

Rodrigo Hernández Zavala

8 de junio de 2025

Ing. Mecatrónica

22310215

Igualdades con rotación y reducción de fondo

Muy buena pregunta. El código de **extracción de fondo en video usando movimiento** tiene como **objetivo detectar qué partes de la imagen están cambiando con el tiempo**, es decir, **distinguir qué es fondo (estático) y qué es figura (en movimiento)**.

🎯 ¿Qué hace exactamente el código?

🔍 1. Captura un frame de referencia (el fondo inicial)

```
python
CopiarEditar
ret, frame_ref = cap.read()
gray_ref = cv2.cvtColor(frame_ref, cv2.COLOR_BGR2GRAY)
gray_ref = cv2.GaussianBlur(gray_ref, (21, 21), 0)
```

- Este frame se considera el **fondo estático**.
 - Se convierte a escala de grises y se suaviza para reducir ruido.
-

⌚ 2. Lee nuevos frames del video (en tiempo real)

```
python
CopiarEditar
ret, frame = cap.read()
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (21, 21), 0)
```

- Convierte el nuevo frame a escala de grises y lo suaviza igual que el fondo.
-

⚖️ 3. Resta el fondo del nuevo frame

```
python
CopiarEditar
delta = cv2.absdiff(gray_ref, gray)
```

- Calcula la **diferencia absoluta** entre el fondo y el nuevo frame.
 - Las áreas que **cambiaron** (es decir, donde hay movimiento) tendrán valores más altos.
-

👤 4. Umbraliza la diferencia

```
python
CopiarEditar
_, thresh = cv2.threshold(delta, 25, 255, cv2.THRESH_BINARY)
```

- Todo lo que **cambió significativamente** se convierte en blanco (255).
 - Lo que **no cambió** se convierte en negro (0).
 - Resultado: una **máscara binaria del movimiento**.
-

5. Mejora la máscara de movimiento

```
python
CopiarEditar
thresh = cv2.dilate(thresh, None, iterations=2)
```

- Diluye la máscara binaria para llenar huecos pequeños.
 - Hace que los objetos en movimiento se vean más sólidos.
-

6. Extrae solo los píxeles en movimiento

```
python
CopiarEditar
movimiento = cv2.bitwise_and(frame, frame, mask=thresh)
```

- Usa la máscara binaria para extraer **solo las zonas en movimiento** del frame original.
-

7. Muestra el resultado

```
python
CopiarEditar
cv2.imshow("Original", frame)
cv2.imshow("Movimiento Detectado", movimiento)
```

- Muestra el video normal y el video con **solo los objetos en movimiento**.
-

¿Qué puedes hacer con esto?

- Detección de personas u objetos en tiempo real.
- Segmentación de figura/fondo sin necesidad de fondo verde.

- Activar eventos o alarmas cuando algo se mueva.
 - Preprocesar imágenes para visión artificial.
-



Limitaciones

Limite	Descripción
Solo detecta movimiento	Si un objeto se queda quieto, desaparece.
Cambios de luz afectan	Luces encendidas o sombras pueden activarlo.
No identifica objetos	Solo detecta que "algo se mueve", no qué es.

Este código usa **visión por computadora** para detectar una figura (por ejemplo, un **triángulo**) dentro de otra imagen (por ejemplo, una escena con varias figuras geométricas), **aunque esté rotada**, usando técnicas de detección de características con **ORB** y **homografía**.



¿Qué hace el código paso a paso?

1. Carga dos imágenes

```
python
CopiarEditar
img_scene = cv2.imread("Figuras.jpg")
img_template = cv2.imread("triangulo.jpg")
```

- Figuras.jpg: imagen general que contiene varias figuras.
 - triangulo.jpg: figura recortada (template) que queremos encontrar dentro de Figuras.jpg.
-

2. Convierte ambas imágenes a escala de grises

```
python
CopiarEditar
gray_scene = cv2.cvtColor(img_scene, cv2.COLOR_BGR2GRAY)
gray_template = cv2.cvtColor(img_template, cv2.COLOR_BGR2GRAY)
```

Esto facilita la detección de bordes y esquinas sin preocuparse por el color.

3. Detecta puntos clave y descriptores con ORB

```
python
CopiarEditar
orb = cv2.ORB_create(nfeatures=2000)
kp1, des1 = orb.detectAndCompute(gray_template, None)
kp2, des2 = orb.detectAndCompute(gray_scene, None)
```

- ORB (Oriented FAST and Rotated BRIEF) busca **puntos clave** únicos (como esquinas).
 - También calcula **descriptores** que permiten comparar esos puntos entre imágenes.
-

4. Compara puntos entre template y escena

```
python
CopiarEditar
bf = cv2.BFMatcher(cv2.NORM_HAMMING)
matches = bf.knnMatch(des1, des2, k=2)
```

- Usa un **emparejador de fuerza bruta** para encontrar los puntos más similares entre ambas imágenes.
 - $k=2$ devuelve los dos mejores matches para cada punto.
-

5. Filtra las mejores coincidencias (Lowe's ratio test)

```
python
CopiarEditar
if m.distance < 0.75 * n.distance:
```

- Se descartan coincidencias ambiguas y se conservan solo las más confiables.
-

6. Calcula la transformación geométrica (homografía)

```
python
CopiarEditar
M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 8.0)
```

- Si hay suficientes matches (>10), se usa **RANSAC** para calcular una **homografía**.

- La homografía permite encontrar **cómo se transformó** (rotó, escaló, trasladó) la figura.
-

7. Dibuja el contorno detectado

```
python
CopiarEditar
img_detectada = cv2.polyline(...)
```

- Toma las esquinas del template y las **proyecta** sobre la imagen Figuras.jpg.
 - Si se encontró, dibuja un **polígono verde** alrededor de la figura detectada.
-

8. Muestra el resultado

```
python
CopiarEditar
cv2.imshow(...)
```

- Abre una ventana con la figura resaltada si fue encontrada.
 - Muestra un mensaje indicando si hubo suficientes coincidencias.
-

Resultado esperado

Si el triangulo.jpg (template) se encuentra **dentro de** Figuras.jpg:

- Verás un **polígono verde** dibujado sobre el triángulo en la escena.
- En pantalla: " Figura detectada con rotación".

Si no:

- Verás la imagen original sin contornos y el mensaje: " No se detectó la figura".
-

¿Por qué es útil?

- Detecta **figuras específicas** incluso si están **rotadas o escaladas**.
- Se puede usar en:

- Reconocimiento de piezas en la industria.
- Localización de objetos en imágenes.
- Comparación de formas en geometría computacional.

Códigos:

```

import cv2

import numpy as np


# Cargar imagen de escena (rotada o no) y template recortado
img_scene = cv2.imread("Ases.jpg")           # Escena original
img_template = cv2.imread("Corazon.jpg")       # Template recortado

if img_scene is None or img_template is None:
    print("Error al cargar las imágenes.")
    exit()

# Convertir a escala de grises
gray_scene = cv2.cvtColor(img_scene, cv2.COLOR_BGR2GRAY)
gray_template = cv2.cvtColor(img_template, cv2.COLOR_BGR2GRAY)

# ORB detector
orb = cv2.ORB_create(nfeatures=2000)

# Detectar keypoints y descriptores
kp1, des1 = orb.detectAndCompute(gray_template, None)
kp2, des2 = orb.detectAndCompute(gray_scene, None)

```

```

# Emparejamiento usando Hamming (ideal para ORB)
bf = cv2.BFMatcher(cv2.NORM_HAMMING)
matches = bf.knnMatch(des1, des2, k=2)

# Filtro Lowe para buenas coincidencias
buenas = []
for m, n in matches:
    if m.distance < 0.75 * n.distance:
        buenas.append(m)

# Si hay suficientes puntos clave, calcular homografía
if len(buenas) > 10:
    src_pts = np.float32([kp1[m.queryIdx].pt for m in buenas]).reshape(-1, 1, 2)
    dst_pts = np.float32([kp2[m.trainIdx].pt for m in buenas]).reshape(-1, 1, 2)

    # Homografía usando RANSAC (permite tolerancia a errores)
    M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 8.0)

    # Obtener esquinas del template y transformarlas a la imagen original
    h, w = gray_template.shape
    pts = np.float32([[0, 0], [w, 0], [w, h], [0, h]]).reshape(-1, 1, 2)
    destino = cv2.perspectiveTransform(pts, M)

    # Dibujar contorno sobre la figura detectada
    img_detectada = cv2.polylines(img_scene.copy(), [np.int32(destino)], True, (0, 255, 0),
3)
    mensaje = f"Figura detectada con rotacion ({len(buenas)} coincidencias)"

else:

```

```
img_detectada = img_scene.copy()
mensaje = "No se detecto la figura (muy pocas coincidencias)"

# Mostrar mensaje y resultado
cv2.putText(img_detectada, mensaje, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)
cv2.imshow("Resultado con soporte a rotacion", img_detectada)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

```
import cv2

# Captura desde cámara
cap = cv2.VideoCapture(0)
ret, frame_ref = cap.read()

if not ret:
    print("No se pudo acceder a la camara.")
    cap.release()
    exit()

# Convertir el primer frame en escala de grises (referencia)
frame_ref_gray = cv2.cvtColor(frame_ref, cv2.COLOR_BGR2GRAY)
frame_ref_gray = cv2.GaussianBlur(frame_ref_gray, (21, 21), 0)

while True:
```

```
ret, frame = cap.read()

if not ret:
    break

gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (21, 21), 0)

# Comparar con el fondo (primer frame)
delta = cv2.absdiff(frame_ref_gray, gray)
_, thresh = cv2.threshold(delta, 25, 255, cv2.THRESH_BINARY)

# Mejorar la máscara
thresh = cv2.dilate(thresh, None, iterations=2)
mask_movimiento = cv2.bitwise_and(frame, frame, mask=thresh)

# Mostrar resultados
cv2.imshow("Original", frame)
cv2.imshow("Movimiento Detectado", mask_movimiento)

key = cv2.waitKey(30)
if key == ord('q'):
    break

cap.release()
cv2.destroyAllWindows()
```

Funcionamiento:



