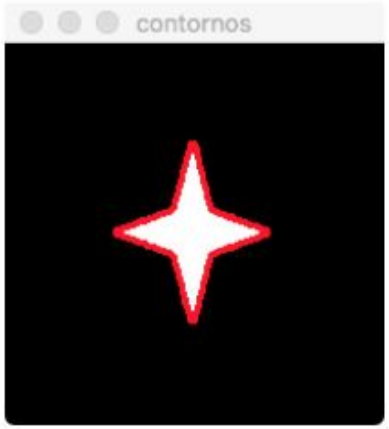
Práctica 1

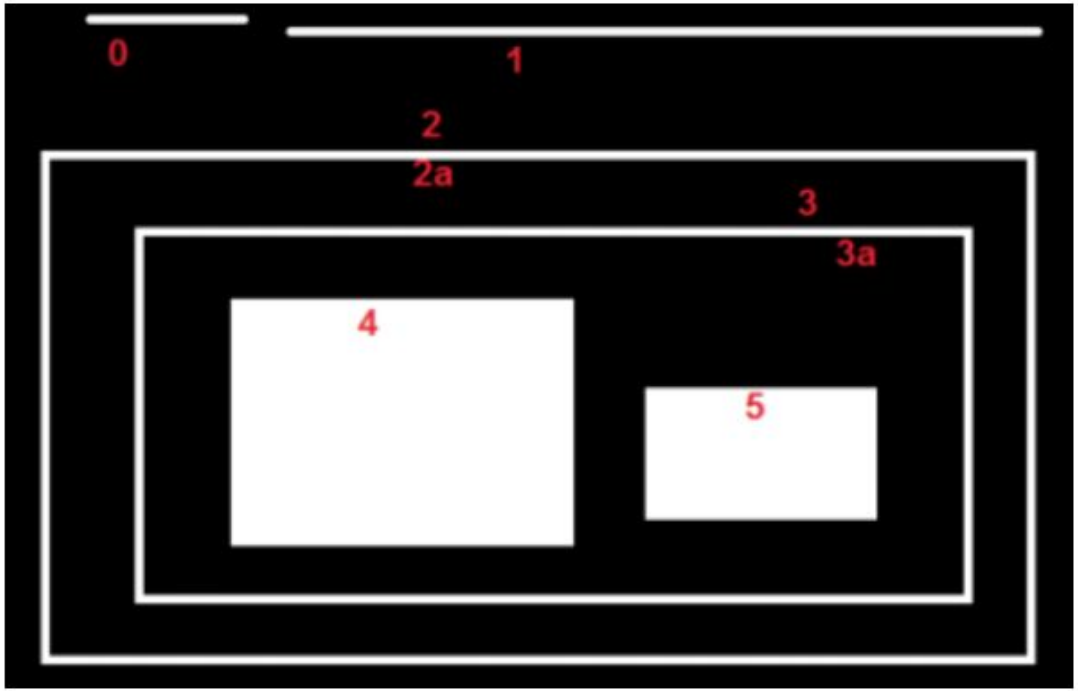
Detección, Descripción y Reconocimiento de formas empleando contornos en OpenCV – Python

- Encontrar contornos en OpenCV:
  - Función: ***findContours***
  - Parámetros de Entrada:
    - 1: imagen binaria sobre la que se va a realizar la búsqueda de contornos
    - 2: Contour Retrieval Mode
      - RETR_LIST
      - RETR_EXTERNAL
      - RETR_CCOMP
      - RETR_TREE
    - 3: Contour Approximation Method
      - CHAIN_APPROX_NONE
      - CHAIN_APPROX_SIMPLE
  - Parámetros de Salida:
    - 1: imagen copia de la imagen de entrada
    - 2: matriz con los contornos encontrados
    - 3: matriz con la jerarquía de los contornos encontrados




```
import numpy as np
import cv2

im = cv2.imread('estrella.jpg',cv2.IMREAD_COLOR)
imgray = cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)

_, thresh = cv2.threshold(imgray,127,255,cv2.THRESH_BINARY)
cv2.imshow('original',thresh)
cv2.waitKey()

_, contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

cv2.drawContours(im,contours,-1,(0,0,255),2)

cv2.imshow('contornos', im)
cv2.waitKey()
```

- Control Retrieval Mode:



- Información más detallada en:
  - https://docs.opencv.org/3.4.0/d9/d8b/tutorial_py_contours_hierarchy.html

- Control Retrieval Mode:
  - RETR_LIST
    - Es el modo más sencillo ya que encuentra todos los contornos sin crear ninguna relación de tipo padre-hijo. En este modo, padres e hijos son iguales y, por lo tanto, son todos contornos (están todos en el mismo nivel de jerarquía)
  - RETR_EXTERNAL
    - En este modo se devuelven únicamente los contornos externos
  - RETR_CCOMP
    - En este modo de funcionamiento todos los contornos se asignan en una jerarquía de 2 niveles: un nivel en el que se encuentran todos los contornos externos de los objetos y otro nivel en el que se encuentran todos los contornos internos
  - RETR_TREE
    - Este modo de funcionamiento es el más empleado ya que nos devuelve un jerarquía completa de todos los contornos detectados en la imagen

- Contour Approximation Method:
  - CHAIN_APPROX_NONE:
    - Se almacenan todos los puntos de cada contorno

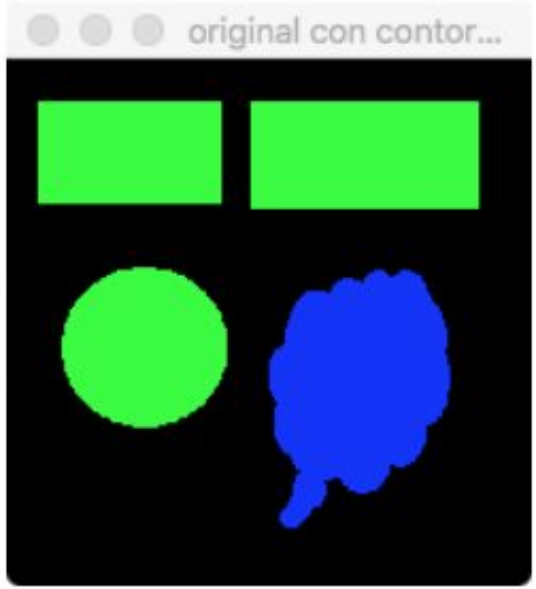
  - CHAIN_APPROX_SIMPLE
    - Se almacenan únicamente aquellos puntos que son suficientes para identificar con claridad cada contorno

- Dibujar contornos en OpenCV:
  - Función: ***drawContours***
  - Parámetros de Entrada:
    - 1: imagen sobre la que se van a dibujar los contornos
    - 2: Contornos a dibujar
      - Le podemos pasar una lista de contornos o un único contorno
    - 3: Índice del contorno a dibujar
      - -1 → todos los contornos
      - 0 → el primer contorno (o el que le hemos pasado si es único
    - 4: Vector BGR para determinar el color con el que se dibuja el contorno
      - (0, 255, 0) → color verde
    - Grosor del contorno a dibujar
      - -1 → no solo pinta el contorno sino que también lo rellena por completo

- Tareas a realizar en la práctica:
  - Abrir imagen formas.png
  - Binarizar imagen mediante umbralización empleando la función threshold
    - Analizar la imagen para establecer el mejor umbral para segmentar todas las figuras presentes en la imagen
  - Detectar los contornos de la imagen empleando la función findContours
    - Probar los distintos modos de recuperación (Control Retrieval Mode)
    - Probar los distintos métodos de aproximación (Contour Approximation Method)
  - Si lo hemos hecho correctamente deberíamos obtener 4 contornos. Para cada uno de ellos:
    - Visualizar la información que contiene
    - Obtener características: perímetro (arcLength), área (contourArea), momentos(moments), ...

- Ejemplo para los datos del primer contorno:

```
cnt = contours[0]
cv2.drawContours(img,[cnt],0,(255,0,0),-1)
M = cv2.moments(cnt)
print(M)
cx = int(M['m10']/M['m00'])
print "El centro de gravedad en x es %r" %cx
cy = int(M['m01']/M['m00'])
print "El centro de gravedad en y es %r" %cy
area = cv2.contourArea(cnt)
print "El area es %r" %area
perimeter = cv2.arcLength(cnt,True)
print "El perimeter es %r" % perimeter
cv2.imshow('original con contorno 1 en azul',img)
cv2.waitKey()
```



- Tareas a realizar en la práctica (continuación):
  - Visualizar y analizar los datos de la jerarquía de contornos

```
_, contours, jerarquia = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
print(jerarquia)
```

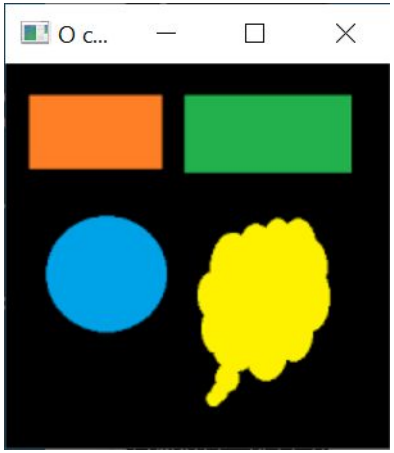
    - En nuestro ejemplo todos los contornos deberían estar al mismo nivel ya que no hay objetos dentro de otros objetos
  - Estudiar las mejores características a emplear para caracterizar adecuadamente los distintos contornos existentes en la imagen
    - Es muy importante que los descriptores sean invariantes con respecto al cambio de tamaño, giro y traslación
      - Para obtener momentos invariantes podemos calcularlos a partir de los momentos obtenidos por la función ***moments*** o emplear la función ***HuMoments***

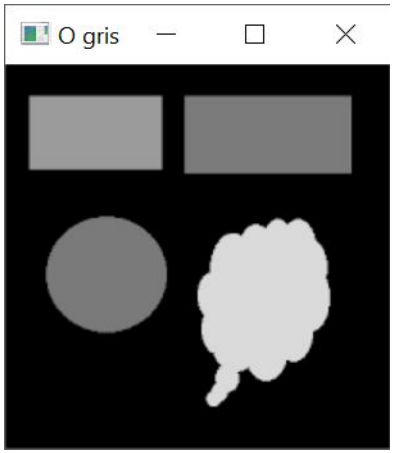
```
HU = cv2.HuMoments(cv2.moments(cnt))
print(HU)
```

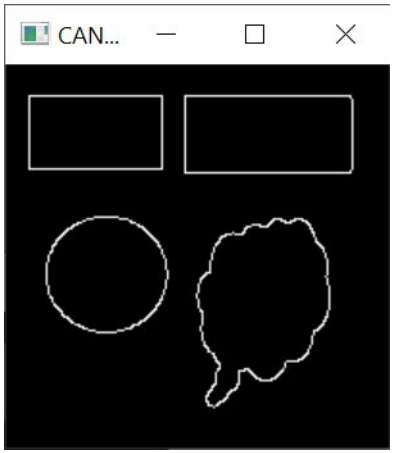
  - Hacer un programa en Python que dada la imagen "formas.png" y una determinada forma (de las 4 existentes en la imagen), devuelva una imagen de salida en la que únicamente aparezca la imagen de la forma determinada

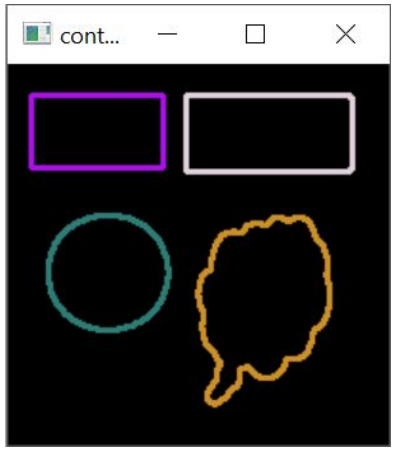
```python
import numpy as np
import cv2
import random as rng

# leemos y mostramos la imagen original en color
im = cv2.imread('formas.png', cv2.IMREAD_COLOR)
cv2.imshow('O color', im)
cv2.waitKey()

# leemos y mostramos la imagen original en niveles de gris
imgray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
cv2.imshow('O gris', imgray)
cv2.waitKey()

# leemos y mostramos la imagen de contornos detectados utilizando Canny
canny = cv2.Canny(imgray, 127, 254)
cv2.imshow('CANNY', canny)
cv2.waitKey()

# obtenemos los datos de los contornos y su jerarquia
contornos, jerarquia = cv2.findContours(canny, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

```
# y los mostramos sobre una imagen vacia
imdraw = np.zeros((canny.shape[0], canny.shape[1]))
for i in range(len(contornos)):
    color = (rng.randint(0,256), rng.randint(0,256), rng.randint(0,256))
    cv2.drawContours(imdraw, contornos, i, color, 2)
cv2.imshow('contornos', imdraw)
cv2.waitKey()

# num contornos detectados
numContornos = len(contornos)
print("num contornos detectados es %r" %numContornos)

# Jerarquia [Next, Previous, First_Child, Parent]
print(jerarquia)

# obtenemos y mostramos los momentos de cada uno de los contornos
# y su centro para reconocer su posición en la imagen
for contorno in contornos:
    M = cv2.moments(contorno)
    print(M)
    cx = int(M['m10']/M['m00'])
    cy = int(M['m01']/M['m00'])
    print('El centro de gravedad del contorno es (', cx, ',', cy, ')')
```









Símbolo del sistema

```
C:\Users\angel\va>python p1.py
num contornos detectados es 4
[[[ 1 -1 -1 -1]
  [ 2  0 -1 -1]
  [ 3  1 -1 -1]
  [-1  2 -1 -1]]]
{'m00': 4626.0, 'm10': 616412.5, 'm01': 568865.1666666666, 'm20': 83548102.83333333, 'm11': 75368810.33333333, 'm02': 72284317.5, 'm30': 11510283417.2
5, 'm21': 10158847285.233334, 'm12': 9520542566.666666, 'm03': 9469687115.45, 'mu20': 1411403.7074686587, 'mu11': -432227.18242901564, 'mu02': 2330236
.6857556403, 'mu30': 1396968.1769065857, 'mu21': 17850.129590153694, 'mu12': -5008424.389982045, 'mu03': 7687270.388593674, 'nu20': 0.0659538264365951
7, 'nu11': -0.02019764892231224, 'nu02': 0.10889019570747233, 'nu30': 0.000959781648748455, 'nu21': 1.2263863337494564e-05, 'nu12': -0.003441018842171
0786, 'nu03': 0.00528150975083592 1}
El centro de gravedad del contorno es ( 133 , 122 )
{'m00': 2995.0, 'm10': 156317.8333333333, 'm01': 326058.5, 'm20': 8904542.333333332, 'm11': 17019918.0, 'm02': 36180735.33333333, 'm30': 542613474.85,
 'm21': 969590568.8333334, 'm12': 1888805071.5333333, 'm03': 4087765694.25, 'mu20': 745856.1837414224, 'mu11': 1968.6644685603678, 'mu02': 683524.8417
640477, 'mu30': 2453.276355266571, 'mu21': -31197.818332001567, 'mu12': -2259.111303716898, 'mu03': 27977.749061584473, 'nu20': 0.08314984448108254, '
nu11': 0.00021947145839174, 'nu02': 0.0762009962919889, 'nu30': 4.997518451941602e-06, 'nu21': -6.355242956619886e-05, 'nu12': -4.601988469060344e-06
, 'nu03': 5.6992893148341444e-05}
El centro de gravedad del contorno es ( 52 , 108 )
{'m00': 3477.0, 'm10': 474481.3333333333, 'm01': 125172.0, 'm20': 66938519.166666664, 'm11': 17081328.0, 'm02': 4969082.833333333, 'm30': 9732181140.6
, 'm21': 2409786690.0, 'm12': 678064655.5333333, 'm03': 212215122.0, 'mu20': 2189443.6182692647, 'mu11': 0.0, 'mu02': 462890.833333333, 'mu30': 4528.5
585861206055, 'mu21': -2.384185791015625e-07, 'mu12': -30555.339194059372, 'mu03': 2.9802322387695312e-08, 'nu20': 0.1811024745686341, 'nu11': 0.0, 'n
u02': 0.03828857462795556, 'nu30': 6.352552514213169e-06, 'nu21': -3.3444781938974917e-16, 'nu12': -4.2862291196753136e-05, 'nu03': 4.1805977423718647
e-17}
El centro de gravedad del contorno es ( 136 , 36 )
{'m00': 2622.0, 'm10': 121923.0, 'm01': 91770.0, 'm20': 6709698.0, 'm11': 4267305.0, 'm02': 3527464.0, 'm30': 408746857.5, 'm21': 234839430.0, 'm12':
164027076.0, 'm03': 145547220.0, 'mu20': 1040278.5, 'mu11': 0.0, 'mu02': 315514.0, 'mu30': 0.0, 'mu21': 0.0, 'mu12': 0.0, 'mu03': 0.0, 'nu20': 0.15131
578947368424, 'nu11': 0.0, 'nu02': 0.045893719806763295, 'nu30': 0.0, 'nu21': 0.0, 'nu12': 0.0, 'nu03': 0.0}
El centro de gravedad del contorno es ( 46 , 35 )

C:\Users\angel\va>
```

Práctica 2

Reconocimiento de formas mediante la Transformada de Hough en OpenCV – Python

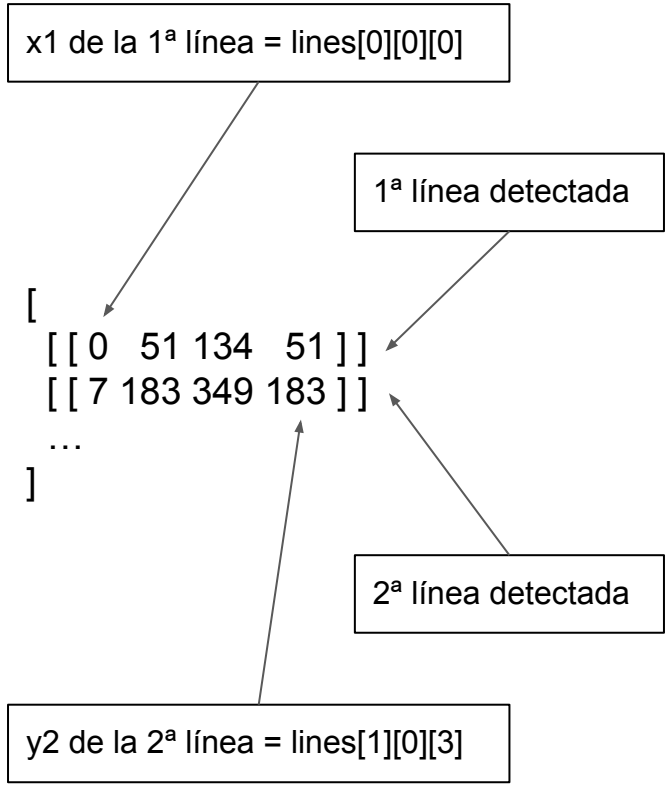
- Encontrar líneas rectas con la transformada estándar:
  - Función: ***HoughLines***
  - Parámetros de Entrada:
    - 1: Imagen binaria sobre la que se va a realizar la búsqueda de líneas rectas
      - Habrá que umbralizar o emplear el método canny para binarizar la imagen si no lo hemos hecho antes
    - 2 y 3: Precisión a emplear en rho y theta respectivamente
      - Precisión del tamaño de cada una de las muestras empleadas a la hora de discretizar tanto el desplazamiento como el ángulo
    - 4: Umbral o mínimo valor del número de votos para considerar que se trata de una línea.
      - El número de votos indica el número de puntos sobre la línea. Por lo tanto, representa la longitud mínima de las líneas a detectar
  - Parámetros de Salida:
    - 1: Devuelve los parámetros rho y theta de cada línea detectada

- Encontrar líneas rectas con la transformada probabilística optimizada:
  - Función: ***HoughLinesP***
  - Diferencias con la transformada estándar:
    - Es una implementación basada en el método descrito en el artículo "Robust Detection of Lines Using the Progressive Probabilistic Hough Transform"
    - Tiene dos parámetros más:
      - minLineLength: longitud mínima de una línea. No se consideran los segmentos más cortos que esta distancia
      - maxLineGap: Distancia máxima permitida entre los segmentos de línea para tratarlos como una sola línea
    - Devuelve directamente los dos puntos finales de cada línea detectada x1, y1, x2, y2

- Un ejemplo de cómo utilizar la Transformada de Hough para detectar líneas:
  - En el tutorial oficial de OpenCV-Python:
  - https://docs.opencv.org/3.4.0/d6/d10/tutorial_py_houghlines.html
  - En la carpeta: opencv/doc/py_tutorials/py_imgproc/py_houghlines
- Un ejemplo de cómo utilizar la Transformada de Hough para detectar circunferencias:
  - En el tutorial oficial de OpenCV-Python:
  - https://docs.opencv.org/3.4.0/da/d53/tutorial_py_houghcircles.html
  - En la carpeta: opencv/doc/py_tutorials/py_imgproc/py_houghcircles
- Un ejemplo de cómo utilizar la función HoughCircles para detectar circunferencias en el que se describen todos sus parámetros:
  - https://www.pyimagesearch.com/2014/07/21/detecting-circles-images-using-opencv-hough-circles/
- Un ejemplo de cómo utilizar la Transformada de Hough para detectar las líneas de la carretera en OpenCV – Python:
  - https://campushippo.com/lessons/detect-highway-lane-lines-with-opencv-and-python-21438a3e2

- Tareas a realizar en la práctica:
  - Elegir una imagen cualquiera en la que haya líneas rectas y utilizar la Transformada de Hough para detectar dichas líneas
  - Elegir una imagen cualquiera en la que haya circunferencias y utilizar la Transformada de Hough para detectar dichas circunferencias

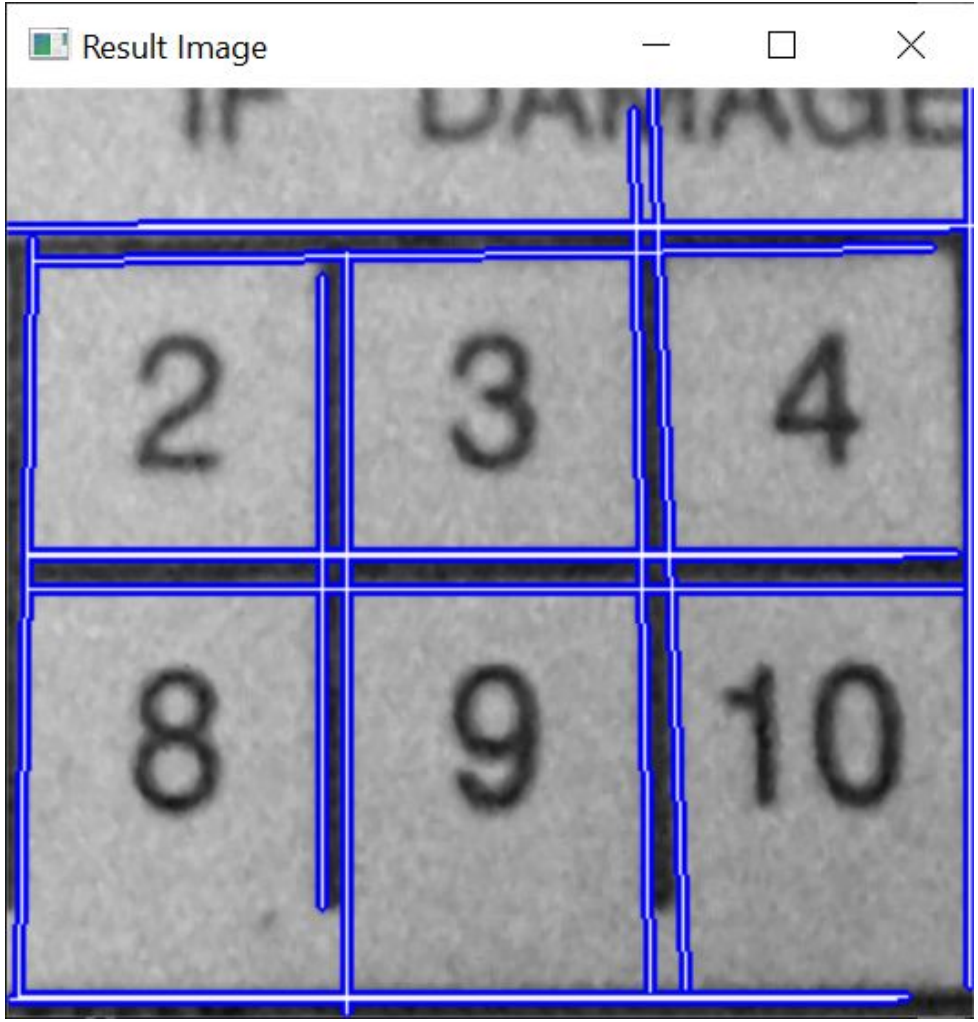
```python
import cv2
import numpy as np

# Leemos la imagen original
img = cv2.imread('numeros.png', cv2.IMREAD_COLOR)

# Convertimos la imagen a niveles de gris
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Encontramos los contornos empleando Canny
edges = cv2.Canny(gray, 50, 200)

# Detectamos las lineas empleando la transforma de Hough
lines = cv2.HoughLinesP(edges, 1, np.pi / 180, 100, minLineLength = 10, maxLineGap = 250)

# Mostramos el numero de lineas encontradas
print ("num lineas encontradas: %r" %len(lines))

# Mostramos el array de 3 dimensiones devuelto por la transformada
print (lines)

# Mostramos los valores x1,y1,x2,y2 dela primera linea detectada
print(lines[0][0][0])
print(lines[0][0][1])
print(lines[0][0][2])
print(lines[0][0][3])
```



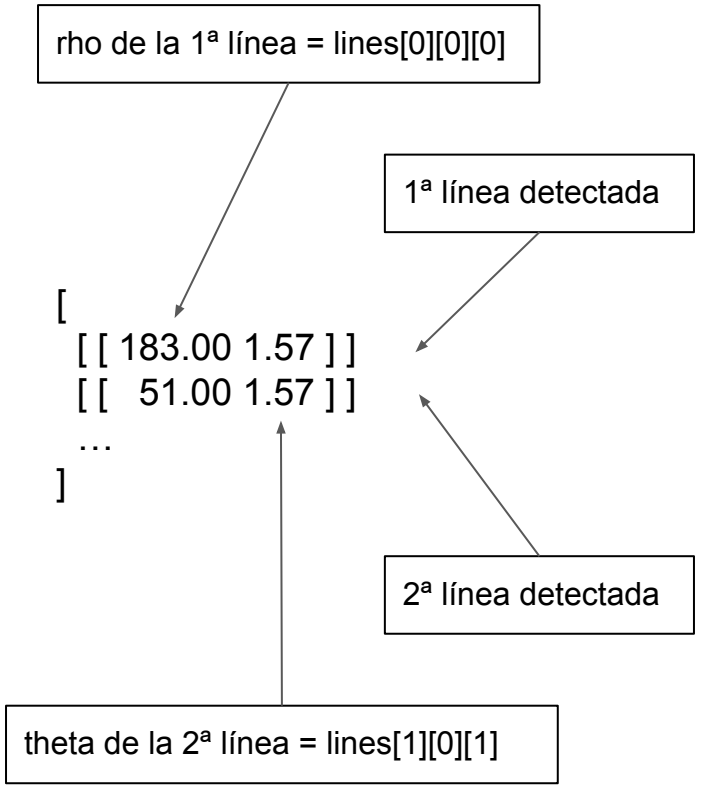
```
# Dibujamos en azul todas las lineas detectadas con un grosor de 3 pixeles
cv2.line(img, (lines[0][0][0], lines[0][0][1]), (lines[0][0][2], lines[0][0][3]), (255, 0, 0), 3)
cv2.line(img, (lines[1][0][0], lines[1][0][1]), (lines[1][0][2], lines[1][0][3]), (255, 0, 0), 3)
cv2.line(img, (lines[2][0][0], lines[2][0][1]), (lines[2][0][2], lines[2][0][3]), (255, 0, 0), 3)
cv2.line(img, (lines[3][0][0], lines[3][0][1]), (lines[3][0][2], lines[3][0][3]), (255, 0, 0), 3)
cv2.line(img, (lines[4][0][0], lines[4][0][1]), (lines[4][0][2], lines[4][0][3]), (255, 0, 0), 3)
cv2.line(img, (lines[5][0][0], lines[5][0][1]), (lines[5][0][2], lines[5][0][3]), (255, 0, 0), 3)
cv2.line(img, (lines[6][0][0], lines[6][0][1]), (lines[6][0][2], lines[6][0][3]), (255, 0, 0), 3)
cv2.line(img, (lines[7][0][0], lines[7][0][1]), (lines[7][0][2], lines[7][0][3]), (255, 0, 0), 3)
cv2.line(img, (lines[8][0][0], lines[8][0][1]), (lines[8][0][2], lines[8][0][3]), (255, 0, 0), 3)
cv2.line(img, (lines[9][0][0], lines[9][0][1]), (lines[9][0][2], lines[9][0][3]), (255, 0, 0), 3)
cv2.line(img, (lines[10][0][0], lines[10][0][1]), (lines[10][0][2], lines[10][0][3]), (255, 0, 0), 3)
cv2.line(img, (lines[11][0][0], lines[11][0][1]), (lines[11][0][2], lines[11][0][3]), (255, 0, 0), 3)
cv2.line(img, (lines[12][0][0], lines[12][0][1]), (lines[12][0][2], lines[12][0][3]), (255, 0, 0), 3)
cv2.line(img, (lines[13][0][0], lines[13][0][1]), (lines[13][0][2], lines[13][0][3]), (255, 0, 0), 3)

# Dibujamos las lineas en sobre la imagen original en blanco y un pixel de grosor
for line in lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(img, (x1, y1), (x2, y2), (255, 255, 255), 1)

# Mostramos la imagen con todas las lineas detectadas
cv2.imshow("Result Image", img)
cv2.waitKey()
```

Seleccionar Símbolo del sistema

```
C:\Users\angel\va\p2>python p2a.py
num lineas encontradas: 14
[[[  0  51 344  51]]

 [[  7 183 349 183]]

 [[  5 332 329 332]]

 [[236   0 248 329]]

 [[  4 333   9  55]]

 [[  8 170 347 170]]

 [[  8 171 327 171]]

 [[ 48  50 353  50]]

 [[  1 333 325 333]]

 [[ 10  63 338  58]]

 [[115 299 115  69]]

 [[229   8 235 330]]

 [[124 338 124  60]]

 [[352 328 352   0]]]
0
51
344
51

C:\Users\angel\va\p2>python p2a.py
```

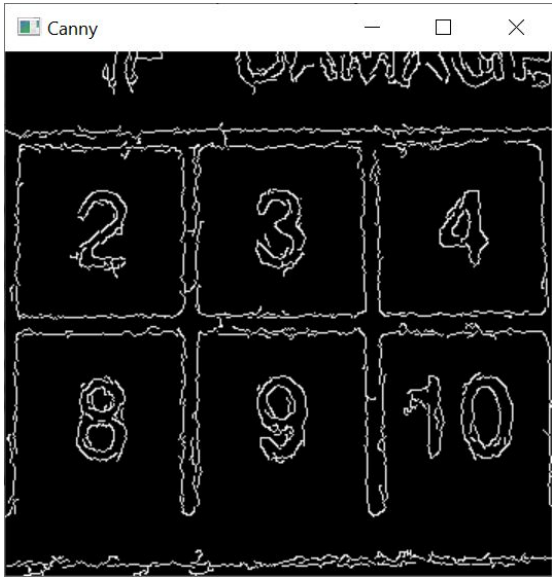


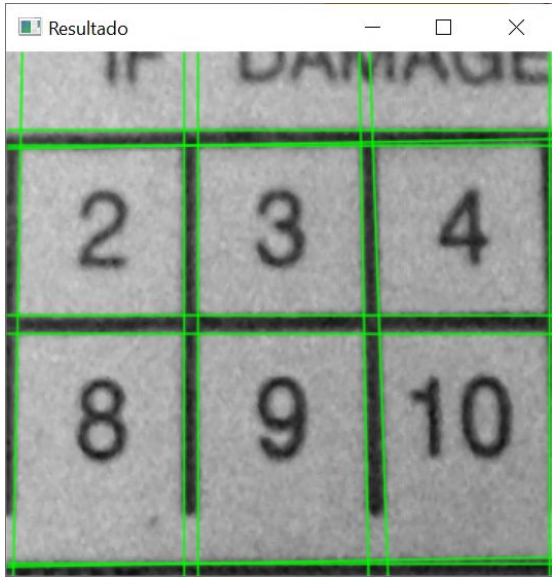
```python
import cv2
import numpy as np

# Leemos y mostramos la imagen original
img = cv2.imread('numeros.png', cv2.IMREAD_COLOR)
cv2.imshow("Original", img)
cv2.waitKey()

# Convertimos la imagen a niveles de gris
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# obtenemos y mostramos la imagen de contornos con Canny
edges = cv2.Canny(gray, 50, 200)
cv2.imshow("Canny", edges)
cv2.waitKey()

# Obtenemos las lineas mediante la transformada de Hough
lines = cv2.HoughLines(edges, 1, np.pi / 180, 100)

# Mostramos el numero de lineas encontradas
print ("num lineas encontradas: %r" %len(lines))

# Mostramos el array de 3 dimensiones devuelto por la transformada
print (lines)

# Mostramos los valores rho,theta de las dos primeras lineas detectadas
print(lines[0][0][0])
print(lines[0][0][1])
print(lines[1][0][0])
print(lines[1][0][1])
```

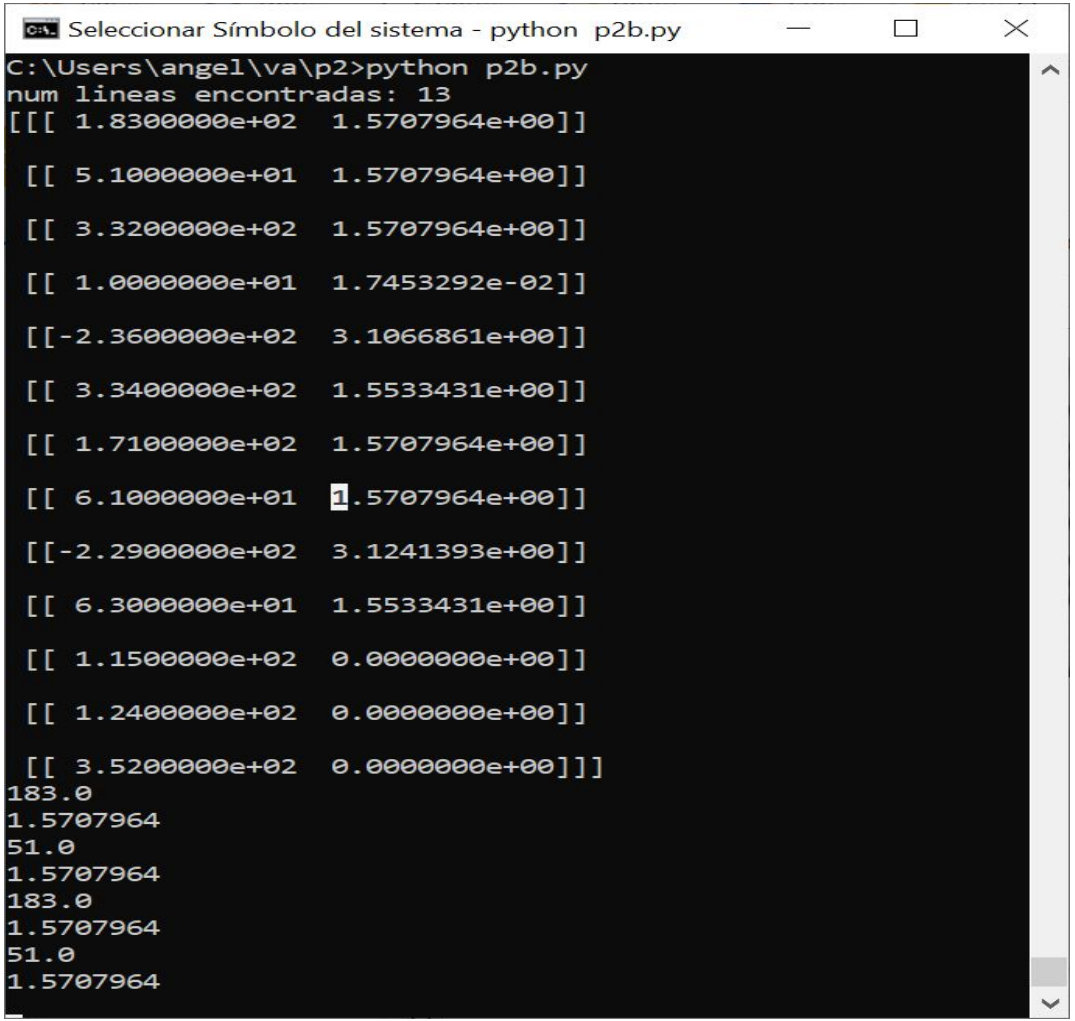


```
# Dibujamos en verde todas las lineas detectadas con un grosor de un pixel
if lines is not None:
    for i in range(0, len(lines)):
        rho = lines[i][0][0]
        print(rho)
        theta = lines[i][0][1]
        print(theta)
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a * rho
        y0 = b * rho
        pt1 = (int(x0 + 1000 * (-b)), int(y0 + 1000 * (a)))
        pt2 = (int(x0 - 1000 * (-b)), int(y0 - 1000 * (a)))
        cv2.line(img, pt1, pt2, (0, 255, 0), 1, cv2.LINE_AA)

# Mostramos la imagen con todas las lineas detectadas
cv2.imshow("Resultado", img)
cv2.waitKey()
```


Original
2
3
4
8
9
10


Canny
2
3
4
8
9
10


Resultado
2
3
4
8
9
10


Seleccionar Símbolo del sistema - python p2b.py
C:\Users\angel\va\p2>python p2b.py
num lineas encontradas: 13
[[[1.8300000e+02 1.5707964e+00]]
[[5.1000000e+01 1.5707964e+00]]
[[3.3200000e+02 1.5707964e+00]]
[[1.0000000e+01 1.7453292e-02]]
[[-2.3600000e+02 3.1066861e+00]]
[[3.3400000e+02 1.5533431e+00]]
[[1.7100000e+02 1.5707964e+00]]
[[6.1000000e+01 1.5707964e+00]]
[[-2.2900000e+02 3.1241393e+00]]
[[6.3000000e+01 1.5533431e+00]]
[[1.1500000e+02 0.0000000e+00]]
[[1.2400000e+02 0.0000000e+00]]
[[3.5200000e+02 0.0000000e+00]]]
183.0
1.5707964
51.0
1.5707964
183.0
1.5707964
51.0
1.5707964