Forma por Igualación de Plantilla (Template Matching)

Encontrar la Forma

OpenCV + Python – Imágenes binarias y escala de grises

Objetivo

Dada una **imagen con un objeto** (binario o gris), y **varios templates** de formas conocidas (círculo, cuadrado, triángulo, estrella...), **identificar cuál template coincide mejor** con el objeto.

Se usan **dos enfoques**:

1. **Template Matching Clásico (correlación)** – para imágenes en **escala de grises**.
2. **Igualación por resta (sujeción + diferencia)** – ideal para **imágenes binarias**.

1. Template Matching Clásico (OpenCV)

python

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# ------------------------------------------------
# 1. Cargar imagen (puede ser binaria o gris)
# ------------------------------------------------
img = cv2.imread('objeto_desconocido.png', cv2.IMREAD_GRAYSCALE)
if img is None:
    raise ValueError("Imagen no encontrada")

# Normalizar (opcional)
img = cv2.normalize(img, None, 0, 255, cv2.NORM_MINMAX)

# ------------------------------------------------
# 2. Definir templates (mismo tamaño o escalables)
# ------------------------------------------------
templates = {
    'circulo':   cv2.imread('templates/circulo.png',
cv2.IMREAD_GRAYSCALE),
    'cuadrado':  cv2.imread('templates/cuadrado.png',
cv2.IMREAD_GRAYSCALE),
    'triangulo': cv2.imread('templates/triangulo.png',
cv2.IMREAD_GRAYSCALE),
    'estrella':  cv2.imread('templates/estrella.png',
cv2.IMREAD_GRAYSCALE)
}

# Asegurar que todos los templates tengan el mismo rango
for name, temp in templates.items():
    templates[name] = cv2.normalize(temp, None, 0, 255,
cv2.NORM_MINMAX)
```

3. Template Matching con múltiples métodos

python

```
methods = [
    cv2.TM_CCOEFF_NORMED,    # Mejor para formas
    cv2.TM_CCORR_NORMED,
    cv2.TM_SQDIFF_NORMED     # Menor = mejor
]

results = {}

for name, template in templates.items():
    h, w = template.shape
    best_val = -np.inf
    best_method = None

    for method in methods:
        res = cv2.matchTemplate(img, template, method)
        min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(res)

        if method in [cv2.TM_SQDIFF, cv2.TM_SQDIFF_NORMED]:
            val = -min_val  # menor diferencia = mejor
        else:
            val = max_val   # mayor correlación = mejor

        if val > best_val:
            best_val = val
            best_method = method
            loc = max_loc if method != cv2.TM_SQDIFF_NORMED else
min_loc

    results[name] = {
        'score': best_val,
        'loc': loc,
        'method': best_method,
        'template': template
    }

# ------------------------------------------------
# 4. Mejor coincidencia
# ------------------------------------------------
best_shape = max(results, key=lambda x: results[x]['score'])
print(f"MEJOR COINCIDENCIA: {best_shape.upper()}")
print(f"  Puntuación: {results[best_shape]['score']:.4f}")
```

5. Visualización

python

```
img_color = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
for name, data in results.items():
    h, w = data['template'].shape
    top_left = data['loc']
    bottom_right = (top_left[0] + w, top_left[1] + h)
    color = (0, 255, 0) if name == best_shape else (0, 0, 255)
    cv2.rectangle(img_color, top_left, bottom_right, color, 2)
    cv2.putText(img_color, f"{name}: {data['score']:.3f}",
                (top_left[0], top_left[1] - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6, color, 2)

plt.figure(figsize=(10, 8))
plt.imshow(cv2.cvtColor(img_color, cv2.COLOR_BGR2RGB))
plt.title(f"Template Matching - Mejor: {best_shape.upper()}")
plt.axis('off')
plt.show()
```

2. Igualación por Resta (Binaria) – Más robusta para formas

python

```
# ------------------------------------------------
# Versión BINARIA: sujeción + resta
# ------------------------------------------------
def binary_template_match(obj_bin, templates_bin, threshold=0.25):
    # Obtener propiedades del objeto
    M = cv2.moments(obj_bin)
    if M['m00'] == 0: return None
    cx_obj = int(M['m10'] / M['m00'])
    cy_obj = int(M['m01'] / M['m00'])
    area_obj = M['m00']

    best_error = np.inf
    best_name = None
    best_aligned = None

    for name, temp_bin in templates_bin.items():
        # Centroid y área del template
        Mt = cv2.moments(temp_bin)
        if Mt['m00'] == 0: continue
        cx_t = int(Mt['m10'] / Mt['m00'])
        cy_t = int(Mt['m01'] / Mt['m00'])
        area_t = Mt['m00']

        # Escalar
        scale = np.sqrt(area_obj / area_t)
        h, w = temp_bin.shape
        M_scale = np.array([[scale, 0, 0], [0, scale, 0]],
dtype=np.float32)
        temp_scaled = cv2.warpAffine(temp_bin, M_scale,
(obj_bin.shape[1], obj_bin.shape[0]))
```

```
        # Centrar
        cx_ts, cy_ts, _ = get_centroid(temp_scaled)
        tx = cx_obj - cx_ts
        ty = cy_obj - cy_ts
        M_trans = np.float32([[1, 0, tx], [0, 1, ty]])
        temp_aligned = cv2.warpAffine(temp_scaled, M_trans,
(obj_bin.shape[1], obj_bin.shape[0]))

        # Resta
        diff = cv2.absdiff(obj_bin, temp_aligned)
        error_pixels = np.count_nonzero(diff)
        total_pixels = area_obj + np.count_nonzero(temp_aligned) -
np.count_nonzero(cv2.bitwise_and(obj_bin, temp_aligned))
        error_rate = error_pixels / total_pixels if total_pixels >
0 else 1

        if error_rate < best_error:
            best_error = error_rate
            best_name = name
            best_aligned = temp_aligned.copy()

    if best_error < threshold:
        return best_name, best_error, best_aligned
    else:
        return "desconocida", best_error, best_aligned

def get_centroid(bin_img):
    M = cv2.moments(bin_img)
    if M['m00'] == 0: return 0, 0, 0
    return int(M['m10']/M['m00']), int(M['m01']/M['m00']),
M['m00']
```

Uso con imágenes binarias

python

```
# Cargar objeto binario
obj_bin = cv2.imread('objeto_binario.png', cv2.IMREAD_GRAYSCALE)
_, obj_bin = cv2.threshold(obj_bin, 127, 255, cv2.THRESH_BINARY)

# Templates binarios
templates_bin = {name: cv2.threshold(t, 127, 255,
cv2.THRESH_BINARY)[1]
                 for name, t in templates.items()}

# Ejecutar
shape, error, aligned = binary_template_match(obj_bin,
templates_bin)
print(f"Forma detectada: {shape.upper()} (error: {error:.3f})")
```

```
# Mostrar
plt.figure(figsize=(12, 4))
plt.subplot(1,3,1); plt.imshow(obj_bin, cmap='gray');
plt.title('Objeto')
plt.subplot(1,3,2); plt.imshow(aligned, cmap='gray');
plt.title(f'Template: {shape}')
plt.subplot(1,3,3); plt.imshow(cv2.absdiff(obj_bin, aligned),
cmap='hot'); plt.title('Diferencia')
plt.show()
```

Comparación de Métodos

Método	Mejor para	Ventajas	Desventajas
TM_CCOEFF_NORMED	Gris, texturas	Rápido, robusto a luz	Sensible a escala/rotación
Resta Binaria	Formas binarias	Muy preciso en siluetas	Requiere alineación
Con escalado + rotación	Formas variables	Máxima precisión	Más lento

Estructura de Carpetas Recomendada

text

```
proyecto/
├── objeto_desconocido.png
├── templates/
│   ├── circulo.png
│   ├── cuadrado.png
│   ├── triangulo.png
│   └── estrella.png
└── template_matching.py
```

Resultado Ejemplo

text

```
MEJOR COINCIDENCIA: CIRCULO
  Puntuación: 0.9421
```

(o con resta binaria: error: 0.087 → **coincide**)

Recomendaciones

- Usa **TM_CCOEFF_NORMED** para imágenes en **gris**.
- Usa **resta binaria + sujeción** para **imágenes binarias**.
- Para **rotación desconocida**, prueba ángulos en pasos de 15°.

- Para **escala variable**, redimensiona el template o usa pirámide.

Función que crea un circulo, o un triangulo o un cuadrado.

python

```python
def create_template(shape, size=(100,100)):
    canvas = np.zeros(size, dtype=np.uint8)
    h, w = size
    c = (w//2, h//2)
    if shape == 'circulo':
        cv2.circle(canvas, c, 40, 255, -1)
    elif shape == 'cuadrado':
        cv2.rectangle(canvas, (20,20), (80,80), 255, -1)
    elif shape == 'triangulo':
        pts = np.array([[50,10], [20,80], [80,80]], np.int32)
        cv2.fillPoly(canvas, [pts], 255)
    return canvas
```