
FIAP

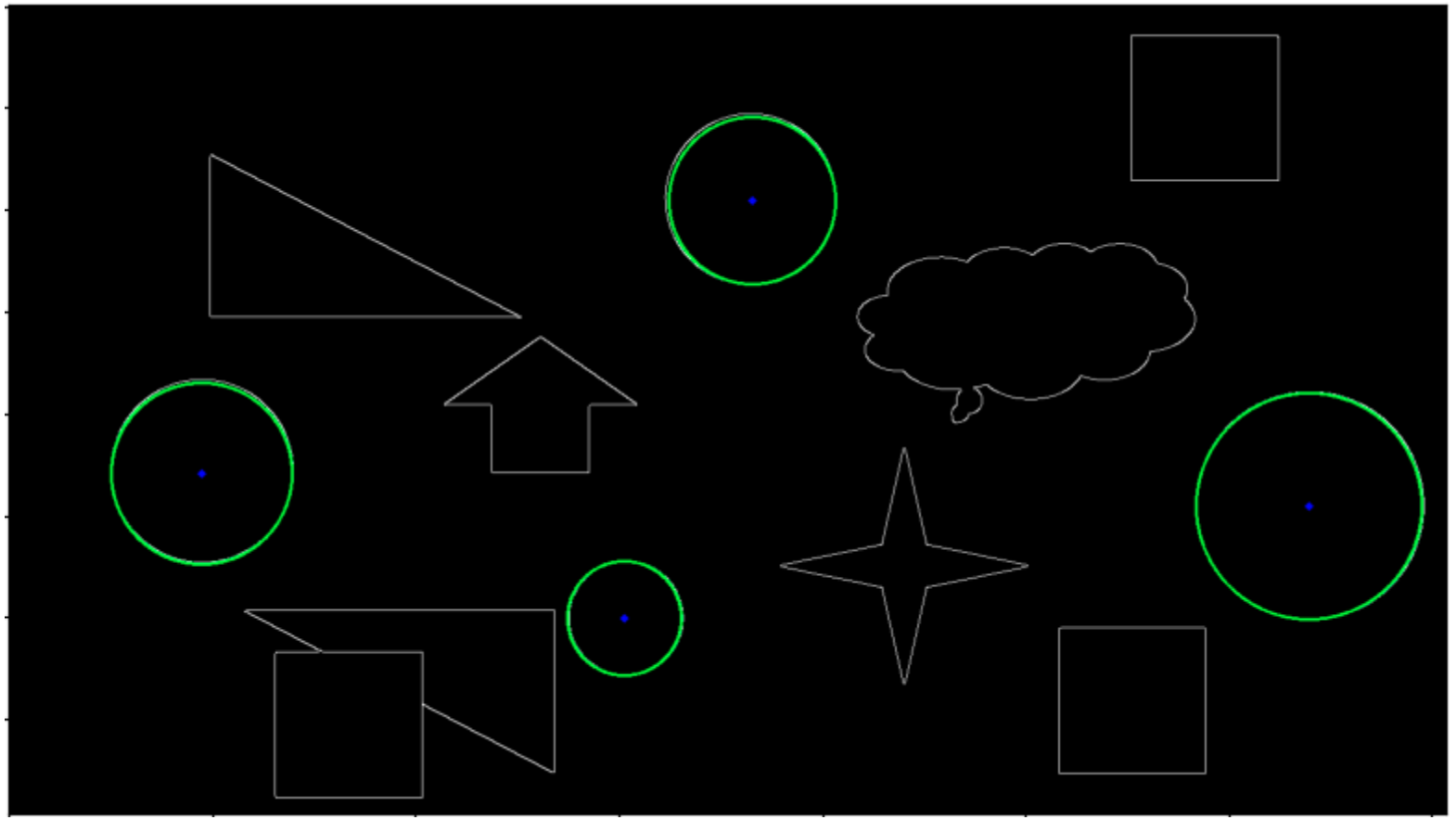
1. Transformada de Hough e morfologia

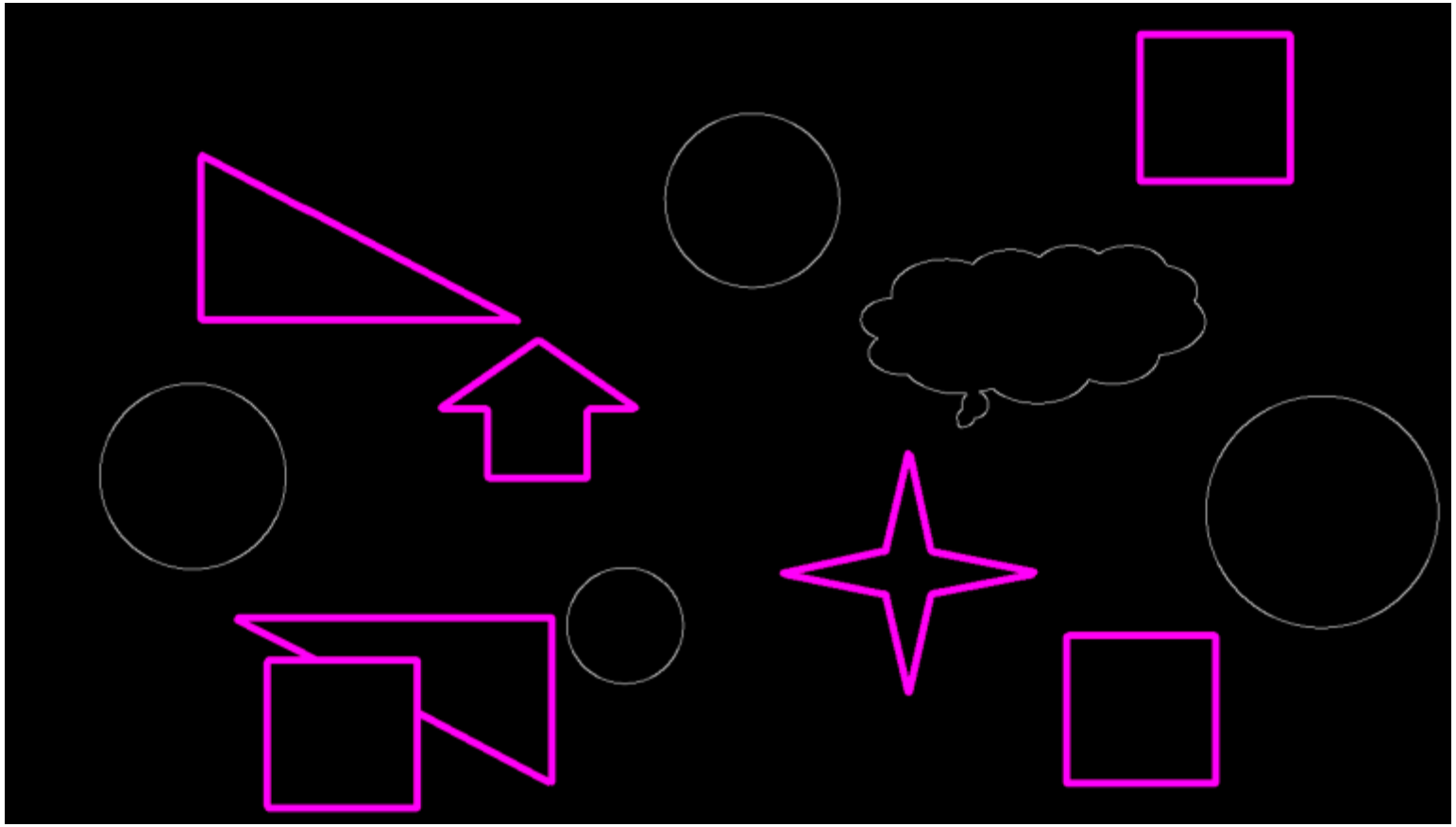
Objetivos da aula:

- Conhecer e praticar a Transformada de Hough para linhas e circulos
- conhecer e praticar com os operadores de dilatação e erosão
- conhecer e praticar com os operadores de abertura e fechamento

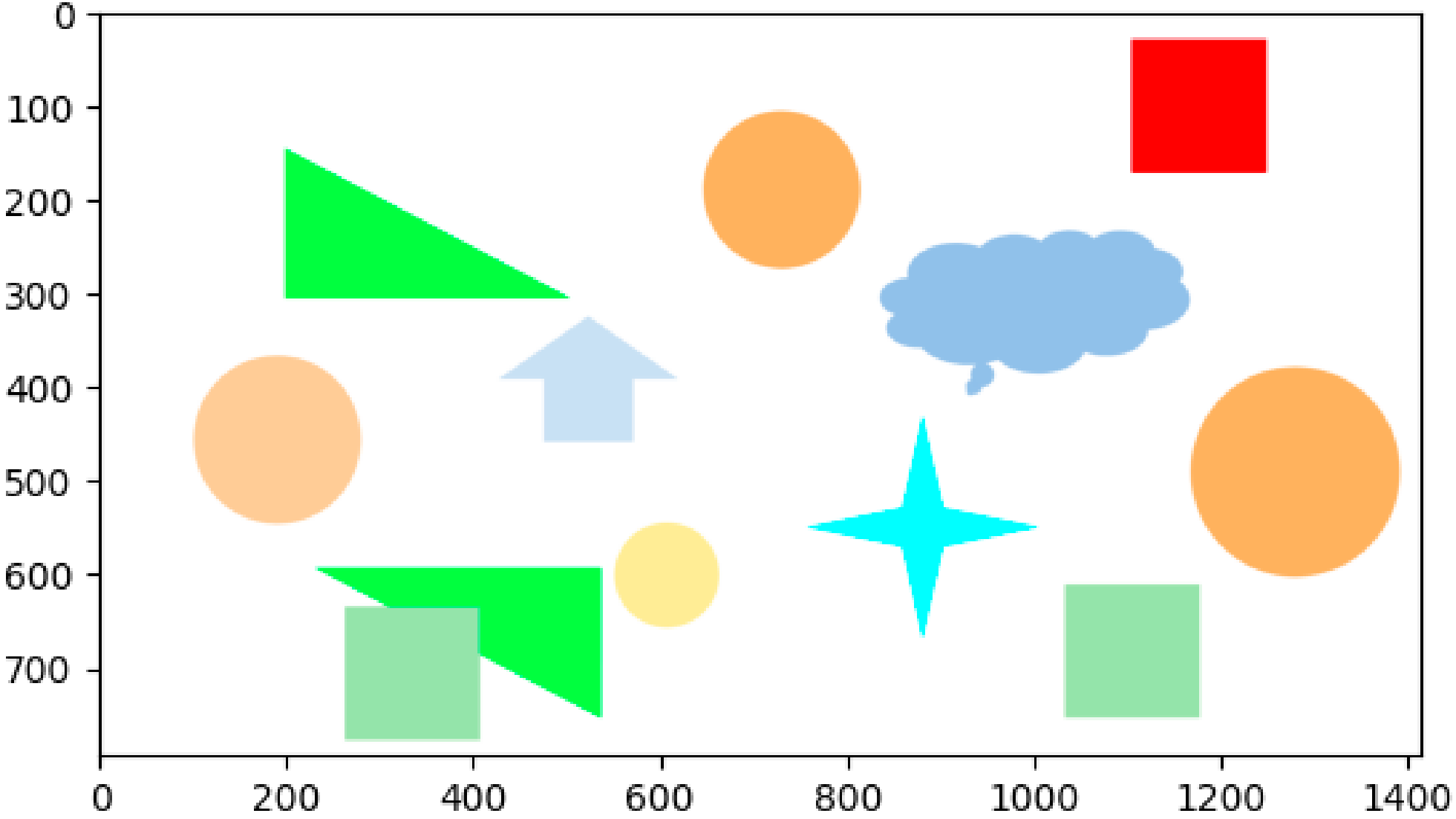
Transformada de Hough

- A transformada de Hough é um método utilizado para reconhecimento de padrões simples como retas e círculos. A aplicação da técnica é feita em contornos de imagem. É uma técnica muito popular e muito poderosa, pois possibilita detectar linhas e círculos em imagens com pouca visibilidade ou com muito ruido.

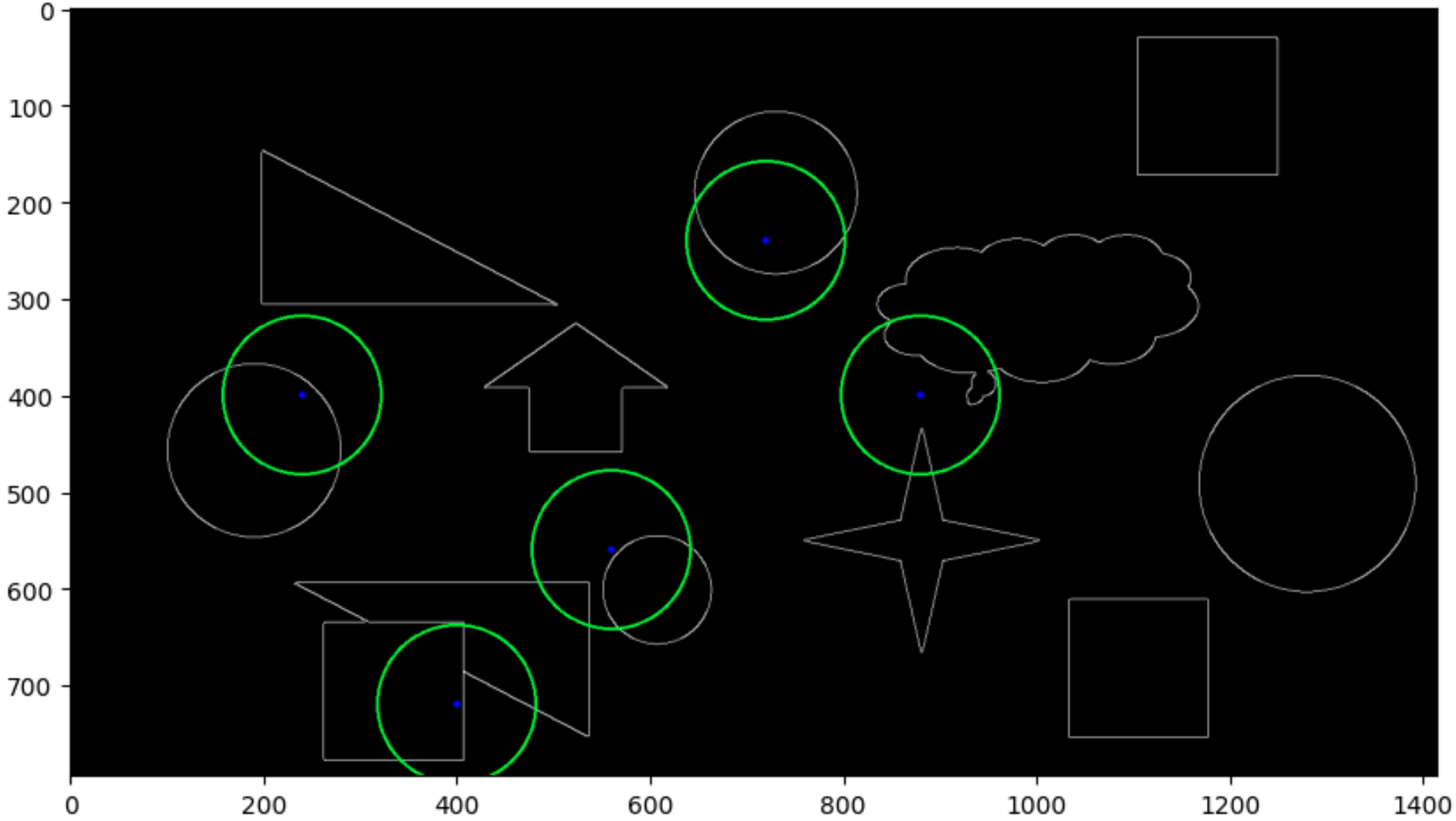




```
%matplotlib inline
import cv2
from matplotlib import pyplot as plt
import numpy as np
import math
img = cv2.imread('formas.png')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)


plt.imshow(img)
```


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200
1400

```
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(img_gray,50,150)
circles=cv2.HoughCircles(edges,cv2.HOUGH_GRADIENT,dp=160,minDist=100,param1=200,param2=100,minRadius=50,maxRadius=150)

bordas_rgb = cv2.cvtColor(edges, cv2.COLOR_GRAY2RGB)
output = bordas_rgb


if circles is not None:
    circles = np.uint16(np.around(circles))
    for i in circles[0,:]:
        # desenha o contorno do circulo
        cv2.circle(output,(i[0],i[1]),i[2],(0,255,0),2)
        # desenha no centro do circulo
        cv2.circle(output,(i[0],i[1]),2,(0,0,255),3)

plt.figure(figsize = (10,10))
plt.imshow(output, cmap="gray")
```


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200
1400

- O resultado não ficou bom, pois há muitos falsos positivos detectados, neste caso precisamos alterar os parâmetros da transformada de hough. Vamos ver o que é cada um deles.

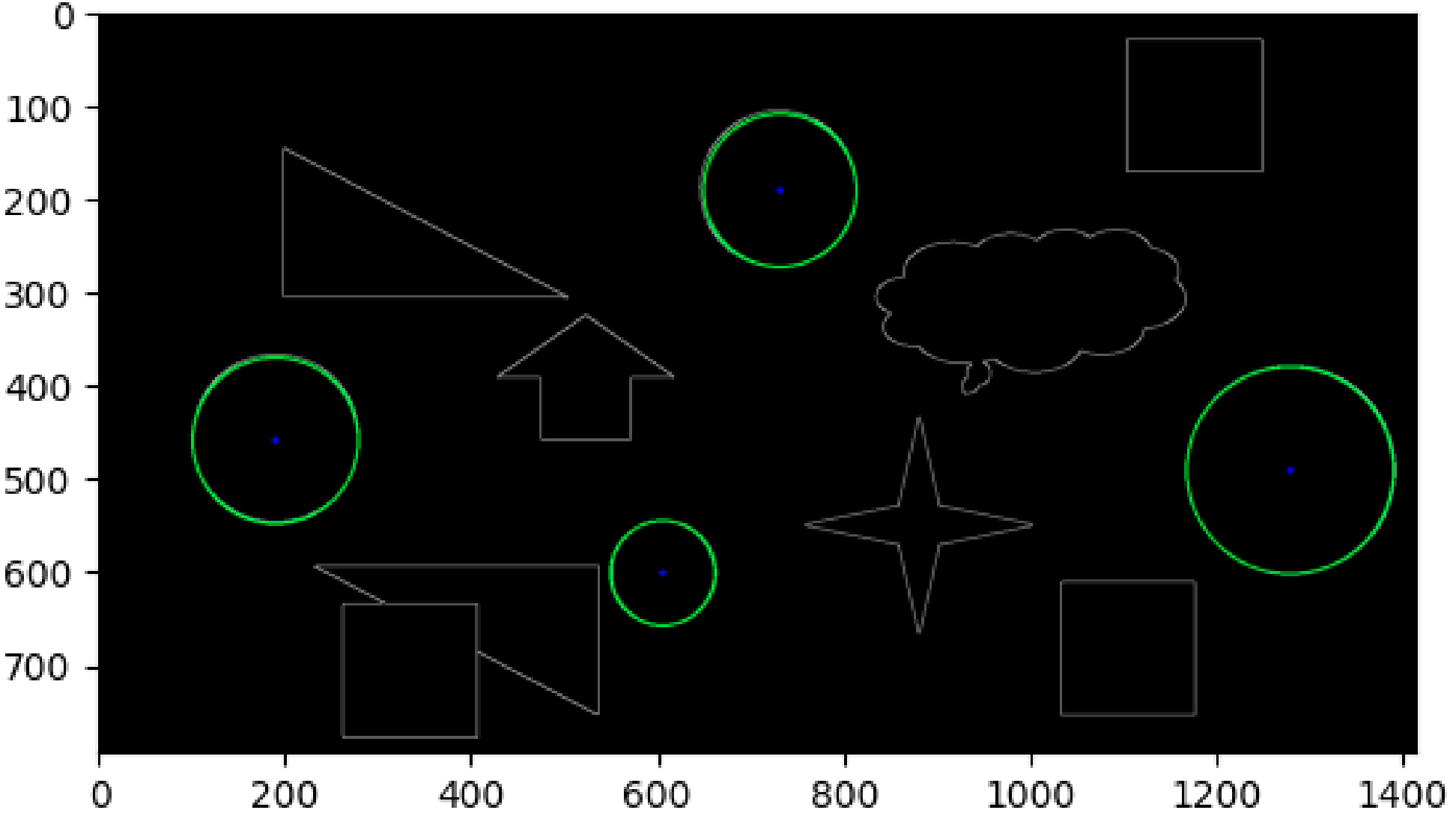
```
circles=cv2.HoughCircles(image,method=cv2.HOUGH_GRADIENT,dp,minDist,param1,param2,minRadius,maxRadius)
```

- image: imagem de entrada na escala de ciza.
- method: Define o metódo de detecção de circulos.
- dp: relação entre o tamanho da imagem e o tamanho do acumulador. Um dp grande "pega" bordas mais tênues.
- minDist: Distância minima entre centros (x,y) dos circulos detectados
- param1: Valor do gradiente usado para lidar com a detecção de bordas
- param2: Limiar do Acumulador usado pelo metódo. Se muito baixo, retorna mais circulos (incluindo círculos falsos). Se mais alto, mais círculos serão potencialmente retornados.
- minRadius: Raio minimo (em pixels).
- maxRadius: Raio máximo (em pixels).

DESAFIO

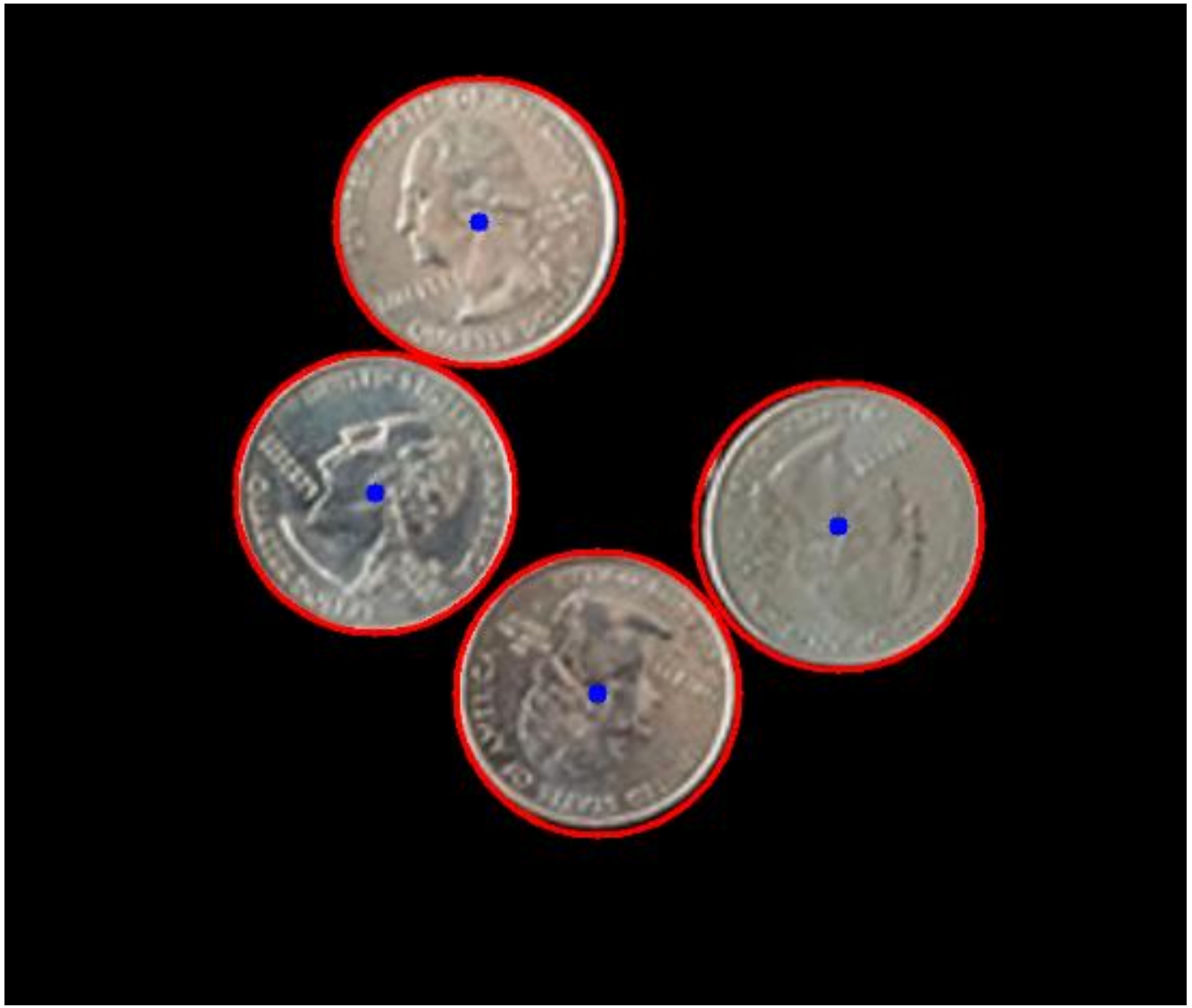
- Faça a alteração dos parâmetros para a transformada de Hough afim de detectar apenas os círculos da imagem.

- **Dica:** Altere um parâmetro por vez e analise o resultado.


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200
1400

DESAFIO

- Modifique o código para detectar, segmentar e exibir a quantidade de moedas de 1 dólar na imagem “coins.png”, temos quatro moedas de 1 dólar.



DETECÇÃO DE RETAS

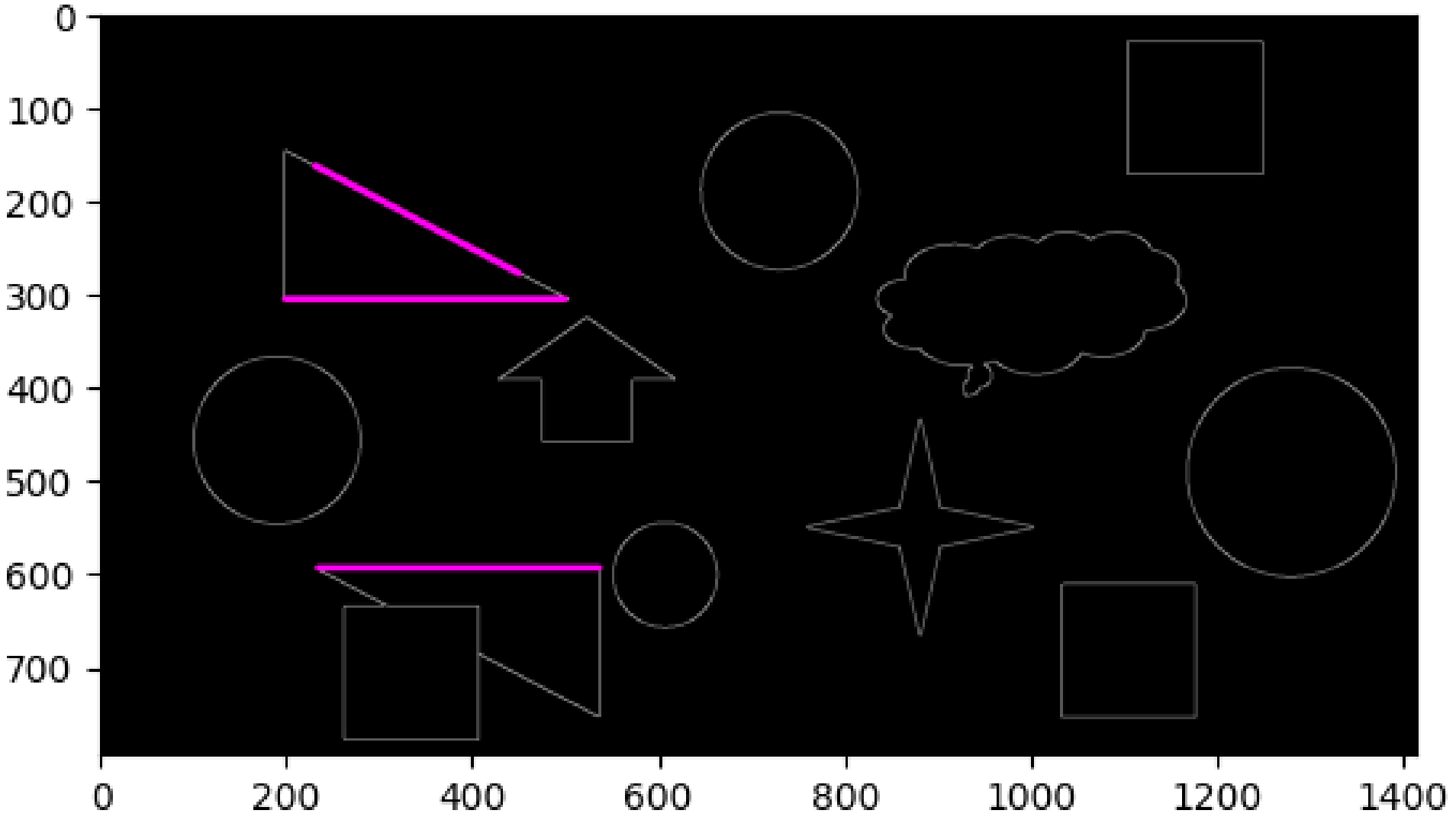
- De forma semelhante ao CV2.HOUGHCIRCLE(), para detecção de retas usamos o cv2.HoughLines() ou cv2.HoughLinesP() o segundo faz uma estimativa probabilística.

```
cv.HoughLinesP( image, rho, theta, threshold[, lines[, minLineLength[, maxLineGap]]])
```

- imagem: Imagem de entrada em escala de cinza.
- rho: Resolução da distância do acumulador em pixeis.
- teta: Resolução do ângulo de rotação do acumulador em radianos, normalmente 1 Grau.
- threshold: Limiar do acumulador. Só são devolvidas as linhas que obtêm votos suficientes (>threshold).
- minLineLineLength: Comprimento mínimo da linha. Segmentos de linha mais curtos do que isso são rejeitados.
- maxLineGap: Distância máxima permitida entre pontos considerados na mesma linha.

Desafio

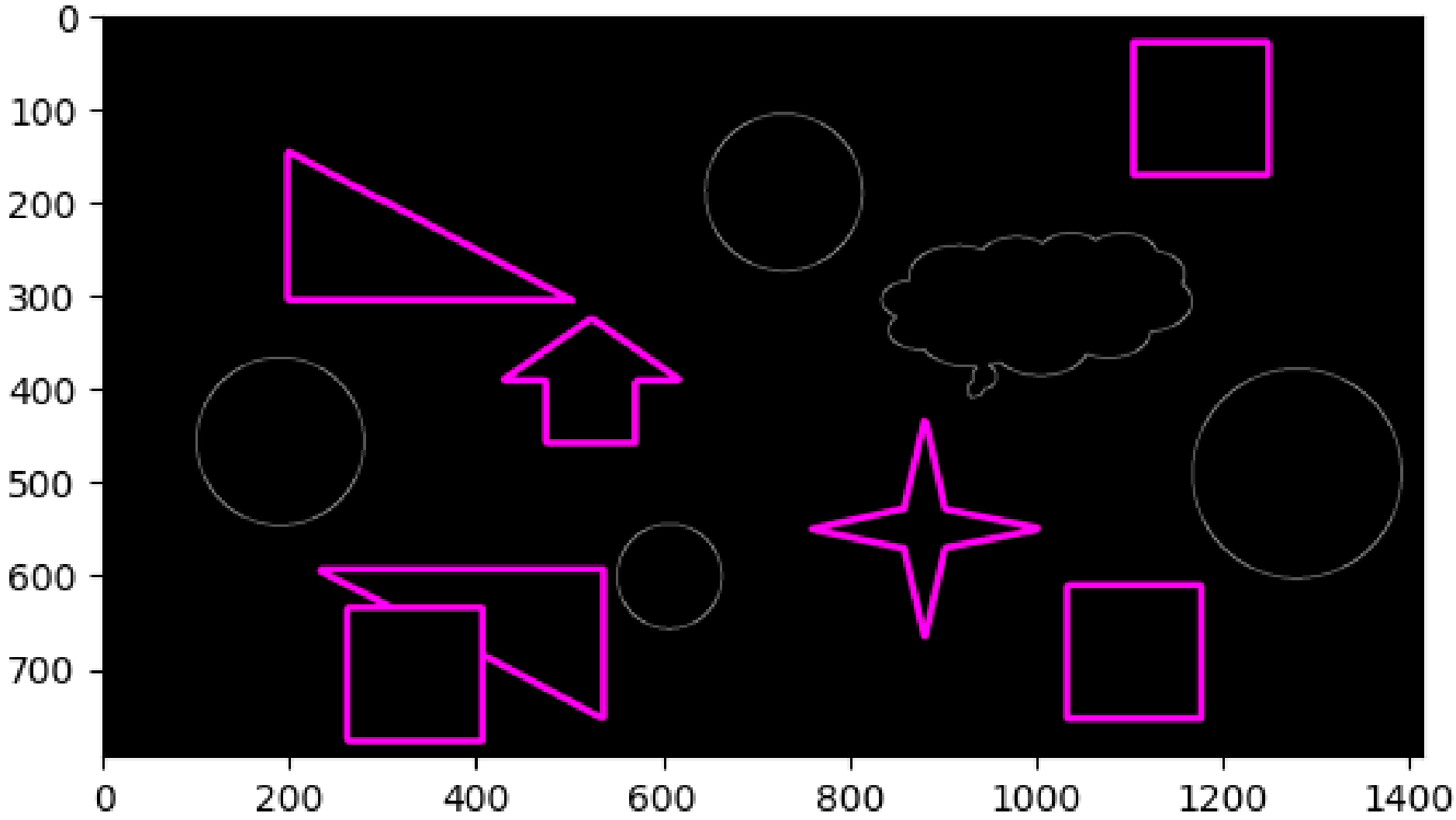
```
img = cv2.imread('formas.png')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)



plt.imshow(img)
plt.show()
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(img_gray,50,150)

lines = cv2.HoughLinesP(edges, 1, math.pi/180.0, 100, np.array([]), 180, 5)

hough_img = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)

for line in lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(hough_img, (x1, y1), (x2, y2), (255, 0, 255), 5)



plt.imshow(hough_img)
```


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200
1400

DESAFIO

- Faça a alteração dos parâmetros para a transformada de Hough afim de detectar apenas as linhas da imagem.

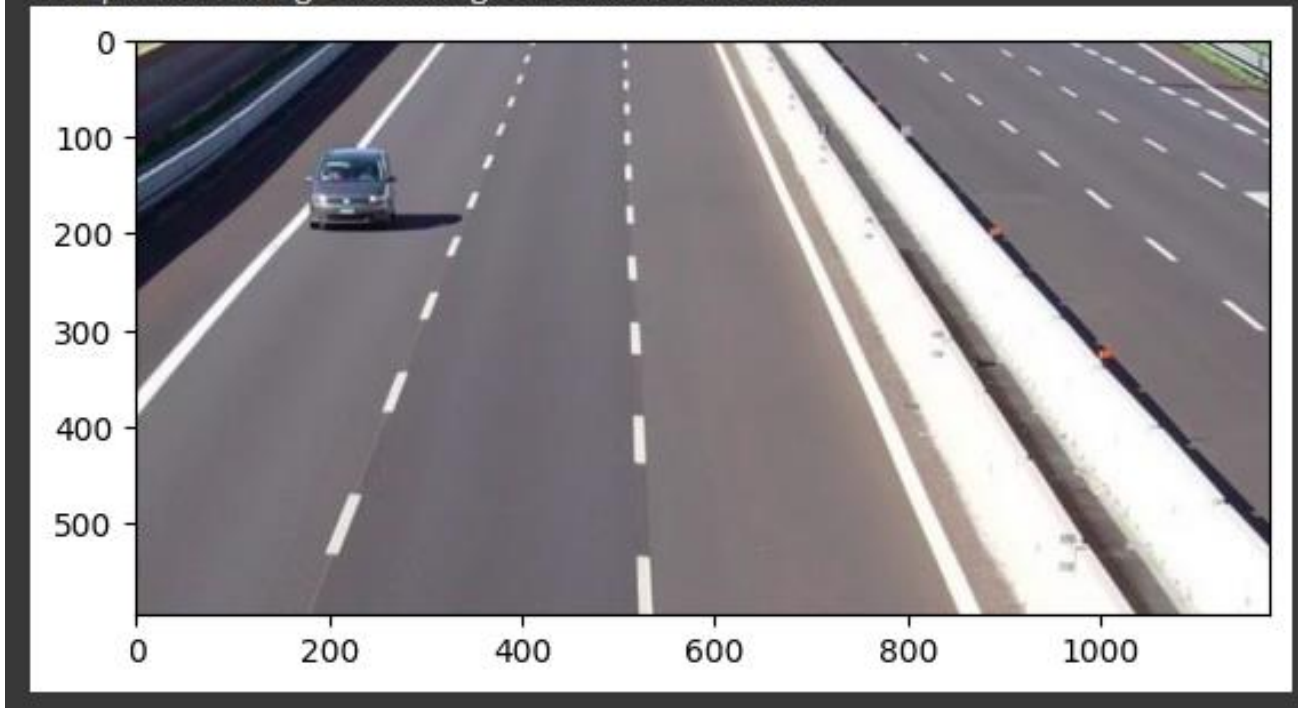
- **Dica:** Altere um parâmetro por vez e analise o resultado.


0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200
1400

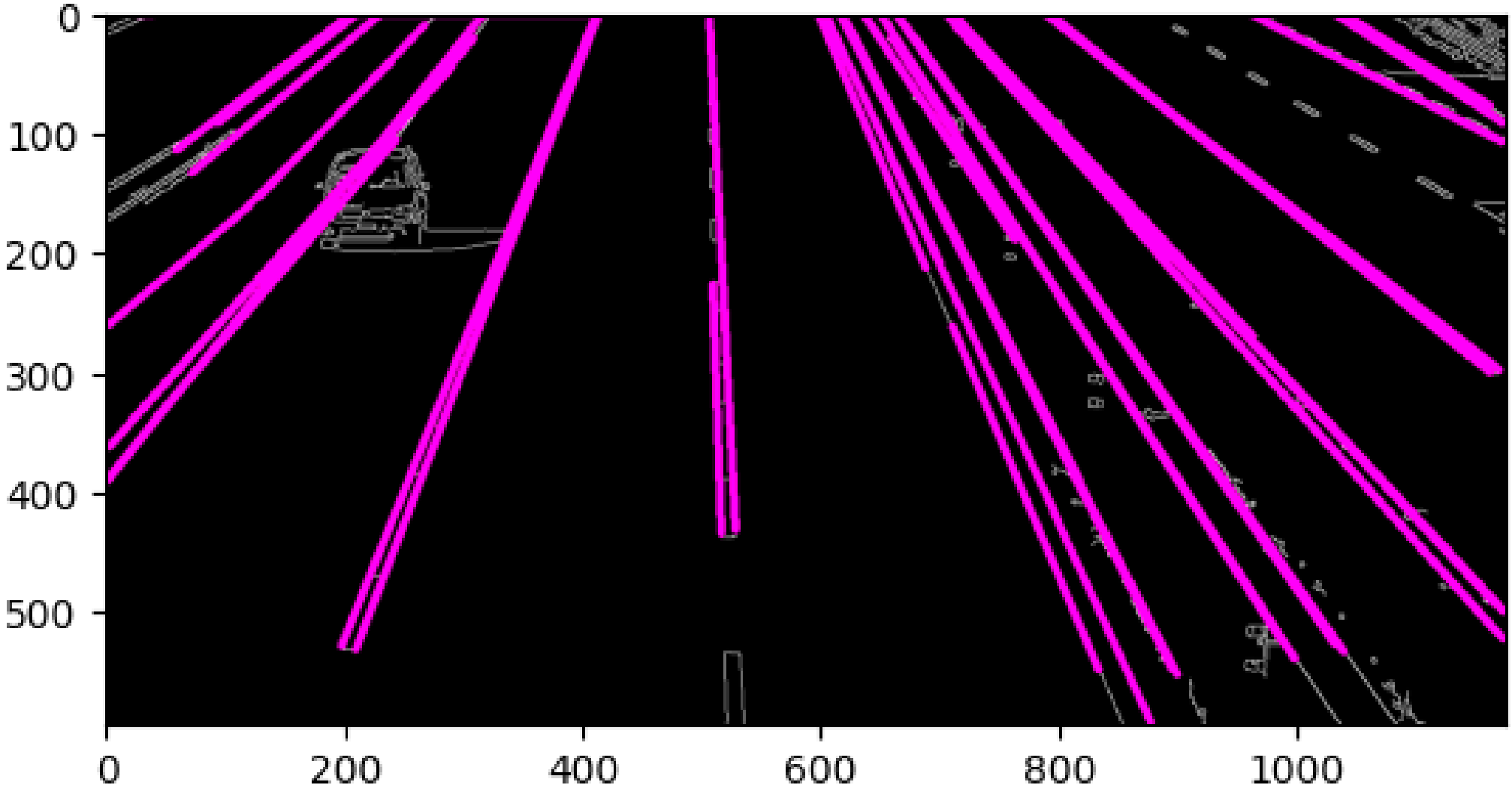
Exemplo mais prático, detecção de faixa por veículos autônomos.

```
img = cv2.imread('rua.png')
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)


plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7cc12335cc70>
```



```
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(img_gray,50,200)
#rho (ρ) e theta (θ).
#Limiar de linhas min
#comprimento mínimo de linha, lacuna máxima permitida entre segmentos de linha.
lines = cv2.HoughLinesP(edges, 1, math.pi/180.0, 99, np.array([]), 3, 120)


hough_img = cv2.cvtColor(edges, cv2.COLOR_GRAY2BGR)

for line in lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(hough_img, (x1, y1), (x2, y2), (255, 0, 255), 5)



plt.imshow(hough_img)
```

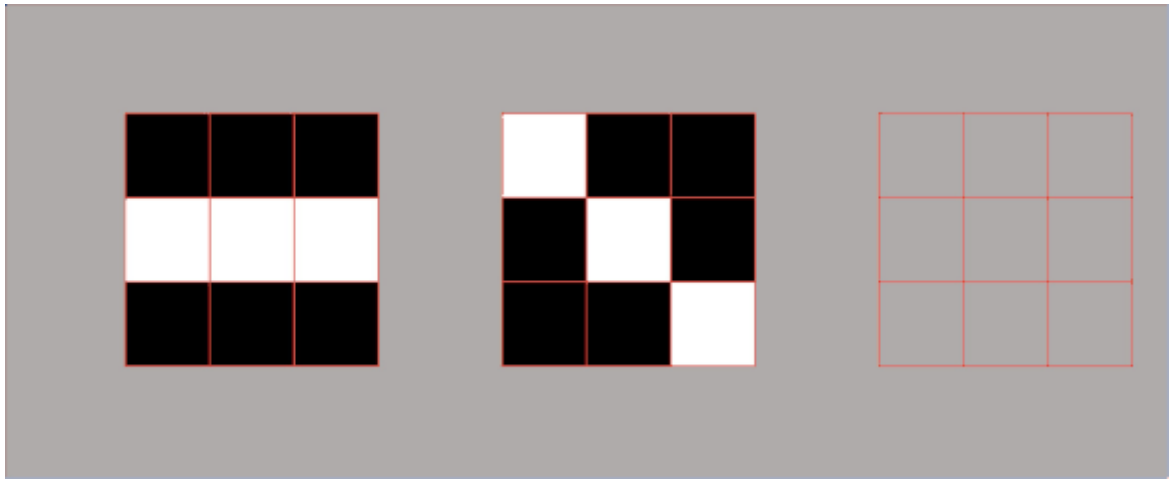

0
100
200
300
400
500
0
200
400
600
800
1000

MORFOLOGIA MATEMÁTICA

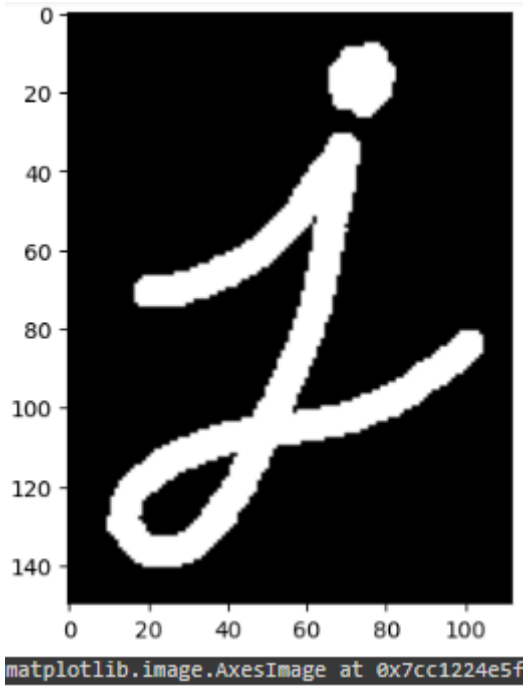
- A Morfologia Matemática (MM) é um modelo teórico para as imagens digitais construídas em cima da teoria dos reticulados e da topologia . É o fundamento do processamento de imagem morfológico, que é baseado nos operadores de deslocamento-invariante (translação invariante) baseados principalmente na adição de Minkowski.

DILATAÇÃO BINÁRIA

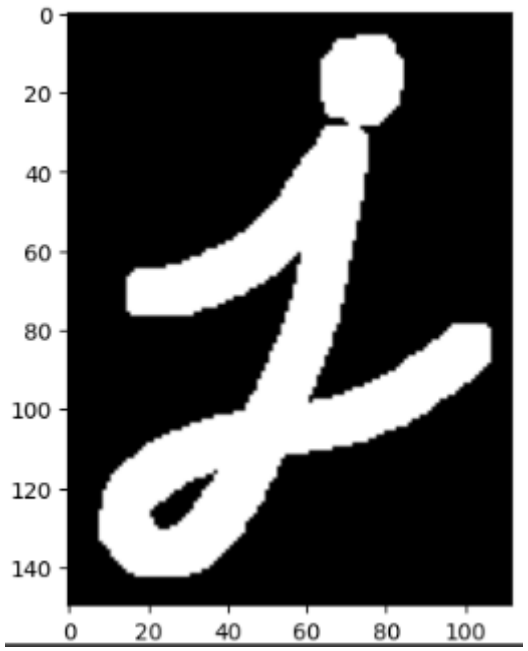
- É uma transformação morfológica que combina dois conjuntos usando adição vetorial. Como o nome diz, o resultado será uma imagem “engordada”.



```python
import cv2
import numpy as np
img = cv2.imread('j.png',0)
plt.imshow(img, cmap="gray")
plt.show()

kernel = np.ones((5,5),np.uint8)

dilation = cv2.dilate(img,kernel,iterations = 1)


plt.imshow(dilation, cmap="gray")
```

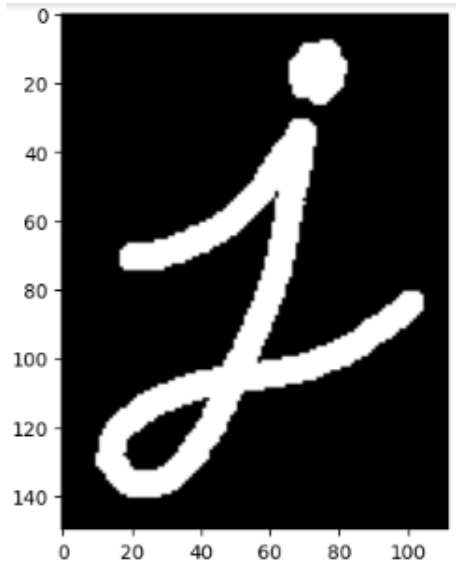



```
matplotlib.image.AxesImage at 0x7cc1224e5f
```

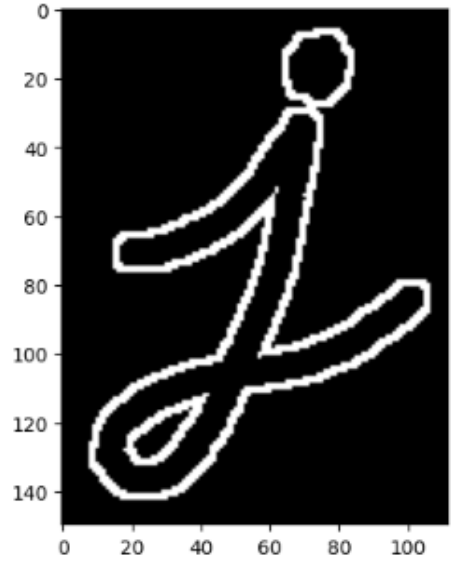


Detectando contorno com dilatação

```
img = cv2.imread('j.png',0)

plt.imshow(img, cmap="gray")
plt.show()

dst = img.copy()
kernel = np.ones((5,5),np.uint8)

dilation = cv2.dilate(img,kernel,iterations = 1)

dst = dilation - img


plt.imshow(dst, cmap="gray")
```

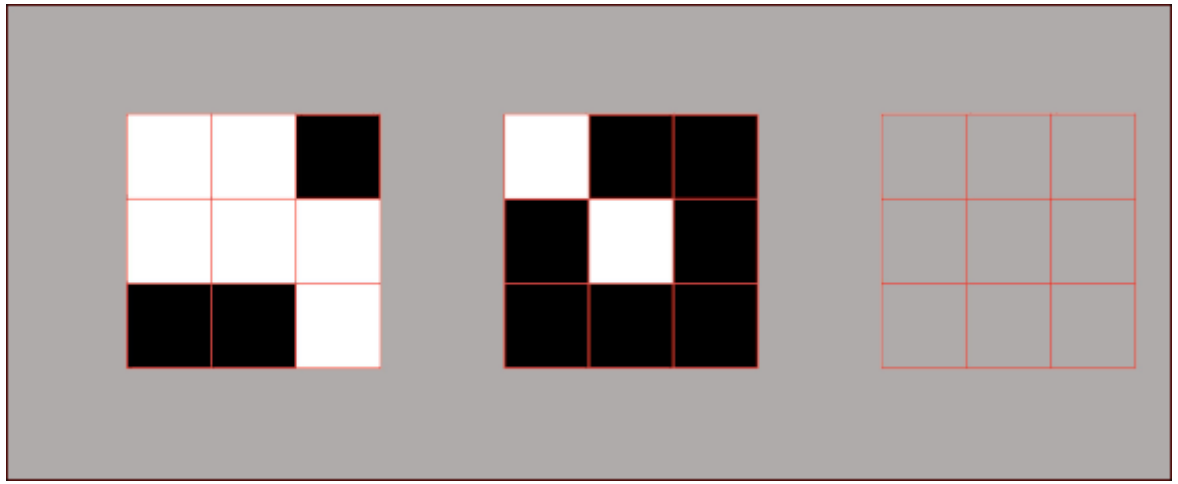


matplotlib.image.AxesImage at 0x7cc1223c2



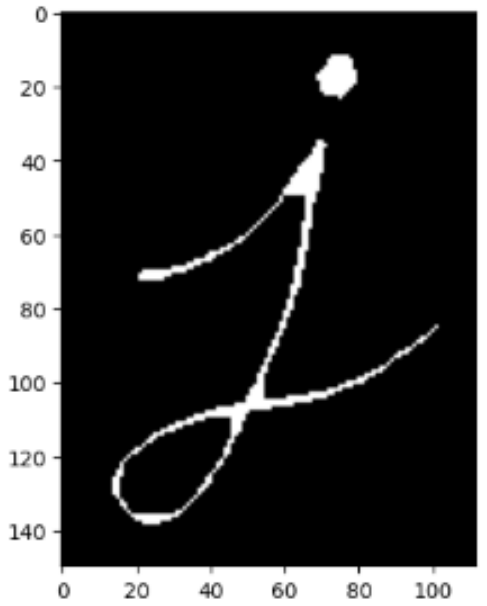
EROSÃO BINÁRIA

- A erosão basicamente encolhe uma imagem e pode ser vista como uma transformação morfológica que combina dois conjuntos usando vetores de subtração. Ela é expressa como a interseção de A e B.



```
img = cv2.imread('j.png',0)
plt.imshow(img, cmap="gray")
plt.show()

dst = img.copy()
kernel = np.ones((6,6),np.uint8)

erode = cv2.erode(img,kernel,iterations = 1)

plt.imshow(erode, cmap="gray")
```

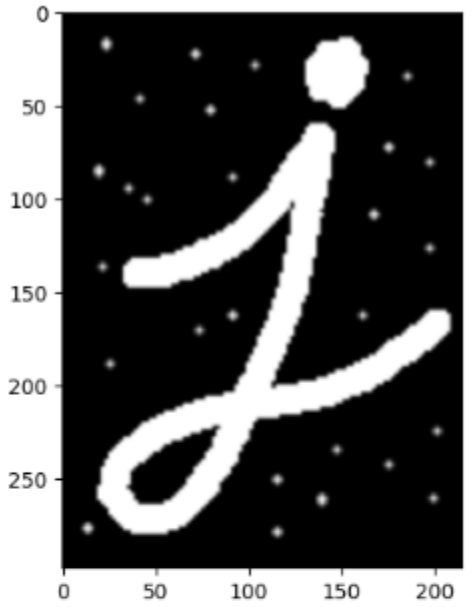

```
atplotlib.image.AxesImage at 0x7cc123452f
```



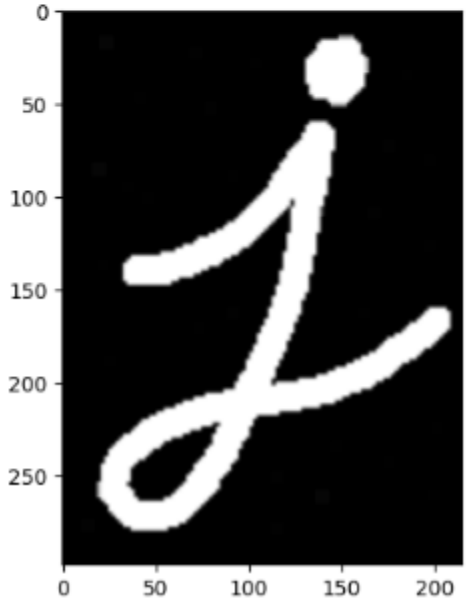
ABERTURA BINÁRIA

- A abertura em geral suaviza o contorno de uma imagem, quebra estreitos e elimina proeminências delgadas, a operação de abertura e usada também para remover ruídos da imagem.

```
img = cv2.imread('j-noise.png',0)
plt.imshow(img, cmap="gray")
plt.show()

dst = img.copy()
kernel = np.ones((7,7),np.uint8)

opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)

plt.imshow(opening, cmap="gray")
```



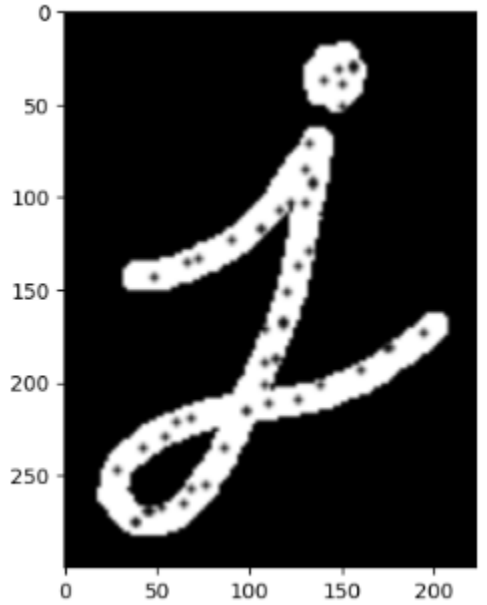
matplotlib.image.AxesImage at 0x7cc12335c



FECHAMENTO BINÁRIO

- O fechamento funde pequenos quebras, alarga golfos estreitos e elimina pequenos orifícios. Se uma abertura cria pequenos vazios na imagem, um fechamento irá preencher ou fechar os vazios, estas operações podem remover muitos dos pixels brancos com ruídos, ou seja basicamente ele e igual a abertura só que primeiramente e feita a dilatação e após e feita a erosão.

```python
img = cv2.imread('holes.png',0)
plt.imshow(img, cmap="gray")
plt.show()

dst = img.copy()
kernel = np.ones((7,7),np.uint8)

closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)

plt.imshow(closing, cmap="gray")
```



matplotlib.image.AxesImage at 0x7cc12323d