1. Momentos Los momentos de imagen le ayudan a calcular algunas características como el centro de masa del objeto, el área del objeto, etc. Consulte la página de Wikipedia sobre Momentos de imagen.

La función cv.moments() proporciona un diccionario de todos los valores de momento calculados. Vea abajo:

import numpy as np  
import cv2 as cv  
img = cv.imread('resources/res\_mario.jpg', cv.IMREAD\_GRAYSCALE)  
assert img is not None, "file could not be read, check with os.path.exists()"  
ret,thresh = cv.threshold(img,127,255,0)  
contours,hierarchy = cv.findContours(thresh, 1, 2)  
cnt = contours[0]  
M = cv.moments(cnt)  
print( M )

{'m00': 0.5, 'm10': 218.16666666666666, 'm01': 117.66666666666666, 'm20': 95193.41666666666, 'm11': 51341.875, 'm02': 27690.916666666664, 'm30': 41536085.050000004, 'm21': 22402171.933333334, 'm12': 12082463.433333334, 'm03': 6516608.800000001, 'mu20': 0.027777777781011537, 'mu11': -0.01388888888322981, 'mu02': 0.027777777781011537, 'mu30': 0.0037037134170532227, 'mu21': -0.0018518534682998578, 'mu12': -0.0018518534974028, 'mu03': 0.0037037041038274765, 'nu20': 0.11111111112404615, 'nu11': -0.05555555553291924, 'nu02': 0.11111111112404615, 'nu30': 0.020951366982159467, 'nu21': -0.010475665161589254, 'nu12': -0.010475665326220357, 'nu03': 0.02095131429859883}

cx = int(M['m10']/M['m00'])  
cy = int(M['m01']/M['m00'])

1. Área de contorno El área del contorno viene dada por la función cv.contourArea() o por momentos, M['m00'] .

area = cv.contourArea(cnt)

1. Perímetro del contorno También se le llama longitud de arco. Se puede descubrir utilizando la función cv.arcLength() . El segundo argumento especifica si la forma es un contorno cerrado (si se pasa como Verdadero) o simplemente una curva.

perimeter = cv.arcLength(cnt,True)

1. Aproximación del contorno Aproxima una forma de contorno a otra forma con menor número de vértices dependiendo de la precisión que especifiquemos. Es una implementación del algoritmo de Douglas-Peucker . Consulte la página de Wikipedia para ver el algoritmo y la demostración.

Para entender esto, suponga que está tratando de encontrar un cuadrado en una imagen, pero debido a algunos problemas en la imagen, no obtuvo un cuadrado perfecto, sino una "mala forma" (como se muestra en la primera imagen a continuación). Ahora puedes usar esta función para aproximar la forma. En esto, el segundo argumento se llama épsilon, que es la distancia máxima desde el contorno hasta el contorno aproximado. Es un parámetro de precisión. Se necesita una selección inteligente de épsilon para obtener el resultado correcto.

epsilon = 0.1\*cv.arcLength(cnt,True)  
approx = cv.approxPolyDP(cnt,epsilon,True)

1. Casco convexo Convex Hull se verá similar a la aproximación de contorno, pero no lo es (ambos pueden proporcionar los mismos resultados en algunos casos). Aquí, la función cv.convexHull() comprueba una curva en busca de defectos de convexidad y la corrige. En términos generales, las curvas convexas son curvas que siempre están abultadas o al menos planas. Y si está abultado por dentro se llama defectos de convexidad. Por ejemplo, consulte la siguiente imagen de la mano. La línea roja muestra el casco convexo de la mano. Las marcas de flecha de doble cara muestran los defectos de convexidad, que son las desviaciones máximas locales del casco respecto de los contornos.

hull = cv.convexHull(points[ hull[ clockwise[ returnPoints]]])

Los puntos son los contornos por los que pasamos. El casco es la salida, normalmente lo evitamos. en el sentido de las agujas del reloj : bandera de orientación. Si es Verdadero, el casco convexo de salida está orientado en el sentido de las agujas del reloj. De lo contrario, se orienta en el sentido contrario a las agujas del reloj. returnPoints : De forma predeterminada, Verdadero. Luego devuelve las coordenadas de los puntos del casco. Si es False, devuelve los índices de los puntos de contorno correspondientes a los puntos del casco.

hull = cv.convexHull(cnt)

1. Comprobando la convexidad Existe una función para comprobar si una curva es convexa o no, cv.isContourConvex() . Simplemente devuelve ya sea Verdadero o Falso. No es un gran trato.

k = cv.isContourConvex(cnt)

1. Rectángulo delimitador Hay dos tipos de rectángulos delimitadores.

7.a. Rectángulo delimitador recto Es un rectángulo recto, no considera la rotación del objeto. Entonces el área del rectángulo delimitador no será mínima. Se encuentra mediante la función cv.boundingRect() .

Sea (x,y) la coordenada superior izquierda del rectángulo y (w,h) su ancho y alto.

x,y,w,h = cv.boundingRect(cnt)  
cv.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)

7.b. Rectángulo girado Aquí, el rectángulo delimitador se dibuja con un área mínima, por lo que también considera la rotación. La función utilizada es cv.minAreaRect() . Devuelve una estructura Box2D que contiene los siguientes detalles: (centro (x,y), (ancho, alto), ángulo de rotación). Pero para dibujar este rectángulo, necesitamos 4 esquinas del rectángulo. Se obtiene mediante la función cv.boxPoints()

rect = cv.minAreaRect(cnt)  
box = cv.boxPoints(rect)  
box = np.int0(box)  
cv.drawContours(img,[box],0,(0,0,255),2)

1. Círculo envolvente mínimo A continuación encontramos el círculo circunstante de un objeto usando la función cv.minEnclosingCircle() . Es un círculo que cubre completamente el objeto con un área mínima.

(x,y),radius = cv.minEnclosingCircle(cnt)  
center = (int(x),int(y))  
radius = int(radius)  
cv.circle(img,center,radius,(0,255,0),2)

1. Ajustar una elipse El siguiente es ajustar una elipse a un objeto. Devuelve el rectángulo rotado en el que está inscrita la elipse.

ellipse = cv.fitEllipse(cnt)  
cv.ellipse(img,ellipse,(0,255,0),2)

1. Colocar una línea De manera similar podemos ajustar una línea a un conjunto de puntos. La imagen de abajo contiene un conjunto de puntos blancos. Podemos aproximarle una línea recta

rows,cols = img.shape[:2]  
[vx,vy,x,y] = cv.fitLine(cnt, cv.DIST\_L2,0,0.01,0.01)  
lefty = int((-x\*vy/vx) + y)  
righty = int(((cols-x)\*vy/vx)+y)  
cv.line(img,(cols-1,righty),(0,lefty),(0,255,0),2)