CENTRO DE ENSEÑANZA TÉCNICA INDUSTRIAL

INGENIERÍA EN MECATRÓNICA

Visión artificial

Práctica 10:
Extracción del fondo y encontrar esquinas

https://github.com/thvvad/Vision-2025/tree/77b1cfc0b3a7078cdff533deb97302780bb3deb3/PRACTICA_10_Fondo%20y%20esquinas

Alumna:
Vanessa Aguirre Diaz

Registro:
22310274

Fecha:
08 de junio del 2025

Objetivo:

De la imagen que deseen separar por medio de un ROI el fondo de la imagen dejando únicamente el ROI al cual se le buscarán todas las esquinas.

Desarrollo teórico:

¿Qué considera OpenCV para detectar una esquina?

El algoritmo cv2.goodFeaturesToTrack() usa el método Shi-Tomasi, que se basa en una mejora del algoritmo de Harris para detección de esquinas. En términos simples:

Una esquina es un punto donde el cambio de intensidad en la imagen es fuerte en todas las direcciones.

Por ejemplo:

- En una línea recta, solo hay cambio en una dirección → no es una esquina.
- En una esquina real (como la unión de dos bordes), hay cambio en múltiples direcciones → sí es una esquina.

corners = cv2.goodFeaturesToTrack(gray, 100, 0.1, 10)

100 → número máximo de esquinas a detectar

- **Aumentar**: el algoritmo buscará más esquinas.
- **Disminuir**: limitará la cantidad de puntos detectados, aunque existan más posibles.

0.1 → calidad mínima aceptada (qualityLevel)

- Este es un umbral relativo comparado con la mejor esquina detectada.
- Por ejemplo:
  - Si la mejor esquina tiene una calidad de 100, y usas 0.1, solo se aceptan las que tengan calidad ≥ 10.
- **Aumentar el valor (por ejemplo, 0.2, 0.3)**:
  - Menos esquinas detectadas, pero de mayor calidad.
- **Disminuir el valor (por ejemplo, 0.01, 0.001)**:
  - Más esquinas, pero se aceptan puntos menos confiables (puede incluir falso positivos).

10 → distancia mínima entre esquinas

- Este valor evita que se detecten esquinas demasiado juntas.
- **Aumentar (por ejemplo, 20 o 30)**:
  - Se obliga al algoritmo a elegir esquinas más separadas, reduciendo su número.
- **Disminuir (por ejemplo, 5)**:
  - Permite esquinas más cercanas, ideal si quieres captar muchos detalles finos.

Parámetro	Efecto de Aumentarlo	Efecto de Disminuirlo
maxCorners	Detecta más esquinas (hasta ese límite)	Menos esquinas detectadas
qualityLevel	Menos esquinas, pero más confiables	Más esquinas, pero menos confiables
minDistance	Esquinas más separadas	Esquinas más juntas

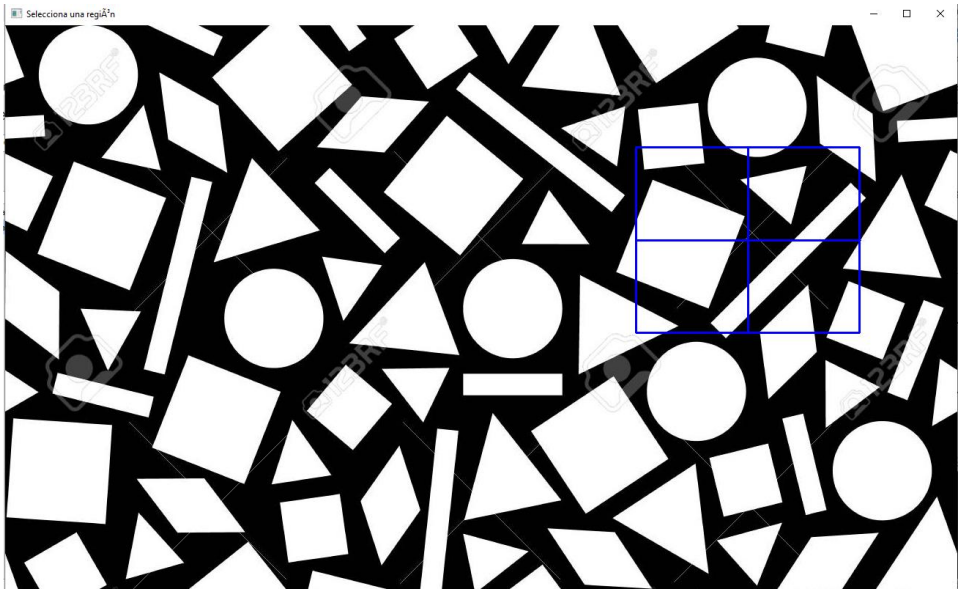
Desarrollo práctico

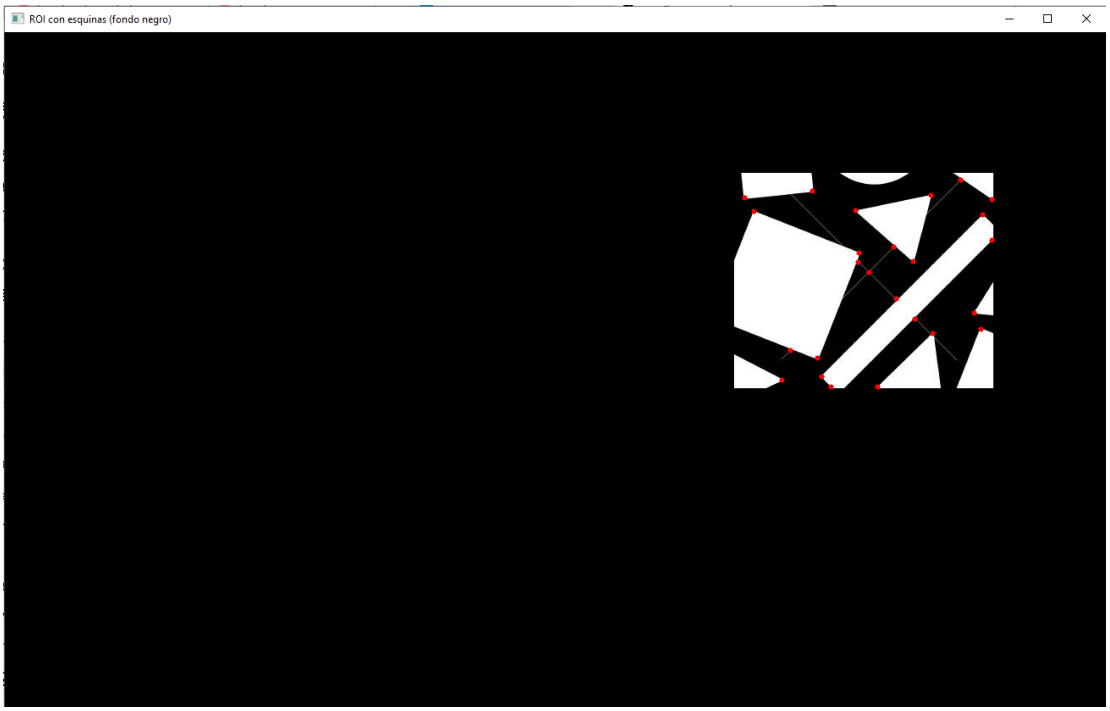
El propósito de esta practica es encontrar las esquinas de las figuras seleccionadas dentro de una roi. Para ello usaremos la función:

corners = cv2.goodFeaturesToTrack(gray, 30, 0.1, 10)

donde iremos cambiando los valores de 30, 0.1 y 10 en base a las características de la imagen que queremos analizar y la precisión con la que queremos que detecte las esquinas.

Con los valores anteriores tenemos el siguiente resultado:





Como podemos notar, la imagen tiene una marca de agua, lo que ocasiona que al cruzarse con las figuras, el programa lo interprete como una línea mas y una posible esquina. Esto podemos interpretarlo como sombras o figuras dentro de la figura que realmente nos interesa, por lo que para un resultado mas limpio, se optó por hacer una umbralización de la roi antes de que pase por la función de detección de esquinas. Es así como obtenemos el siguiente resultado con los parámetros anteriores:

Este valor de umbral también debe ajustarse según las características de la imagen, el color de las figuras y las características que se desean tomar en cuenta y cuales no para el programa. Si es muy alto pueden llegar a perderse algunas figuras o parte de estas, y si es muy bajo pueden colarse detalles no deseados.

El código final queda como:

```
# Practica 10: Detección de esquinas en una ROI seleccionada manualmente

import numpy as np
import cv2

# Carga la imagen original
img = cv2.imread('fig.jpg')

# Crea una imagen negra del mismo tamaño
mascara_negra = np.zeros_like(img)

# Permite seleccionar la región de interés (ROI)
x, y, w, h = cv2.selectROI("Selecciona una region", img)

# Extrae la ROI
```

```
roi = img[y:y+h, x:x+w]

# Convierte la ROI a escala de grises
roi_gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)

# Aplica umbralización binaria (blanco y negro)
_, roi_bin = cv2.threshold(roi_gray, 150, 255, cv2.THRESH_BINARY)

# Convierte la imagen umbralizada a 3 canales para poder insertarla de nuevo
roi_bin_color = cv2.cvtColor(roi_bin, cv2.COLOR_GRAY2BGR)

# Aplica detección de esquinas en la imagen umbralizada
gray_float = np.float32(roi_bin)  # convertir a float32 para goodFeaturesToTrack
corners = cv2.goodFeaturesToTrack(gray_float, 30, 0.1, 10)

# Dibuja los puntos detectados directamente sobre la ROI en blanco y negro
if corners is not None:
    corners = np.int0(corners)
    for corner in corners:
        x_r, y_r = corner.ravel()
        cv2.circle(roi_bin_color, (x_r, y_r), 3, (0, 0, 255), -1)

# Inserta la ROI procesada (con umbral y esquinas) en su lugar dentro de la imagen negra
mascara_negra[y:y+h, x:x+w] = roi_bin_color

# Muestra la imagen final: solo la ROI en B/N y esquinas visibles, lo demás en negro
cv2.imshow('ROI binarizada con esquinas (fondo negro)', mascara_negra)

# Espera a que se presione una tecla
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Conclusión:

En esta práctica se aplicó el algoritmo Shi-Tomasi para detectar esquinas en una región de interés seleccionada por el usuario, aislando el resto de la imagen considerada como fondo, la cual no nos interesaba analizar. Además, se implementó una conversión a escala de grises y una umbralización binaria, lo que permitió resaltar únicamente las características más relevantes dentro de la roi como es la forma del objeto en su totalidad, omitiendo detalles como diseños internos, dibujos o relieves de la superficie que pudieran interpretarse como más esquinas.

Con esto obtuve un código que nos muestra los puntos donde se detectan esquinas, y me permitió experimentar y jugar con los parámetros de la función, así como con varias imágenes, para ver de qué modo se podía mejorar y que valores debía tener la función según las características de la imagen para obtener el resultado deseado.

Bibliografía:

https://pythonprogramming.net/corner-detection-python-opencv-tutorial/