Igualación por Sujeción y Resta

Objetivo

Dada una **imagen binaria con un objeto**, y un **patrón (template) de forma conocida**, queremos:

1. **Alinear (sujetar)** el template con el objeto.
2. **Restar** el template del objeto (o viceversa).
3. **Analizar la diferencia** → si es mínima ≈ **la forma coincide**.

Método: Igualación por Sujeción y Resta

Paso a paso:

Paso	Descripción
1	Extraer el **objeto binario** de la imagen (ya segmentado).
2	Tener un **template binario** de la forma esperada (ej. círculo, triángulo).
3	**Centrar ambos** (usar centroides).
4	**Escalar el template** al mismo tamaño que el objeto (área o bounding box).
5	**Restar**: `diff =
6	**Contar píxeles diferentes → error de forma**
7	Si error < umbral → **coincidencia de forma**

Implementación en Python + OpenCV

python

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

# ------------------------------------------------
# 1. Crear imagen binaria con objeto desconocido
# ------------------------------------------------
h, w = 200, 200
img = np.zeros((h, w), dtype=np.uint8)

# Objeto desconocido: un rectangulo redondeado (casi elipse)
cv2.rectangle(img, (60, 80), (140, 130), 255, -1)
cv2.circle(img, (70, 95), 15, 255, -1)
cv2.circle(img, (130, 115), 15, 255, -1)

# ------------------------------------------------
# 2. Template: forma conocida (ej. elipse perfecta)
# ------------------------------------------------
template = np.zeros((h, w), dtype=np.uint8)
cv2.ellipse(template, center=(100, 105), axes=(45, 25), angle=0,
            startAngle=0, endAngle=360, color=255, thickness=-1)
```

```
# ------------------------------------------------
# 3. Encontrar contornos y centroides
# ------------------------------------------------
def get_centroid(binary):
    M = cv2.moments(binary)
    if M['m00'] == 0: return None
    cx = int(M['m10'] / M['m00'])
    cy = int(M['m01'] / M['m00'])
    return cx, cy, M['m00']

contours, _ = cv2.findContours(img, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
obj_contour = contours[0]
cx_obj, cy_obj, area_obj = get_centroid(img)

temp_contour, _ = cv2.findContours(template, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
cx_temp, cy_temp, area_temp = get_centroid(template)

# ------------------------------------------------
# 4. Escalar y alinear template al objeto
# ------------------------------------------------
scale = np.sqrt(area_obj / area_temp)
h_temp, w_temp = template.shape

# Matriz de transformación: traslación + escala
M_scale = np.array([[scale, 0, 0],
                    [0, scale, 0]], dtype=np.float32)
template_scaled = cv2.warpAffine(template, M_scale, (w, h))

# Recalcular centro del template escalado
cx_ts, cy_ts, _ = get_centroid(template_scaled)

# Trasladar para centrar con el objeto
tx = cx_obj - cx_ts
ty = cy_obj - cy_ts
M_trans = np.float32([[1, 0, tx], [0, 1, ty]])
template_aligned = cv2.warpAffine(template_scaled, M_trans, (w,
h))

# ------------------------------------------------
# 5. Resta (diferencia absoluta)
# ------------------------------------------------
diff = cv2.absdiff(img, template_aligned)
error_pixels = np.sum(diff > 0)
total_pixels = area_obj + np.sum(template_aligned > 0) -
np.sum(cv2.bitwise_and(img, template_aligned) > 0)
shape_error = error_pixels / total_pixels  # error normalizado

# ------------------------------------------------
# 6. Visualizacion
```

```
# ------------------------------------------------
plt.figure(figsize=(15, 5))

plt.subplot(1, 4, 1)
plt.imshow(img, cmap='gray')
plt.title('Objeto Desconocido')
plt.axis('off')

plt.subplot(1, 4, 2)
plt.imshow(template, cmap='gray')
plt.title('Template (Elipse)')
plt.axis('off')

plt.subplot(1, 4, 3)
plt.imshow(template_aligned, cmap='gray')
plt.title('Template Alineado')
plt.axis('off')

plt.subplot(1, 4, 4)
plt.imshow(diff, cmap='hot')
plt.title(f'Diferencia\nError = {shape_error:.3f}')
plt.colorbar()
plt.axis('off')

plt.tight_layout()
plt.show()

# ------------------------------------------------
# 7. Decision
# ------------------------------------------------
threshold = 0.25  # umbral empírico
if shape_error < threshold:
    print(f"Forma COINCIDE (error = {shape_error:.3f} <
{threshold}) → ELIPSE")
else:
    print(f"Forma NO coincide (error = {shape_error:.3f} ≥
{threshold})")
```

Resultado Esperado

```
Forma COINCIDE (error = 0.187 < 0.25) → ELIPSE
```

Aunque el objeto no es una elipse perfecta, el **error es bajo** → se clasifica como **elipsoide**.

Ventajas del Método

Ventaja	Descripción
Simple y rápido	Solo resta binaria
Interpretable	El error es % de píxeles distintos

Ventaja	Descripción
Robusto a ruido	Si se usa erosión/dilatación previa
No requiere aprendizaje	Solo necesitas el template

Mejoras Opcionales

python
```
# 1. Tolerancia a rotación (probar multiples angulos)
angles = [0, 15, 30, 45, 60, 75, 90]
best_error = float('inf')
best_aligned = None

for angle in angles:
    M_rot = cv2.getRotationMatrix2D((cx_obj, cy_obj), angle,
scale)
    temp_rot = cv2.warpAffine(template, M_rot, (w, h))
    diff = cv2.absdiff(img, temp_rot)
    error = np.sum(diff > 0)
    if error < best_error:
        best_error = error
        best_aligned = temp_rot
```
python
```
# 2. Preprocesamiento: cerrar agujeros, suavizar contornos
obj_clean = cv2.morphologyEx(img, cv2.MORPH_CLOSE,
kernel=np.ones((3,3)))
```

Aplicaciones

- Control de calidad (¿la pieza tiene la forma correcta?)
- Clasificación de formas binarias
- Detección de defectos por desviación de forma
- Robótica: verificar agarre de objetos

Resumen: Algoritmo Final

```
1. Segmentar objeto → binario
2. Crear template binario de forma ideal
3. Centrar (centroides)
4. Escalar (área)
5. [Opcional] Rotar (múltiples ángulos)
6. Restar: diff = |obj - temp|
7. error = píxeles_diferentes / píxeles_totales
8. if error < umbral → "forma igual"
```