

UNIVERSIDAD NACIONAL DE ROSARIO

FACULTAD DE CIENCIAS EXACTAS, INGENIERÍA Y AGRIMENSURA

Tecnicatura Universitaria en Inteligencia Artificial

Procesamiento de Imágenes

Trabajo Práctico N° 3: Cinco Dados

Docentes:

Gonzalo Sad

Juan Manuel Calle

Joaquín Allione

Estudiantes:

Alejandro Armas

Facundo Ferreira Dacámara

Gabriel Soda

14 de Diciembre de 2025

Introducción

El presente trabajo práctico tiene como objetivo el desarrollo de algoritmos para detectar y analizar automáticamente tiradas de dados en secuencias de video.

Ejercicio:

Se propone el análisis de secuencias de video denominadas `tirada_<id>.mp4`, las cuales muestran tiradas de 5 dados en movimiento.

Los objetivos específicos son:

1. **Detección de frames estáticos:** Desarrollar un algoritmo que identifique automáticamente los frames donde los dados se encuentran detenidos después de una tirada.
2. **Lectura de valores:** Determinar el número obtenido en cada dado y mostrar los resultados por terminal.
3. **Generación de videos anotados:** Crear videos de salida donde los dados en reposo aparezcan con:
 - Bounding box asociado
 - Nombre identificatorio
 - Valor obtenido en la cara superior

Nota: El script debe informar y mostrar los resultados en cada una de las etapas de procesamiento.

Metodología y Resultados

2.1 Identificación de momentos estáticos

El primer desafío consiste en detectar automáticamente los frames en los cuales los dados se encuentran completamente detenidos tras una tirada. La Figura 1 ilustra un ejemplo de frame objetivo: los cinco dados en reposo sobre la superficie de juego.

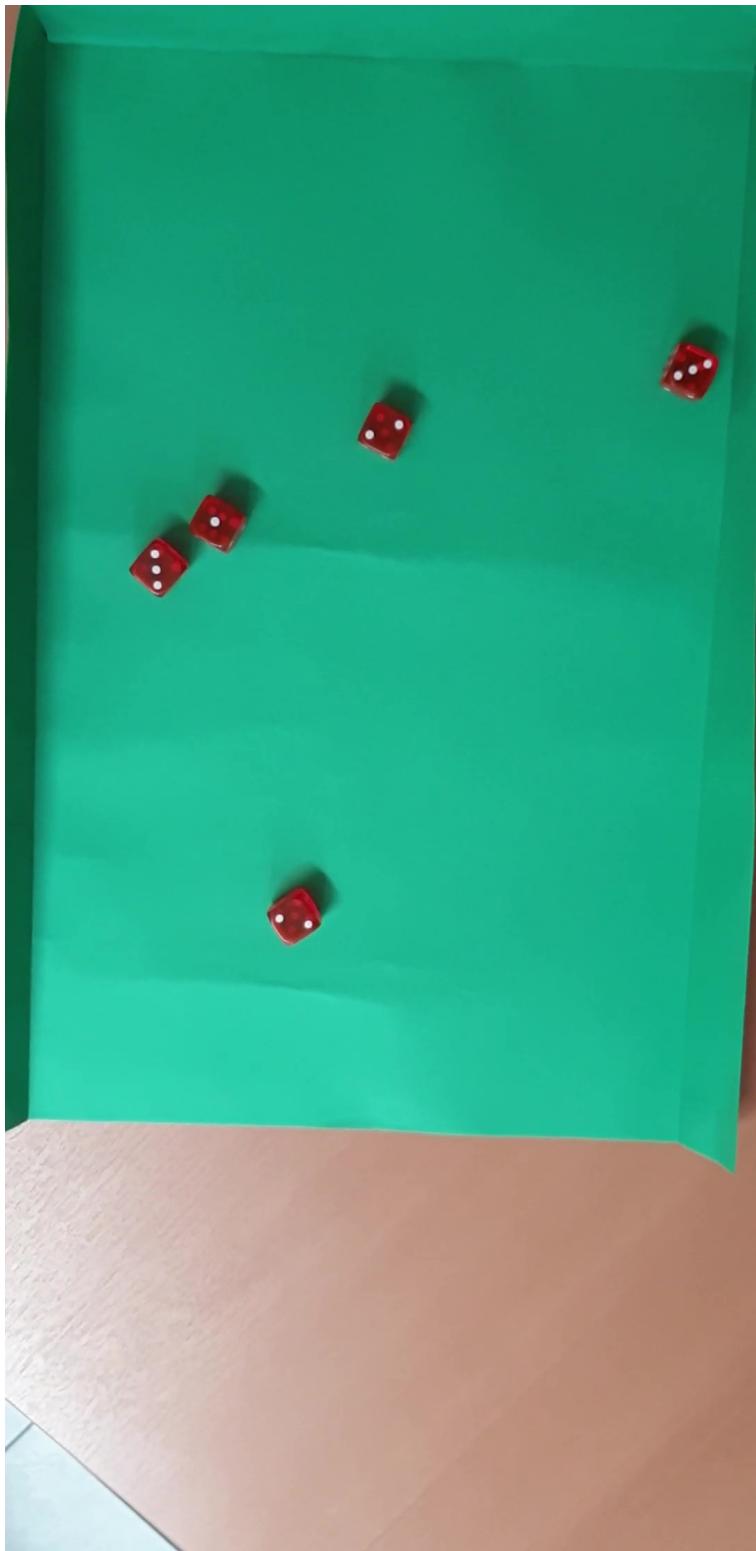


Figura 1: Frame representativo donde los dados se encuentran en estado estático después de una tirada.

2.2 Segmentación por color en espacio HSV

Para aislar los datos del fondo, se implementó una segmentación basada en el espacio de color HSV, aprovechando el contraste entre el rojo de los datos y el fondo verde de la superficie donde los mismos son lanzados.

Representación HSV

En Python ➡ H: [0 - 180] - S: [0 - 255] - V: [0 - 255]

```
hsv = cv2.cvtColor(rgb, cv2.COLOR_RGB2HSV)
```

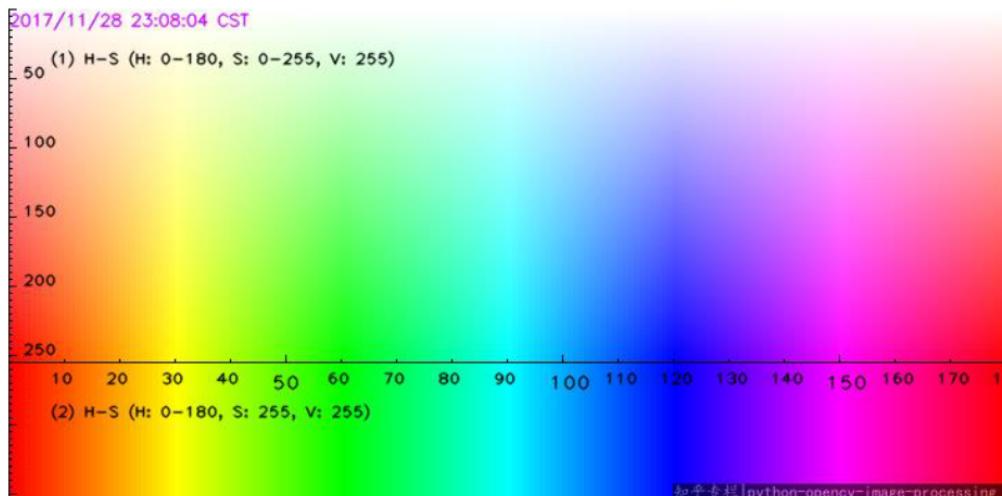


Figura 2: Rango de valores HSV utilizado como referencia para la detección del color rojo de los datos.

2.3 Calibración de parámetros de segmentación

Se desarrolló una herramienta interactiva para la calibración de los umbrales de segmentación, permitiendo ajustar dinámicamente los parámetros hasta obtener una máscara óptima.

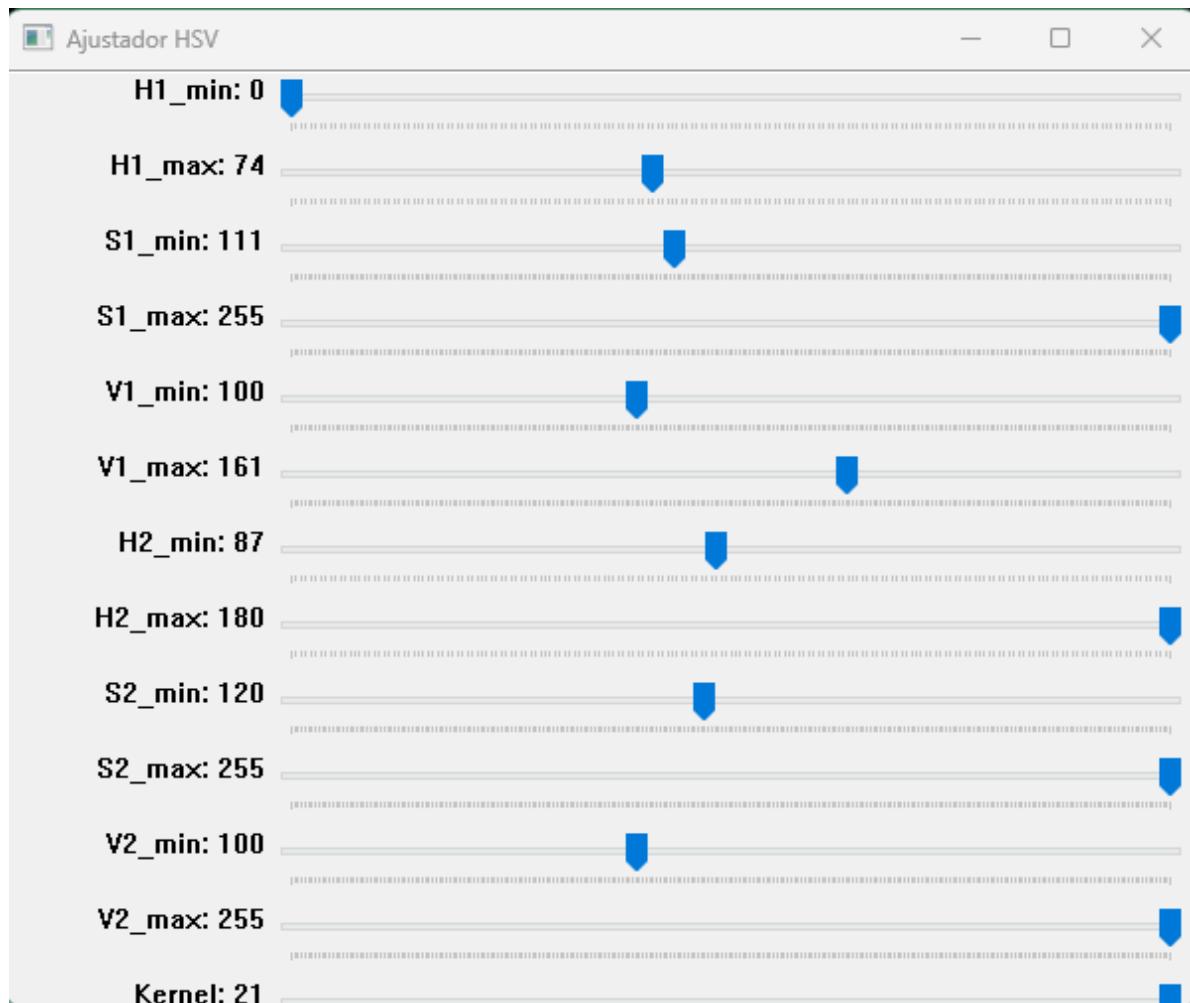


Figura 3: Interfaz interactiva para la definición y ajuste de parámetros de segmentación.

A continuación se muestra la máscara resultado que más cumplía con captar los datos

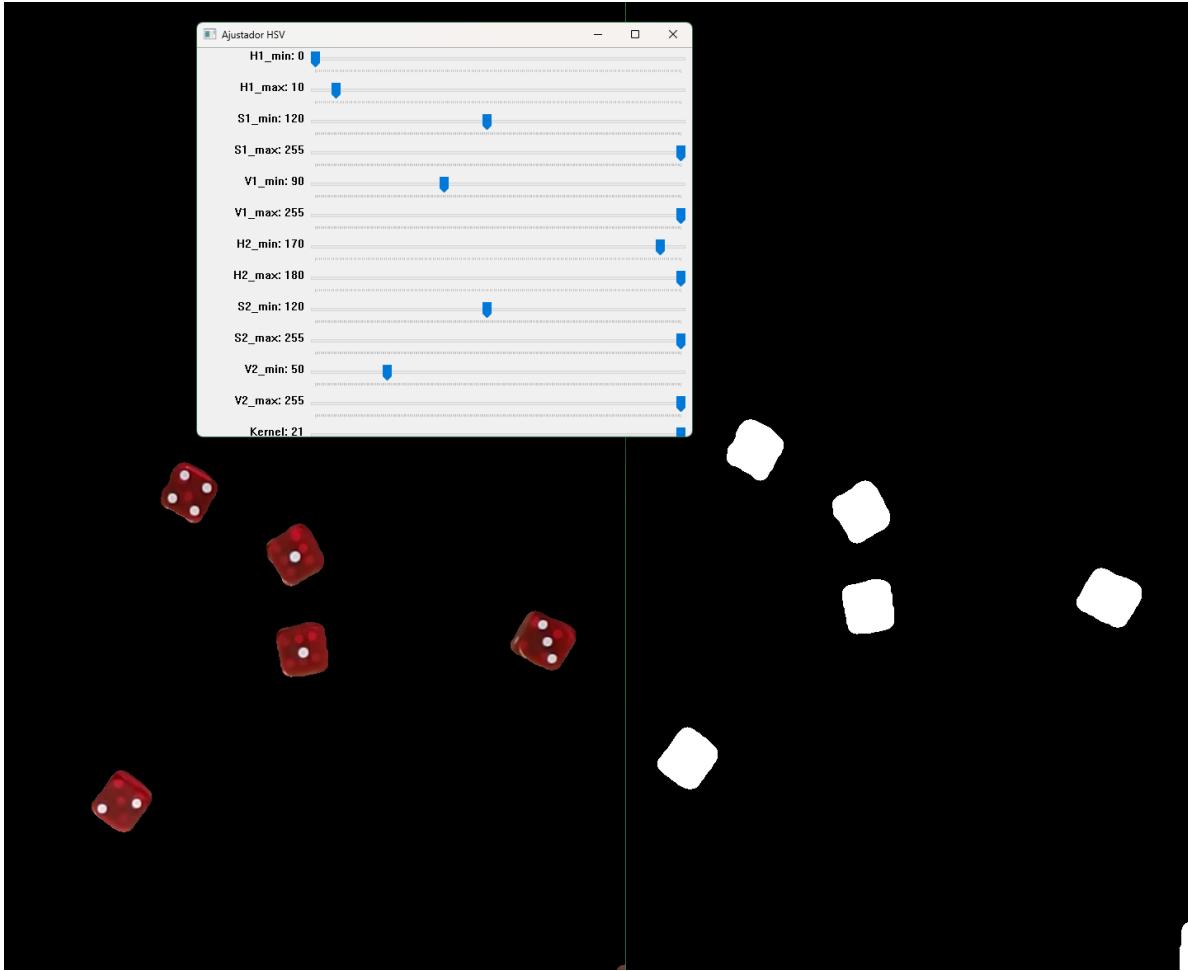


Figura 4: Máscara binaria obtenida con los parámetros calibrados, mostrando la correcta segmentación de los dados.

La máscara resultante con los parámetros optimizados logra aislar efectivamente el color rojo de los dados, eliminando el fondo verde y otros elementos de la escena.

2.4 Aplicación de la máscara y detección de componentes

La máscara fue aplicada sistemáticamente al conjunto de frames, permitiendo la identificación de regiones de interés mediante técnicas de análisis de contornos.

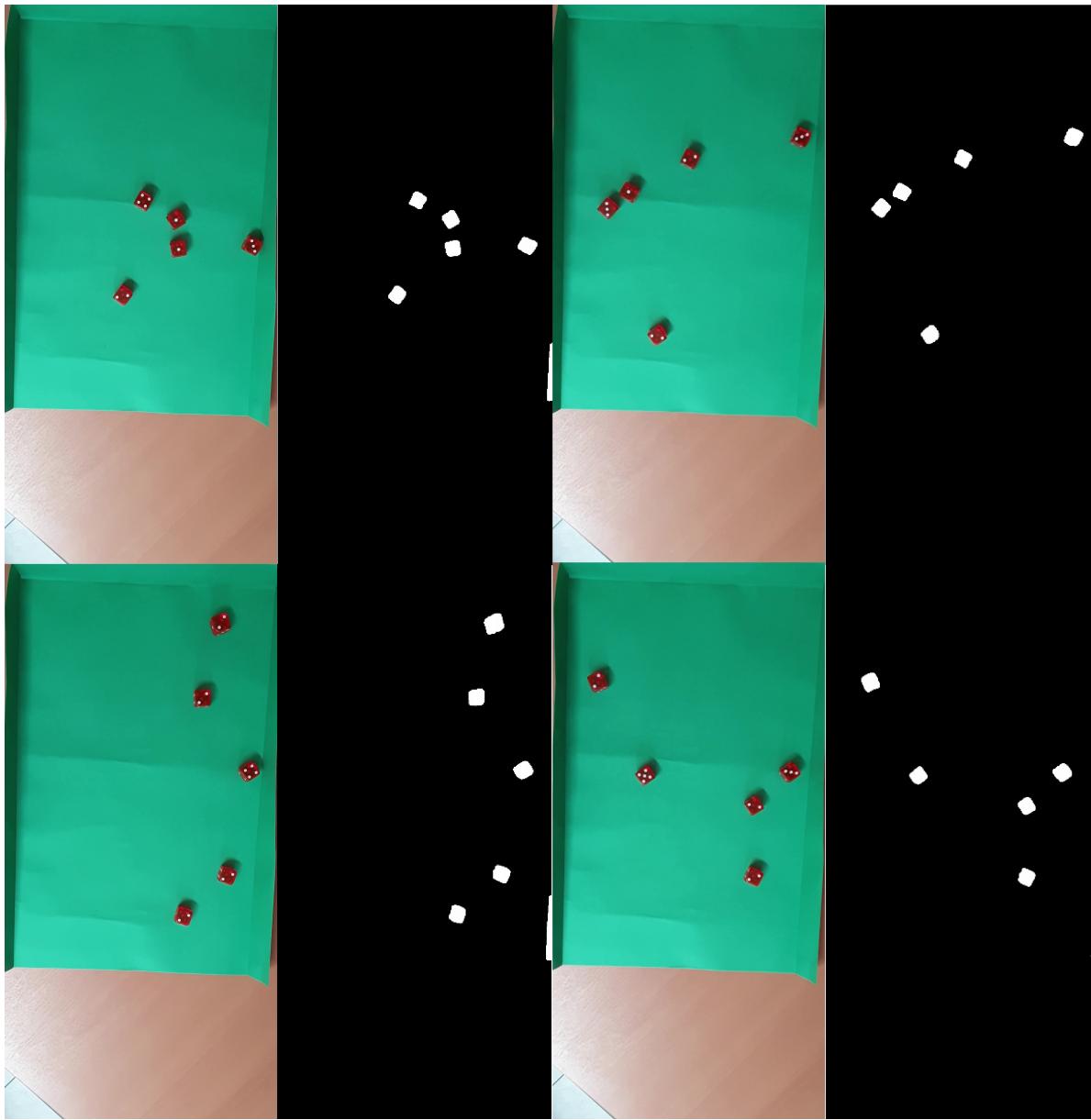


Figura 5: Máscara aplicada sobre frames del video, facilitando la detección de dados mediante FindContours y Connected Components with Stats para la identificación de los puntos (pips) en las caras de los dados.

2.5 Problemática del autofocus y análisis estadístico de movimiento

Durante el procesamiento de la secuencia `tirada_2.mp4`, se identificó una complicación técnica significativa: el sistema de autofocus de la cámara genera ajustes continuos incluso cuando los datos se encuentran estáticos. Este comportamiento introduce variaciones en la imagen que dificultan la detección confiable de frames en reposo mediante métodos convencionales de diferencia entre frames.

Para caracterizar este fenómeno, se realizó un análisis estadístico cuantitativo del cambio entre frames consecutivos, evaluando métricas de variación a lo largo de toda la secuencia.

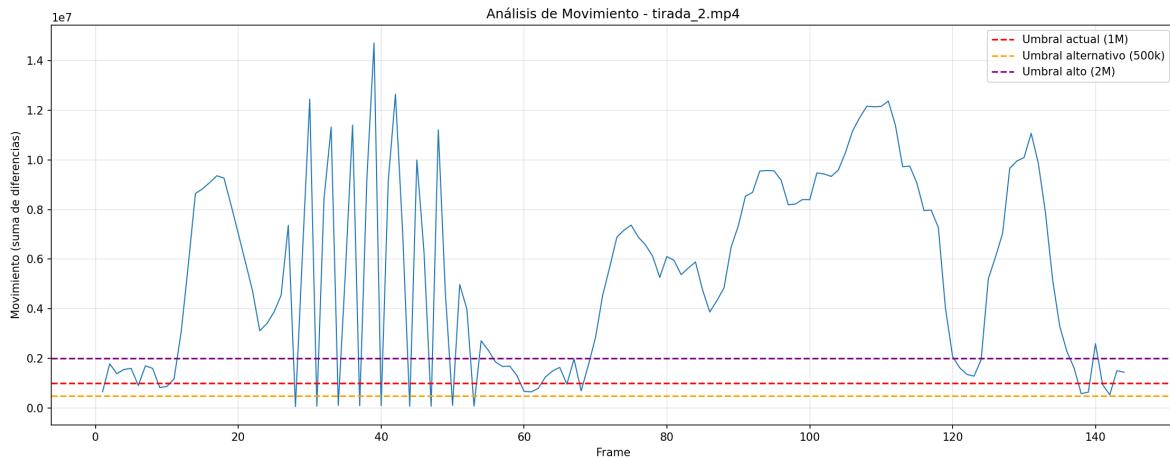


Figura 6: Análisis temporal de la variación entre frames en la secuencia `tirada_2.mp4`.

Los picos constantes evidencian las fluctuaciones introducidas por el autofocus, impidiendo la identificación clara de períodos de estabilidad incluso cuando los datos están en reposo. Este análisis permitió comprender las limitaciones del enfoque basado únicamente en diferencia de frames y motivó el desarrollo de estrategias alternativas para la detección robusta de momentos estáticos.

3. Solución Propuesta

3.1 Estrategia basada en máscaras robustas

Tras identificar las limitaciones del análisis de diferencias de frames completos, se adoptó un enfoque centrado en la **segmentación robusta mediante máscaras de color**. La premisa fundamental es que si la máscara roja es suficientemente precisa y limpia, las componentes conectadas de dicha máscara corresponderán directamente a los datos en la escena. A esta

decisión se arribó luego de observar la efectividad en el filtrado de colores rojos, generando una máscara que siempre coincide con los datos.

Este cambio de paradigma simplifica significativamente el problema: en lugar de detectar datos como objetos geométricos, se confía en que **las regiones rojas detectadas son los datos**, eliminando la necesidad de validaciones complejas de forma o tamaño.

3.2 Preprocesamiento y filtrado morfológico

Para garantizar la robustez de la máscara ante variaciones de iluminación y el autofocus, se implementó una pipeline de procesamiento que incluye:

3.2.1 Filtrado Gaussiano

```
frame = cv2.GaussianBlur(frame, (7, 7), 0)
```

El filtrado Gaussiano con kernel 7×7 demostró ser **la optimización más crítica** del sistema. Este paso elimina ruido de alta frecuencia y suaviza variaciones menores causadas por el autofocus, estabilizando significativamente las máscaras resultantes.

3.2.2 Segmentación en HSV con rangos duales

```
lower_red1 = np.array([0, 120, 100])
upper_red1 = np.array([5, 255, 255])

lower_red2 = np.array([175, 120, 100])
upper_red2 = np.array([180, 255, 255])
```

Dado que el rojo se encuentra en ambos extremos del espectro de matiz en HSV, se utilizan dos rangos complementarios que luego se combinan mediante operación OR.

3.2.3 Operaciones morfológicas

```
kernel_close = np.ones((11, 11), np.uint8)
mask_solid = cv2.morphologyEx(mask_red, cv2.MORPH_CLOSE, kernel_close)

kernel_open = np.ones((5, 5), np.uint8)
mask_solid = cv2.morphologyEx(mask_solid, cv2.MORPH_OPEN, kernel_open)
```

- **Cierre (11×11):** Une regiones rojas cercanas y elimina los puntos blancos (pips) del interior de los datos, creando blobs sólidos.
- **Apertura (5×5):** Elimina ruido residual de la mesa sin afectar significativamente las regiones de datos.

3.3 Optimización mediante reducción de resolución

Un aspecto fundamental del diseño es el **procesamiento a resolución reducida** (factor 4) para las tareas de detección, reservando la resolución completa únicamente para la identificación de valores de los datos (una vez segmentados).

```
frame = cv2.resize(frame_hd, dsize=(int(width/4), int(height/4)))
```

Ventajas: - Aceleración significativa del procesamiento (reducción de $\sim 16 \times$ en píxeles) - Menor consumo computacional para operaciones morfológicas - Diferencias entre frames más estables ante variaciones menores

Compensación de precisión: - Padding de 5 píxeles en bounding boxes - Escalado $4 \times$ de coordenadas para volver a resolución original - Identificación de valores en HD sobre ROIs recortados

3.4 Criterio de estabilidad

```
def diferencia_frames(frame1, frame2):
    delta = cv2.absdiff(frame1, frame2)
    cambios_pixels = cv2.countNonZero(delta)
    return np.sum(cambios_pixels)
```

Se considera que los datos están estáticos cuando: - La diferencia entre máscaras consecutivas es **50 píxeles** - Esta condición se debe mantener durante **>10 frames consecutivos**

Justificación de parámetros:

Parámetro	Valor	Justificación
Umbral de diferencia	50 píxeles	Tolerancia al autofocus del video <code>tirada_2.mp4</code> . Valores menores (20-40) funcionan en videos sin autofocus pero fallan en este caso crítico.

Parámetro	Valor	Justificación
Frames consecutivos	10 frames	A 30 fps 0.33s. Elimina falsos positivos por movimientos transitorios (e.g., manos que se retiran o se cruzan para recoger los dados).

Flujo de procesamiento optimizado

```

while cap.isOpened():
    ret, frame_hd = cap.read()
    if not ret: break
    frame = cv2.resize(frame_hd, dsize=(int(width/4), int(height/4)))
    frame_count += 1
    frame_mask = mascara_roja(frame)

    if not frame_mask.any():
        for f in buffer_frames:
            out.write(f)
        buffer_frames = []
        out.write(frame_hd)
        continue

    if diferencia_frames(frame_prev_mask, frame_mask) <= 50: # Frame
        similar al anterior
        frame_prev_mask = frame_mask.copy()
        if frames_permitidos < 5:
            frames_permitidos += 1
        buffer_frames.append(frame_hd)

        if len(buffer_frames) == 5:
            frame_estatico_mask = frame_mask.copy() # Guardar máscara de
un frame 'del medio'

    elif frames_permitidos and len(buffer_frames) > 5: # Frames
        permitidos
            # No copia la mascara actual, sigue usando la del ultimo frame
            similar.
            frames_permitidos -= 1
            buffer_frames.append(frame_hd)

else: # Frame diferente al anterior y no hay frames permitidos

```

```

frame_prev_mask = frame_mask.copy()

if len(buffer_frames) > 10: # Asume datos quietos. Procesa buffer
    con anotaciones
    print(' Dados quietos entre frames', frame_count -
          len(buffer_frames), 'y', frame_count - 1)
    procesar_frames_estaticos(buffer_frames, frame_estatico_mask,
                               out)

else: # Escribe sin procesar.
    for f in buffer_frames:
        out.write(f)

buffer_frames = []
out.write(frame_hd)

```

Características clave:

El enfoque propuesto provee un uso eficiente de los recursos mediante un procesamiento optimizado del video.

Los frames de las tiradas se leen, se procesan y se escriben en tiempo real en caso de no contener ningún dato o detectar movimiento, y se almacenan en un buffer en caso de detectar estaticidad. Al flushear el buffer (Porque se rompió la cadena de frames similares) se determina si son o no los *frames de datos quietos* según la cantidad de elementos. (Si hay más de 10 elementos en el buffer asumimos que lo son)

El algoritmo cuenta también con una mejora simple pero eficaz: Por cada frame similar consecutivo, permite agregar un frame no similar al buffer, esto fué necesario para tolerar pequeños ruidos como ajustes de foco de la cámara. A esto lo llamamos “**Frame Permitido**”

Se requiere que hayan al menos 3 frames similares REALES en el buffer antes de permitir introducir el primer frame permitido.

3.5 Segmentación y lectura de valores

3.5.1 Estrategia de cálculo único

Como ya se mencionó, si se detecta que el buffer contiene más de 10 elementos, se asume que en esos frames están los datos quietos y se calculan **Componentes conectadas una sola vez** sobre la máscara de un frame estático representante, se escalan las coordenadas a la resolución original y se guardan junto con el valor del dato. Luego a cada *frame de datos quietos* se le escribe dicha información, segmentando y clasificando cada dato.

```

def procesar_frames_estaticos(buffer_frames, frame_estatico_mask, out):
    """
    Procesa una lista de frames estáticos de datos quietos.
    Escribe los frames procesados en el video de salida.
    """
    PADDING = 5
    dados = []

    num, labels, stats, centroids =
    cv2.connectedComponentsWithStats(frame_estatico_mask, connectivity=8)

    for i in range(1, num):
        x, y, w, h, area = stats[i]
        x -= PADDING
        y -= PADDING
        w += 2 * PADDING
        h += 2 * PADDING

        x *= 4; y *= 4; w *= 4; h *= 4 # Volver a resolución original hd

        dado_roi = buffer_frames[0][y:y+h, x:x+w]
        valor = identificar_valor(dado_roi)
        print(f'    Dado identificado. Valor: {valor}')
        dados.append((x, y, w, h, valor))

    print(f' Generando video anotado con {len(dados)} datos identificados.')

    for f in buffer_frames: # Anotar y guardar cada frame de datos quietos
        for (x, y, w, h, valor) in dados:
            cv2.rectangle(f, (x, y), (x+w, y+h), (255, 255, 0), 3)
            cv2.putText(f, str(valor), (x, y-10),
                       cv2.FONT_HERSHEY_SIMPLEX, 1.5, (0, 255, 255), 3)
        out.write(f)

```

Ventajas de este enfoque: - Evita recalcular componentes conectadas frame a frame - Elimina inconsistencias en detección (e.g., cambios de valor cuando una mano pasa por delante) - Garantiza estabilidad visual en el video anotado - Reduce el costo computacional

3.5.2 Detección de puntos blancos

```

def identificar_valor(dado_roi):
    roi_hsv = cv2.cvtColor(dado_roi, cv2.COLOR_BGR2HSV)

    # Umbralización para blanco
    lower_white = np.array([0, 0, 180])
    upper_white = np.array([180, 60, 255])
    mask_white = cv2.inRange(roi_hsv, lower_white, upper_white)

    # Limpieza con kernel elíptico
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))
    mask_white = cv2.morphologyEx(mask_white, cv2.MORPH_OPEN, kernel)

    # Conteo de componentes conectadas
    n_labels, labels, stats, centroids = cv2.connectedComponentsWithStats(
        mask_white, connectivity=8
    )

    return n_labels - 1 # Excluir fondo

```

La detección de valores se realiza en **resolución completa** para maximizar la precisión. Se utiliza:

- Segmentación HSV con rangos amplios de blanco
- Morfología de apertura con kernel elíptico (respeta la forma circular de los pips)
- Conteo de componentes conectadas (cada punto = una componente)

4. Resultados

4.1 Análisis de desempeño por video

El algoritmo fue evaluado sobre cuatro secuencias de video con características diversas:

Video	Frames estáticos detectados	Datos identificados	Valores correctos
tirada_1.mp4	66 a 90	5	5/5
tirada_2.mp4	59 a 72	5	5/5
tirada_3.mp4	68 a 91	5	5/5
tirada_4.mp4	48 a 88	5	5/5

4.2 Resultados

Video 1:

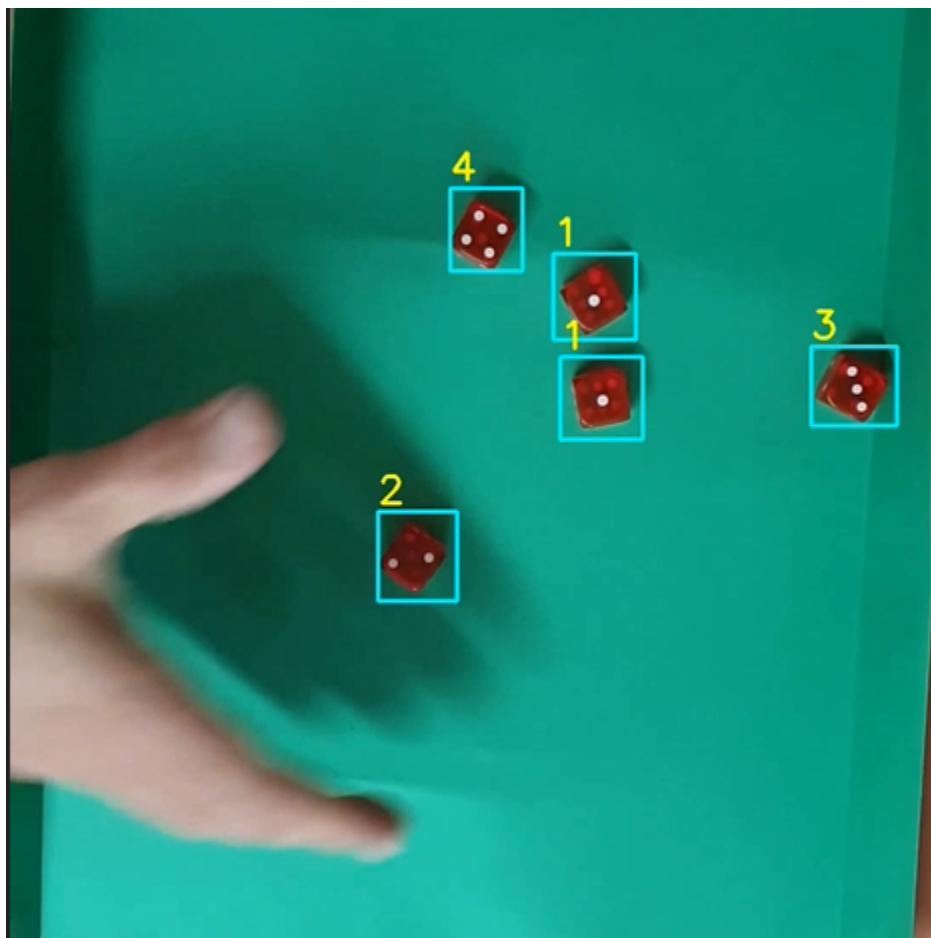


Figura 7: Frames anotados del video tirada_1.mp4 mostrando detección exitosa de los cinco dados con sus valores respectivos.

Se observa en la imagen que la robustez que provee la máscara se percibe en que únicamente son tomados los píxeles donde están los dados para las detecciones, permitiendo detectar dados en reposo incluso cuando la mano del operador se cruza para retirarlos. Lo mismo se repite en las siguientes capturas.

Video 2:

El video tirada_2.mp4 representa el caso más desafiante debido al autofocus de la cámara. El umbral de 50 píxeles fue calibrado específicamente para este escenario.

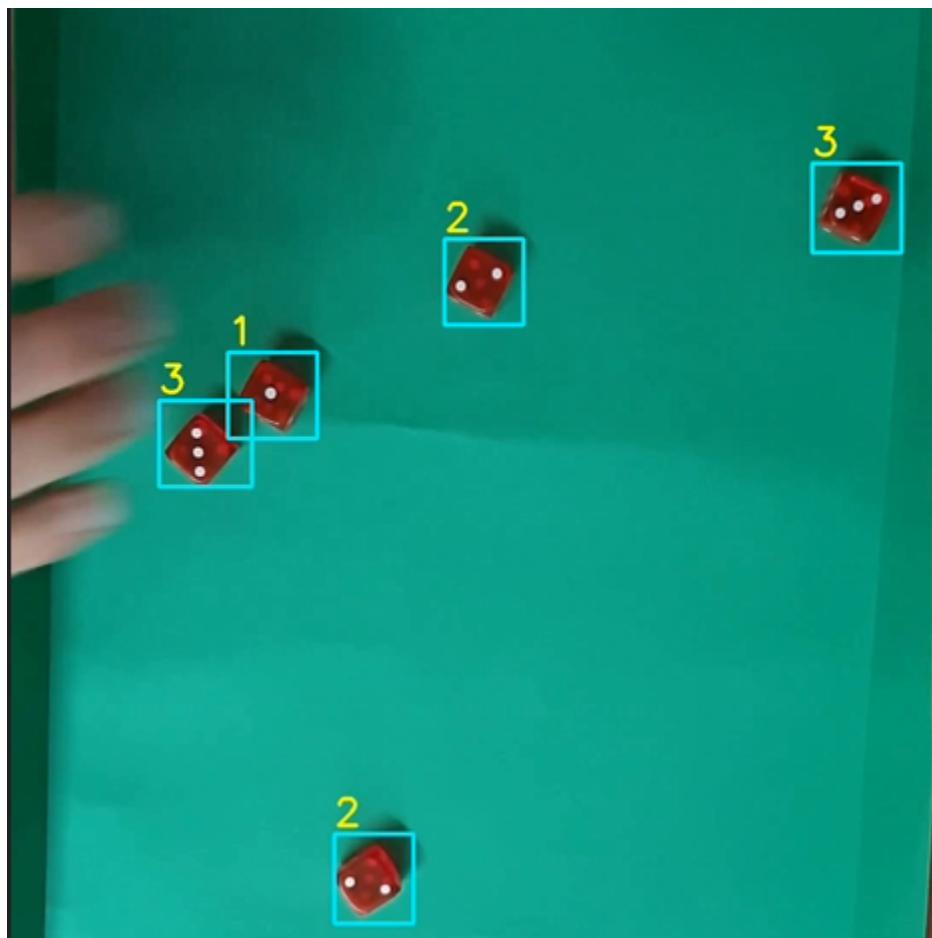


Figura 8: Detección robusta en tirada_2.mp4 a pesar de las variaciones introducidas por el autofocus.

Video 3:

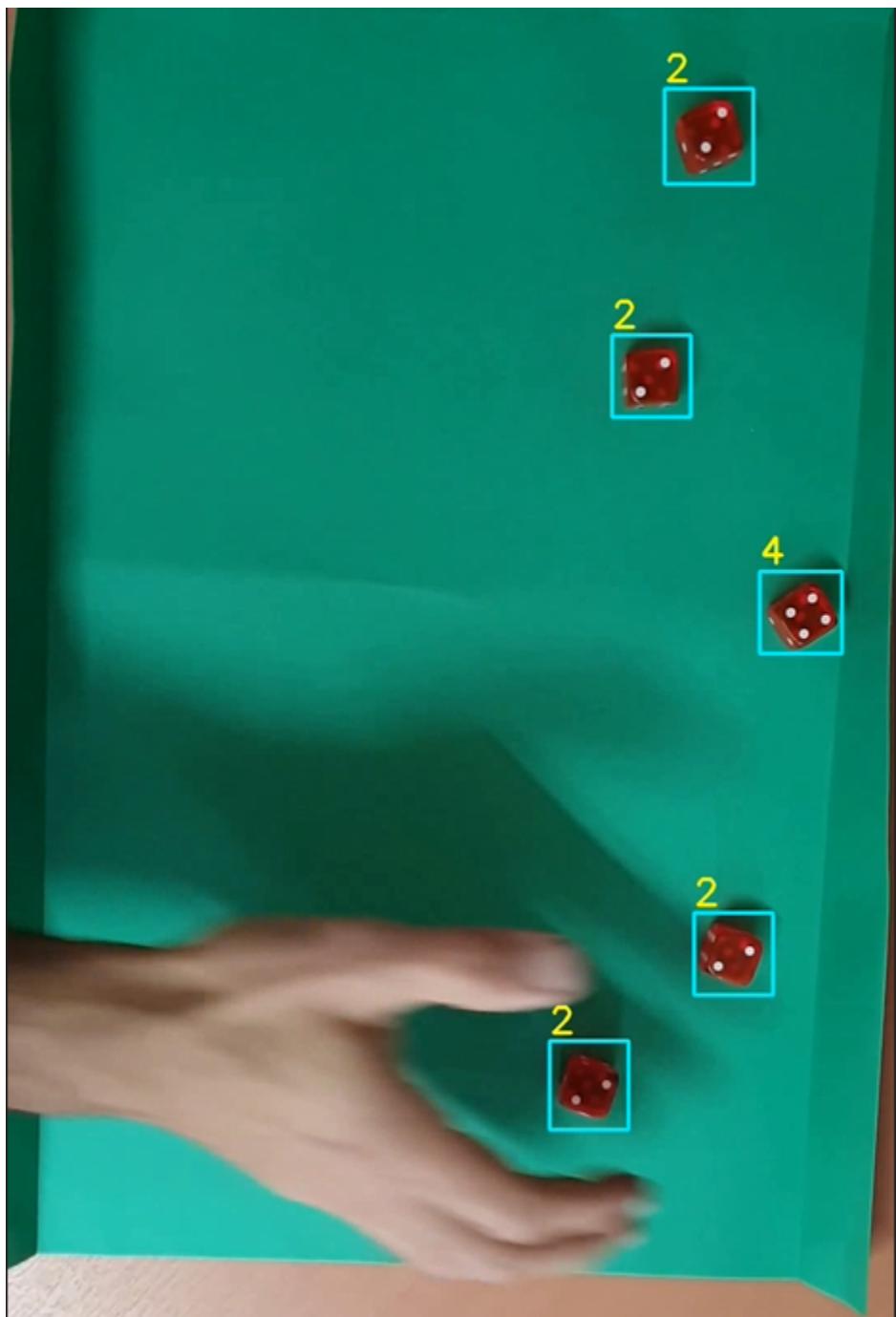


Figura 9: Resultados del video tirada_3.mp4 con bounding boxes y valores correctamente identificados.

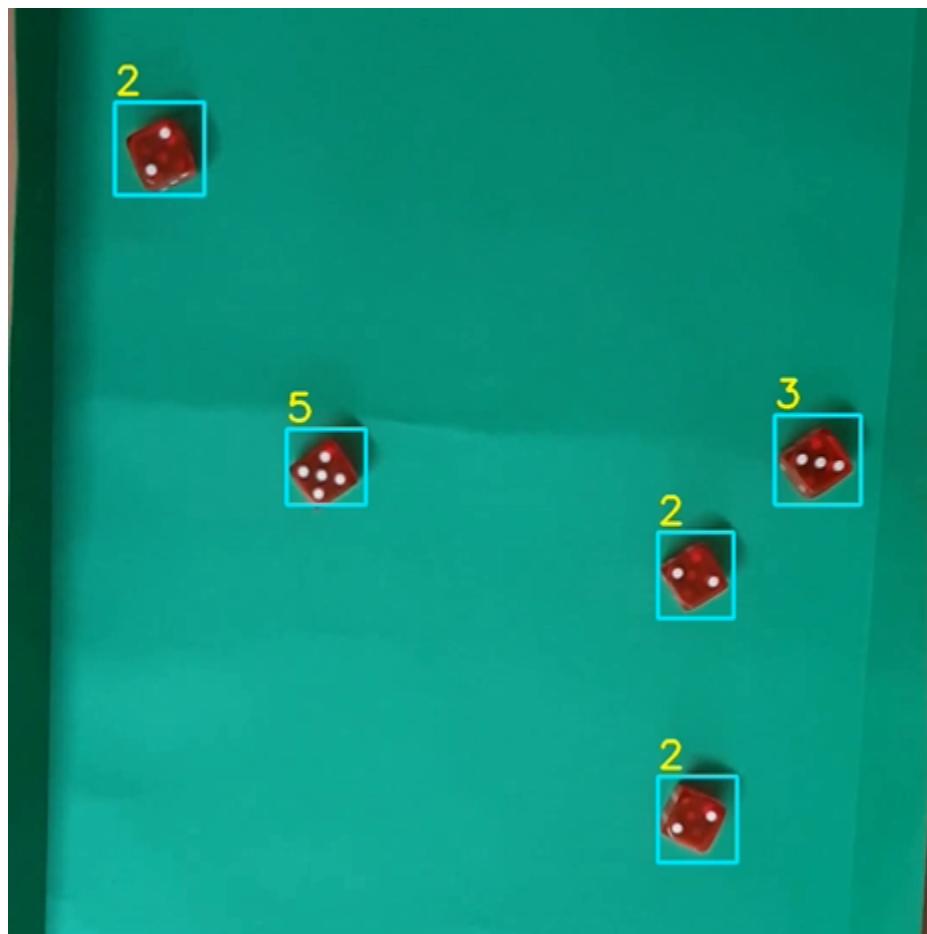


Figura 10: Procesamiento del video tirada_4.mp4 mostrando detección correcta hasta proximidad del final del video.

4.3 Ejemplo de salida por terminal

Se muestran los resultados detallados por la terminal:

```
Procesando video: tirada_1.mp4
Dados quietos entre frames 65 y 86
Dado identificado. Valor: 4
Dado identificado. Valor: 1
Dado identificado. Valor: 3
Dado identificado. Valor: 1
Dado identificado. Valor: 2
Generando video anotado con 5 dados identificados.
```

```
Video guardado: tirada_1-anotado.mp4
```

```
Procesando video: tirada_2.mp4
```

```
Datos quietos entre frames 56 y 72
```

```
    Dado identificado. Valor: 3
```

```
    Dado identificado. Valor: 2
```

```
    Dado identificado. Valor: 1
```

```
    Dado identificado. Valor: 3
```

```
    Dado identificado. Valor: 2
```

```
Generando video anotado con 5 datos identificados.
```

```
Video guardado: tirada_2-anotado.mp4
```

5.1 Fortalezas del enfoque propuesto

1. **Robustez ante autofocus:** La combinación de filtrado Gaussiano y umbral permisivo permite operar con variaciones de enfoque automático.
2. **Simplicidad conceptual:** Confiar en la calidad de la máscara simplifica significativamente la lógica de detección.
3. **Cálculo único de valores:** Evita inconsistencias temporales y reduce el costo computacional.

5.2 Limitaciones identificadas

1. **Parámetros hardcodeados:** El umbral de 50 píxeles y 10 frames consecutivos fueron ajustados empíricamente. También el ratio de reducción de resolución y los tamaños de kernel para las distintas operaciones aplicadas. Una calibración automática basada en características del video mejoraría la generalización.

6. Conclusiones

Se desarrolló exitosamente un sistema automático de detección y análisis de tiradas de datos en video, cumpliendo los tres objetivos propuestos:

1. **Detección automática de frames estáticos** mediante máquina de estados con criterio de 15 frames consecutivos con diferencia igual o mayor a 50 píxeles.
2. **Lectura correcta de valores** utilizando segmentación de pips en resolución HD con conteo de componentes conectadas.

3. Generación de videos anotados con bounding boxes y valores superpuestos sobre los datos detectados.

El enfoque basado en **máscaras robustas** demostró ser superior a métodos de detección geométrica directa, simplificando significativamente la arquitectura del sistema. Las optimizaciones implementadas (procesamiento a resolución reducida, escritura on-the-fly, cálculo único de valores) permiten escalabilidad y eficiencia computacional.

El principal desafío enfrentado—el autofocus automático de la cámara—fue resuelto mediante ajuste de parámetros y filtrado Gaussiano agresivo, validando la robustez del diseño propuesto ante condiciones no ideales de captura.

El sistema desarrollado es completamente funcional para el escenario propuesto y sienta las bases para extensiones futuras hacia detección multi-tirada, calibración adaptativa y generalización a diferentes configuraciones de color.