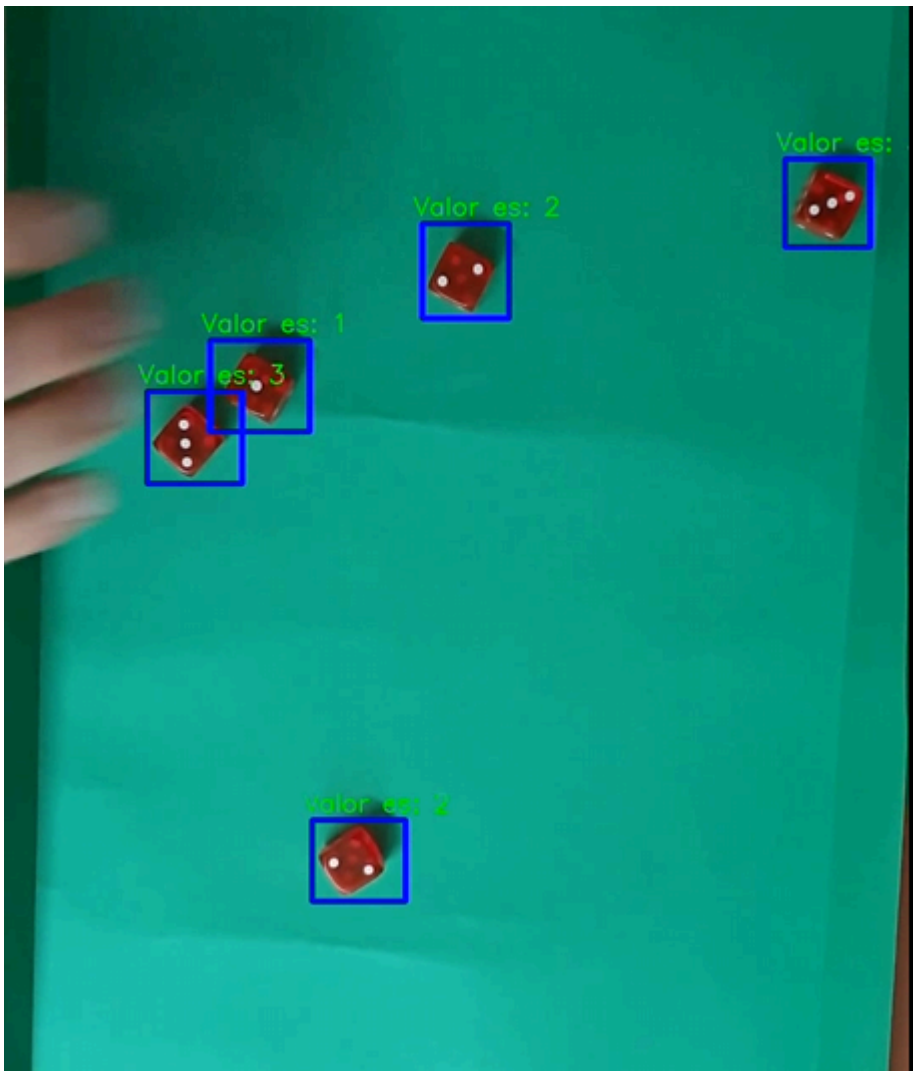
En primer lugar, se abre el archivo de video utilizando OpenCV, extrayendo parámetros clave como la resolución, los cuadros por segundo (FPS) y el número total de cuadros. Estos datos se utilizan para configurar tanto el análisis como la creación del video de salida, asegurando que conserve las mismas características técnicas que el original. A continuación, el sistema detecta automáticamente la región útil del cuadro (donde se encuentran los dados) mediante el análisis del canal rojo en una columna central, eliminando del análisis aquellas áreas que no contienen información relevante.

El análisis de cada cuadro recortado se realiza a través de la función procesar_imagenes. En el caso de los dados estacionarios, se evalúa su cara visible para determinar su valor, y esta información se anota directamente sobre el cuadro. Los dados detectados son resaltados con bounding boxes y etiquetas que indican su valor.

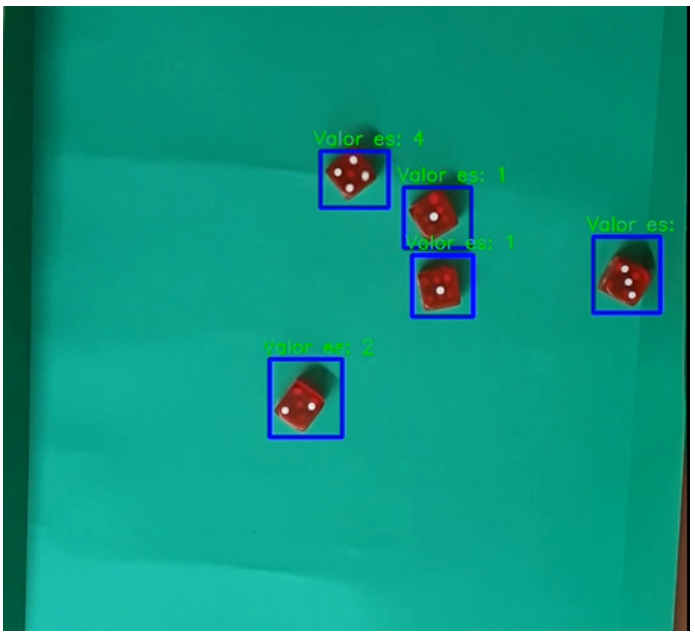
Finalmente, el cuadro procesado se reintegra al video completo y se escribe en el archivo de salida. Una vez que se procesan todos los cuadros o se encuentra un final abrupto en el video, los recursos se liberan adecuadamente para garantizar el cierre seguro de los archivos. Este enfoque permite analizar automáticamente el contenido del video y generar un resultado final visualmente informativo, útil para aplicaciones en análisis de juegos o experimentos donde se requiere seguimiento preciso de objetos.

Resultados

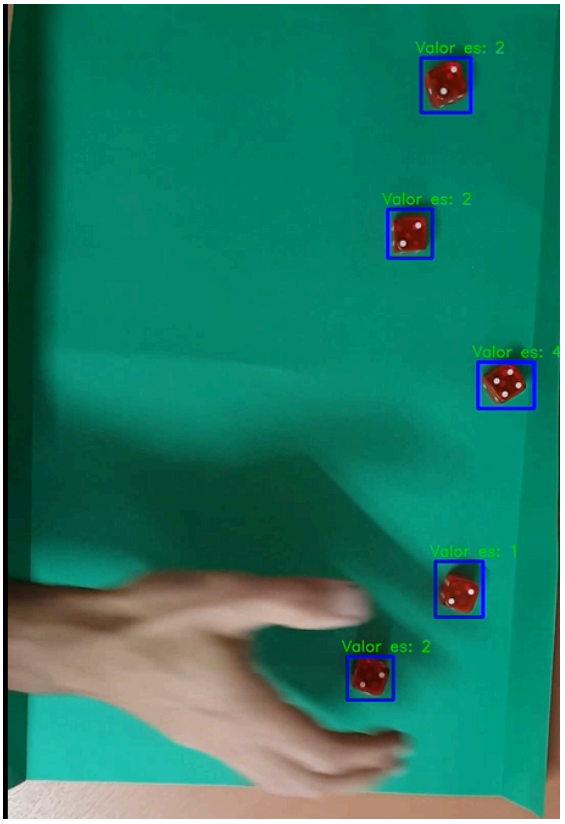
imgvid2


Valor es:
Valor es: 2
Valor es: 1
Valor es: 3
Valor es: 2

imgvid1



imgvid3



imgvid4