ceti

CENTRO DE ENSEÑANZA
TÉCNICA INDUSTRIAL

Maestro: Mauricio Alejandro Cabrera Arellano

Alumno: Alejandro Retana Rubio 22110315

Materia: Visión Artificial

Practica 11 y 11.1

Fecha: 09-06-2025

11. Detección y Emparejamiento de Características con ORB y Brute-Force

1. Visión por Computadora y Reconocimiento de Patrones

En el campo de la visión por computadora, uno de los objetivos principales es reconocer objetos o partes específicas dentro de una imagen. Para ello, se buscan **características locales**, como esquinas, bordes o patrones repetitivos, que sean únicos y distinguibles.

Estas características pueden ser comparadas entre distintas imágenes para:

- Detectar similitudes
- Reconocer objetos
- Realizar seguimiento de movimiento
- Empalmar imágenes (panoramas)

2. ¿Qué es ORB?

ORB (Oriented FAST and Rotated BRIEF) es un algoritmo que combina dos técnicas:

- **FAST (Features from Accelerated Segment Test):** para detectar puntos clave en la imagen.
- **BRIEF (Binary Robust Independent Elementary Features):** para describir la apariencia local alrededor de esos puntos mediante un descriptor binario.

ORB mejora estos métodos al hacerlos:

- **Invariantes a la rotación** (puede detectar un objeto aunque esté girado)
- **Robustos al cambio de escala**
- **Eficientes computacionalmente**, lo que permite usarlos en tiempo real

ORB es ampliamente utilizado como alternativa a SIFT y SURF, ya que no tiene restricciones de licencia y es más rápido.

3. Keypoints y Descriptores

- Un **keypoint** es una coordenada en la imagen que representa un punto relevante (por ejemplo, una esquina).

- Un **descriptor** es un vector (en ORB, binario) que codifica la apariencia local alrededor del keypoint.

Al detectar keypoints y generar descriptores, podemos comparar dos imágenes y encontrar las zonas que se parecen.

4. Brute-Force Matcher

El **Brute-Force Matcher (BFMatcher)** compara cada descriptor de una imagen con todos los de la otra para encontrar las mejores coincidencias. En este caso se utiliza la métrica **Hamming**, ideal para descriptores binarios como los de ORB.

Al usar `crossCheck=True`, el emparejamiento es más estricto: una coincidencia solo se acepta si ambos descriptores se reconocen mutuamente como el mejor emparejamiento.

5. Visualización de Coincidencias

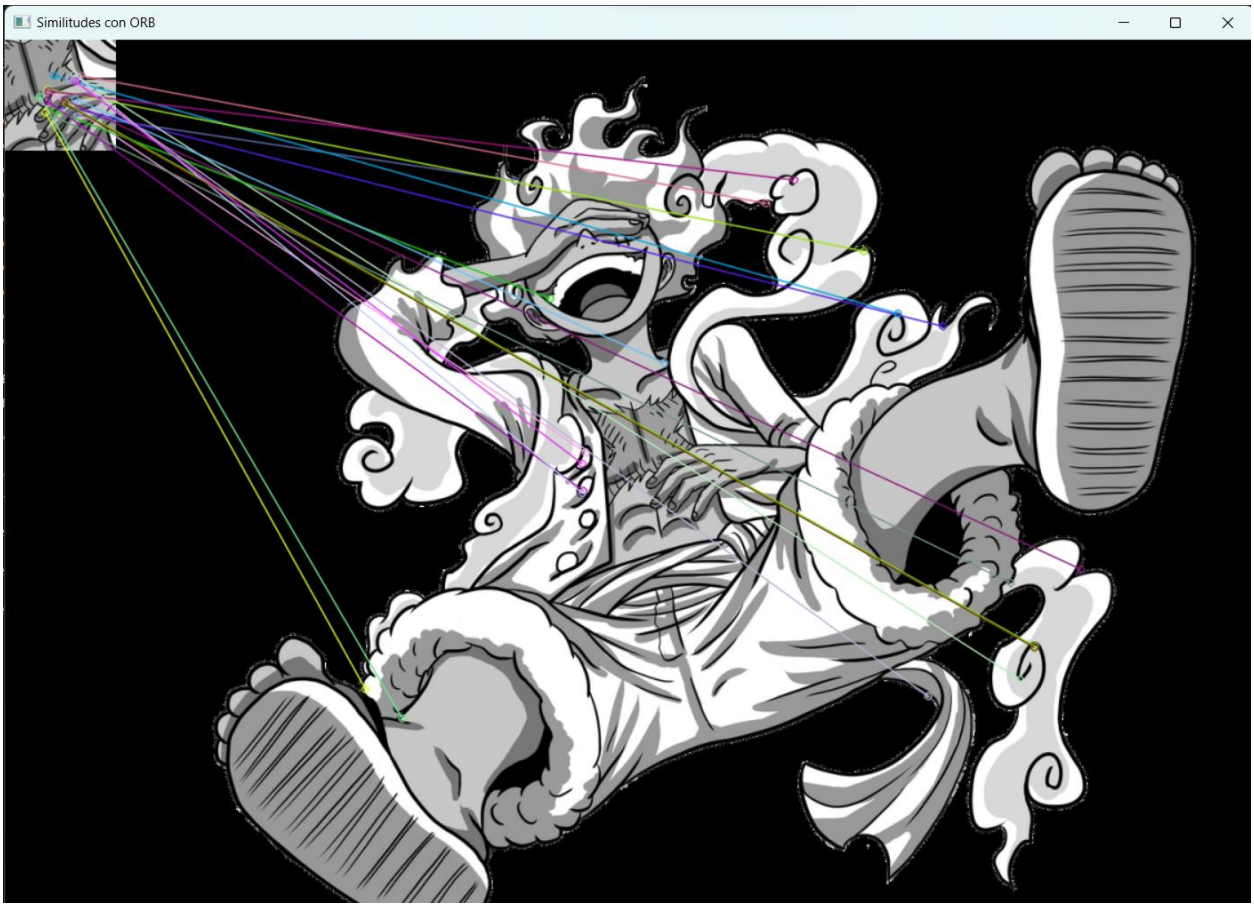
Luego de encontrar las mejores coincidencias, se visualizan conectando los puntos clave coincidentes entre las dos imágenes mediante líneas. Esto permite al usuario ver claramente si hay similitud entre la imagen de búsqueda y la imagen plantilla.

Aplicaciones de ORB y Emparejamiento de Características

- Reconocimiento facial
- Detección de objetos en video o imágenes
- Realidad aumentada
- Empalme de imágenes (panoramas)
- Seguimiento de objetos
- Análisis forense de imágenes

```
mport cv2  # Librería OpenCV para procesamiento de imágenes

# --------- Cargar las imágenes a procesar ---------
img1 = cv2.imread('template.png', 0)  # Cargar la imagen plantilla (a buscar), en
escala de grises
img2 = cv2.imread('luffy5.png', 0)    # Cargar la imagen completa donde se
realizará la búsqueda, también en grises
```

```python
# Trabajar en escala de grises facilita el análisis y reduce el costo computacional

# --------- Inicializar el detector ORB ---------
orb = cv2.ORB_create()
# ORB: Oriented FAST and Rotated BRIEF
# Es un detector y descriptor eficiente, rápido y robusto frente a rotación y cambios de escala

# --------- Detectar puntos clave y calcular descriptores ---------
kp1, des1 = orb.detectAndCompute(img1, None)  # Puntos clave y descriptores de la plantilla
kp2, des2 = orb.detectAndCompute(img2, None)  # Puntos clave y descriptores de la imagen principal
# Los keypoints representan regiones importantes; los descriptores codifican su apariencia local

# --------- Crear el objeto Brute-Force Matcher ---------
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
# NORM_HAMMING: métrica usada para comparar descriptores binarios como los de ORB
# crossCheck=True: la coincidencia se valida en ambos sentidos (mayor precisión)

# --------- Realizar el emparejamiento entre descriptores ---------
matches = bf.match(des1, des2)  # Buscar coincidencias entre descriptores de ambas imágenes

# --------- Ordenar coincidencias según la distancia ---------
matches = sorted(matches, key=lambda x: x.distance)
# Se priorizan las coincidencias con menor distancia (más similares)

# --------- Dibujar las 20 mejores coincidencias encontradas ---------
resultado = cv2.drawMatches(img1, kp1, img2, kp2, matches[:20], None, flags=2)
# Visualizar líneas que conectan los puntos coincidentes entre ambas imágenes

# --------- Mostrar el resultado final ---------
cv2.imshow('Similitudes con ORB', resultado)  # Mostrar la ventana con coincidencias visuales
cv2.waitKey(0)  # Esperar a que el usuario presione una tecla para cerrar
cv2.destroyAllWindows()  # Cerrar todas las ventanas de OpenCV
```



11.1 **Detección de Movimiento por Diferencia de Fondo**

1. Introducción al Análisis de Video

El análisis de video es una técnica fundamental dentro del procesamiento de imágenes y visión por computadora. Permite extraer información relevante de secuencias de imágenes, como la presencia de objetos en movimiento, seguimiento de personas o eventos inusuales. Uno de los métodos más básicos y efectivos para detectar movimiento es el análisis por **diferencia de fondo**.

2. ¿Qué es la Detección de Movimiento?

La detección de movimiento consiste en identificar cambios significativos entre imágenes consecutivas de una escena. En particular, se busca detectar cuándo y dónde aparece un objeto que antes no estaba presente o que ha cambiado de posición.

Este tipo de análisis tiene aplicaciones en:

- Sistemas de videovigilancia
- Cámaras inteligentes (CCTV)
- Robótica autónoma
- Detección de intrusos
- Reconocimiento de actividad

3. Diferencia de Fondo

El método de **diferencia de fondo (background subtraction)** se basa en comparar cada nuevo fotograma (frame) de un video con una imagen considerada como el fondo de referencia (una imagen sin movimiento o con la escena vacía).

Pasos generales:

1. Se toma un frame inicial como imagen de fondo.
2. Cada nuevo frame se convierte a escala de grises para simplificar el análisis.
3. Se calcula la **diferencia absoluta** entre el fondo y el frame actual.
4. Se aplica un **umbral** para convertir los cambios relevantes en una imagen binaria (blanco = movimiento, negro = sin cambio).

Este proceso permite visualizar las regiones donde ocurrió un cambio significativo, lo cual se interpreta como movimiento.

4. Umbralización (Thresholding)

La **umbralización** es una técnica que convierte una imagen en escala de grises en una imagen binaria. Se define un valor límite (threshold) y todos los píxeles con valores superiores se ponen en blanco, mientras que los inferiores se colocan en negro.

En este caso, se usa un umbral de **30**, lo que significa que solo los cambios fuertes entre fondo y frame actual se consideran como movimiento.

5. Visualización de Resultados

Se presentan dos ventanas:

- El video original, capturado en tiempo real desde la cámara.
- Una versión en blanco y negro que muestra únicamente las zonas donde hay movimiento.

Esto permite una interpretación visual sencilla y rápida de los cambios en la escena.

Ventajas del Método de Diferencia de Fondo

- Fácil de implementar
- Eficiente computacionalmente
- Funciona en tiempo real
- Útil para escenas con fondo estático

Limitaciones

- Sensible a cambios de iluminación
- No es adecuado si el fondo cambia constantemente
- Requiere una buena elección del fondo inicial

```
import cv2  # Librería OpenCV para manejo de video e imágenes

# --------- Captura de video ---------
cap = cv2.VideoCapture(0)  # Inicia la cámara web (puedes usar un archivo de
video: 'video.mp4')
# El número 0 indica la cámara por defecto; si usas un video, reemplázalo por la
ruta del archivo

# --------- Leer el primer frame como fondo de referencia ---------
ret, fondo = cap.read()  # Captura el primer frame
fondo_gray = cv2.cvtColor(fondo, cv2.COLOR_BGR2GRAY)  # Convierte ese frame a
escala de grises
# Esta imagen se usará como fondo "estático" para comparar contra los siguientes
frames

# --------- Bucle principal ---------
```

```python
while True:
    ret, frame = cap.read()  # Captura frame por frame en tiempo real
    if not ret:
        break  # Si no se pudo leer el frame, se sale del bucle

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  # Convierte el frame actual a escala de grises

    # --------- Detección de movimiento ---------
    diff = cv2.absdiff(fondo_gray, gray)  # Calcula la diferencia absoluta entre el fondo y el frame actual

    # --------- Aplicar umbral ---------
    _, thresh = cv2.threshold(diff, 30, 255, cv2.THRESH_BINARY)
    # Los cambios mayores al umbral (30) se convierten en blanco (255), el resto en negro
    # Esto resalta las zonas donde ocurrió movimiento

    # --------- Mostrar los resultados ---------
    cv2.imshow('Video Original', frame)         # Muestra el video original en color
    cv2.imshow('Movimiento detectado', thresh)  # Muestra las zonas con movimiento detectado

    # --------- Salir al presionar ESC ---------
    if cv2.waitKey(1) & 0xFF == 27:  # Espera una tecla cada 1 ms; 27 es el código ASCII de ESC
        break

# --------- Liberar recursos ---------
cap.release()  # Libera la cámara o archivo de video
cv2.destroyAllWindows()  # Cierra todas las ventanas de OpenCV
```


Movimiento detectado