Análisis de imagen

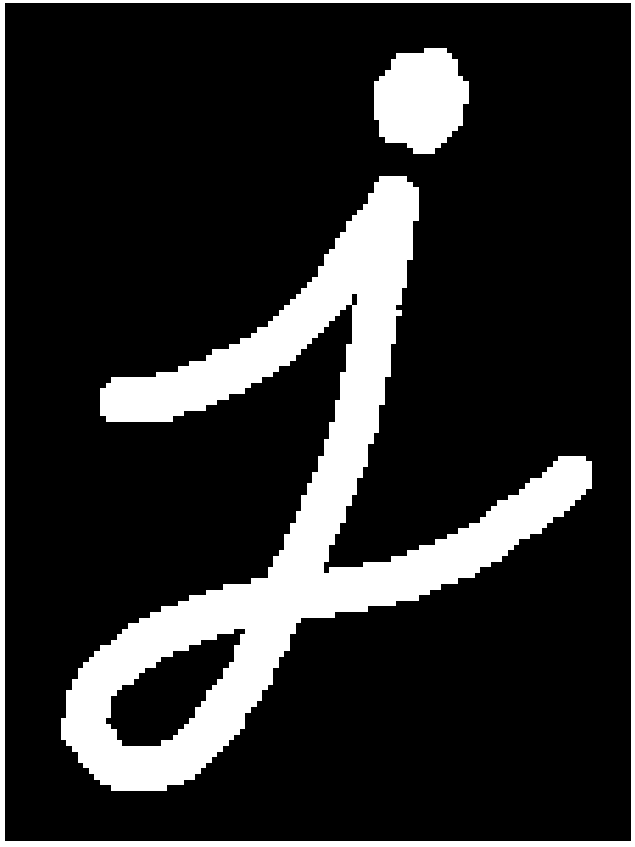
Araguás, Gastón Redolfi, Javier

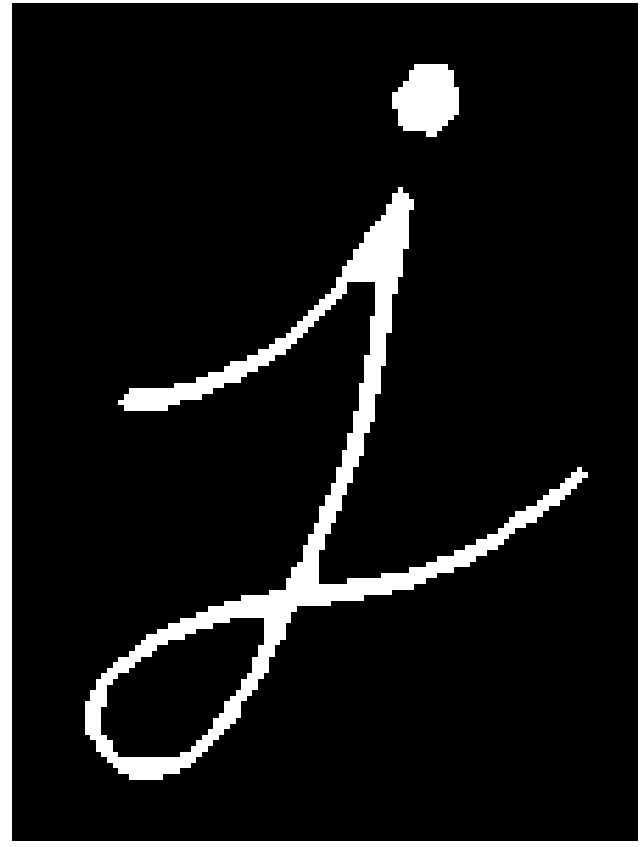
15 de Mayo del 2019

Transformaciones Morfológicas

- Son operaciones simples que se realizan sobre las imágenes.
- Están basadas en un kernel, núcleo o filtro.
- Las operadores más comunes son:
  - Erosión
  - Dilatación
- Además pueden ser combinadas para hacer
  - Apertura
  - Closing (Cerrado?)
  - Gradiente morfológico

Erosión

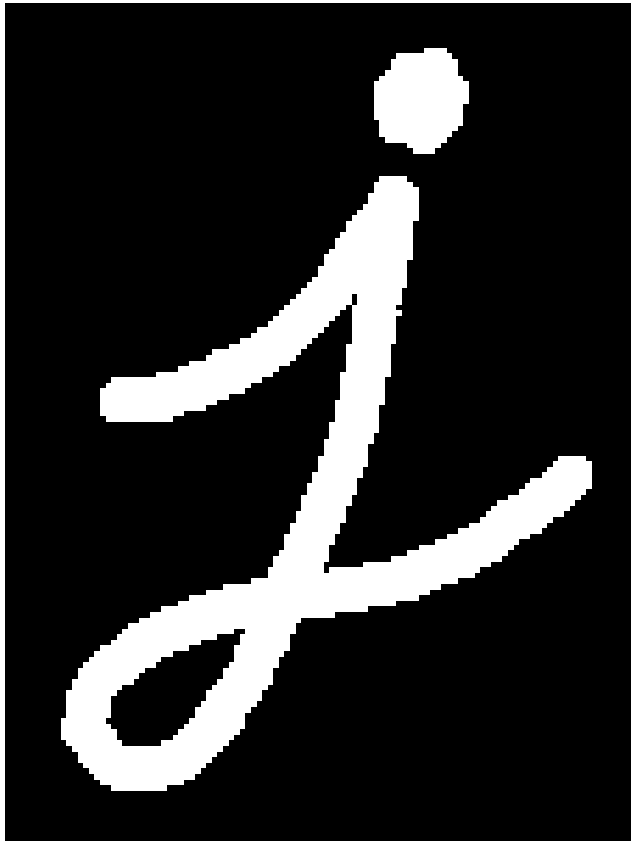




Código para aplicar erosión

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-


import cv2
import numpy as np

img = cv2.imread('j.png', 0)
kernel = np.ones((5, 5), np.uint8)
eroded = cv2.erode(img, kernel, iterations=1)

cv2.imwrite('j_erode.png', eroded)
```

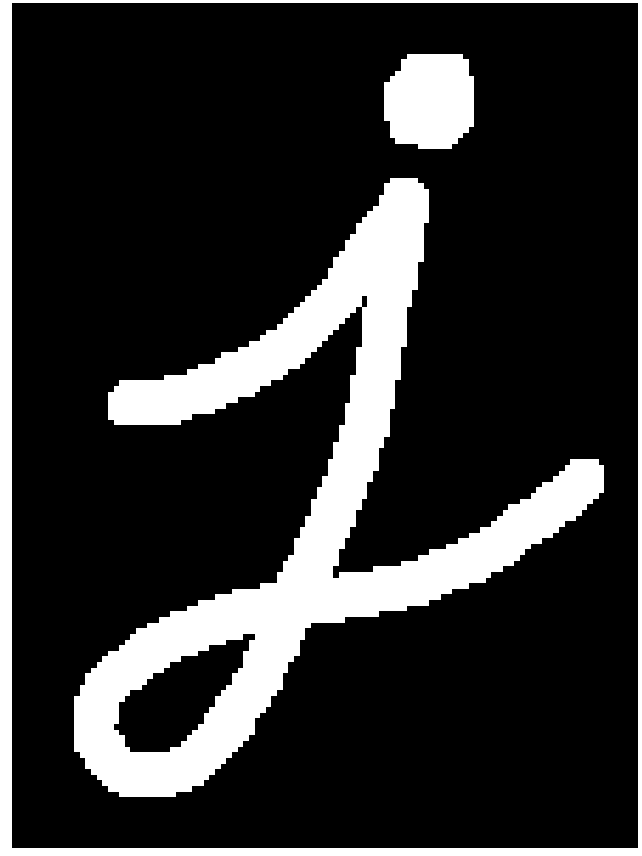
Dilatación



Código para aplicar dilatación

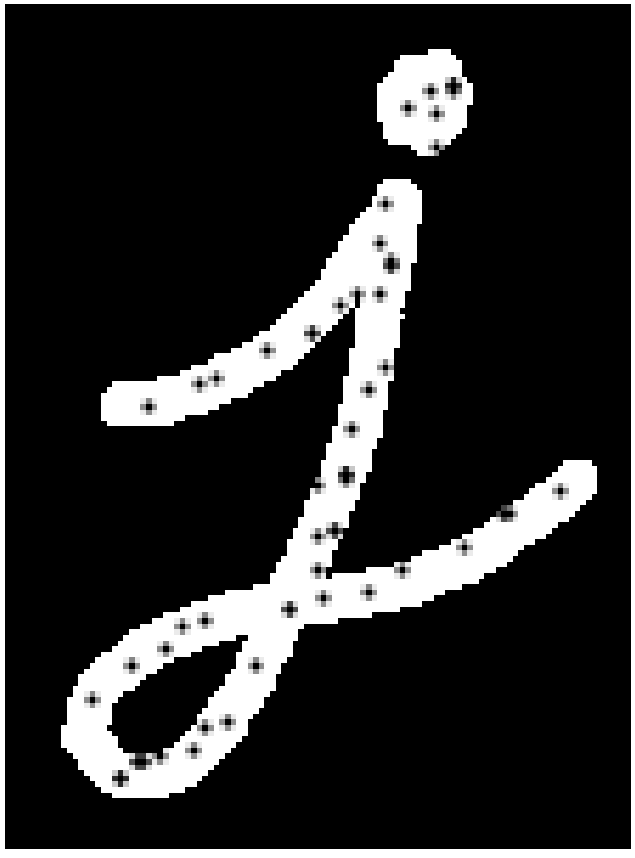
```
#! /usr/bin/env python
# -*- coding: utf-8 -*-


import cv2
import numpy as np

img = cv2.imread('j.png', 0)
kernel = np.ones((5, 5), np.uint8)
dilated = cv2.dilate(img, kernel, iterations=1)

cv2.imwrite('j_dilate.png', dilated)
```

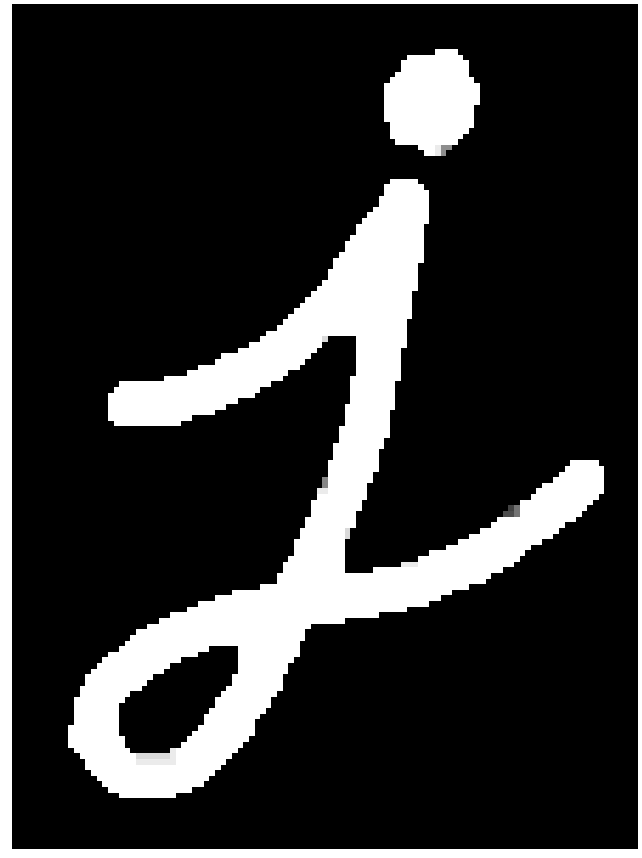
Apertura



Código para aplicar apertura

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-


import cv2
import numpy as np

img = cv2.imread('j_noise.png', 0)
kernel = np.ones((5, 5), np.uint8)
opened = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)

cv2.imwrite('j_opened.png', opened)
```

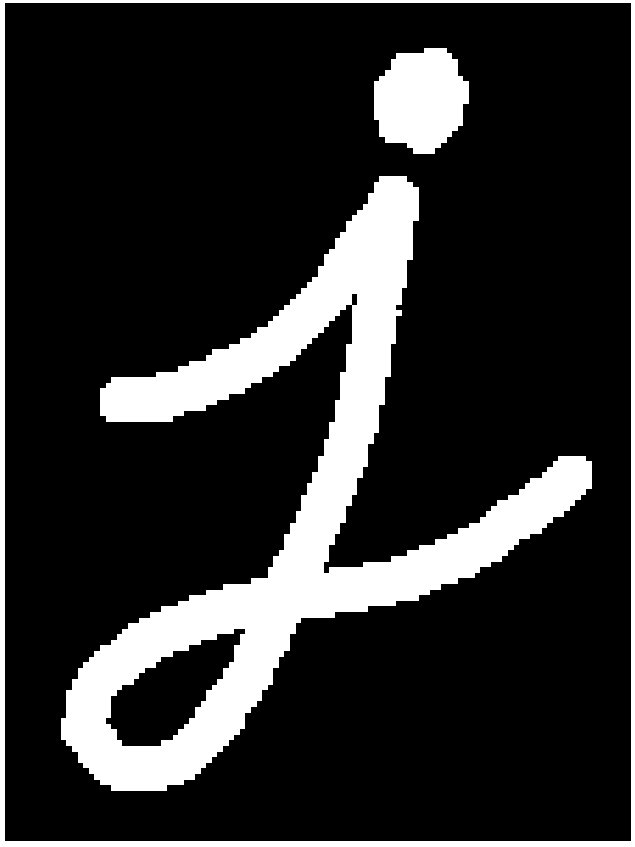


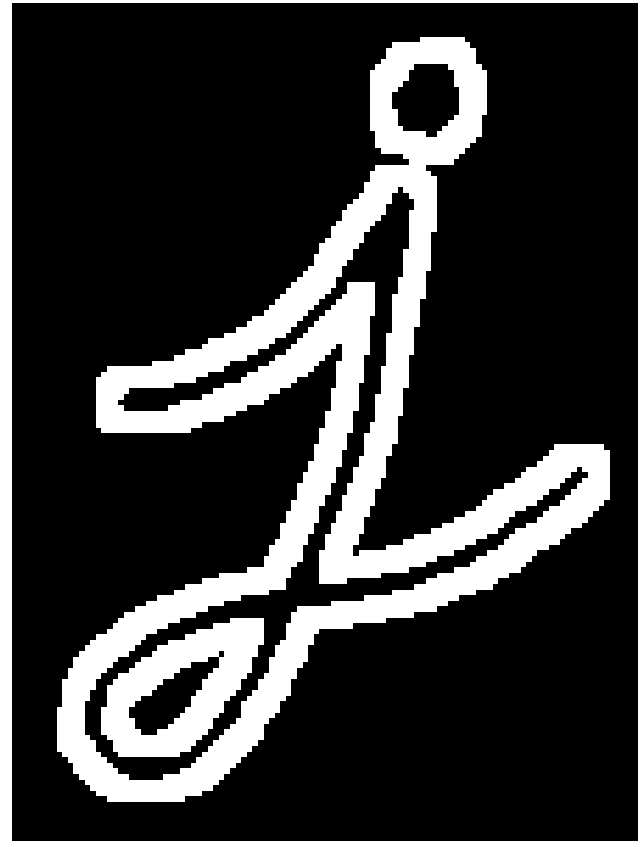


Código para aplicar Cerrado

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-


import cv2
import numpy as np

img = cv2.imread('j_noise_1.png', 0)
kernel = np.ones((5, 5), np.uint8)
closed = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)

cv2.imwrite('j_closed.png', closed)
```

Gradiente morfológico





Código para aplicar gradiente morfológico

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-


import cv2
import numpy as np

img = cv2.imread('j.png', 0)
kernel = np.ones((5, 5), np.uint8)
gradient = cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel)

cv2.imwrite('j_gradient.png', gradient)
```

Gradientes

- Aplican filtros que calculan el gradiente de las imágenes.
- Los gradientes representan saltos bruscos.
- Decimos que son detectores de bordes.
- Los más comunes son:
  - Sobel
  - Laplaciano
  - Canny (no es solo gradiente)

Sobel

- Calcula un suavizado gaussiano más una derivada direccional.

Sobre el eje x

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

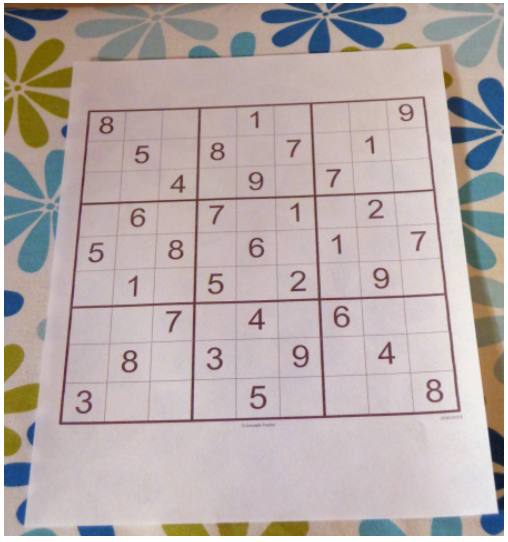
Sobre el eje y

$$S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

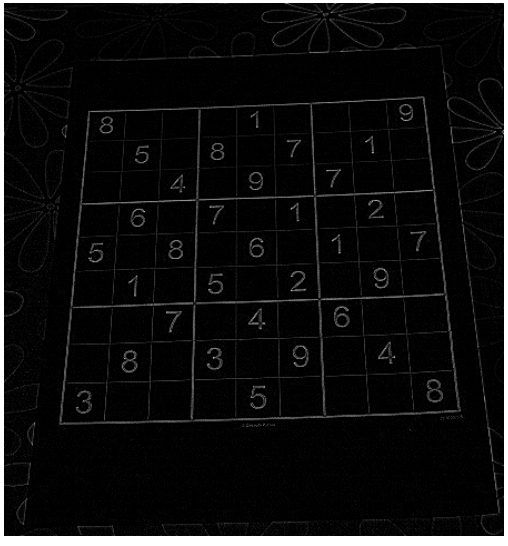
Laplaciano

- Calcula un kernel del tipo.

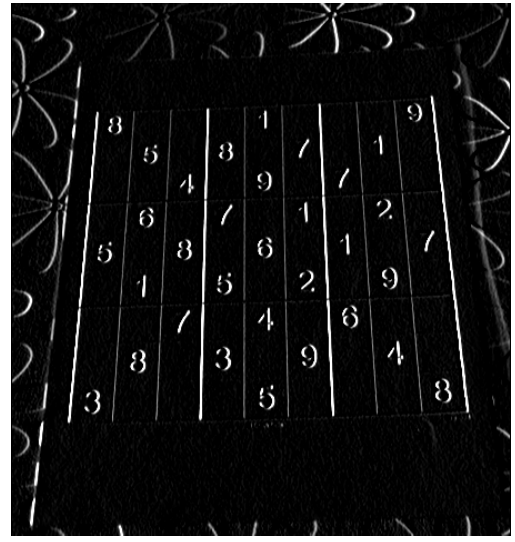
$$S_x = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$



(a) Original



(b) Laplaciano



(c) Sobel x

(d) Sobel y

Código para Sobel y laplaciano

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-


import cv2

img = cv2.imread('sudoku.jpeg', 0)

laplacian = cv2.Laplacian(img, cv2.CV_64F)

sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3)
sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3)

cv2.imwrite('laplacian.png', laplacian)
cv2.imwrite('sobelx.png', sobelx)
cv2.imwrite('sobely.png', sobely)
```

Canny

- Algoritmo multi-etapa.
  1. Reducción de ruido con un kernel gaussiano.
  2. Gradiente en x e y con Sobel.
  3. Supresión de no-máximo.
  4. Umbralizado con histéresis.

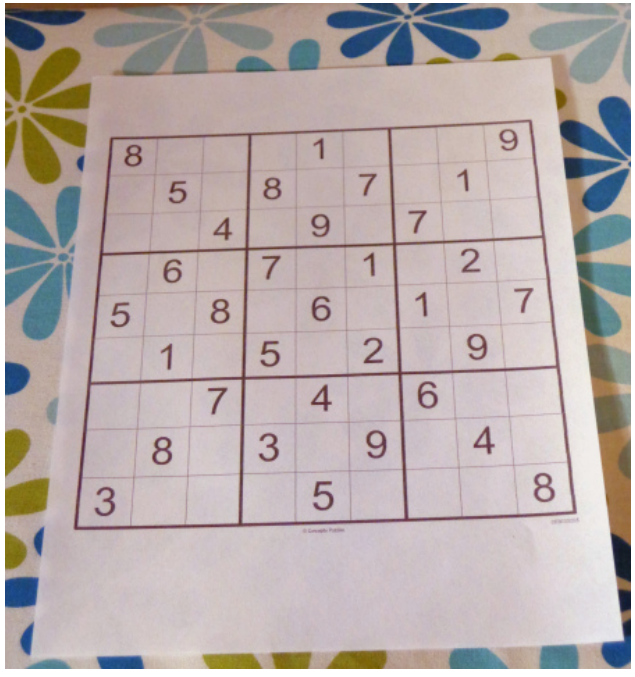
Código para Canny

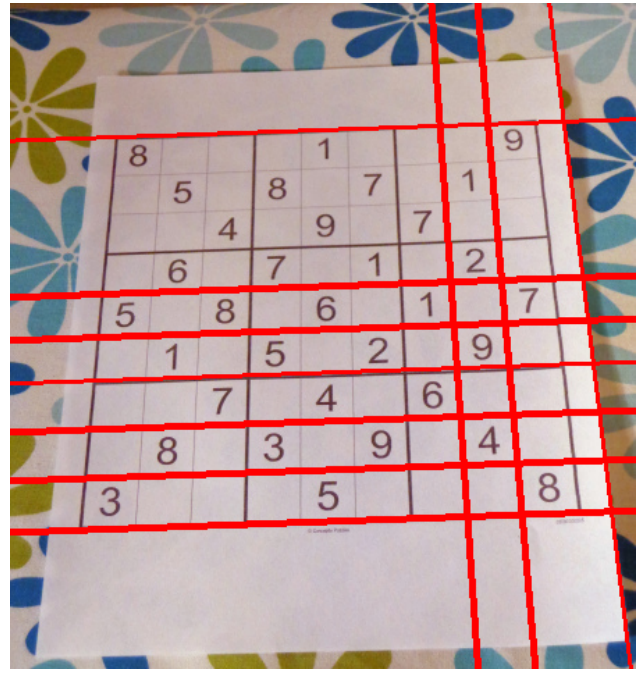
```
#! /usr/bin/env python
# -*- coding: utf-8 -*-

import cv2

img = cv2.imread('lena.png', 0)
canny = cv2.Canny(img, 100, 200)

cv2.imwrite('lena_canny.png', canny)
```

Transformada de Hough

- Permite detectar formas que puedan ser representadas en forma matemática.
- Lo más común es usarla para detectar:
  - Líneas
  - Círculos

Hough para líneas

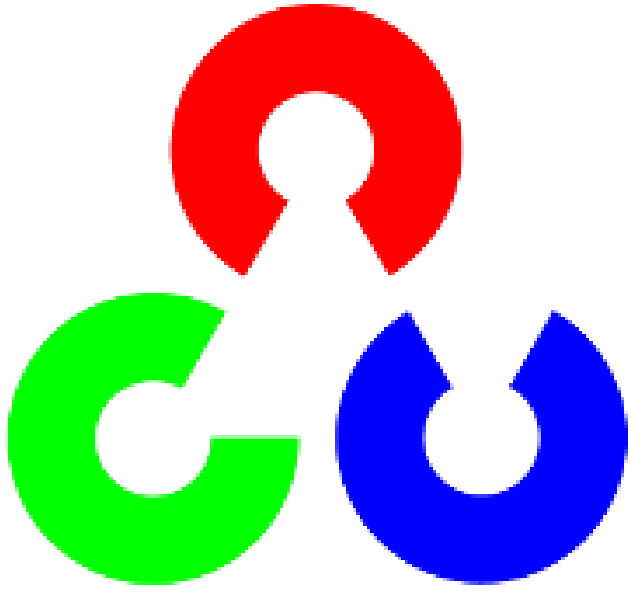


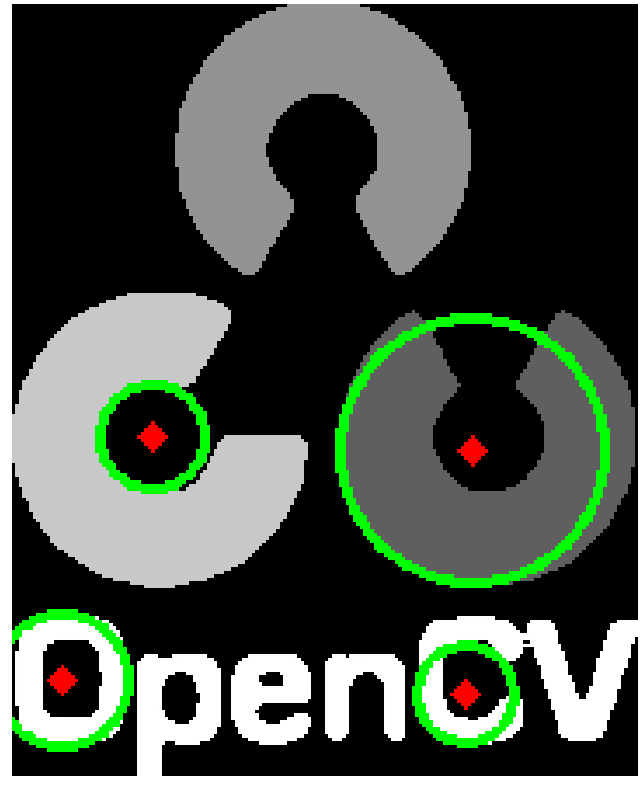


Código para detectar líneas con Hough

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-


import cv2
import numpy as np
img = cv2.imread('sudoku.jpeg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 50, 150, apertureSize=3)
lines = cv2.HoughLines(edges, 1, np.pi/180, 200)

for line in lines:
    rho, theta = line[0]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    cv2.line(img, (x1, y1), (x2, y2), (0, 0, 255), 2)

cv2.imwrite('sudoku_hough_lines.png', img)
```

Hough para círculos





Código para detectar círculos con Hough

```
#! /usr/bin/env python
# -*- coding: utf-8 -*-


import cv2
import numpy as np

img = cv2.imread('opencv-logo-white.png', 0)
img = cv2.medianBlur(img, 5)

cimg = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
circles = cv2.HoughCircles(img, cv2.HOUGH_GRADIENT, 1, 20,
                           param1=50, param2=30,
                           minRadius=0, maxRadius=0)
circles = np.uint16(np.around(circles))

for i in circles[0, :]:
    cv2.circle(cimg, (i[0], i[1]), i[2], (0, 255, 0), 2)
    cv2.circle(cimg, (i[0], i[1]), 2, (0, 0, 255), 3)

cv2.imwrite('opencv_hough_circles.png', cimg)
```