



Rodrigo Hernández Zavala

27 de mayo de 2025

22310215

Practica 7: Remoción de Ruido

◆ 1. Captura de video desde la cámara

```
python
CopiarEditar
cap = cv2.VideoCapture(0)
```

Esto abre la cámara (índice 0 = cámara por defecto). Cada fotograma capturado es procesado dentro de un bucle.

◆ 2. Conversión de BGR a HSV

```
python
CopiarEditar
hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
```

OpenCV carga las imágenes en formato **BGR** (Blue, Green, Red), pero para detectar colores es mucho más fácil usar el espacio de color **HSV**:

- **H (Hue)**: tono de color (0-179 en OpenCV)
- **S (Saturation)**: saturación del color
- **V (Value)**: brillo

HSV separa mejor el color de la iluminación, ideal para segmentar colores específicos.

◆ 3. Creación de máscaras por color

```
python
CopiarEditar
mask_red = cv2.inRange(hsv, lower_red1, upper_red1) + cv2.inRange(hsv,
lower_red2, upper_red2)
```

`cv2.inRange()` crea una **máscara binaria**:

- Píxeles que están dentro del rango HSV → blanco (255)
- Píxeles fuera del rango → negro (0)

Como el color rojo está **dividido en dos rangos** en HSV (cerca de 0° y 180°), combinamos dos máscaras con +.

◆ 4. Simulación de Falsos Positivos (F+) y Falsos Negativos (F-)

- **F+ = Ruido blanco simulado**

```
python
CopiarEditar
cv2.circle(fplus, (100, 100), 10, 255, -1)
```

Se dibujan **círculos blancos** en la máscara binaria, simulando **errores de detección** donde hay "algo" aunque no debería.

- **F- = Huecos negros simulados**

```
python
CopiarEditar
cv2.rectangle(fminus, (300, 200), (320, 220), 0, -1)
```

Se dibujan **rectángulos negros** sobre una región blanca para simular **áreas omitidas incorrectamente**.

◆ 5. Aplicación de filtros morfológicos

Estos se aplican sobre las máscaras **con ruido (F+ o F-)**.

Filtro	Función principal
Erosión	Elimina manchas pequeñas blancas (reduce objetos)
Dilatación	Rellena huecos, expande zonas blancas
Opening	Erosión + dilatación → elimina F+ (ruido blanco)
Closing	Dilatación + erosión → rellena F- (huecos negros)
TopHat	Extrae detalles claros sobre fondo oscuro
BlackHat	Extrae detalles oscuros sobre fondo claro

Ejemplo:

```
python
CopiarEditar
cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
```

◆ 6. Visualización en mosaico con etiquetas

Para entender qué filtro estás viendo:

- Cada máscara procesada es convertida a BGR (para mostrar color).
 - Se le agrega texto con `cv2.putText()` para indicar el nombre del filtro.
 - Se combinan con `np.hstack()` y `np.vstack()` en un mosaico de 2 filas \times 3 columnas.
-

◆ 7. Resultado final

Se muestran **cuatro ventanas**:

1. `original`: la imagen de la cámara.
2. `Rojo - F+`: efectos aplicados a la máscara roja con falsos positivos.
3. `Verde - F-`: efectos aplicados a la máscara verde con huecos.
4. `Azul - F+`: lo mismo para azul con ruido blanco.

Cada una muestra:

- `TopHat`
 - `BlackHat`
 - Dilatación
 - Erosión
 - `Opening`
 - `Closing`
-

📌 ¿Y el filtro "lineal"?

Los **filtros lineales** como **GaussianBlur** o **media** no están en este ejemplo, pero son otra forma de reducción de ruido (por promedio de píxeles). Aquí usamos filtros **morfológicos**, que son más precisos para imágenes binarias (como las máscaras).

Código:

```
import cv2  
  
import numpy as np
```

```
# Inicializar cámara
cap = cv2.VideoCapture(0)

if not cap.isOpened():

    print("Error: No se puede acceder a la cámara.")

    exit()

# Crear kernel morfológico
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (100, 100))

# Aplicar filtros morfológicos + blur
def aplicar_morfologia(mask):

    blur = cv2.GaussianBlur(mask, (5, 5), 0) # Filtro lineal

    return {

        'Blur': blur,

        'TopHat': cv2.morphologyEx(blur, cv2.MORPH_TOPHAT, kernel),

        'BlackHat': cv2.morphologyEx(blur, cv2.MORPH_BLACKHAT, kernel),

        'Dilatacion': cv2.dilate(blur, kernel, iterations=1),

        'Erosion': cv2.erode(blur, kernel, iterations=1),

        'Opening': cv2.morphologyEx(blur, cv2.MORPH_OPEN, kernel),

        'Closing': cv2.morphologyEx(blur, cv2.MORPH_CLOSE, kernel)

    }

# Etiquetar cada imagen con su nombre y redimensionarla a tamaño estándar
def etiquetar(img, texto, size=(320, 240)):

    img_resized = cv2.resize(img, size)

    img_color = cv2.cvtColor(img_resized, cv2.COLOR_GRAY2BGR)
```

```
cv2.putText(img_color, texto, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,  
          0.9, (0, 255, 255), 2, cv2.LINE_AA)  
  
return img_color
```

```
# Simular ruido de F+ y F-
```

```
def agregar_ruido_falsos(mask, color_id):  
  
    fplus = mask.copy()  
  
    fminus = mask.copy()  
  
    cv2.circle(fplus, (100 + color_id * 60, 100), 10, 255, -1)  
  
    cv2.circle(fplus, (200 + color_id * 60, 150), 10, 255, -1)  
  
    cv2.rectangle(fminus, (300, 200), (320, 220), 0, -1)  
  
    cv2.rectangle(fminus, (400, 250), (420, 270), 0, -1)  
  
    return fplus, fminus
```

```
# Construir mosaico con etiquetas
```

```
def mosaico_color(diccionario):  
  
    etiquetas = ['Blur', 'TopHat', 'BlackHat', 'Dilatacion', 'Erosion', 'Opening', 'Closing']  
  
    etiquetadas = [etiquetar(diccionario[nombre], nombre) for nombre in etiquetas]  
  
    fila1 = np.hstack(etiquetadas[:4]) # 4 imágenes  
  
    fila2 = np.hstack(etiquetadas[4:]) # 3 imágenes  
  
    # Agregar imagen negra para emparejar  
  
    negro = np.zeros_like(etiquetadas[0])  
  
    fila2 = np.hstack([fila2, negro]) # completar con imagen vacía  
  
    return np.vstack((fila1, fila2))
```

```
# Redimensionar ventanas
```

```
def escalar(img, ancho=1280, alto=480):
    return cv2.resize(img, (ancho, alto))

while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame = cv2.resize(frame, (640, 480))
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Máscaras HSV por color
    mask_red = cv2.inRange(hsv, np.array([0, 120, 70]), np.array([10, 255, 255])) + \
               cv2.inRange(hsv, np.array([170, 120, 70]), np.array([180, 255, 255]))
    mask_green = cv2.inRange(hsv, np.array([40, 40, 40]), np.array([70, 255, 255]))
    mask_blue = cv2.inRange(hsv, np.array([100, 150, 0]), np.array([140, 255, 255]))

    # Simular errores F+ y F-
    red_fplus, _ = agregar_ruido_falsos(mask_red, 0)
    _, green_fminus = agregar_ruido_falsos(mask_green, 1)
    blue_fplus, _ = agregar_ruido_falsos(mask_blue, 2)

    # Aplicar filtros
    red_fx = aplicar_morfologia(red_fplus)
    green_fx = aplicar_morfologia(green_fminus)
    blue_fx = aplicar_morfologia(blue_fplus)
```

```

# Mostrar resultados en 4 ventanas

cv2.imshow("Original", cv2.resize(frame, (960, 540)))

cv2.imshow("Rojo - F+ (con filtros)", escalar(mosaico_color(red_fx)))

cv2.imshow("Verde - F- (con filtros)", escalar(mosaico_color(green_fx)))

cv2.imshow("Azul - F+ (con filtros)", escalar(mosaico_color(blue_fx)))

# Presionar 'q' para salir

if cv2.waitKey(1) & 0xFF == ord('q'):

    break

cap.release()

cv2.destroyAllWindows()

```

Imagenes del funcionamiento:



