CENTRO DE ENSEÑANZA TÉCNICA INDUSTRIAL

INGENIERÍA EN MECATRÓNICA

Visión artificial

Práctica 9:
Template matching

https://github.com/thvvad/Vision-2025/tree/0ad34c2dda5e920af25ef3df235d0c2ea7781b5b/PRACTICA_9_Temple%20Matching

Alumna:
Vanessa Aguirre Diaz

Registro:
22310274

Fecha:
08 de junio del 2025

Objetivo:

Crear un template (marcara del ROI a detectar).
Encontrar por lo menos 2 ROI's con un minimo de detección de .85

Desarrollo teórico:

¿Qué es el Template Matching en visión artificial?

Template Matching (en español: coincidencia con plantilla) es una técnica de visión por computadora que se utiliza para encontrar una subimagen específica (una "plantilla") dentro de una imagen más grande.

Es como buscar una pieza de rompecabezas dentro de toda la imagen, comparándola con la forma original que estás buscando.

El proceso básico es:

1. Seleccionar una plantilla (subimagen pequeña que contiene el objeto o patrón que deseas encontrar).
2. Compararla en todas las posiciones posibles dentro de la imagen original.
3. Calcular una puntuación de similitud en cada punto (usando alguna función matemática).
4. Localizar las áreas donde la puntuación es más alta: esas son las posiciones más probables donde se encuentra la plantilla.

La función principal es:

cv2.matchTemplate(imagen, plantilla, método)

Y los métodos más comunes son:

Método	**¿Qué hace?**	**Valor alto = mejor?**
cv2.TM_CCOEFF_NORMED	Correlación de coeficientes normalizada	✔ Sí
cv2.TM_SQDIFF_NORMED	Diferencia cuadrática normalizada	✘ No (valor bajo = mejor)
cv2.TM_CCORR_NORMED	Correlación cruzada normalizada	✔ Sí

El Template Matching funciona bien cuando:

- La plantilla tiene el mismo tamaño, orientación y forma exacta que el objeto buscado.
- La imagen no tiene cambios de escala, rotación o iluminación bruscos.

No es bueno si:

- El objeto aparece más grande/pequeño.
- Está rotado.
- Hay deformaciones o sombras.

Para casos más complejos, se puede usar:

- Multi-scale template matching (escalas diferentes).
- ORB/SIFT/SURF (detectores de características).
- Redes neuronales CNN (para reconocimiento más general).

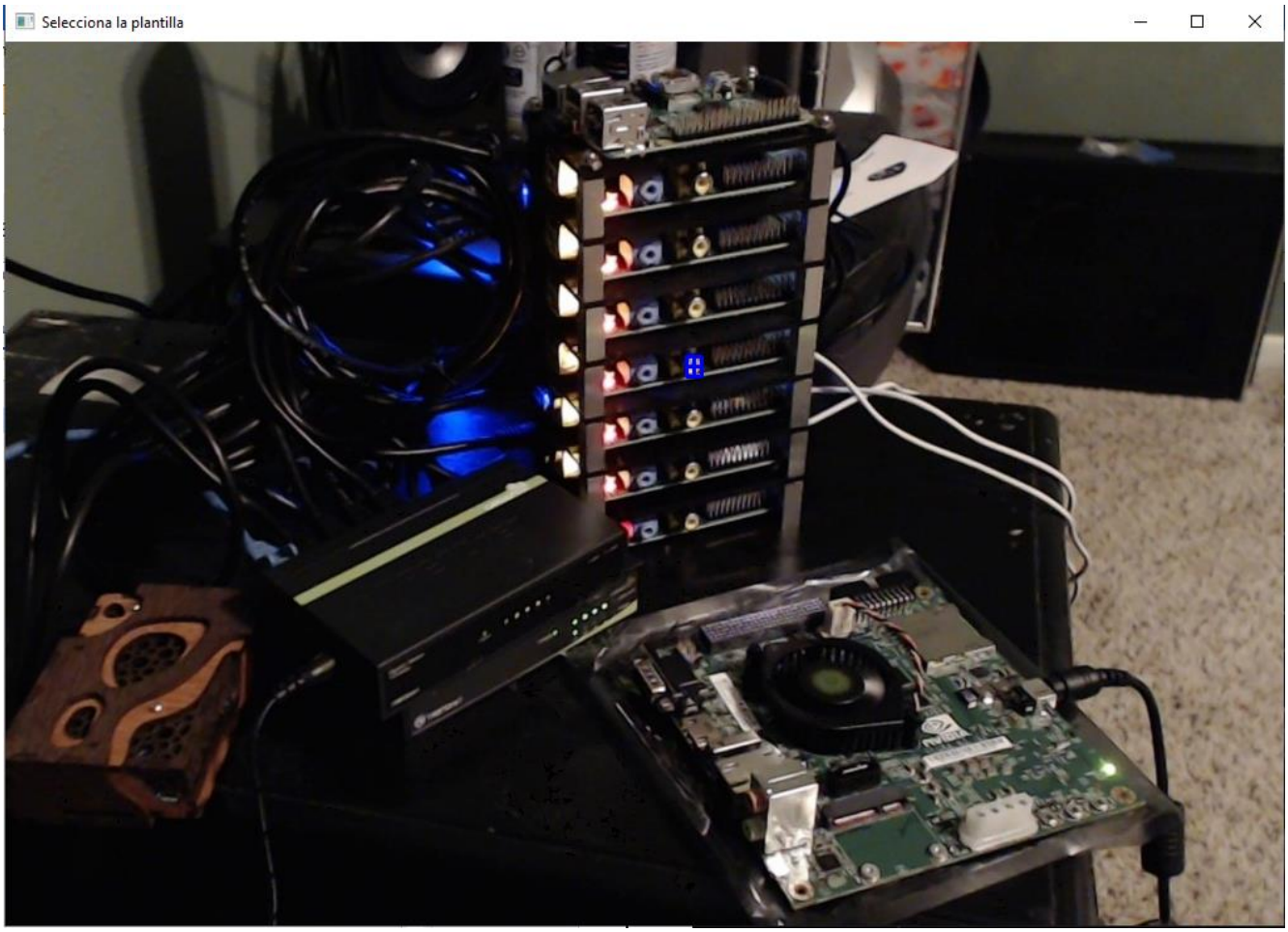
Desarrollo práctico

Partiendo del tutorial, este nos da un código que modifique para poder seleccionar la región de interés de forma manual. El código queda así:

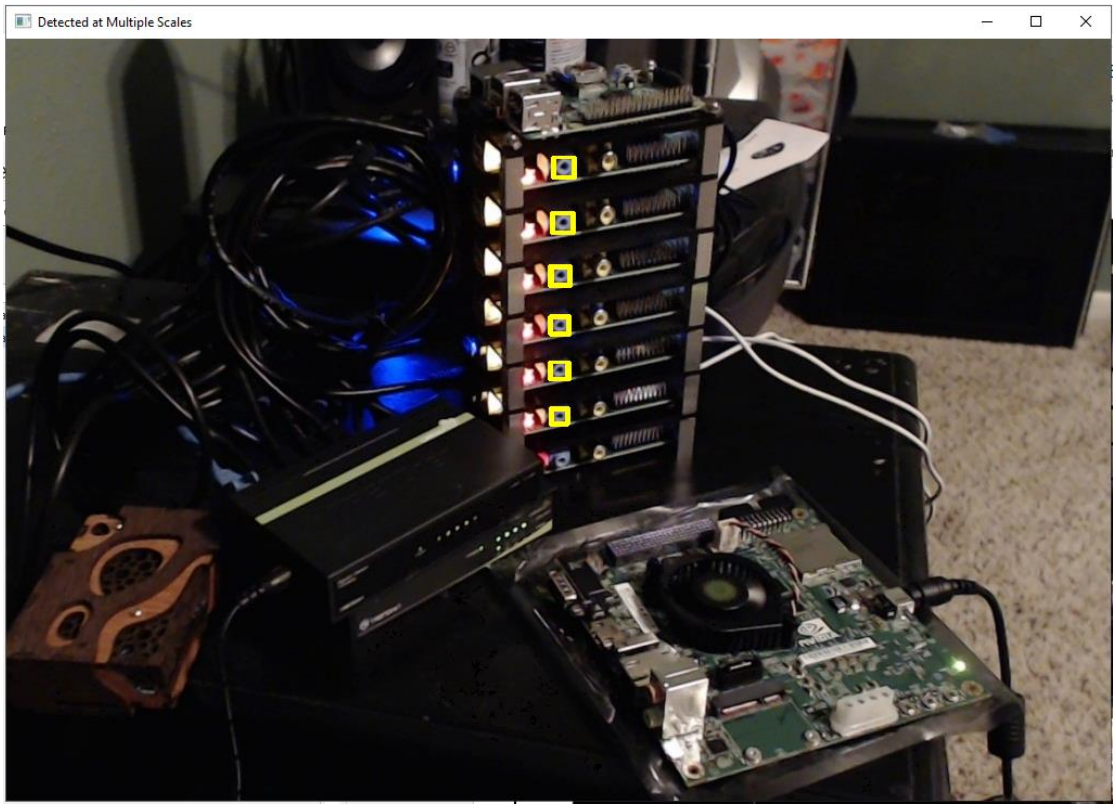
```
# Practica 9: Temple Matching con roi manual

import cv2
import numpy as np

# Cargar imagen original a color
img_rgb = cv2.imread('opencv-template-matching-python-tutorial.jpg')
img_display = img_rgb.copy()

# Selección manual de la plantilla
roi = cv2.selectROI("Selecciona la plantilla", img_display)
template = img_rgb[int(roi[1]):int(roi[1]+roi[3]), int(roi[0]):int(roi[0]+roi[2])]
template_gray = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)

# Convertimos imagen original a escala de grises
img_gray_original = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)

threshold = 0.85  # Umbral de coincidencia
scales = np.linspace(0.5, 1.5, 20)  # Escalas de 50% a 150%, con 20 pasos

# Para cada escala
for scale in scales:
    # Escalamos la imagen original
    resized = cv2.resize(img_gray_original, None, fx=scale, fy=scale, interpolation=cv2.INTER_LINEAR)

    # Si la imagen escalada es más pequeña que la plantilla, ya no tiene sentido continuar
```

```
    if resized.shape[0] < template_gray.shape[0] or resized.shape[1] < template_gray.shape[1]:
        continue

    # Aplicamos template matching
    res = cv2.matchTemplate(resized, template_gray, cv2.TM_CCOEFF_NORMED)
    loc = np.where(res >= threshold)

    # Por cada coincidencia, calculamos la posición en la imagen original
    for pt in zip(*loc[::-1]):
        top_left = (int(pt[0] / scale), int(pt[1] / scale))
        bottom_right = (int((pt[0] + template_gray.shape[1]) / scale),
                        int((pt[1] + template_gray.shape[0]) / scale))
        cv2.rectangle(img_rgb, top_left, bottom_right, (0, 255, 255), 2)

# Mostrar resultado
cv2.imshow('Detected at Multiple Scales', img_rgb)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

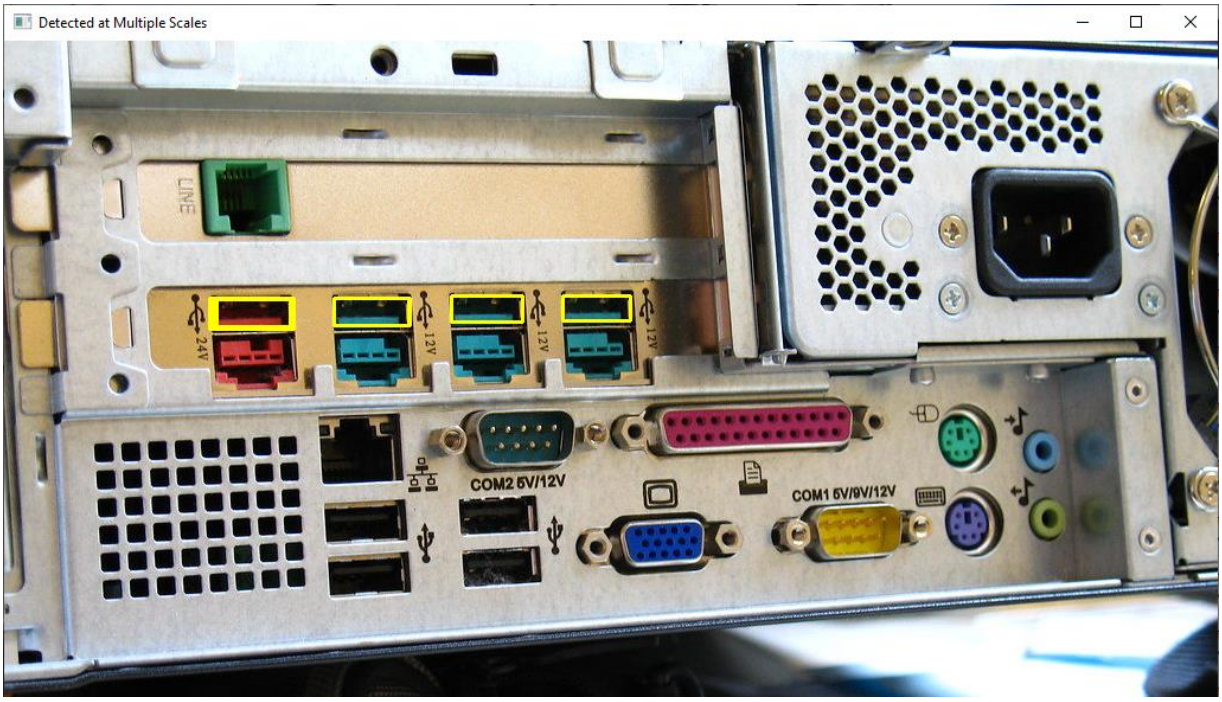
El resultado obtenido, seleccionando el conector amarillo, es:



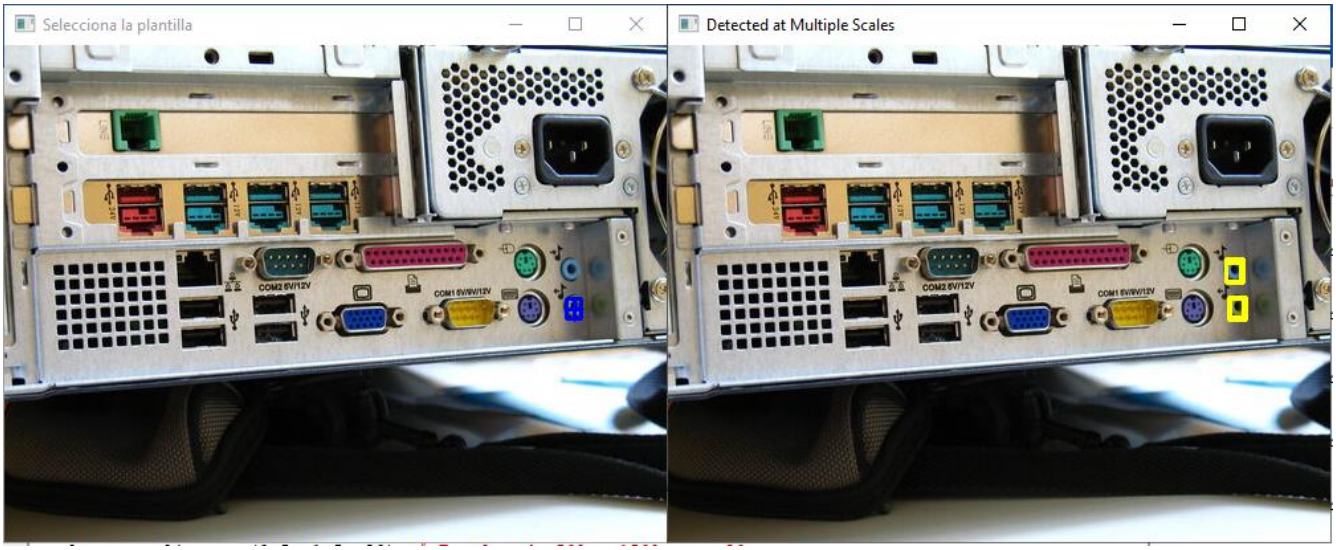
Con el conector azul:



También se probó con otras imágenes, seleccionando la entrada usb:



También se seleccionó la entrada de audio:

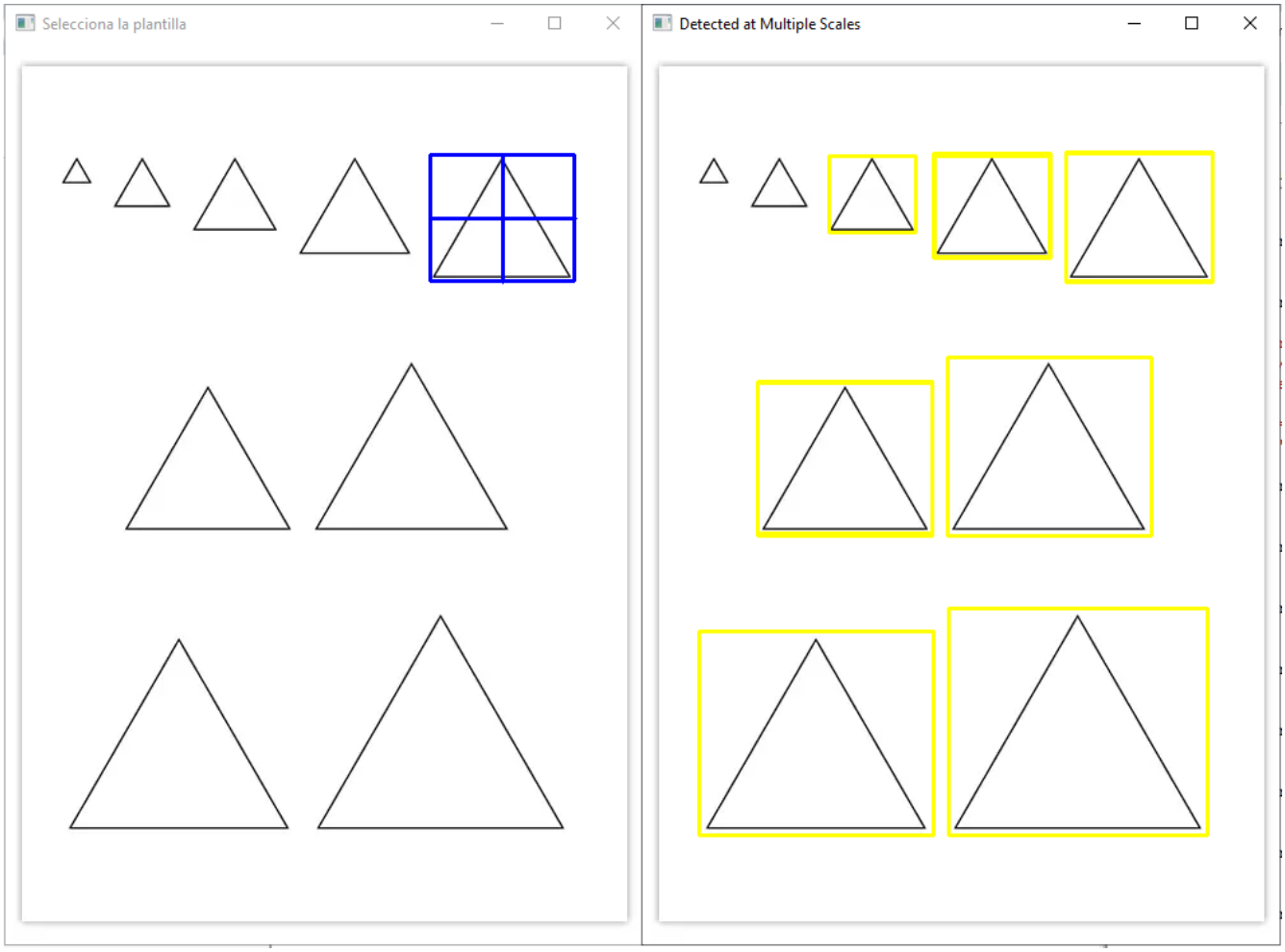


Con esta nueva imagen podemos ver que no importa el color que tenga el conector, o el objeto seleccionado, si encuentra uno con la misma forma aunque tengan diferente color, igual lo toma en cuenta.

Luego probe con una imagen cuya figura era la misma pero de diferente tamaño, y ahí ya no encontró coincidencias, esto debido a que para que este programa lo tome como igual, la figura debe tener las mismas dimensiones. Con esto en cuenta se hizo un nuevo diseño que multiplica la región de interés por varios escalares y compara cada nueva referencia con toda la imagen para ver si encuentra la misma figura aunque sea de diferente tamaño. El código quedo de la siguiente forma:

```
# Practica #9: Temple Matching ´para mismo objeto, diferente escala
import cv2
import numpy as np

# Cargar imagen original a color
img_rgb = cv2.imread('tri.jpg')
img_display = img_rgb.copy()

# Selección manual de la plantilla
roi = cv2.selectROI("Selecciona la plantilla", img_display)
template = img_rgb[int(roi[1]):int(roi[1]+roi[3]), int(roi[0]):int(roi[0]+roi[2])]
template_gray = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)

# Convertimos imagen original a escala de grises
img_gray_original = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)

threshold = 0.8  # Umbral de coincidencia
scales = np.linspace(0.5, 2, 50)  # Escalas de 50% a 150%, con 20 pasos

# Para cada escala
for scale in scales:
    # Escalamos la imagen original
    resized = cv2.resize(img_gray_original, None, fx=scale, fy=scale, interpolation=cv2.INTER_LINEAR)

    # Si la imagen escalada es más pequeña que la plantilla, ya no tiene sentido continuar
    if resized.shape[0] < template_gray.shape[0] or resized.shape[1] < template_gray.shape[1]:
        continue

    # Aplicamos template matching
    res = cv2.matchTemplate(resized, template_gray, cv2.TM_CCOEFF_NORMED)
    loc = np.where(res >= threshold)

    # Por cada coincidencia, calculamos la posición en la imagen original
    for pt in zip(*loc[::-1]):
        top_left = (int(pt[0] / scale), int(pt[1] / scale))
        bottom_right = (int((pt[0] + template_gray.shape[1]) / scale),
                        int((pt[1] + template_gray.shape[0]) / scale))
        cv2.rectangle(img_rgb, top_left, bottom_right, (0, 255, 255), 2)

# Mostrar resultado
cv2.imshow('Detected at Multiple Scales', img_rgb)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Con este código obtenemos el siguiente resultado:



Conclusión:

En esta práctica se implementó la técnica de template matching utilizando OpenCV con el objetivo de detectar un objeto específico dentro de una imagen. A través de la conversión de imágenes a escala de grises y la comparación por similitud mediante la función cv2.matchTemplate, se logró identificar correctamente regiones que coincidían con la plantilla dada. Esta técnica demostró ser eficaz para encontrar objetos con la misma forma, tamaño y orientación, aunque tiene limitaciones cuando existen variaciones en escala, rotación o iluminación. En el caso de esta práctica, se intentó solucionar la variación de tamaño multiplicando la imagen de referencia por un rango de escalares y comparar cada uno en la imagen a ver si se encontraba otra coincidencia de forma pero de distinto tamaño.

Bibliografía:

https://pythonprogramming.net/template-matching-python-opencv-tutorial/