ST. FRANCIS INSTITUTE OF TECHNOLOGY

Mount Poinsur, S.V.P. Road, Borivali (W), Mumbai - 400 103.

CLASS / BATCH: TE-EXTCA /TA-3 SEM: VI

NAME: Om Kadam ROLL NO: 41

SUBJECT: IPMV

EXPERIMENT NO. : 05

TITLE: Morphological Operations

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

Correction Parameters	Marks Allotted	Correction Parameters	Marks Allotted
Submission on time [10%]		Experimental Result [30%]	
Conclusion / Result Analysis/ Discussion [30%]		Post Experiment Exercise [20%]	
Presentation of Write-Up [10%]			

TOTAL MARKS:

FACULTY SIGNATURE (WITH DATE)

Roll No. _____

Experiment – 05: Morphological operations

Date: _____

1. **Aim :** To implement image Morphological operations, Erosion and Dilation

2. **Requirements:** Python

3. **Pre-Experiment Exercise**

3.1 Brief Theory

Morphological image processing pursues the goals of removing these imperfections by accounting for the form and structure of the image. Morphological operations rely only on the relative ordering of pixel values, not on their numerical values, and therefore are especially suited to the processing of binary images. Morphological operations can also be applied to greyscale images such that their light transfer functions are unknown and therefore their absolute pixel values are of no or minor interest. The two principal morphological operations are dilation and erosion. Dilation allows objects to expand, thus potentially filling in small holes and connecting disjoint objects. Erosion shrinks objects by etching away (eroding) their boundaries. These operations can be customized for an application by the proper selection of the structuring element, which determines exactly how the objects will be dilated or eroded.

Dilation:

The dilation is an operation used to grow or thicken objects in binary images. The dilation process is performed by laying the structuring element B on the image A and sliding it across the image in a manner similar to convolution. The difference is in the operation performed. It is best described in a sequence of steps:

If the origin of the structuring element coincides with a 'white' pixel in the image, there is no change; move to the next pixel.

If the origin of the structuring element coincides with a 'black' in the image, make black all pixels from the image covered by the structuring element.

$$A \oplus B(x,y) = \max_{i,j \in B} \{A(x+i, y+j) + B(i,j)\}$$

The structuring element can have any shape. Typical shapes are presented below:

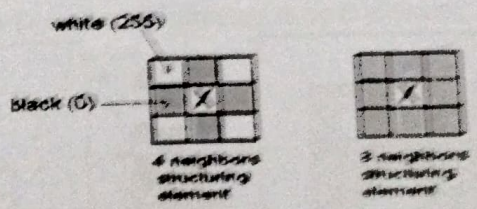


Fig. 1: Typical shapes of the structuring elements

An example is shown in Fig. 2. Note that with a dilation operation, all the 'black' pixels in the original image will be retained, any boundaries will be expanded, and small holes will be filled.

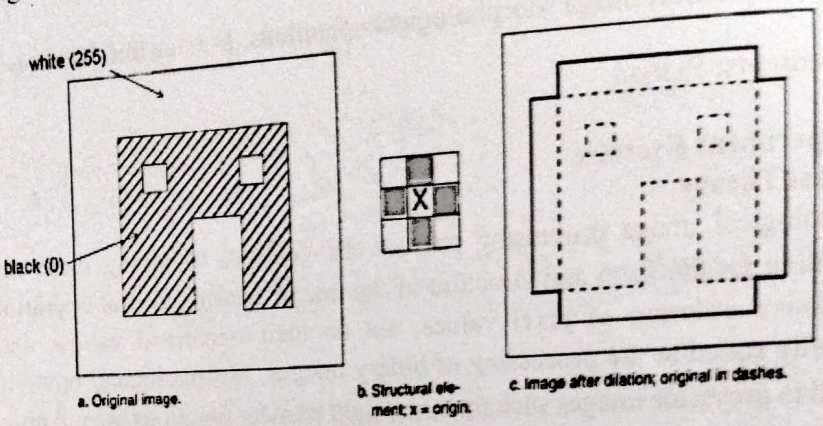


Fig. 2 Illustration of the dilatation process

Erosion:

Erosion is used to shrink or thin objects in binary images. The erosion process is similar to dilation, but we turn pixels to 'white', not 'black'. Erosion can be used to remove isolated features. As before, slide the structuring element across the image and then follow these steps:

If the origin of the structuring element coincides with a 'white' pixel in the image, there is no change; move to the next pixel.

If the origin of the structuring element coincides with a 'black' pixel in the image, and at least one of the 'black' pixels in the structuring element falls over a white pixel in the image, then change the 'black' pixel in the image (corresponding to the position on which the center of the structuring element falls) from 'black' to a 'white'.

$$A \ominus B(x,y) = \min_{i,j \in B} \{A(x+i, y+j) - B(i,j)\}$$

In Fig. 3 the only remaining pixels are those that coincide to the origin of the structuring element where the entire structuring element was contained in the existing object. Because the structuring element is 3 pixels wide, the 2-pixel-wide right leg of the image object was eroded away, but the 3-pixel-wide left leg retained some of its center pixels.

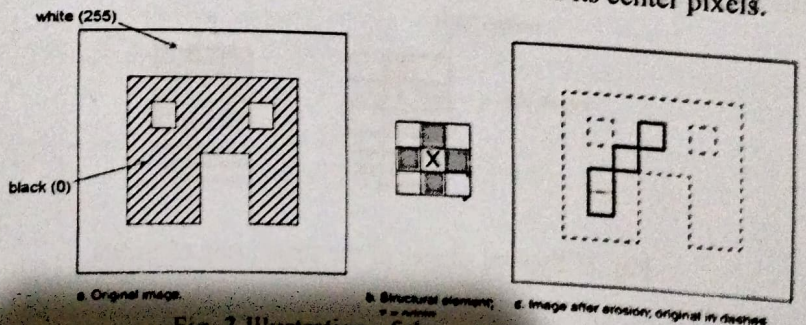


Fig. 3 Illustration of the erosion process

Boundary Extraction:

If A is an image and structuring element is B then Boundary Extraction can be given as,

Boundary (A) = A - (A ⊖ B)

4. Laboratory Exercise

4.1 Algorithm :

1. Accept the binary as an input and 3 x 3 structuring element with all ones
2. Perform the required morphological operation.
3. Display all input and output images.

5. Post Experiment

5.1 Conclusion

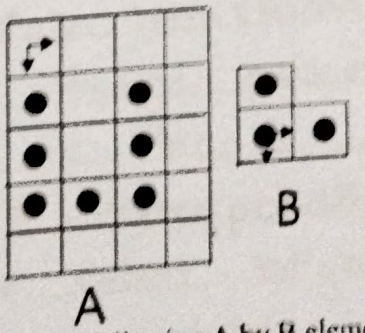
Morphological operations are a powerful tool for image processing that can be used to manipulate the shape and structure of image features. These include erosion, dilation, opening, closing and hit-or-miss transformation.

5.2 Questions

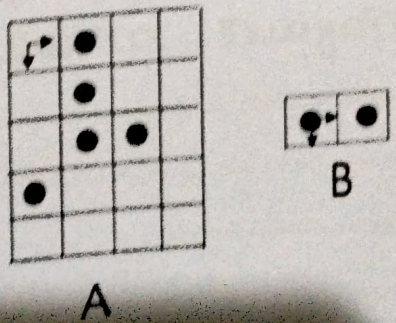
i. What are the advantage of opening and closing morphological operation?

ii. Define Hit or Miss Transform.

iii. Find the opening of A by B



iv. Find the dilation and erosion of following A by B element



Opening :-
Noise Removal : Opening is effective in removing small, noisy regions while preserving the shape and size of larger objects.
Break Connections : It can break narrow connections between objects, helping to separate objects that are touching

Closing :-
Filling holes : Closing is useful for filling small holes or gaps in objects, making them more ~~uniform~~. solid and complete.
Smoothing contours : It can smooth out the contours of objects, making them more uniform.

2) The Hit-or-miss transform is a morphological operation used in image processing to detect specific patterns or shapes within given binary pattern with the corresponding pixels in the input image. The result highlights locations where the pattern matches making it useful for precise pattern recognition and shape detection task.

3)

A =

0	0	0	0
1	0	1	0
1	0	1	0
1	1	1	0
0	0	0	0

B =

1	
1	1

Eroding A with B

0	0	0	0
0	0	0	0
0	0	0	0
1	0	0	0
0	0	0	0

Dilating the eroded with B

0	0	0	0
0	0	0	0
1	0	0	0
1	1	0	0
0	0	0	0

A =

0	1	0	0
0	1	0	0
0	1	1	0
1	0	0	0
0	0	0	0

B =

1	1

Dilation =

0	1	1	0
0	1	1	0
0	1	1	1
1	1	0	0
0	0	0	0

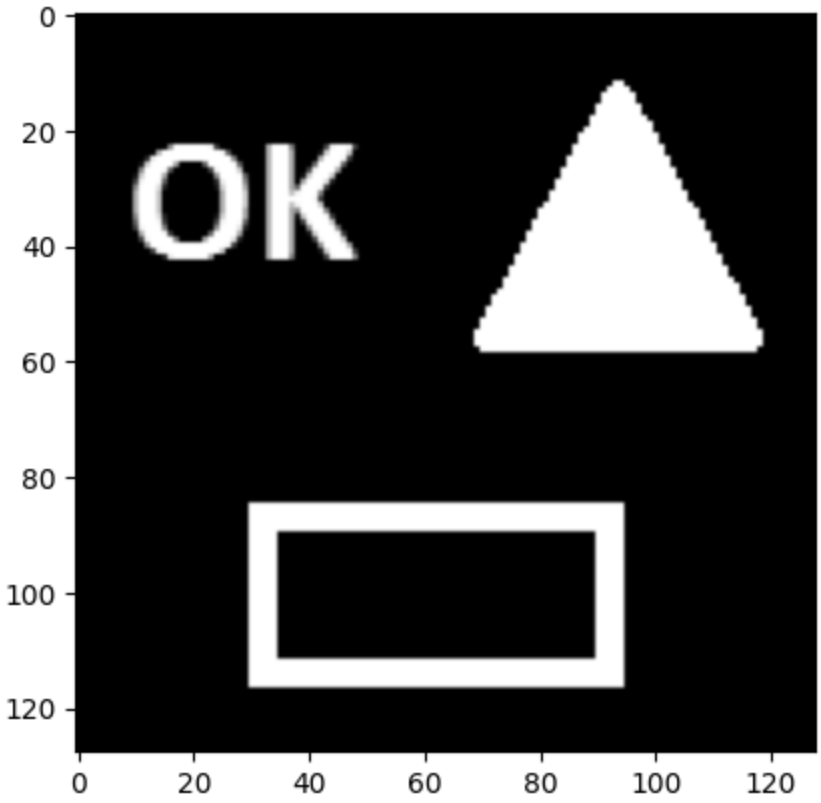
Erosion =

0	0	0	0
0	0	0	0
0	1	0	0
0	0	0	0
0	0	0	0

In [ ]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from io import BytesIO
from PIL import Image
from google.colab.patches import cv2_imshow
from google.colab import files
# uploaded = files.upload()
```

In [ ]:
```
img = cv2.imread('OM.png',0)
plt.imshow(img,cmap=plt.cm.gray)
```

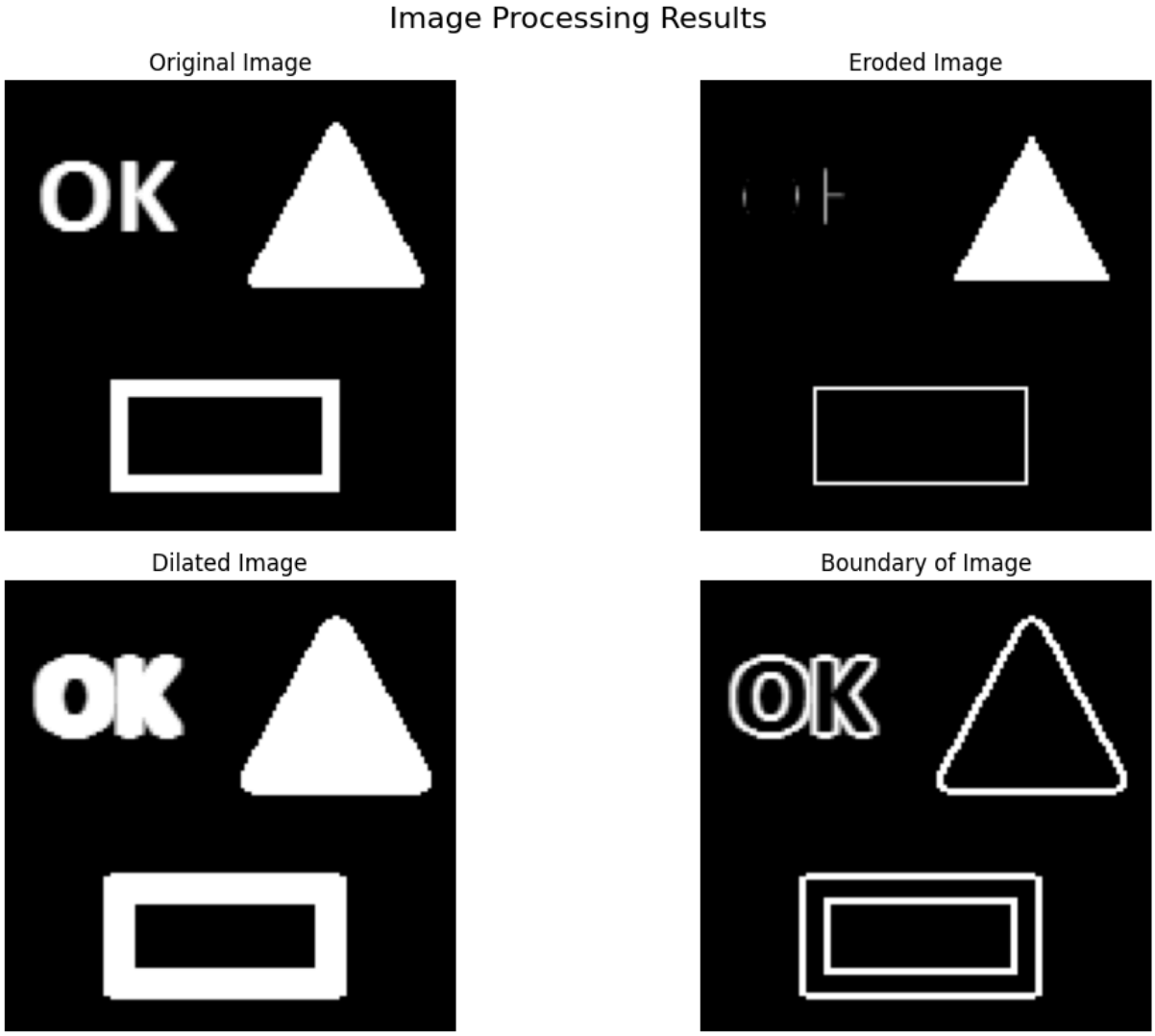
Out[ ]: <matplotlib.image.AxesImage at 0x7d8cd80b5540>



In [ ]:
```
m,n= img.shape
```

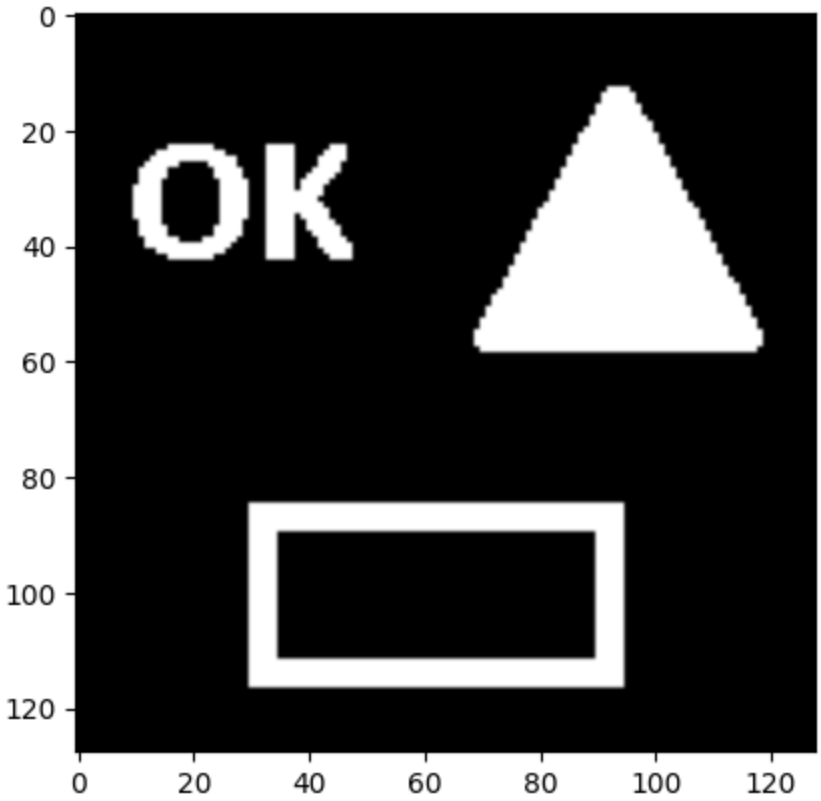
In [ ]:
```
kernel = np.ones((5,5),np.uint8)
kernel1=cv2.getStructuringElement(cv2.MORPH_RECT,(5,5))
kernel2 =cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
kernel3= cv2.getStructuringElement(cv2.MORPH_CROSS,(5,5))
img_erosion = cv2.erode(img, kernel, iterations=1)
img_dilate = cv2.dilate(img, kernel2, iterations=1)
img_boundary = img_dilate-img


fig = plt.figure(figsize=(12, 8))

# Set the title for the entire figure
fig.suptitle('Image Processing Results', fontsize=16)
```

```
# Add subplots
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
ax4 = fig.add_subplot(2, 2, 4)

# Set titles for subplots
ax1.set_title('Original Image', fontsize=12)
ax2.set_title('Eroded Image', fontsize=12)
ax3.set_title('Dilated Image', fontsize=12)
ax4.set_title('Boundary of Image', fontsize=12)

# Display images on subplots
ax1.imshow(img, cmap='gray')
ax2.imshow(img_erosion, cmap='gray')
ax3.imshow(img_dilate, cmap='gray')
ax4.imshow(img_boundary, cmap='gray')

# Remove ticks from subplots
ax1.axis('off')
ax2.axis('off')
ax3.axis('off')
ax4.axis('off')

# Adjust layout
plt.tight_layout()

# Show the figure
plt.show()
```



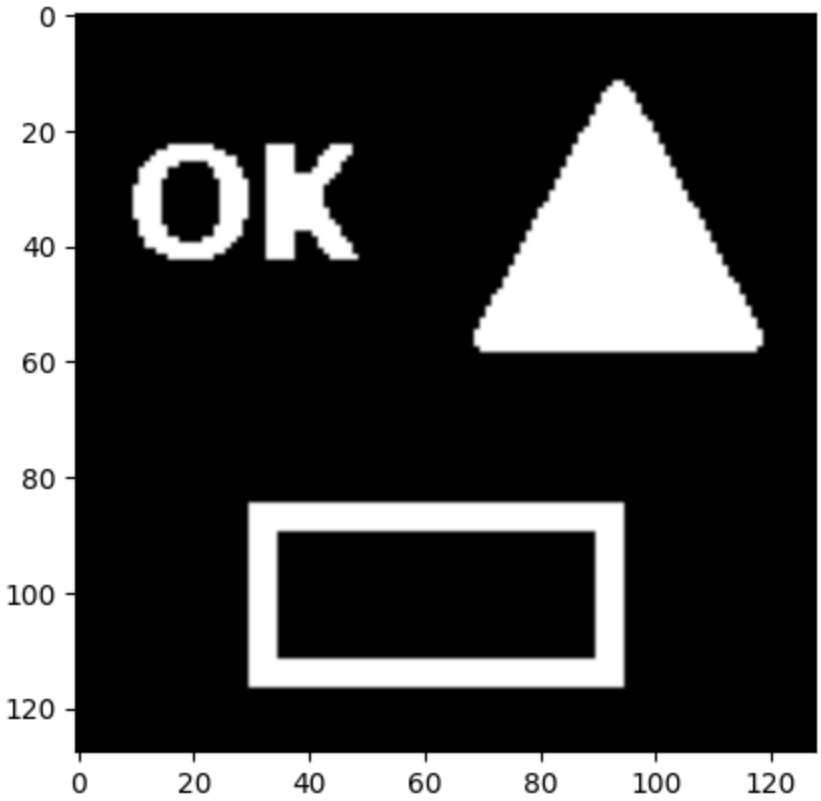
In [ ]:
```
#Opening & Closing
binr = cv2.threshold(img, 0, 255,
                     cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]

# define the kernel
kernel = np.ones((3, 3), np.uint8)

# opening the image
opening = cv2.morphologyEx(binr, cv2.MORPH_OPEN,
                           kernel, iterations=1)
# print the output
plt.imshow(opening, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7d8cd7c9d150>
```



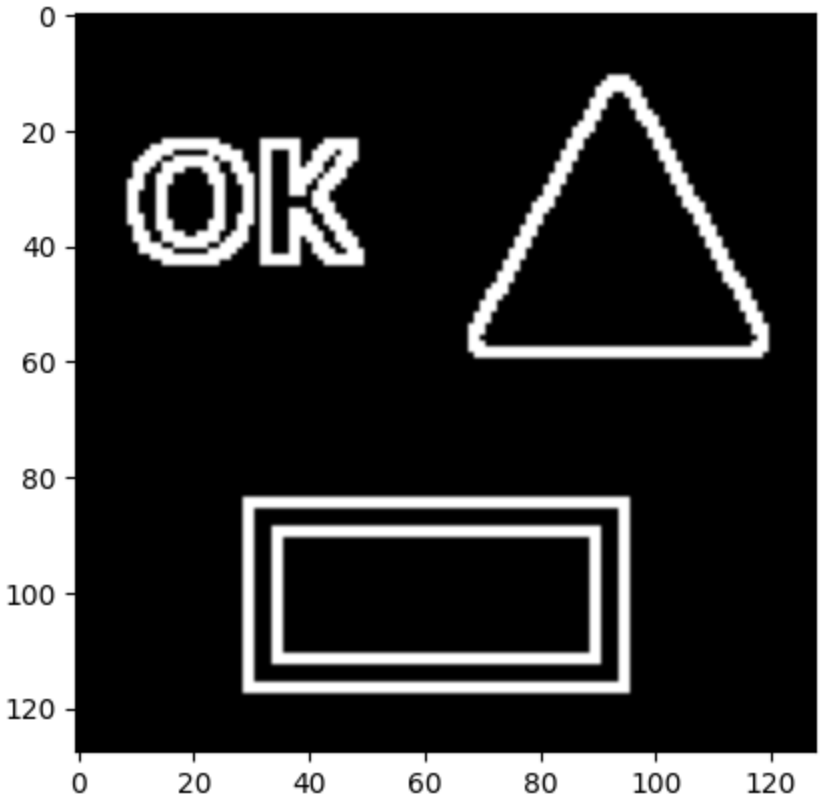
In [ ]:
```
binr = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]

# define the kernel
kernel = np.ones((3, 3), np.uint8)

# opening the image
closing = cv2.morphologyEx(binr, cv2.MORPH_CLOSE, kernel, iterations=1)

# print the output
plt.imshow(closing, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7d8cd7c9ee60>
```



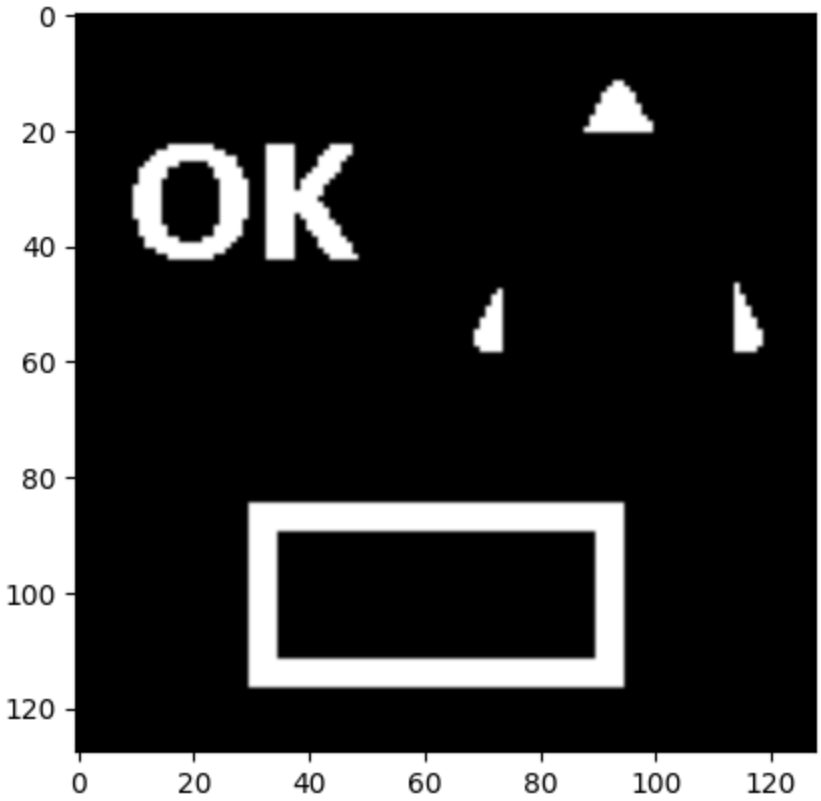
In [ ]:
```
#Morphological Gradient
binr = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]

# define the kernel
kernel = np.ones((3, 3), np.uint8)

# invert the image
invert = cv2.bitwise_not(binr)

# use morph gradient
morph_gradient = cv2.morphologyEx(invert,
                                  cv2.MORPH_GRADIENT,
                                  kernel)


# print the output
plt.imshow(morph_gradient, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7d8cd8387640>
```



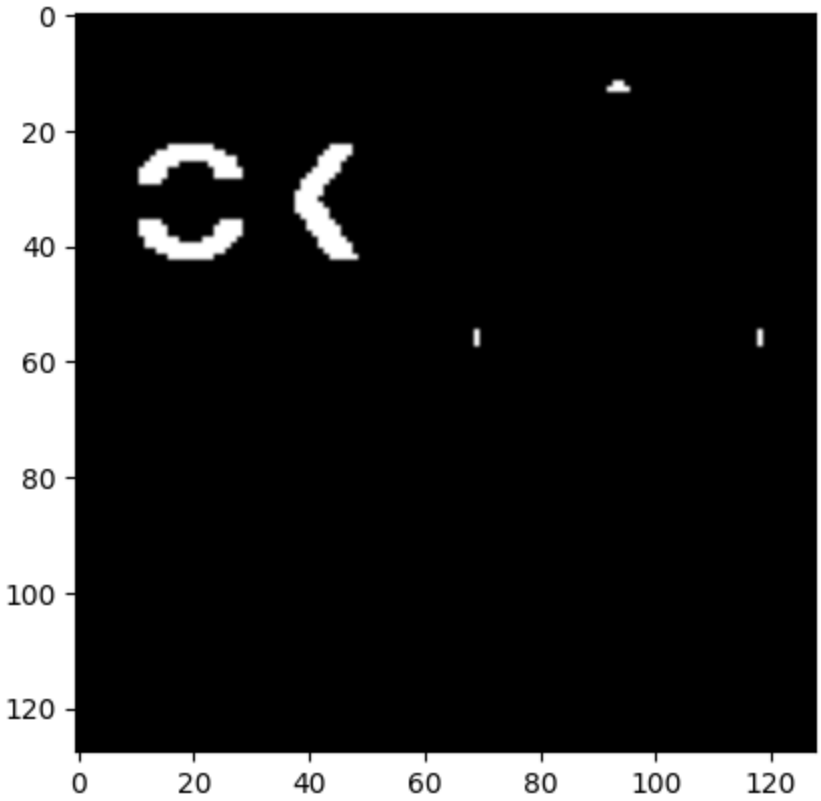
In [ ]:
```
#Top hat
binr = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]

# define the kernel
kernel = np.ones((13, 13), np.uint8)

# use morph gradient
morph_gradient = cv2.morphologyEx(binr,
                                  cv2.MORPH_TOPHAT,
                                  kernel)
# print the output
plt.imshow(morph_gradient, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7d8cd8d84400>
```

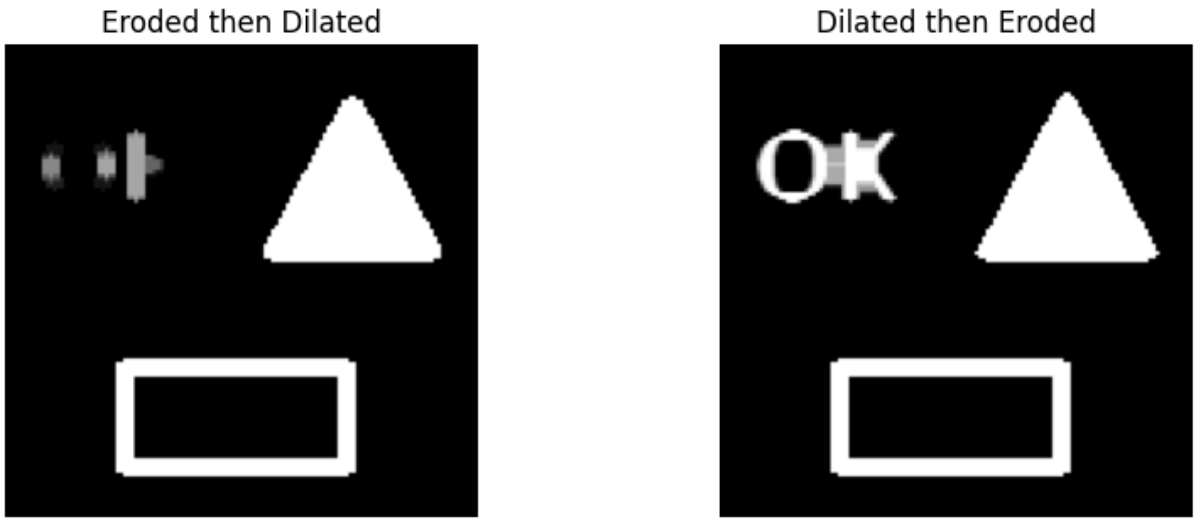


In [ ]:
```
# Black Hat
binr = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]

# define the kernel
kernel = np.ones((5, 5), np.uint8)

# invert the image
invert = cv2.bitwise_not(binr)

# use morph gradient
morph_gradient = cv2.morphologyEx(invert,
                                  cv2.MORPH_BLACKHAT,
                                  kernel)
# print the output
plt.imshow(morph_gradient, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7d8cd8ef2fb0>
```

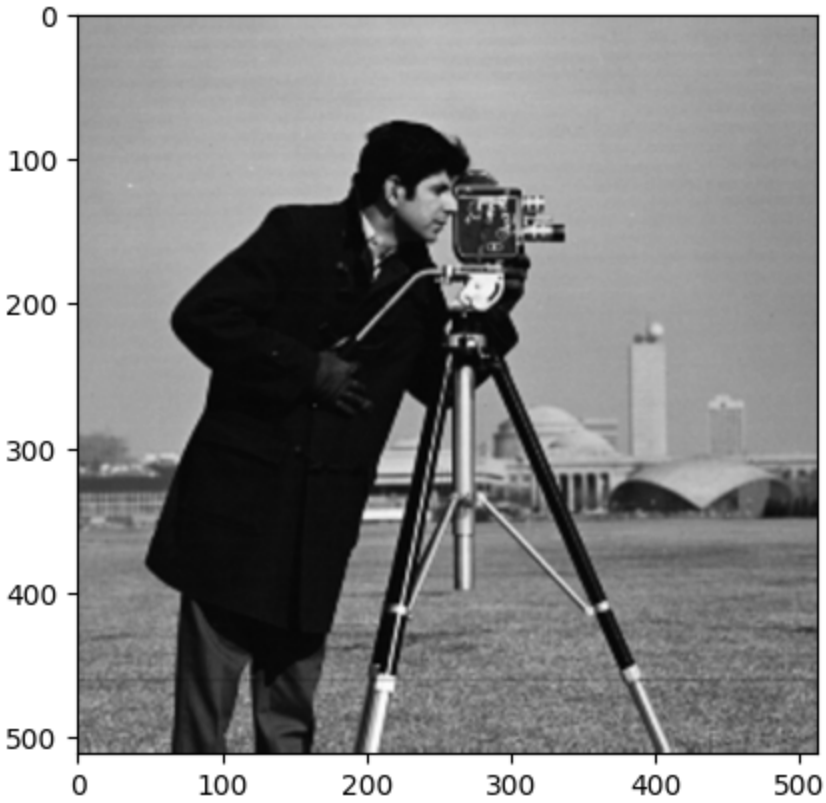


In [ ]:
```
img_eroded_dilate = cv2.dilate(img_erosion, kernel2, iterations=1)
img_dilated_erode = cv2.erode(img_dilate, kernel, iterations=1)


fig = plt.figure(figsize=(10, 8))

# Set the title for the entire figure
fig.suptitle('Opening & Closing Operations', fontsize=16)

# Add subplots
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)

# Set titles for subplots
ax1.set_title('Eroded then Dilated', fontsize=12)
ax2.set_title('Dilated then Eroded', fontsize=12)

# Display images on subplots
ax1.imshow(img_eroded_dilate, cmap='gray')
ax2.imshow(img_dilated_erode, cmap='gray')

# Remove ticks from subplots
ax1.axis('off')
ax2.axis('off')

# Show the figure
plt.show()
```



In [ ]:
```
img = cv2.imread('cameraman.tif',0)
plt.imshow(img,cmap=plt.cm.gray)
```

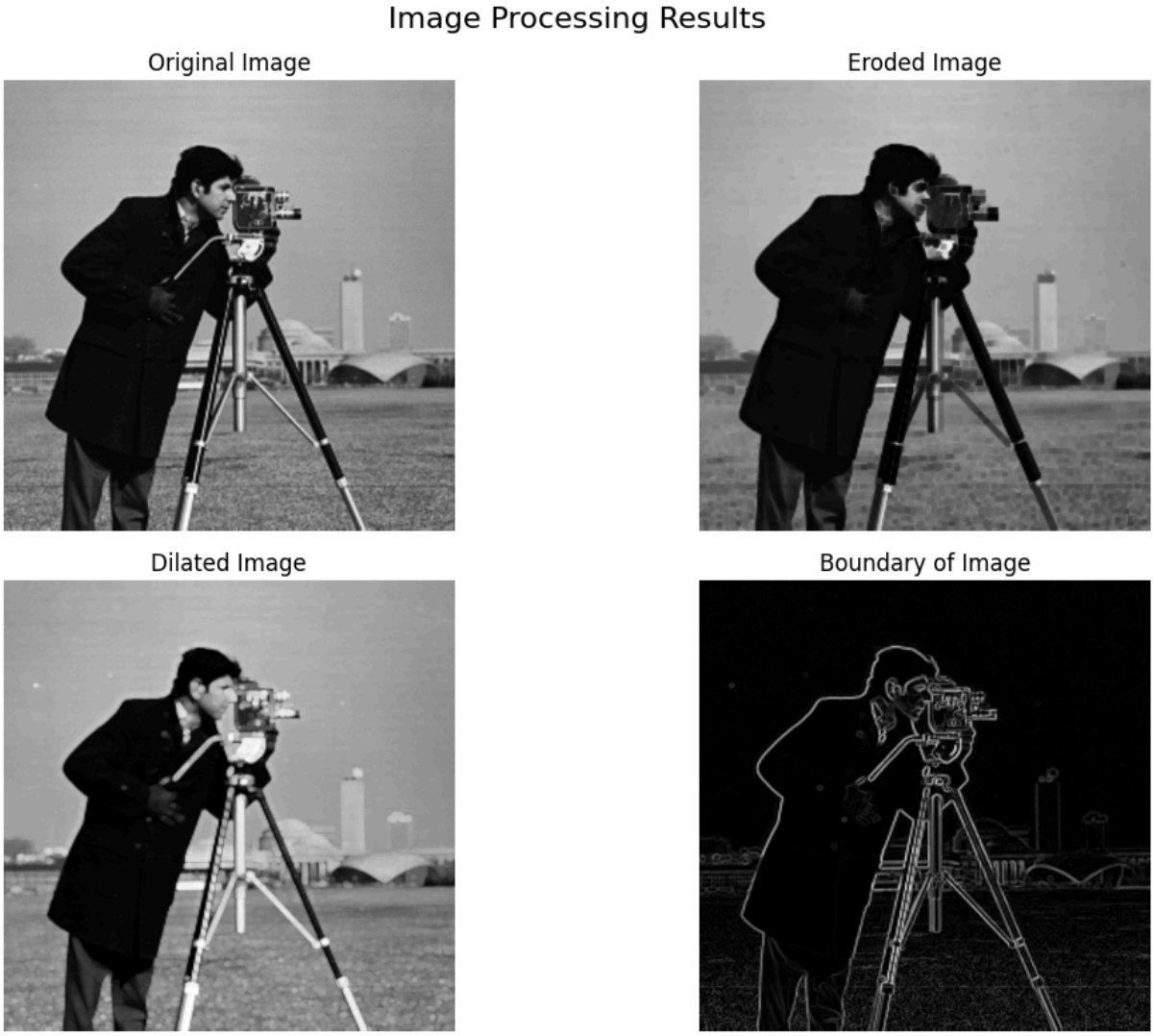
Out[ ]:
```
<matplotlib.image.AxesImage at 0x7d8cd8cecbe0>
```



In [ ]:
```
m,n= img.shape
```

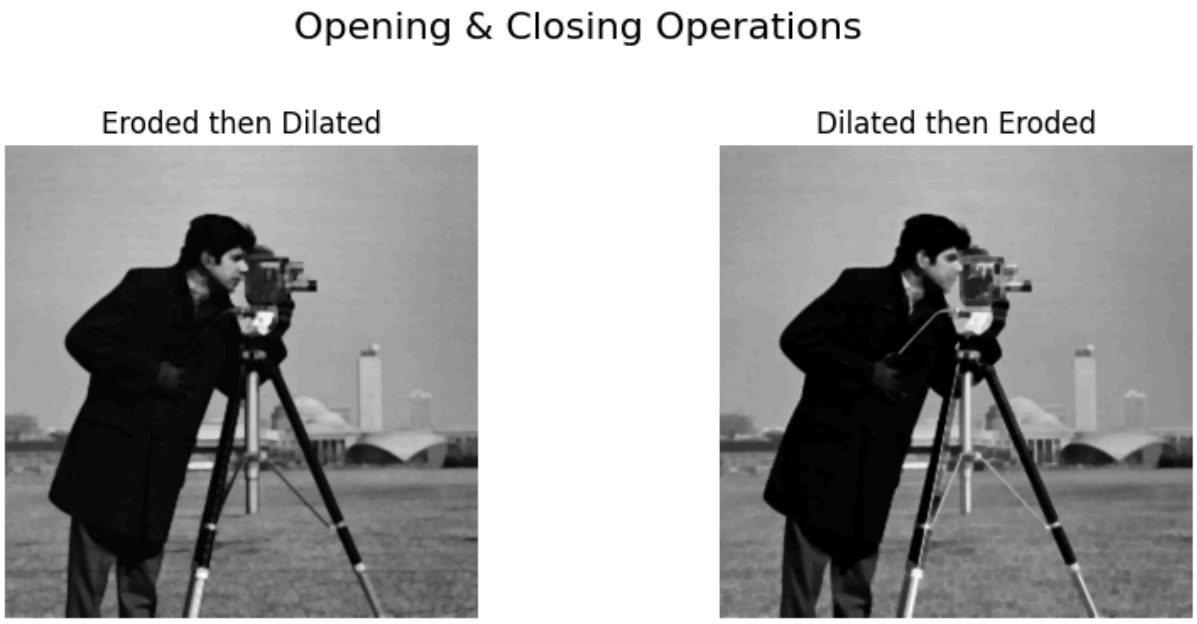
In [ ]:
```
img_erosion = cv2.erode(img, kernel, iterations=1)
img_dilate = cv2.dilate(img, kernel2, iterations=1)
img_boundary = img_dilate-img
```

```
fig = plt.figure(figsize=(12, 8))

# Set the title for the entire figure
fig.suptitle('Image Processing Results', fontsize=16)

# Add subplots
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)
ax3 = fig.add_subplot(2, 2, 3)
ax4 = fig.add_subplot(2, 2, 4)

# Set titles for subplots
ax1.set_title('Original Image', fontsize=12)
ax2.set_title('Eroded Image', fontsize=12)
ax3.set_title('Dilated Image', fontsize=12)
ax4.set_title('Boundary of Image', fontsize=12)

# Display images on subplots
ax1.imshow(img, cmap='gray')
ax2.imshow(img_erosion, cmap='gray')
ax3.imshow(img_dilate, cmap='gray')
ax4.imshow(img_boundary, cmap='gray')

# Remove ticks from subplots
ax1.axis('off')
ax2.axis('off')
ax3.axis('off')
ax4.axis('off')

# Adjust layout
plt.tight_layout()

# Show the figure
plt.show()
```



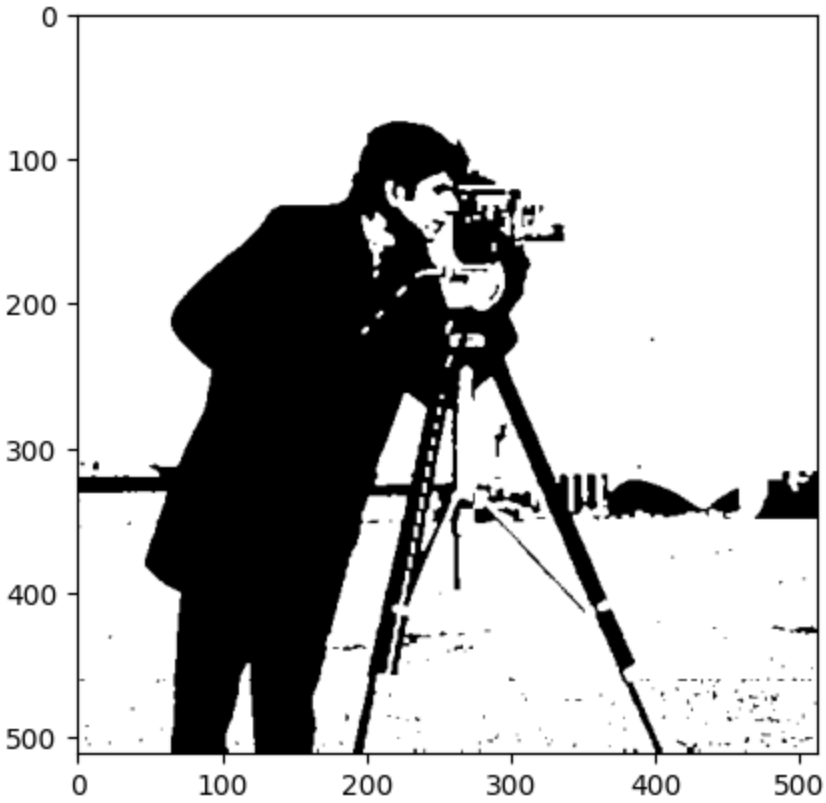
In [ ]:
```
img_eroded_dilate = cv2.dilate(img_erosion, kernel2, iterations=1)
img_dilated_erode = cv2.erode(img_dilate, kernel, iterations=1)


fig = plt.figure(figsize=(10, 8))

# Set the title for the entire figure
fig.suptitle('Opening & Closing Operations', fontsize=16)

# Add subplots
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 2)

# Set titles for subplots
ax1.set_title('Eroded then Dilated', fontsize=12)
ax2.set_title('Dilated then Eroded', fontsize=12)

# Display images on subplots
ax1.imshow(img_eroded_dilate, cmap='gray')
ax2.imshow(img_dilated_erode, cmap='gray')

# Remove ticks from subplots
ax1.axis('off')
```

```
ax2.axis('off')

# Show the figure
plt.show()
```



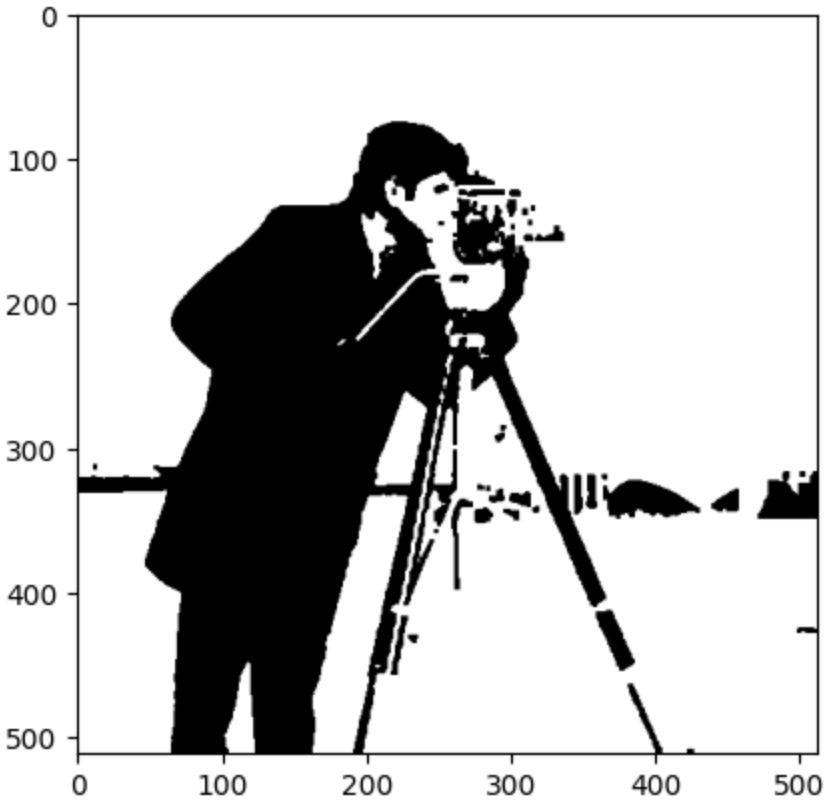
In [ ]:
```
#Opening & Closing
binr = cv2.threshold(img, 0, 255,
                     cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]

# define the kernel
kernel = np.ones((3, 3), np.uint8)

# opening the image
opening = cv2.morphologyEx(binr, cv2.MORPH_OPEN,
                           kernel, iterations=1)
# print the output
plt.imshow(opening, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7d8cd732bee0>
```



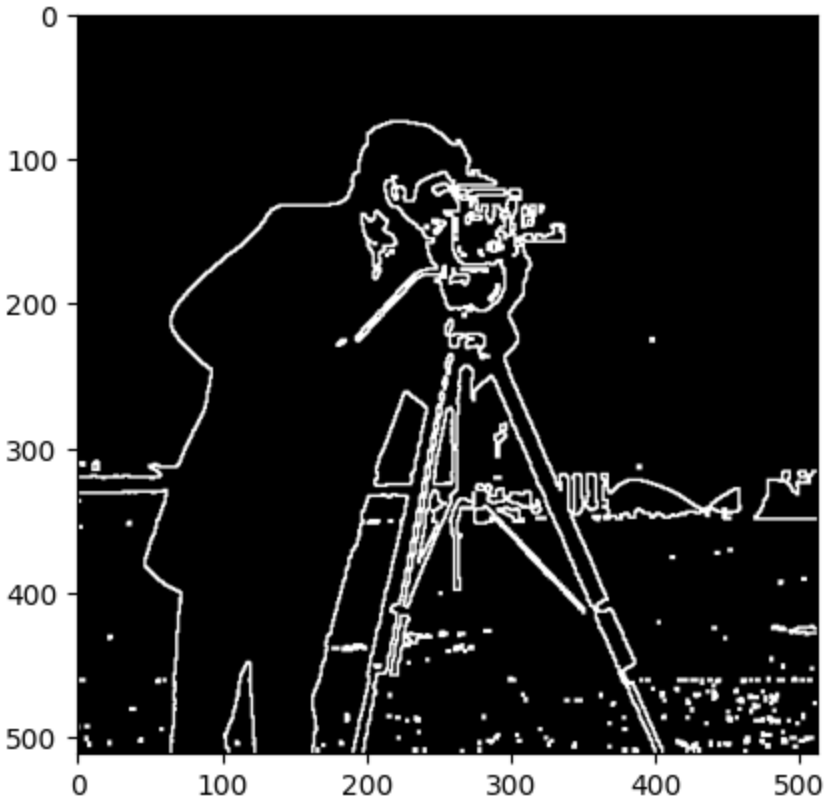
In [ ]:
```
binr = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]

# define the kernel
kernel = np.ones((3, 3), np.uint8)

# opening the image
closing = cv2.morphologyEx(binr, cv2.MORPH_CLOSE, kernel, iterations=1)

# print the output
plt.imshow(closing, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7d8cd705ca90>
```



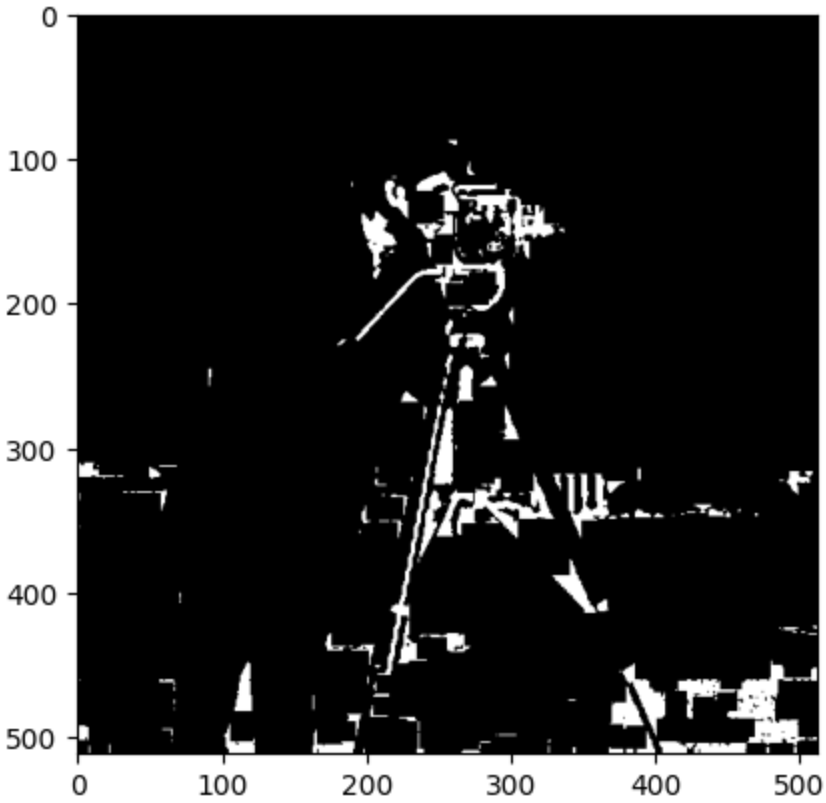
In [ ]:
```
#Morphological Gradient
binr = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]

# define the kernel
kernel = np.ones((3, 3), np.uint8)

# invert the image
invert = cv2.bitwise_not(binr)

# use morph gradient
morph_gradient = cv2.morphologyEx(invert,
                                  cv2.MORPH_GRADIENT,
                                  kernel)

# print the output
plt.imshow(morph_gradient, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7d8cd70bbee0>
```



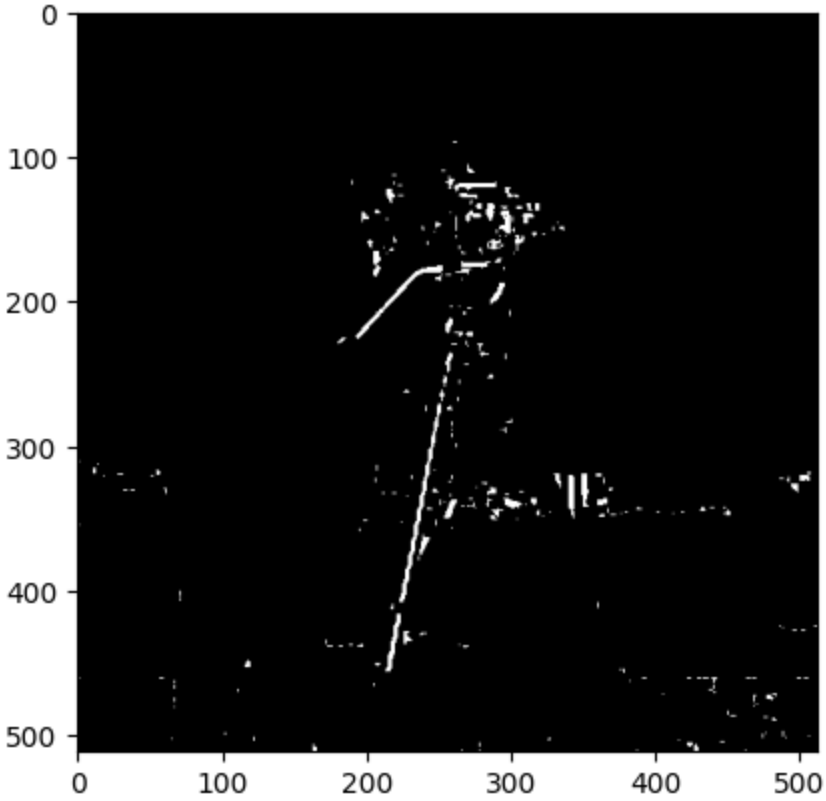
In [ ]:
```
#Top hat
binr = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]

# define the kernel
kernel = np.ones((13, 13), np.uint8)

# use morph gradient
morph_gradient = cv2.morphologyEx(binr,
                                  cv2.MORPH_TOPHAT,
                                  kernel)
# print the output
plt.imshow(morph_gradient, cmap='gray')
```

Out[ ]: <matplotlib.image.AxesImage at 0x7d8cd7136020>



In [ ]:
```
# Black Hat
binr = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]

# define the kernel
kernel = np.ones((5, 5), np.uint8)

# invert the image
invert = cv2.bitwise_not(binr)

# use morph gradient
morph_gradient = cv2.morphologyEx(invert,
                                  cv2.MORPH_BLACKHAT,
                                  kernel)
# print the output
plt.imshow(morph_gradient, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x7d8cd916cdf0>
```


0
100
200
300
400
500
0
100
200
300
400
500