opencv-1

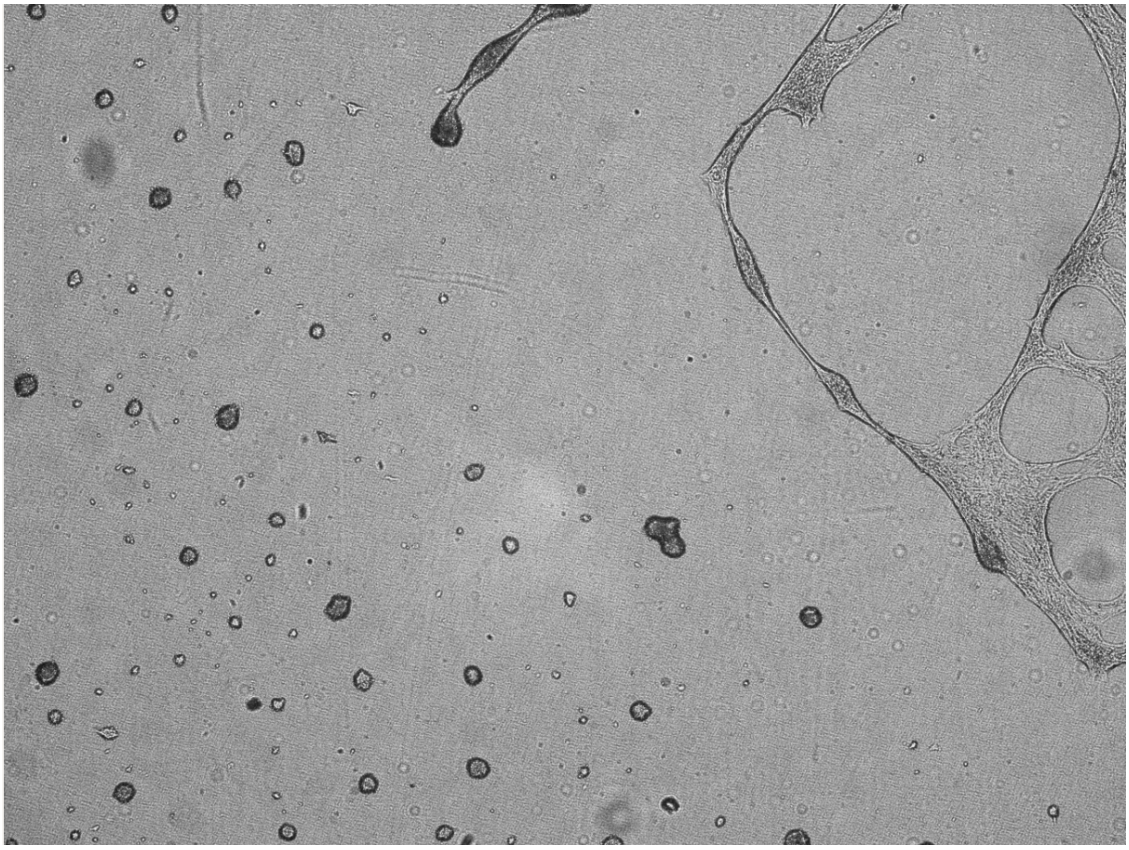
October 14, 2024

[1]:
```
!pip install opencv-python
!pip install opencv-python matplotlib
```

```
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-
packages (4.10.0.84)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-
packages (from opencv-python) (1.26.4)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-
packages (4.10.0.84)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (3.7.1)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-
packages (from opencv-python) (1.26.4)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.7)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (24.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from matplotlib) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.4)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-
packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

[5]:
```
import cv2
import glob
from google.colab.patches import cv2_imshow


image_path = "/content/img.png"
```

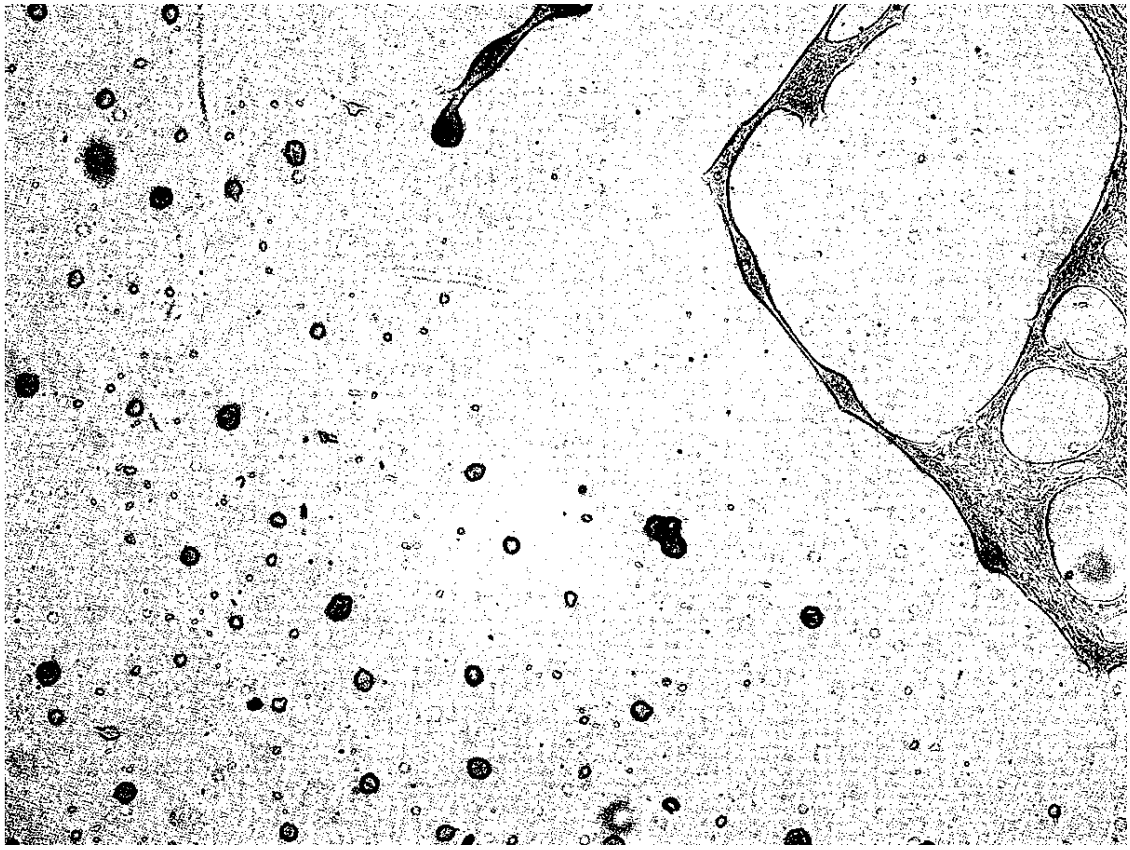
```
images = glob.glob(image_path)

for image_file in images:
    image = cv2.imread(image_file)
    cv2_imshow(image)
```



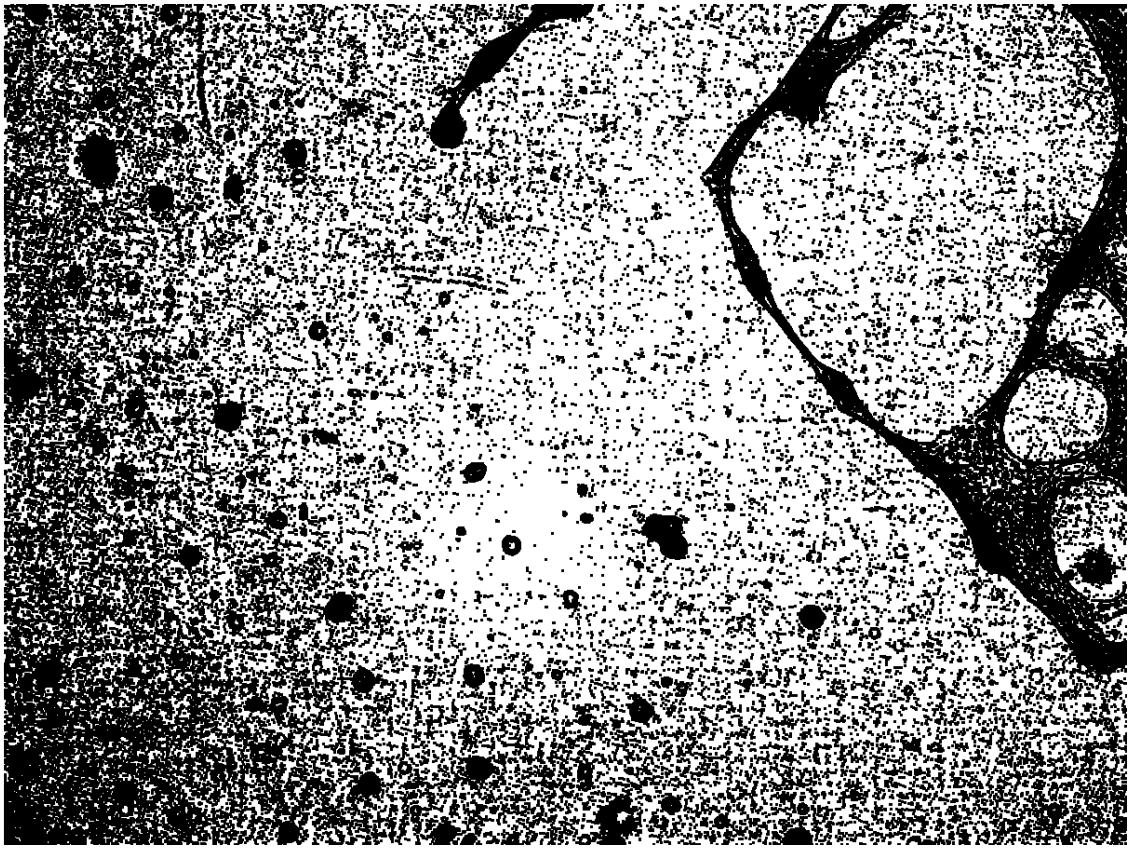
1 FILTROS

[6]:
```
image = cv2.imread("/content/img.png", cv2.IMREAD_GRAYSCALE)
```

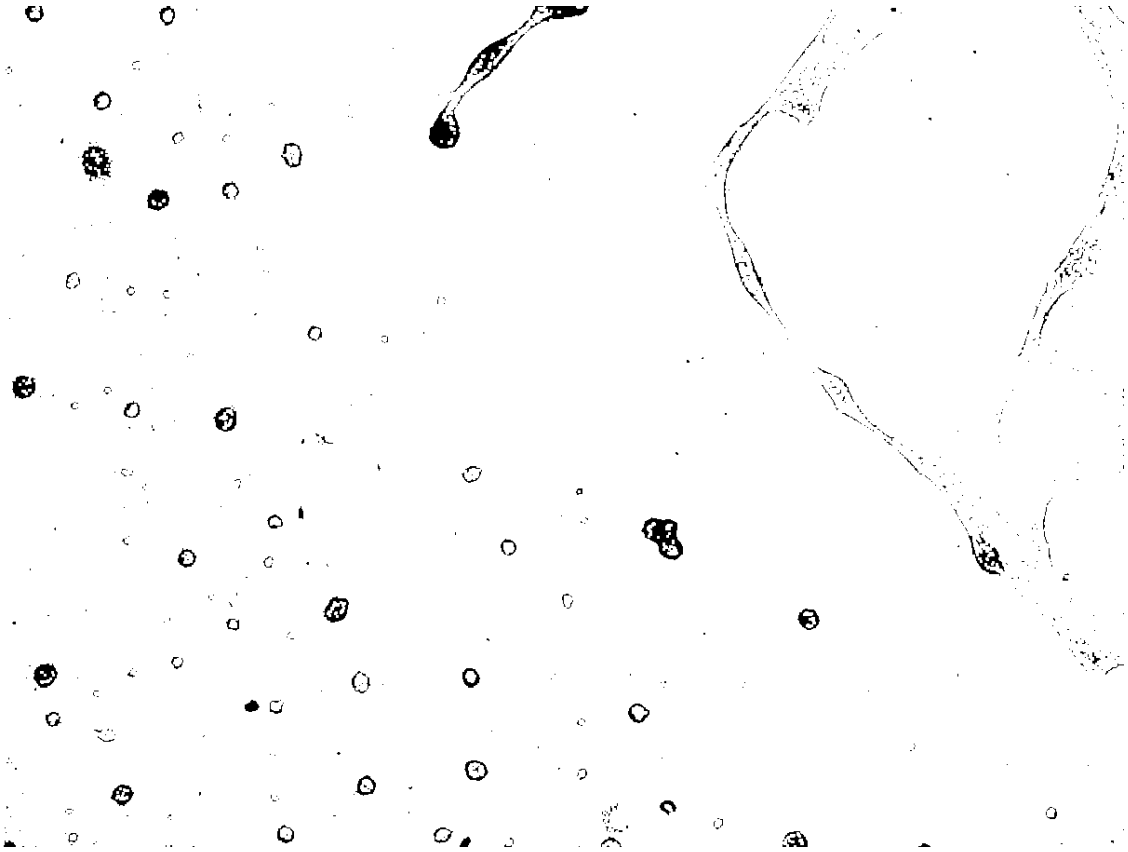
[7]:
```
_, binary_image = cv2.threshold(image, 135, 255, cv2.THRESH_BINARY)
cv2_imshow(binary_image)
```



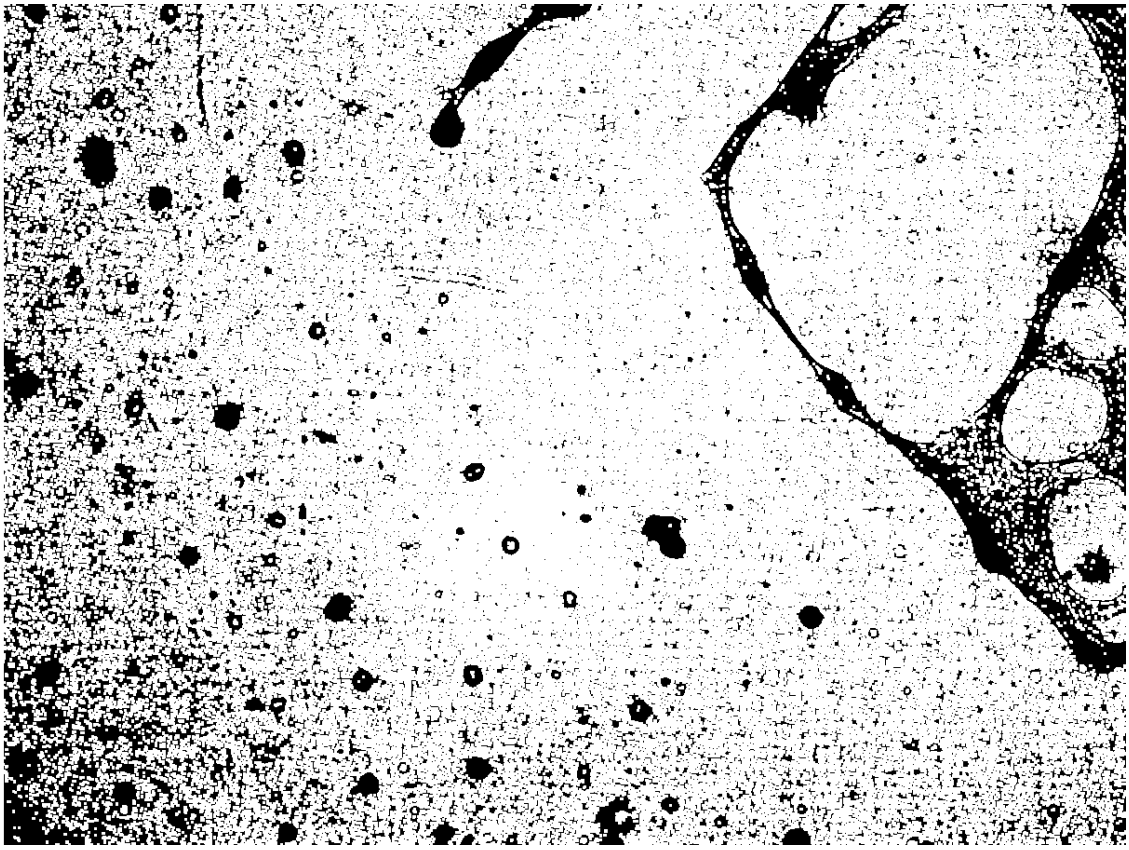
[8]:
```
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))

erosion = cv2.erode(binary_image, kernel, iterations=1)

cv2_imshow(erosion)
```



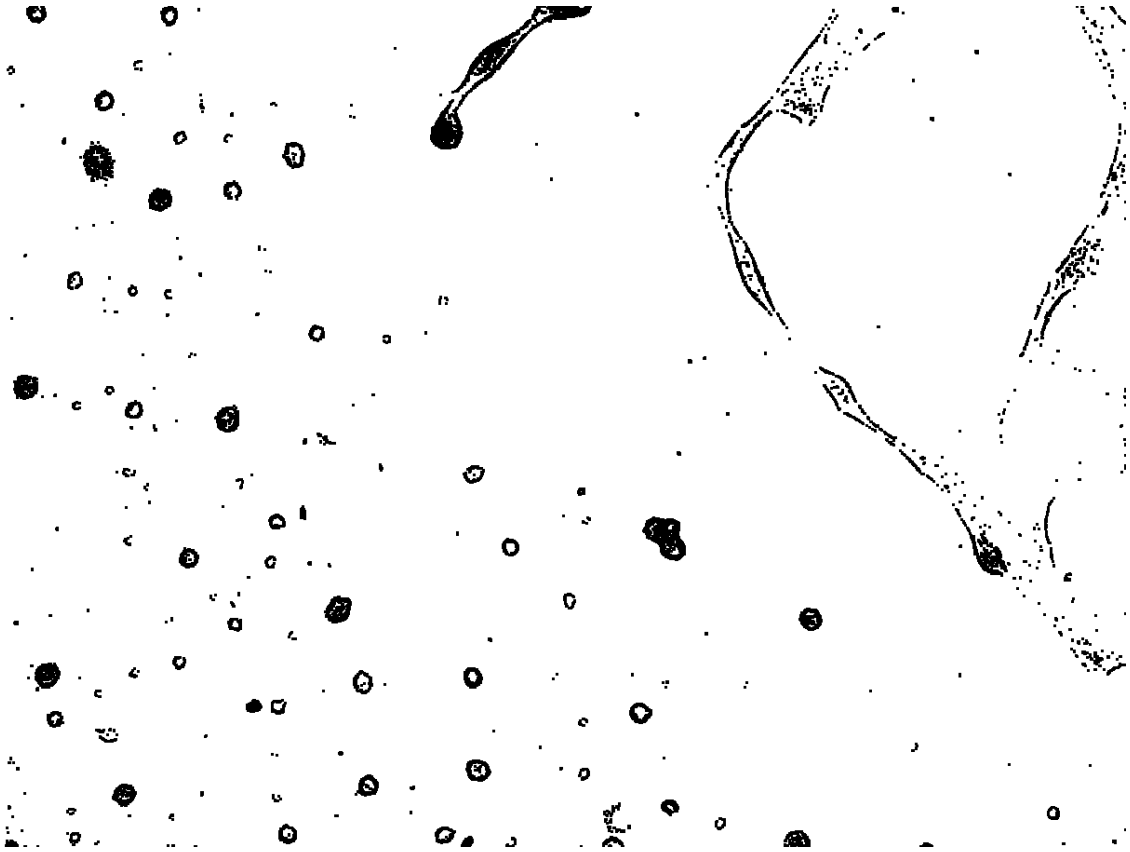
[9]:
```
dilatation = cv2.dilate(binary_image, kernel, iterations=1)


cv2_imshow(dilatation)
```



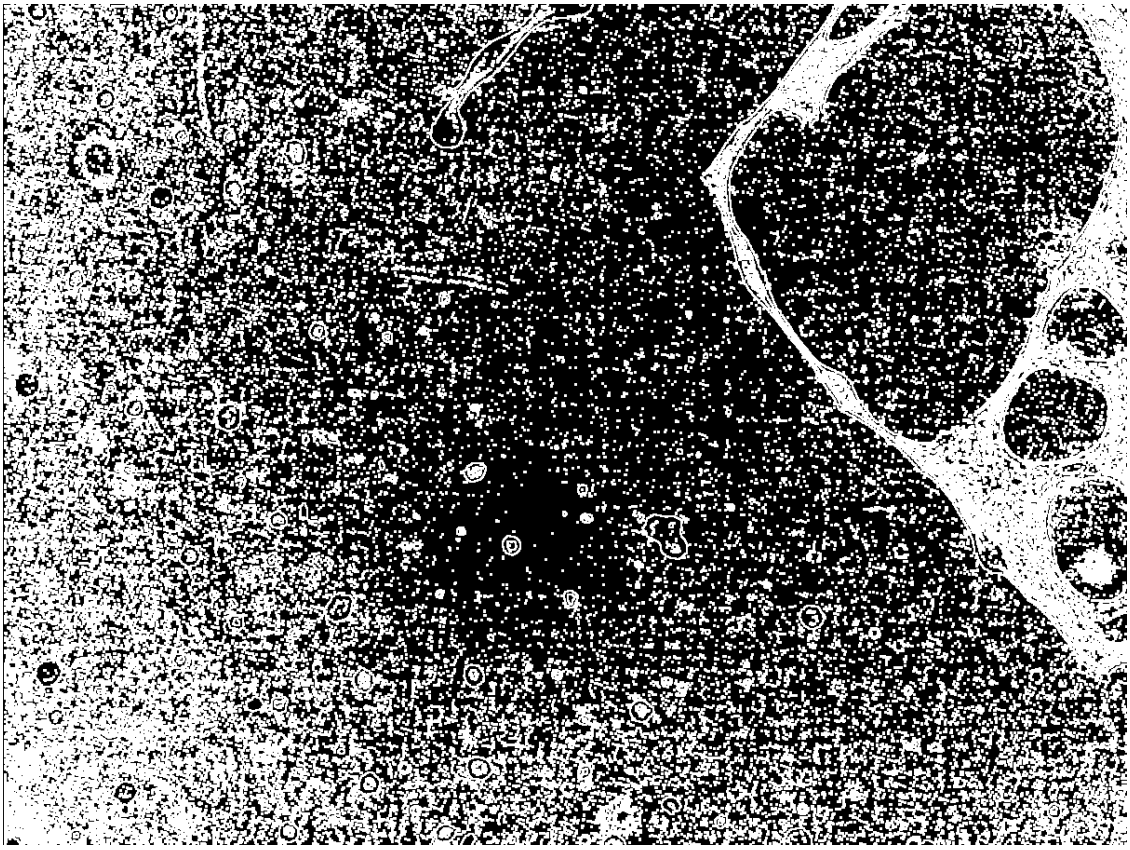
[10]:
```
filter = cv2.morphologyEx(binary_image, cv2.MORPH_OPEN, kernel)

cv2_imshow(filter)
```



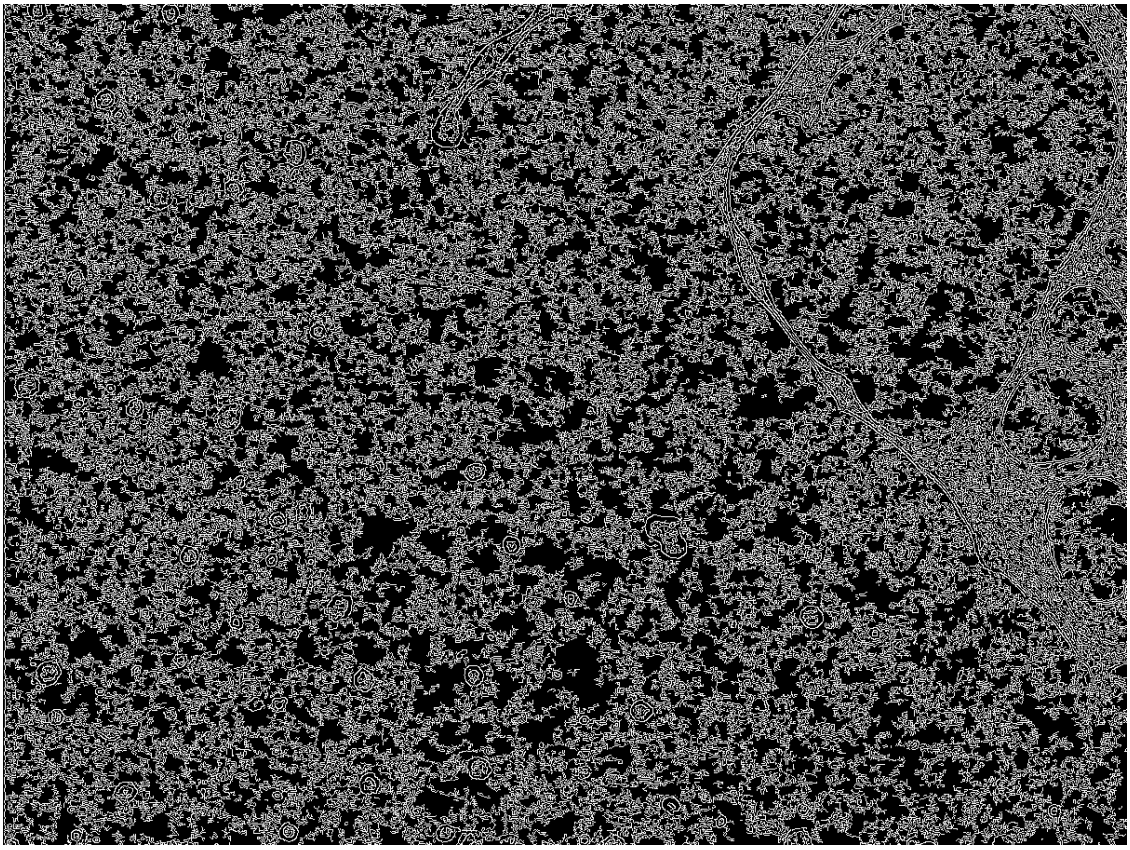
[11]:
```
filter2 = cv2.morphologyEx(binary_image, cv2.MORPH_CLOSE, kernel)

cv2_imshow(filter2)
```



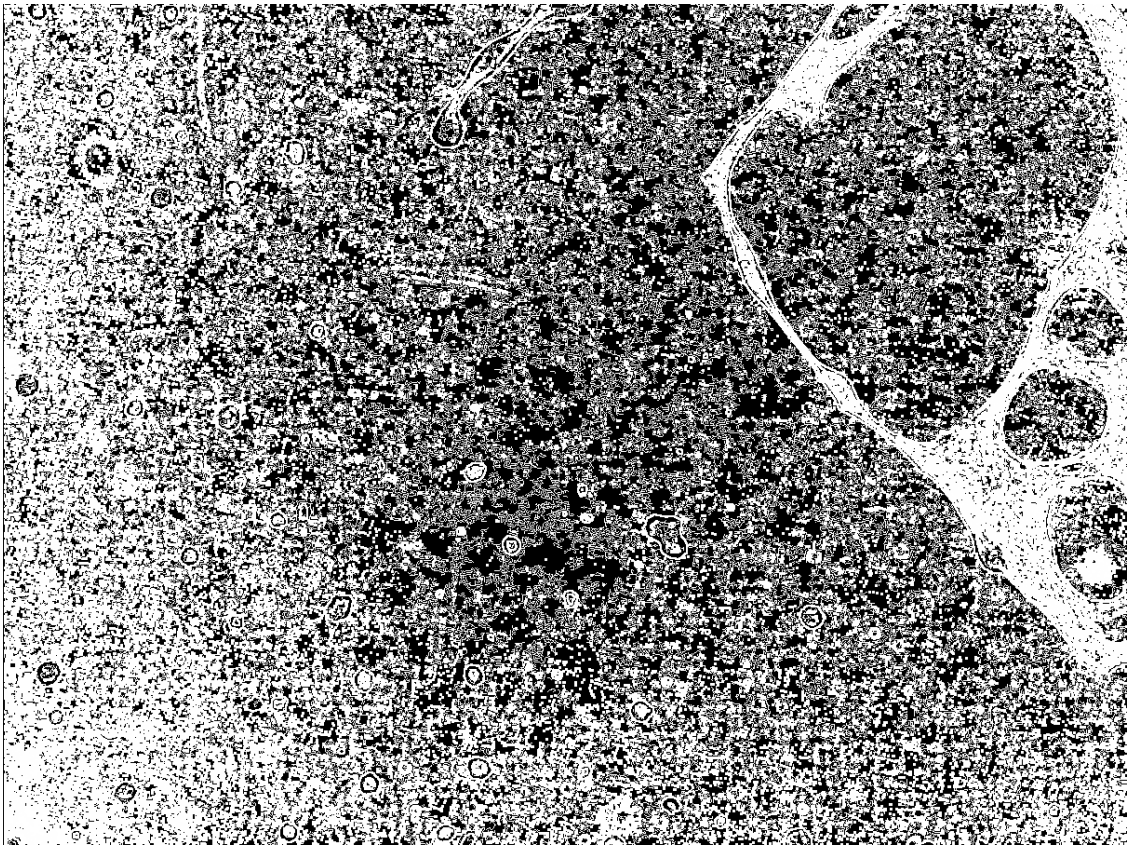
[13]:
```
filtro4 = cv2.morphologyEx(binary_image, cv2.MORPH_GRADIENT, kernel)

cv2_imshow(filtro4)
```



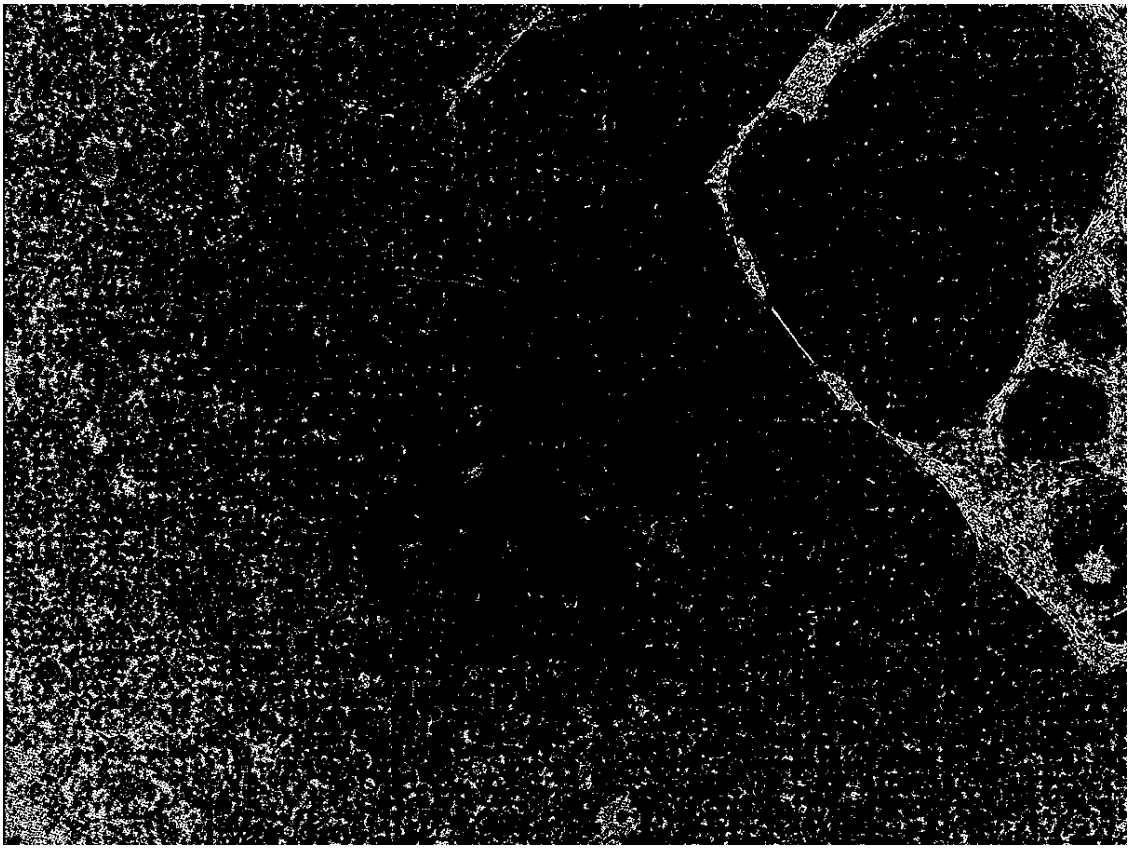
[12]:
```
edges = cv2.Canny(image, 255, 1)
cv2_imshow(edges)
```



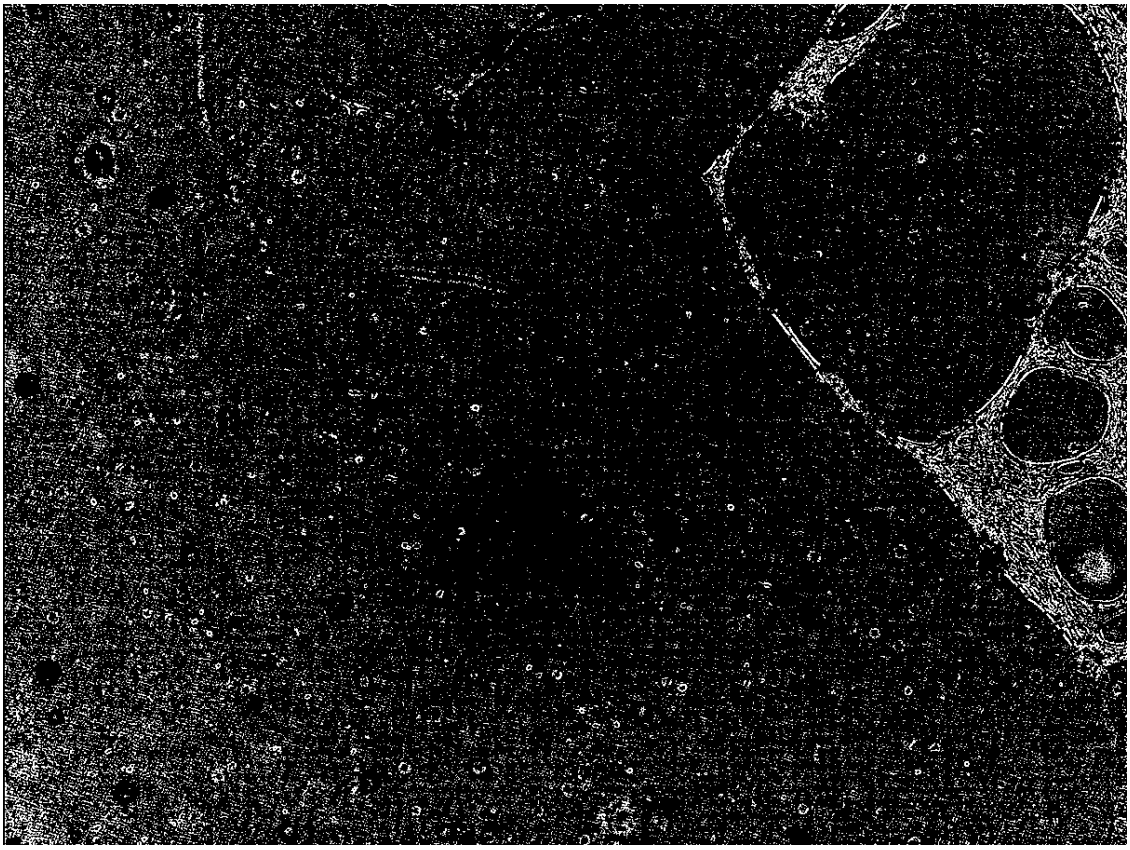
[14]:
```
f = filtro4 + edges
cv2_imshow(f)
```



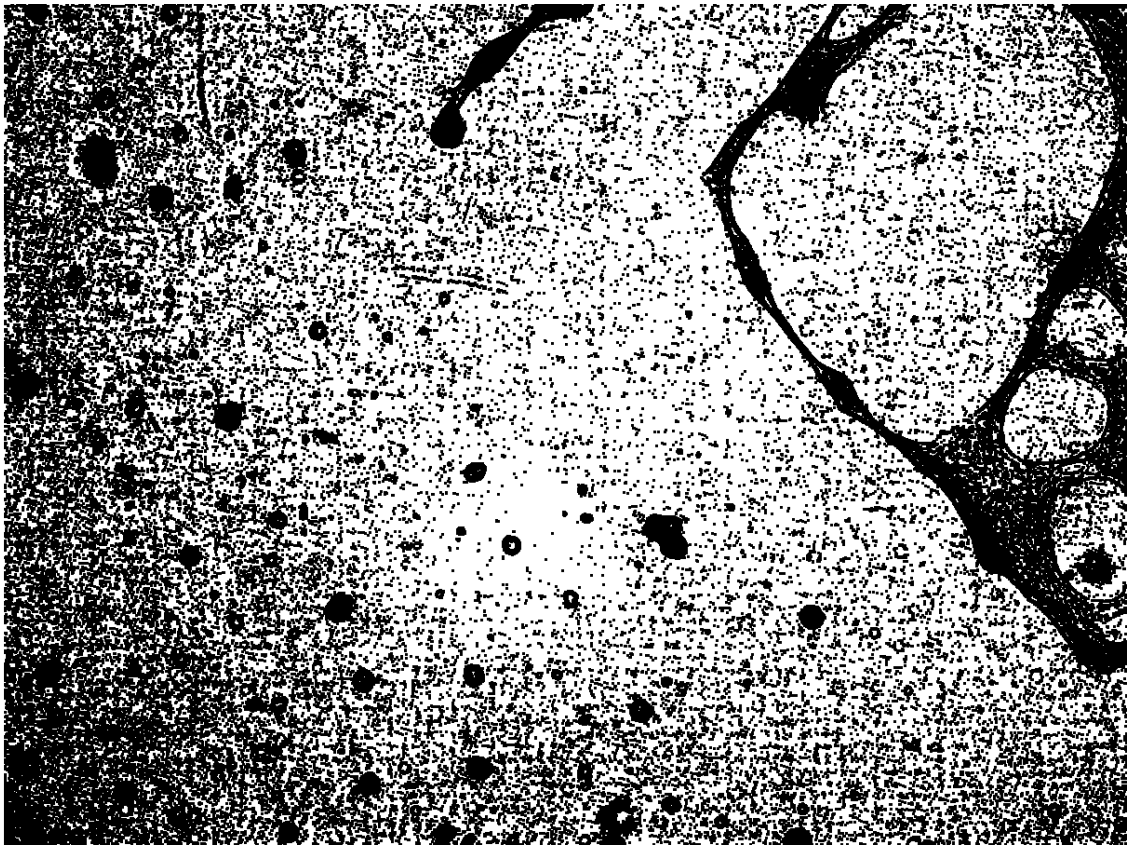
[15]:
```
filtro5 = cv2.morphologyEx(binary_image, cv2.MORPH_TOPHAT, kernel)

cv2_imshow(filtro5)
```



[16]:
```
filtro6 = cv2.morphologyEx(binary_image, cv2.MORPH_BLACKHAT, kernel)

cv2_imshow(filtro6)
```



[17]:
```
filtro8 = cv2.morphologyEx(binary_image, cv2.MORPH_HITMISS, kernel)

cv2_imshow(filtro8)
```

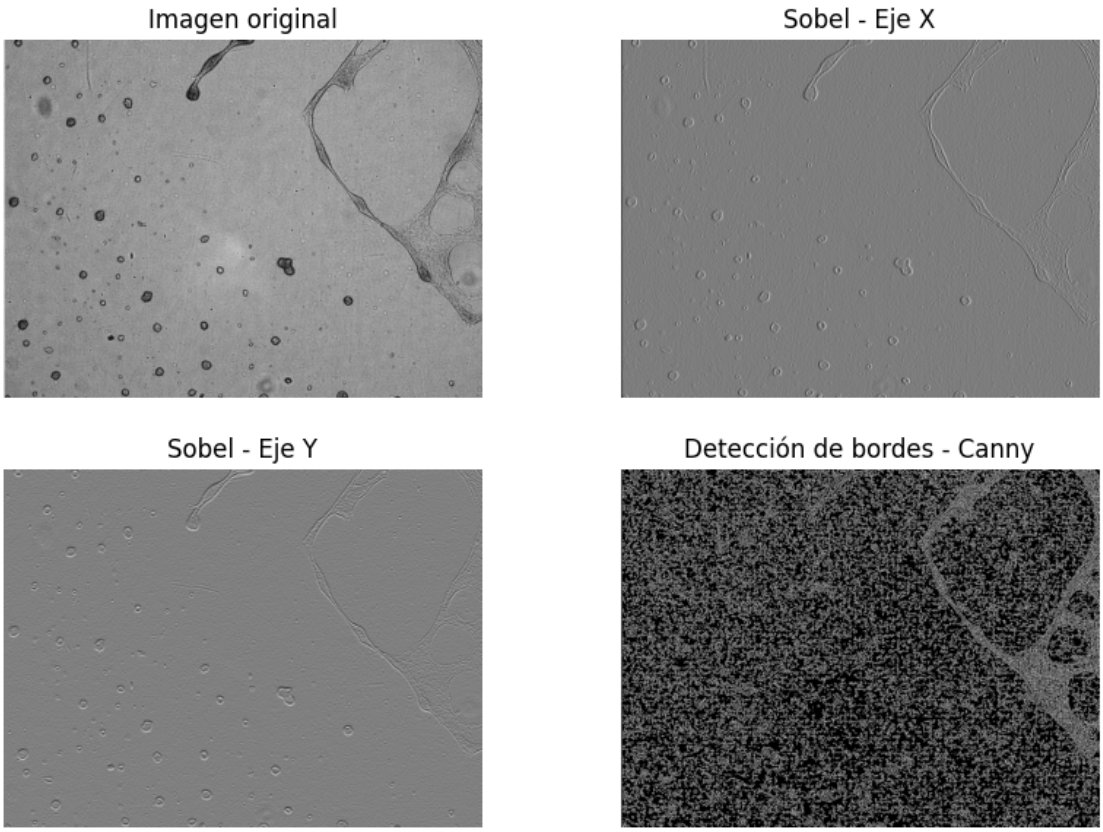


2 Modelo

[18]:
```
image = cv2.imread('/content/img.png', cv2.IMREAD_GRAYSCALE)
```

[19]:
```
import cv2
import matplotlib.pyplot as plt
import numpy as np




image = cv2.imread('/content/img.png', cv2.IMREAD_GRAYSCALE)


sobelx = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=5)
sobely = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=5)


edges_canny = cv2.Canny(image, 100, 200)
```

```
plt.figure(figsize=(10, 7))

plt.subplot(2, 2, 1), plt.imshow(image, cmap='gray')
plt.title('Imagen original'), plt.axis('off')

plt.subplot(2, 2, 2), plt.imshow(sobelx, cmap='gray')
plt.title('Sobel - Eje X'), plt.axis('off')

plt.subplot(2, 2, 3), plt.imshow(sobely, cmap='gray')
plt.title('Sobel - Eje Y'), plt.axis('off')

plt.subplot(2, 2, 4), plt.imshow(edges_canny, cmap='gray')
plt.title('Detección de bordes - Canny'), plt.axis('off')

plt.show()
```



3 SAM2

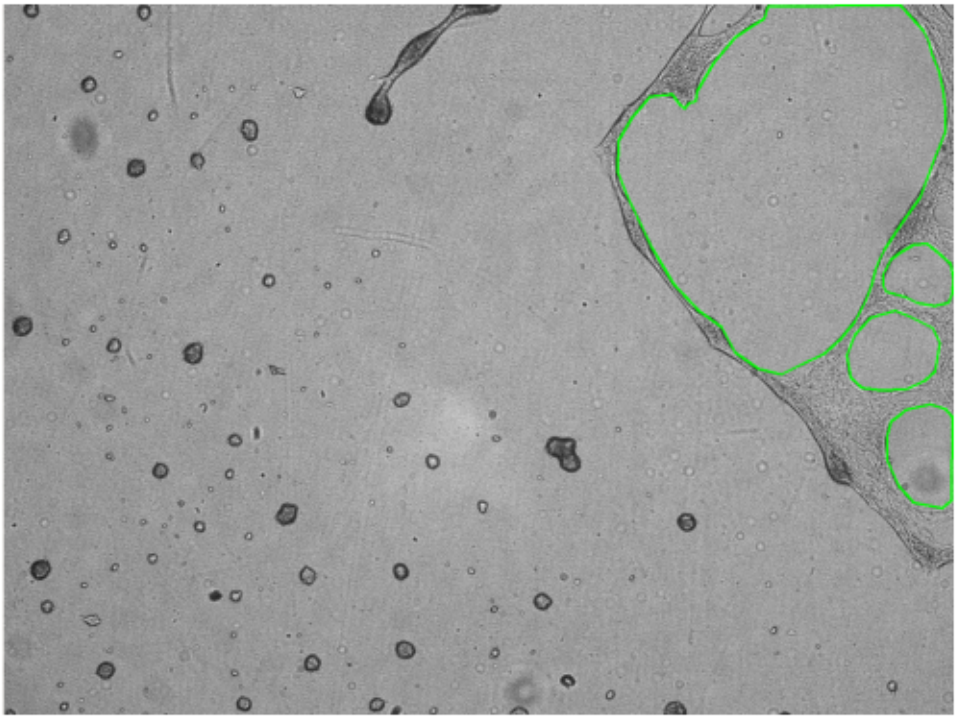
[ ]:
```
pip install git+https://github.com/facebookresearch/segment-anything.git
```

[ ]:
```
from segment_anything import SamPredictor, sam_model_registry
import cv2
import matplotlib.pyplot as plt
import numpy as np


image = cv2.imread('sample_data/img.png')


sam = sam_model_registry["vit_b"](checkpoint="/content/sample_data/
 ↪sam_vit_b_01ec64.pth")
predictor = SamPredictor(sam)


predictor.set_image(image)



input_point = np.array([[100, 150]])
input_label = np.array([1])

masks, scores, logits = predictor.predict(point_coords=input_point,␣
 ↪point_labels=input_label, multimask_output=True)


plt.imshow(masks[0], cmap='gray')
plt.title('Miotubos segmentados')
plt.show()
```

4 Binary mask

[24]:
```
import json
import numpy as np
import cv2
import os


image = cv2.imread('/content/img.png')
```

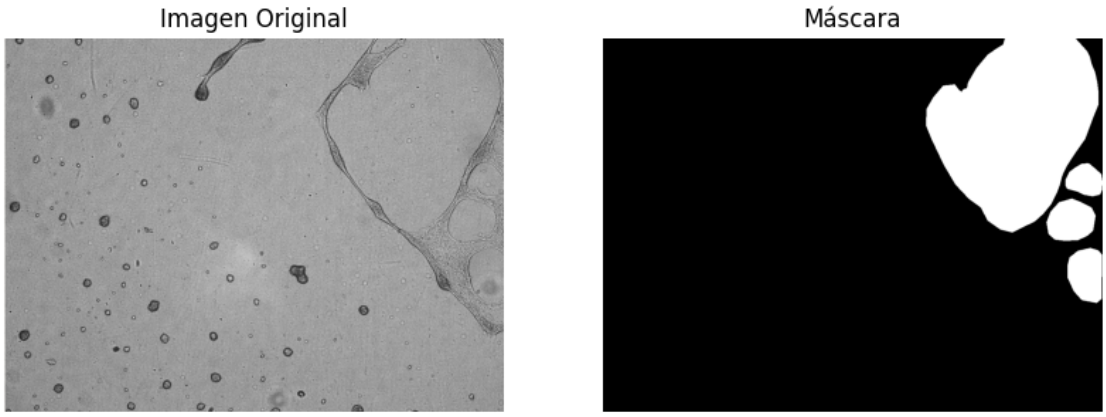
```
with open('/content/imgET.json') as f:
    annotations = json.load(f)


for shape in annotations['shapes']:
    if shape['label'] == 'miotubo':
        points = np.array(shape['points'], dtype=np.int32)


        cv2.polylines(image, [points], isClosed=True, color=(0, 255, 0),␣
 ↪thickness=2)


image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)


plt.imshow(image_rgb)
plt.axis('off')
plt.show()
```



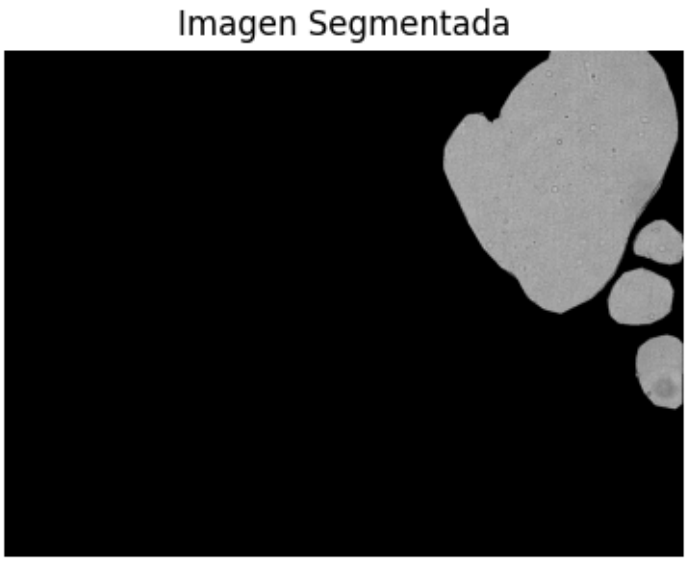
[25]:
```
def json_to_mask(json_path, output_path):

    with open(json_path, 'r') as f:
```

```
        data = json.load(f)


    img_height = data['imageHeight']
    img_width = data['imageWidth']


    mask = np.zeros((img_height, img_width), dtype=np.uint8)


    for shape in data['shapes']:
        points = np.array(shape['points'], dtype=np.int32)
        cv2.fillPoly(mask, [points], color=255)


    cv2.imwrite(output_path, mask)


json_file = "/content/imgET.json"
output_mask = "sample_data/mask.png"


json_to_mask(json_file, output_mask)
```

[26]:
```
import cv2
import matplotlib.pyplot as plt


image = cv2.imread("/content/img.png")
mask = cv2.imread("/content/sample_data/mask.png", cv2.IMREAD_GRAYSCALE)


plt.figure(figsize=(10, 5))


plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title("Imagen Original")
plt.axis('off')


plt.subplot(1, 2, 2)
plt.imshow(mask, cmap='gray')
plt.title("Máscara")
plt.axis('off')

plt.show()
```

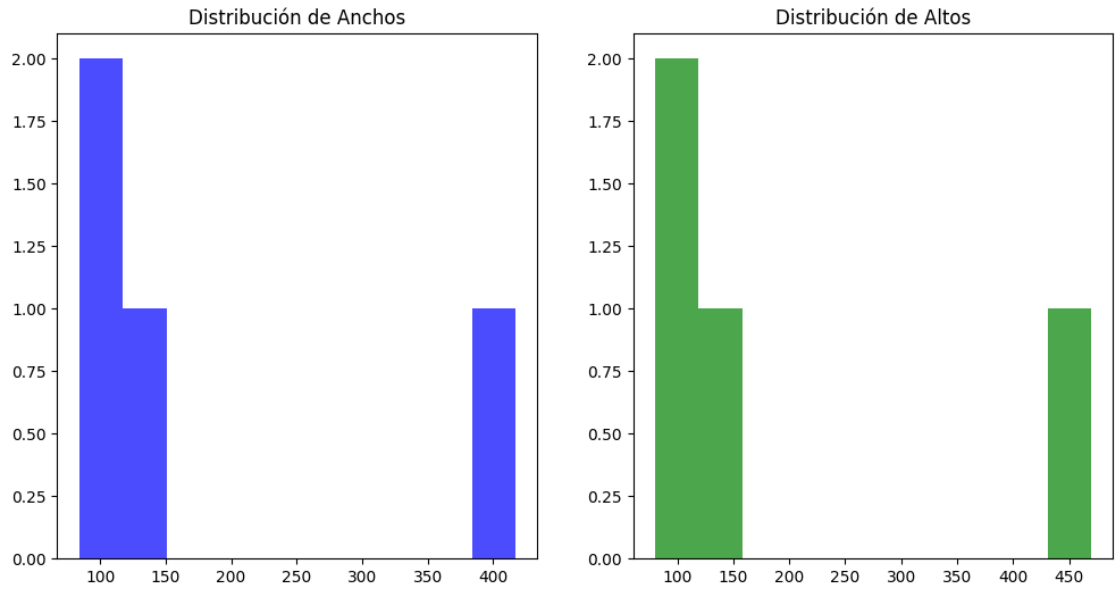


[27]:
```
import numpy as np


binary_mask = np.where(mask > 0, 1, 0)


segmented_image = cv2.bitwise_and(image, image, mask=binary_mask.astype(np.
 ↪uint8))


plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(segmented_image, cv2.COLOR_BGR2RGB))
plt.title("Imagen Segmentada")
plt.axis('off')

plt.show()
```



[28]:
```
contours, _ = cv2.findContours(binary_mask.astype(np.uint8), cv2.RETR_EXTERNAL,␣
↪cv2.CHAIN_APPROX_SIMPLE)


widths = []
heights = []


for contour in contours:

    x, y, w, h = cv2.boundingRect(contour)


    widths.append(w)
    heights.append(h)


plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.hist(widths, bins=10, color='blue', alpha=0.7)
plt.title("Distribución de Anchos")

plt.subplot(1, 2, 2)
plt.hist(heights, bins=10, color='green', alpha=0.7)
plt.title("Distribución de Altos")

plt.show()
```

```
print(f"Tamaño de la muestra: {len(widths)}")
```




```
Tamaño de la muestra: 4
```

[29]:
```
contours, _ = cv2.findContours(binary_mask.astype(np.uint8), cv2.RETR_EXTERNAL,␣
↪cv2.CHAIN_APPROX_SIMPLE)


centroids = []


for contour in contours:
    M = cv2.moments(contour)
    if M["m00"] != 0:
        cx = int(M["m10"] / M["m00"])
        cy = int(M["m01"] / M["m00"])
        centroids.append((cx, cy))

image_with_distances = image.copy()


for i in range(len(centroids)):
    for j in range(i + 1, len(centroids)):
```

```
        distance = np.sqrt((centroids[j][0] - centroids[i][0])**2 +␣
↪(centroids[j][1] - centroids[i][1])**2)


        cv2.line(image_with_distances, centroids[i], centroids[j], (0, 255, 0),␣
↪1)


        mid_point = ((centroids[i][0] + centroids[j][0]) // 2, (centroids[i][1]␣
↪+ centroids[j][1]) // 2)
        cv2.putText(image_with_distances, f'{int(distance)}', mid_point, cv2.
↪FONT_HERSHEY_SIMPLEX, 0.4, (255, 0, 0), 1)


plt.figure(figsize=(10, 10))
plt.imshow(cv2.cvtColor(image_with_distances, cv2.COLOR_BGR2RGB))
plt.title("Distancias entre blobs visualizadas")
plt.axis('off')
plt.show()
```