Encontrar su forma por igualación por otras extensiones de la Transformada de Hough.

Encontrar la Forma

(Métodos avanzados: PHT, SHT, CHT, RHT, GHT con escala/rotación, etc.)

La **Transformada de Hough** clásica detecta **líneas y círculos**. Sus **extensiones** permiten detectar **formas más complejas** con **rotación, escala, oclusión y ruido**.

Lista de Extensiones de Hough

Extensión	Nombre	Detecta	Invariancias
PHT	Probabilistic Hough Transform	Líneas	Ruido, oclusión
PPHT	Progressive Probabilistic HT	Líneas	Más rápido
SHT	Standard Hough Transform (clásica)	Líneas	—
CHT	Circular Hough Transform	Círculos	—
RHT	Randomized Hough Transform	Líneas, círculos	Ruido, menos memoria
GHT	Generalized Hough Transform	Formas arbitrarias	Rotación, escala (con R-Table)
AGHT	Adaptive GHT	Formas variables	Escala dinámica
MSHT	Multi-Scale Hough	Líneas en múltiples escalas	Escala

Implementación Completa: 5 Extensiones en 1 Script

python

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

# ------------------------------------------------
# 1. Cargar imagen de prueba
# ------------------------------------------------
img_color = cv2.imread('formas_mixtas.png')
if img_color is None:
    # Generar imagen sintética si no existe
    h, w = 400, 500
    img = np.zeros((h, w), dtype=np.uint8)
    # Línea
    cv2.line(img, (50, 50), (350, 300), 255, 2)
    # Círculo
    cv2.circle(img, (200, 150), 60, 255, 2)
    # Elipse rotada
    cv2.ellipse(img, (300, 250), (70, 40), 45, 0, 360, 255, 2)
    # Estrella (forma arbitraria)
    star = np.array([[300,100], [320,150], [370,150], [330,190],
[340,240],
```

```
                    [300,210], [260,240], [270,190], [230,150],
[280,150]], np.int32)
    cv2.fillPoly(img, [star], 255)
    cv2.imwrite('formas_mixtas.png', img)
    img_color = cv2.imread('formas_mixtas.png')

img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(img_gray, 50, 150, apertureSize=3)

# ------------------------------------------------
# 2. EXTENSIÓN 1: Probabilistic Hough Transform (PHT)
# ------------------------------------------------
lines_pht = cv2.HoughLinesP(edges, rho=1, theta=np.pi/180,
threshold=50,
                            minLineLength=50, maxLineGap=10)

# ------------------------------------------------
# 3. EXTENSIÓN 2: Randomized Hough Transform (RHT) - Círculos
# ------------------------------------------------
def randomized_hough_circles(edges, iterations=1000, min_dist=50,
min_radius=20, max_radius=100):
    h, w = edges.shape
    points = np.column_stack(np.where(edges > 0))
    if len(points) < 3: return None

    best_circle = None
    best_votes = 0

    for _ in range(iterations):
        # Tomar 3 puntos aleatorios
        idx = np.random.choice(len(points), 3, replace=False)
        p1, p2, p3 = points[idx]

        # Calcular centro del círculo
        try:
            x1, y1 = p1[1], p1[0]
            x2, y2 = p2[1], p2[0]
            x3, y3 = p3[1], p3[0]

            d = 2 * (x1*(y2 - y3) + x2*(y3 - y1) + x3*(y1 - y2))
            if abs(d) < 1e-5: continue
            ux = ((x1**2 + y1**2)*(y2 - y3) + (x2**2 + y2**2)*(y3
- y1) + (x3**2 + y3**2)*(y1 - y2)) / d
            uy = ((x1**2 + y1**2)*(x3 - x2) + (x2**2 + y2**2)*(x1
- x3) + (x3**2 + y3**2)*(x2 - x1)) / d
            cx, cy = int(ux), int(uy)
            r = int(np.mean([np.linalg.norm(p1 - [cy, cx]),
                             np.linalg.norm(p2 - [cy, cx]),
                             np.linalg.norm(p3 - [cy, cx])]))
        except:
            continue
```

```
        if not (min_radius <= r <= max_radius): continue
        if not (0 <= cx < w and 0 <= cy < h): continue

        # Contar votos en borde
        mask = np.zeros((h, w), dtype=np.uint8)
        cv2.circle(mask, (cx, cy), r, 1, 1)
        votes = np.sum(mask & edges) // 255
        if votes > 10 and votes > best_votes:
            dists = [np.linalg.norm(np.array([cx, cy]) - other)
for other in
                     [(c[0], c[1]) for c in (best_circle[:2] if
best_circle else [])]]
            if not best_circle or all(d > min_dist for d in
dists):
                best_circle = (cx, cy, r)
                best_votes = votes

    return [(best_circle[0], best_circle[1], best_circle[2])] if
best_circle else None

circles_rht = randomized_hough_circles(edges, iterations=500)

# ------------------------------------------------
# 4. EXTENSIÓN 3: Generalized Hough con Escala + Rotación
# ------------------------------------------------
def ght_with_scale_rotation(template, edges, scales=[0.8,1.0,1.2],
angles=np.arange(0,360,30)):
    h, w = template.shape
    ref_x, ref_y = w//2, h//2
    temp_edges = cv2.Canny(template, 50, 150)
    r_table_base = {}
    contours, _ = cv2.findContours(temp_edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_NONE)
    for cnt in contours:
        for p in cnt:
            dx = p[0][0] - ref_x
            dy = p[0][1] - ref_y
            angle = np.arctan2(dy, dx)
            r_table_base.setdefault(round(angle, 2),
[]).append((dx, dy))

    accumulator = np.zeros(edges.shape, dtype=np.float32)
    edge_points = np.column_stack(np.where(edges > 0))

    for scale in scales:
        for angle in angles:
            M = cv2.getRotationMatrix2D((ref_x, ref_y), angle,
scale)
            for base_angle, vectors in r_table_base.items():
                for dx, dy in vectors:
                    x2 = ref_x + dx * np.cos(angle * np.pi/180) -
dy * np.sin(angle * np.pi/180)
```

```
                    y2 = ref_y + dx * np.sin(angle * np.pi/180) +
dy * np.cos(angle * np.pi/180)
                    dx_s = int(x2 - ref_x)
                    dy_s = int(y2 - ref_y)
                    for py, px in edge_points:
                        rx = int(px - dx_s)
                        ry = int(py - dy_s)
                        if 0 <= rx < edges.shape[1] and 0 <= ry <
edges.shape[0]:
                            accumulator[ry, rx] += 1

    return accumulator

# Template estrella
star_temp = np.zeros((80,80), dtype=np.uint8)
pts =
np.array([[40,10],[48,35],[70,35],[52,50],[58,75],[40,60],[22,75],
[28,50],[10,35],[32,35]], np.int32)
cv2.fillPoly(star_temp, [pts], 255)
acc_star = ght_with_scale_rotation(star_temp, edges,
scales=[0.9,1.1], angles=[0,45])

# ------------------------------------------------
# 5. EXTENSIÓN 4: Ajuste de Elipse (no Hough, pero complementario)
# ------------------------------------------------
contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
ellipses = []
for cnt in contours:
    if len(cnt) >= 5:
        ellipse = cv2.fitEllipse(cnt)
        ellipses.append(ellipse)

# ------------------------------------------------
# 6. Visualización
# ------------------------------------------------
output = img_color.copy()

# PHT: líneas verdes
if lines_pht is not None:
    for x1,y1,x2,y2 in lines_pht[:,0]:
        cv2.line(output, (x1,y1), (x2,y2), (0,255,0), 2)

# RHT: círculos azules
if circles_rht:
    for x,y,r in circles_rht:
        cv2.circle(output, (x,y), r, (255,0,0), 2)
        cv2.circle(output, (x,y), 2, (0,0,255), 3)

# GHT: estrella (pico en acumulador)
acc_norm = cv2.normalize(acc_star, None, 0, 255,
cv2.NORM_MINMAX).astype(np.uint8)
```

```
_, thresh = cv2.threshold(acc_norm, 100, 255, cv2.THRESH_BINARY)
star_peaks, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
for peak in star_peaks:
    M = cv2.moments(peak)
    if M['m00'] > 0:
        cx = int(M['m10']/M['m00'])
        cy = int(M['m01']/M['m00'])
        cv2.circle(output, (cx, cy), 15, (0,255,255), 2)

# Elipses: amarillas
for ell in ellipses:
    cv2.ellipse(output, ell, (0,255,255), 2)

plt.figure(figsize=(18, 10))

plt.subplot(2, 3, 1)
plt.imshow(cv2.cvtColor(img_color, cv2.COLOR_BGR2RGB))
plt.title('Original')
plt.axis('off')

plt.subplot(2, 3, 2)
plt.imshow(edges, cmap='gray')
plt.title('Bordes')
plt.axis('off')

plt.subplot(2, 3, 3)
plt.imshow(acc_norm, cmap='hot')
plt.title('Acumulador GHT (Estrella)')
plt.colorbar()
plt.axis('off')

plt.subplot(2, 3, 4)
plt.imshow(output[:,:,::-1])
plt.title('Detecciones:\nVerde=PHT, Azul=RHT, Amarillo=Elipse,
Cian=GHT')
plt.axis('off')

plt.tight_layout()
plt.show()
```

Resumen de Extensiones

Extensión	OpenCV	Ventaja
PHT / PPHT	HoughLinesP	Rápido, robusto a ruido
RHT	Manual	Menos memoria, evita falsos positivos
GHT	Manual	Formas arbitrarias
GHT + Escala/Rotación	Manual	Invariante a transformaciones
fitEllipse	cv2.fitEllipse	Elipses perfectas

Comparación de Rendimiento

Método	Velocidad	Memoria	Formas	Ruido
Hough Clásica	Alta	Alta	Líneas, círculos	Baja
PHT	Muy alta	Baja	Líneas	Alta
RHT	Media	Muy baja	Círculos	Alta
GHT	Baja	Alta	Cualquier	Media

Recomendaciones

Caso	Mejor método
Líneas en ruido	**PHT**
Círculos múltiples	**RHT** o HoughCircles
Forma personalizada	**GHT**
Rotación/escala	**GHT con R-Table por ángulo/escala**
Elipses	fitEllipse

Conclusión

Las extensiones de Hough permiten ir más allá de líneas y círculos, detectando formas complejas con robustez a transformaciones y ruido. La más poderosa es GHT, pero PHT y RHT son ideales para aplicaciones en tiempo real.

Detector universal que combina todas las extensiones y devuelve: {'lineas': 3, 'circulos': 1, 'estrella': True}:

python

```
detector = HoughExtensions()
results = detector.detect_all(image)
```