Image Processing Lab

Subject Code: MCALE231

A Project Report Submitted in Fulfilment of the Degree of

MASTER

In COMPUTER APPLICATION

Year 2023-2024

By

Mr. KETAN RAVINDRA DAKI

(Application Id- 78749)

Semester-II

Under the Guidance of

Prof. Prashant Londhe

Institute of Distance and Open Learning
Vidya Nagari, Kalina, Santacruz East – 400098.
University of Mumbai

PCP Center
[Satish Pradhan Dyan sadhana College, Thane]

Institute of Distance and Open Learning
Vidya Nagari, Kalina, Santacruz East – 400098.

CERTIFICATE

This to certify that, **"KETAN RAVINDRA DAKI"** appearing **Master's in computer application (Semester II) Application ID: 78749** has satisfactorily completed the prescribed practical of **Image Processing Lab** as laid down by the University of Mumbai for the academic year 2023-24.

Teacher In Charge	External Examiner	Coordinator – M.C.A

Date:
Place:

Index

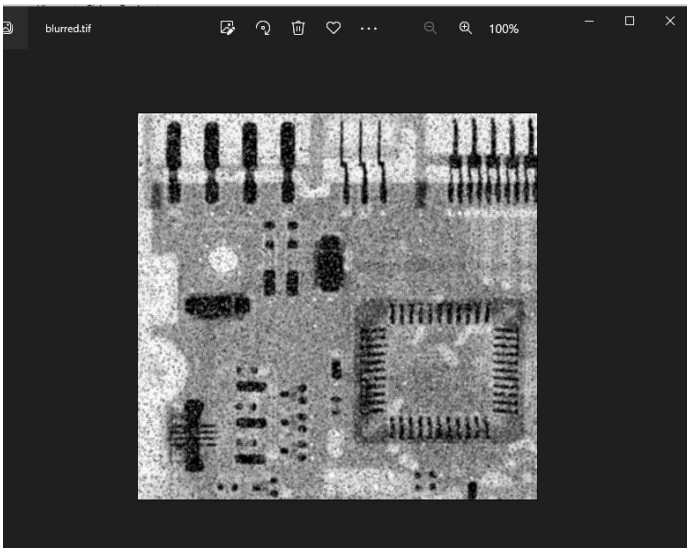
Exercise	Topic	Date	Signature
1	Programs for image enhancement using spatial domain filters. Program for Average spatial Filter.		
2	To Find DFT/FFT forward and inverse transform of image.		
3	To find DCT forward and inverse transform of image.		
4	Morphological operational: Dilation, Erosion, Opening, Closing.		
5	The detection of discontinuities – Point, Line and Edge detections, Hough transform, Thresholding, Region based segmentation chain codes		

Practical 1

Aim : Programs for image enhancement using spatial domain filters. Program for Average spatial Filter.

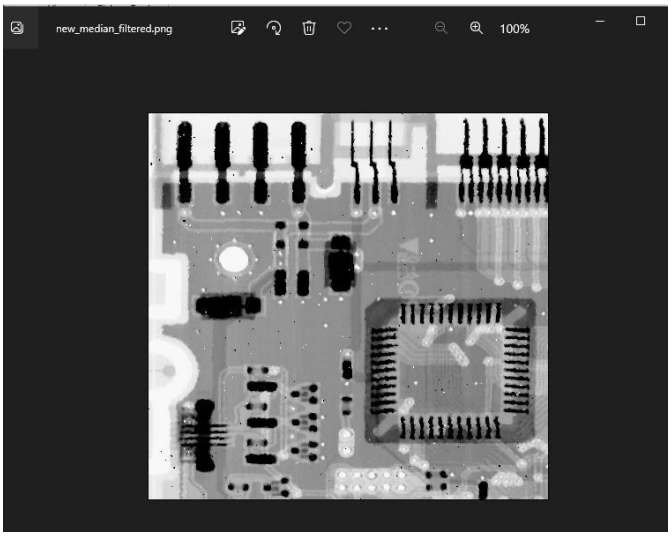
Code:-

```
import cv2
import numpy as np


# Read the image
img = cv2.imread('sample.png', 0)

# Obtain number of rows and columns
# of the image
m, n = img.shape

# Develop Averaging filter(3, 3) mask
mask = np.ones([3, 3], dtype = int)
mask = mask / 9

# Convolve the 3X3 mask over the image
img_new = np.zeros([m, n])

for i in range(1, m-1):
    for j in range(1, n-1):
        temp = img[i-1, j-1]*mask[0, 0]+img[i-1, j]*mask[0, 1]+img[i-1, j + 1]*mask[0,
2]+img[i, j-1]*mask[1, 0]+ img[i, j]*mask[1, 1]+img[i, j + 1]*mask[1, 2]+img[i + 1, j-
1]*mask[2, 0]+img[i + 1, j]*mask[2, 1]+img[i + 1, j + 1]*mask[2, 2]

        img_new[i, j]= temp

img_new = img_new.astype(np.uint8)
cv2.imwrite('blurred.tif', img_new)
```




```
import cv2
import numpy as np


# Read the image
img_noisy1 = cv2.imread('sample.png', 0)

# Obtain the number of rows and columns
# of the image
m, n = img_noisy1.shape

# Traverse the image. For every 3X3 area,
# find the median of the pixels and
# replace the center pixel by the median
img_new1 = np.zeros([m, n])

for i in range(1, m-1):
    for j in range(1, n-1):
        temp = [img_noisy1[i-1, j-1],
                img_noisy1[i-1, j],
                img_noisy1[i-1, j + 1],
                img_noisy1[i, j-1],
                img_noisy1[i, j],
                img_noisy1[i, j + 1],
                img_noisy1[i + 1, j-1],
                img_noisy1[i + 1, j],
                img_noisy1[i + 1, j + 1]]

        temp = sorted(temp)
        img_new1[i, j]= temp[4]

img_new1 = img_new1.astype(np.uint8)
```

```
cv2.imwrite('new_median_filtered.png', img_new1)
```

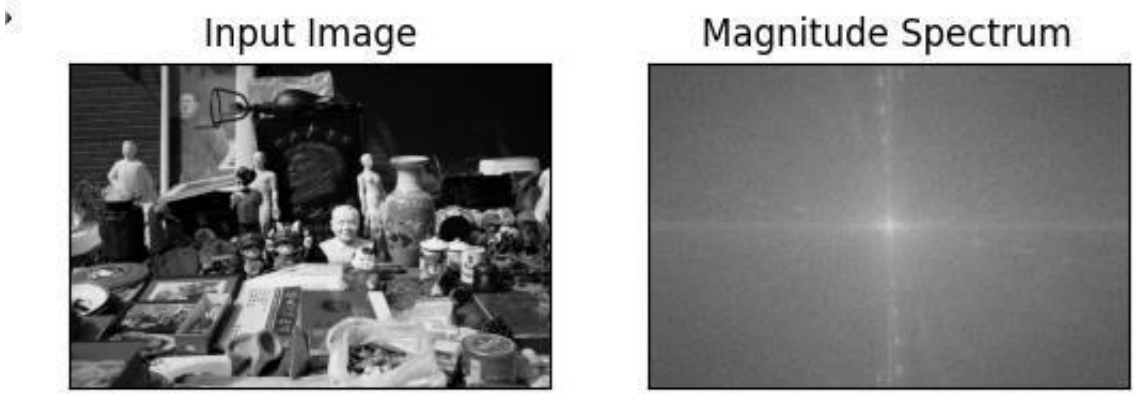


Practical 2

Aim :- To Find DFT/FFT forward and inverse transform of image.

Code:-

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

# read input image
img = cv2.imread('film.jpg',0)

# find the discrete fourier transform of the image
dft = cv2.dft(np.float32(img),flags = cv2.DFT_COMPLEX_OUTPUT)

# shift zero-frequency component to the center of the spectrum
dft_shift = np.fft.fftshift(dft)
magnitude_spectrum = 20*np.log(cv2.magnitude(dft_shift[:,:,0],dft_shift[:,:,1]))
# visualize input image and the magnitude spectrum
plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
```



Practical 3

Aim :- To find DCT forward and inverse transform of image.

```
import math

pi = 3.142857
m = 8
n = 8

# Function to find discrete cosine transform and print it
def dctTransform(matrix):

    # dct will store the discrete cosine transform
    dct = []
    for i in range(m):
        dct.append([None for _ in range(n)])

    for i in range(m):
        for j in range(n):

            # ci and cj depends on frequency as well as
            # number of row and columns of specified matrix
            if (i == 0):
                ci = 1 / (m ** 0.5)
            else:
                ci = (2 / m) ** 0.5
            if (j == 0):
                cj = 1 / (n ** 0.5)
            else:
                cj = (2 / n) ** 0.5

            # sum will temporarily store the sum of
            # cosine signals
            sum = 0
            for k in range(m):
                for l in range(n):

                    dct1 = matrix[k][l] * math.cos((2 * k + 1) * i * pi / (
                        2 * m)) * math.cos((2 * l + 1) * j * pi / (2 * n))
                    sum = sum + dct1

            dct[i][j] = ci * cj * sum

    for i in range(m):
        for j in range(n):
            print(dct[i][j], end="\t")
        print()
```

```
# Driver code
matrix = [[255, 255, 255, 255, 255, 255, 255, 255],
          [255, 255, 255, 255, 255, 255, 255, 255],
          [255, 255, 255, 255, 255, 255, 255, 255],
          [255, 255, 255, 255, 255, 255, 255, 255],
          [255, 255, 255, 255, 255, 255, 255, 255],
          [255, 255, 255, 255, 255, 255, 255, 255],
          [255, 255, 255, 255, 255, 255, 255, 255],
          [255, 255, 255, 255, 255, 255, 255, 255]]

dctTransform(matrix)
```

```
2039.9999999999995      -1.1681078033445202     1.1910101606541048      -
1.2306043961129725      1.2892204000077006      -1.3705578971374432
        1.480259193374122       -1.626904189710877
-1.1681078033447012     0.000668860706007024    -0.0006819746385176018
        0.0007046463715241202   -0.0007382100046555706
        0.0007847840071448786   -0.0008475991738947641
        0.000931568372211089
1.1910101606542707      -0.00068197463848918    0.0006953456876459541 -
0.0007184619311288998   0.0007526836253646252   -0.000800170775129061
        0.0008642175194708557   -0.0009498330491997109
-1.2306043961130728     0.0007046463715241202   -0.0007184619311786378
        0.0007423466567360038   -0.0007777060254028356
        0.0008267718496846044   -0.0008929477797785523
        0.0009814095333116057
1.2892204000077108      -0.0007382100047408358  0.0007526836253433089  -
0.0007777060253886248   0.0008147496273664956   -0.0008661525492072997
        0.0009354805634451679   -0.0010281559168010546
-1.3705578971374133     0.0007847840071093515   -0.000800170775129061
        0.0008267718496810517   -0.0008661525492108524
        0.0009207985046231215   -0.000994500454542191
        0.0010930227377929924
1.4802591933741172      -0.0008475991738450261  0.0008642175194708557  -
0.000892947779782105    0.0009354805634309571   -0.000994500454552849
        0.0010741016076369903   -0.001180509746856906
-1.6269041897109071     0.0009315683721755619   -0.0009498330491766183
        0.000981409533283184    -0.0010281559167850673
        0.0010930227377778934   -0.0011805097468391423
        0.0012974594325978472
```

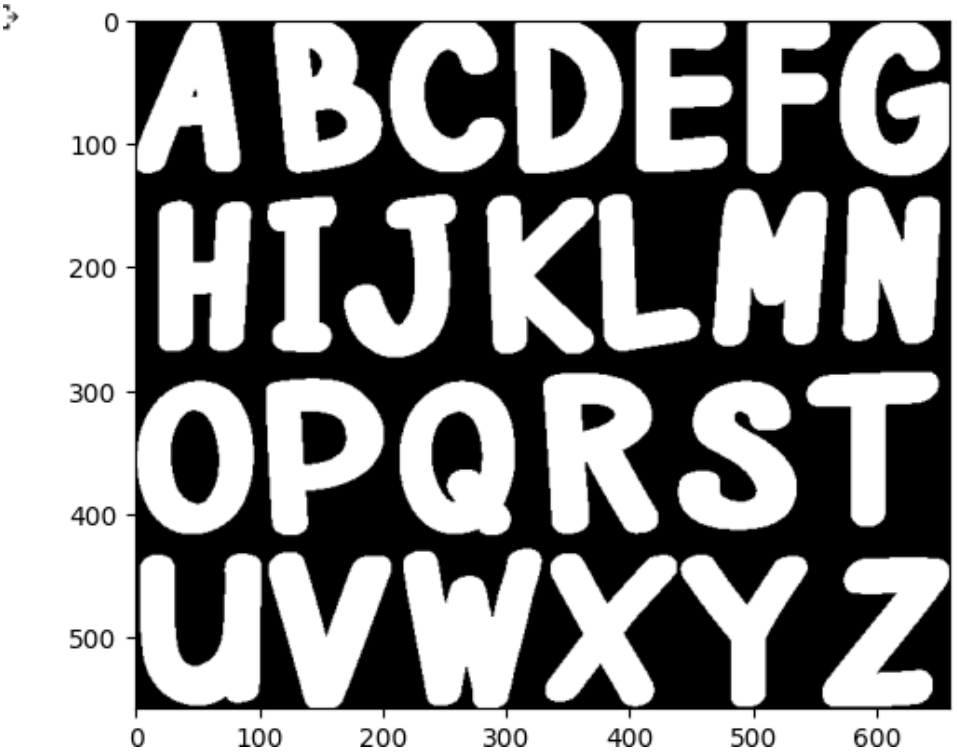
Practical 4

Aim : Morphological operational: Dilation, Erosion, Opening, Closing.

```
# import the necessary packages
import cv2
import numpy as np
import matplotlib.pyplot as plt

# read the image
img = cv2.imread(r"1.jpg", 0)

# binarize the image
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)

# erode the image
erosion = cv2.erode(invert, kernel,
                    iterations=1)

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

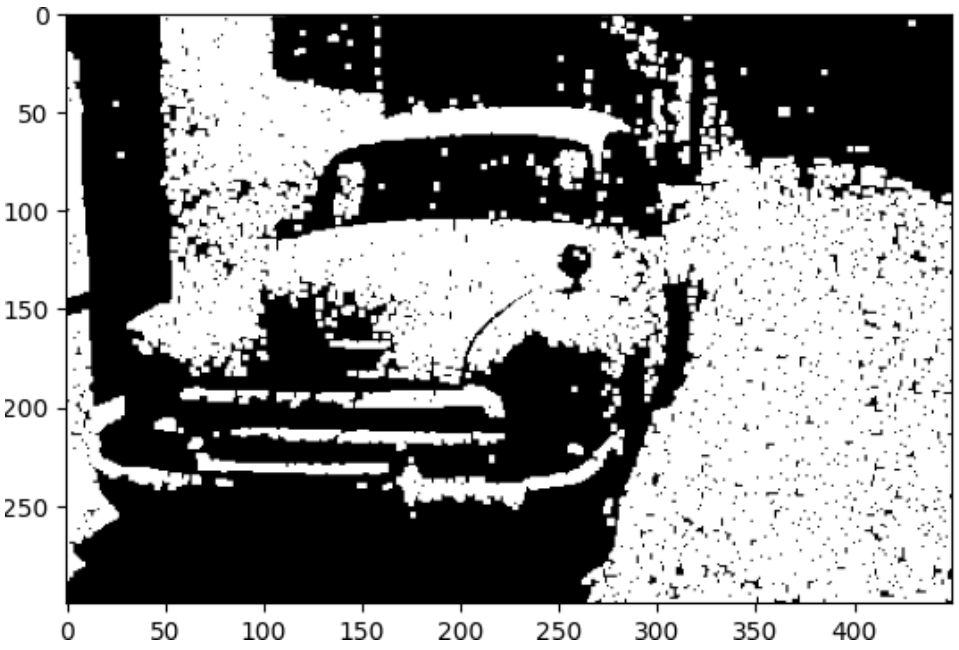

```
import cv2

# read the image
img = cv2.imread(r"1.jpg", 0)
```

```
# binarize the image
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)

# dilate the image
dilation = cv2.dilate(invert, kernel, iterations=1)

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

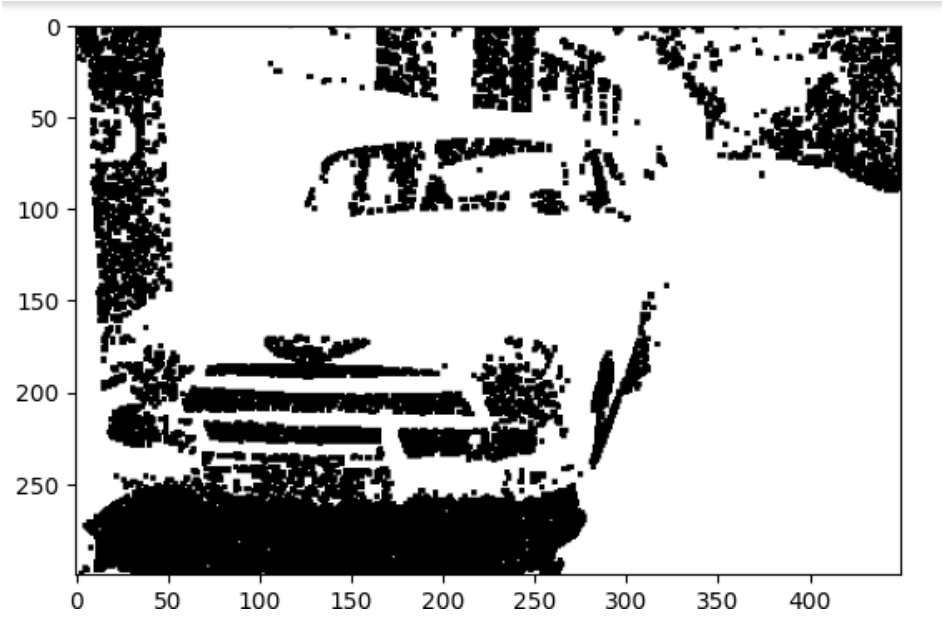



```
import cv2

# read the image
img = cv2.imread(r"2.png", 0)

# binarize the image
binr = cv2.threshold(img, 0, 255,
                     cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]

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

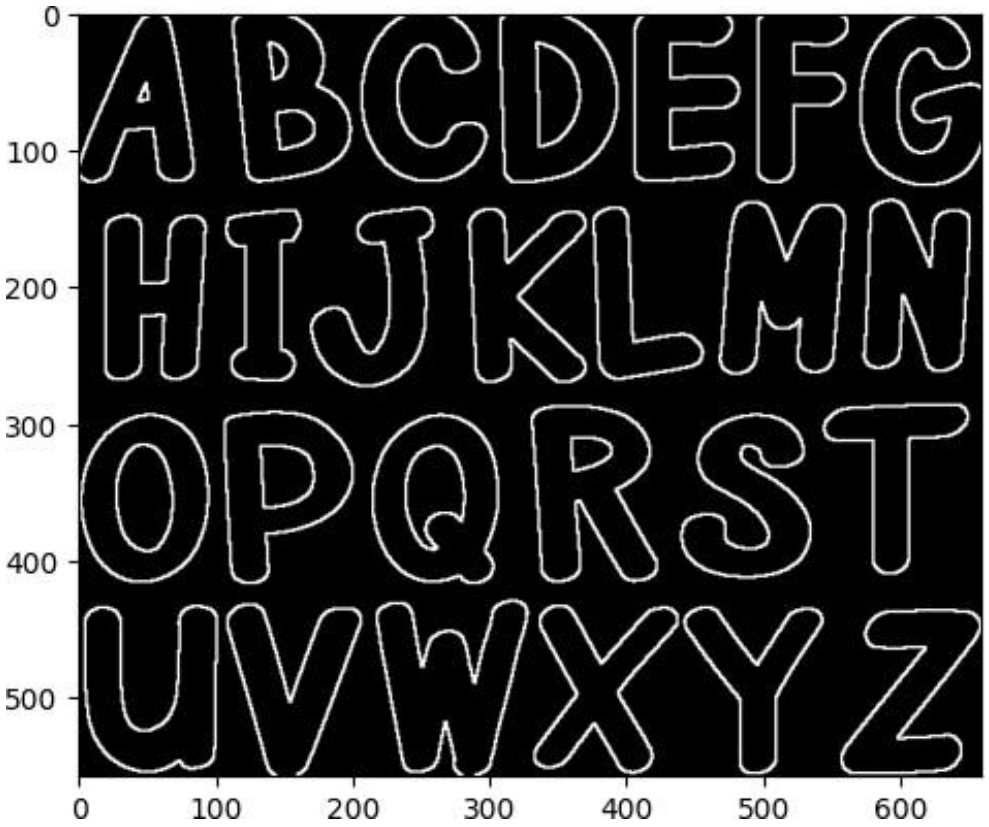
```
# opening the image
opening = cv2.morphologyEx(binr, cv2.MORPH_OPEN,
                           kernel, iterations=1)
# print the output
plt.imshow(opening, cmap='gray')
```



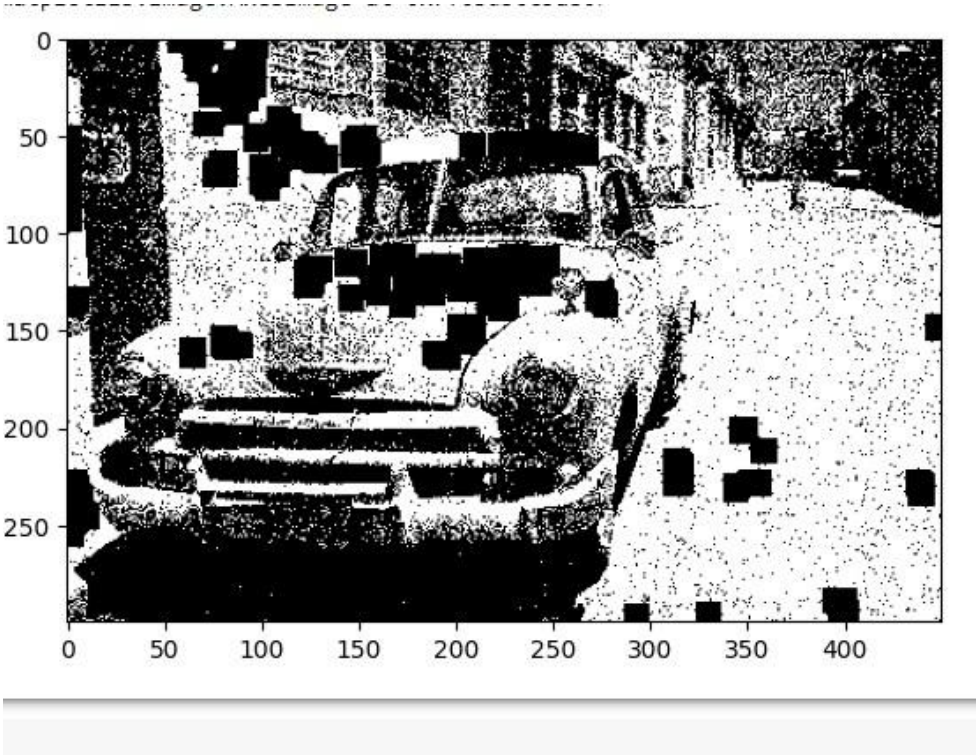

```
import cv2

# read the image
img = cv2.imread(r"2.png", 0)

# binarize the image
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')
```



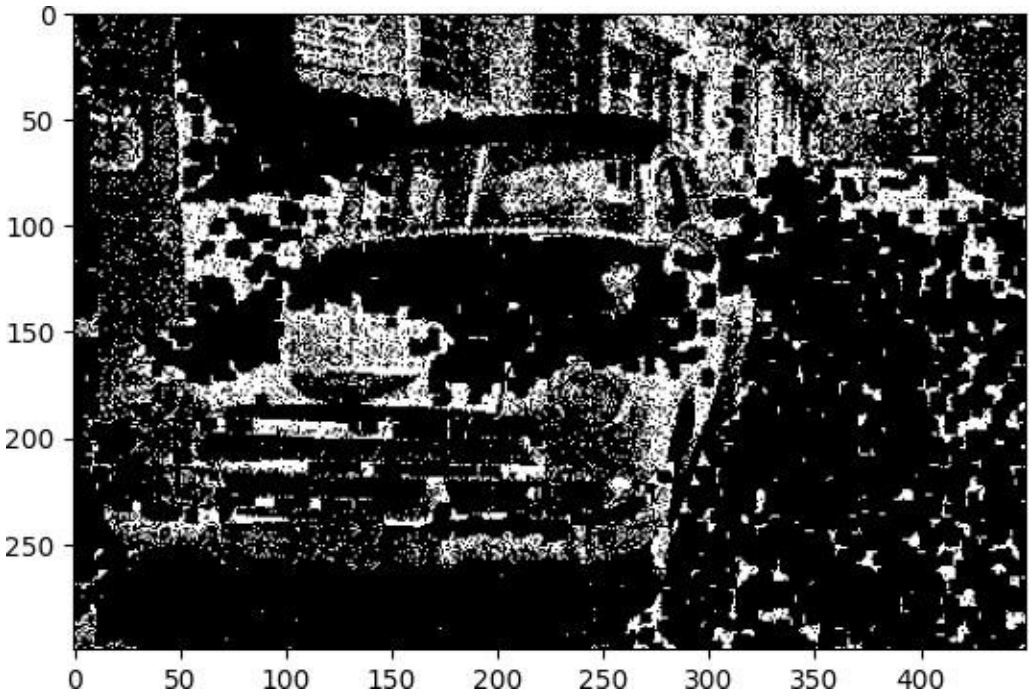

```
import cv2

# read the image
img = cv2.imread(r"1.jpg", 0)

# binarize the image
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')
```




```
# import the necessary packages
import cv2

# read the image
img = cv2.imread("2.png", 0)

# binarize the image
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')
```




```
# import the necessary packages
import cv2

# read the image
img = cv2.imread("2.png", 0)

# binarize the image
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')
```

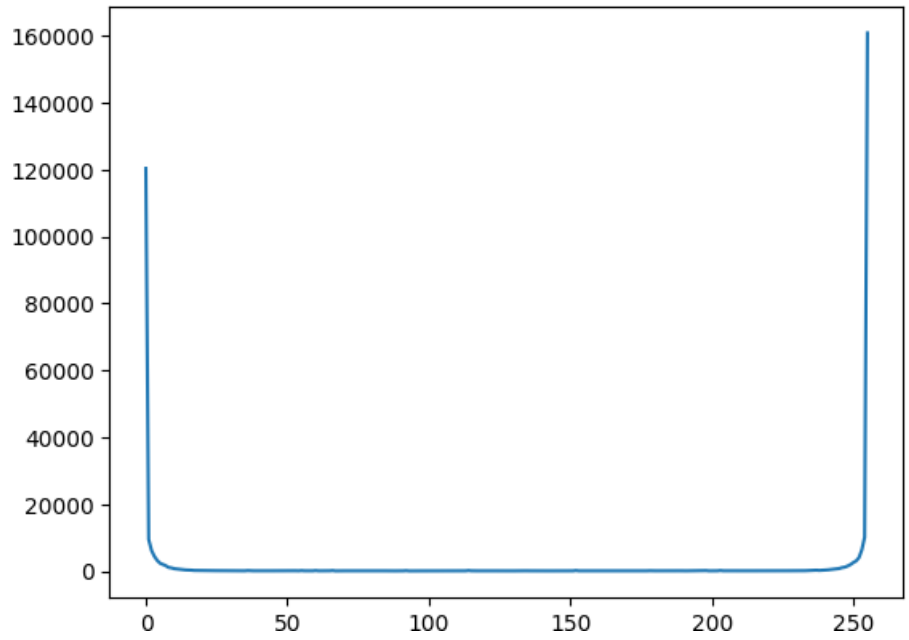

0
50
100
150
200
250
0
50
100
150
200
250
300
350
400

Pratical 5

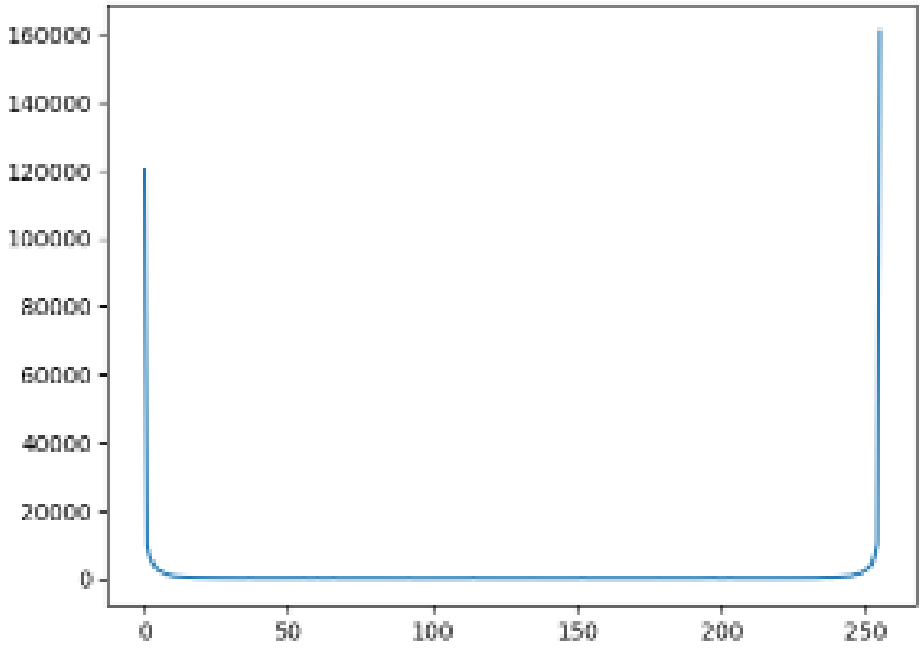
Aim :- The detection of discontinuities – Point, Line and Edge detections, Hough transform, Thresholding, Region based segmentation chain codes

Code:-

```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt

img = cv2.imread('1.jpg')
img_gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
histogram,bin_edges = np.histogram(img_gray,bins=256,range=(0,256))
fig = plt.plot(histogram)
plt.show()
threshold_value = 130

ret,imgt = cv2.threshold(img_gray,threshold_value,255,cv2.THRESH_BINARY)
cv2_imshow(imgt)
```




```
img = cv2.imread('1.jpg')
img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
hist,bin_edges = np.histogram(img,bins=256,range=(0,256))
plt.plot(hist)
plt.show()
```

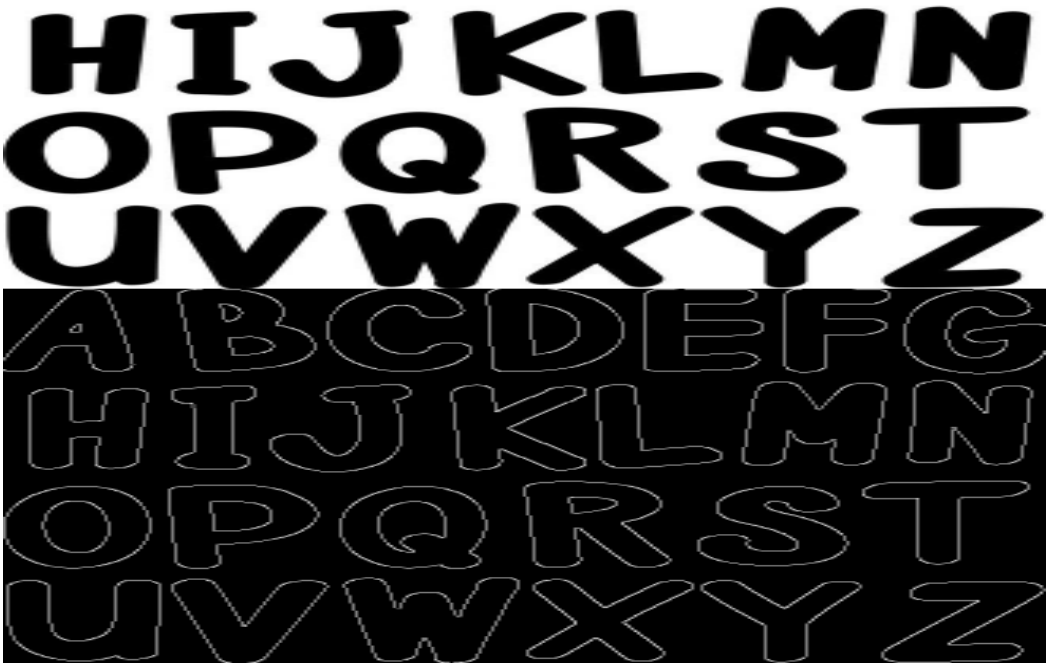

160000
140000
120000
100000
80000
60000
40000
20000
0
0
50
100
150
200
250


A B C D E F G
H I J K L M N
O P Q R S T
U V W X Y Z

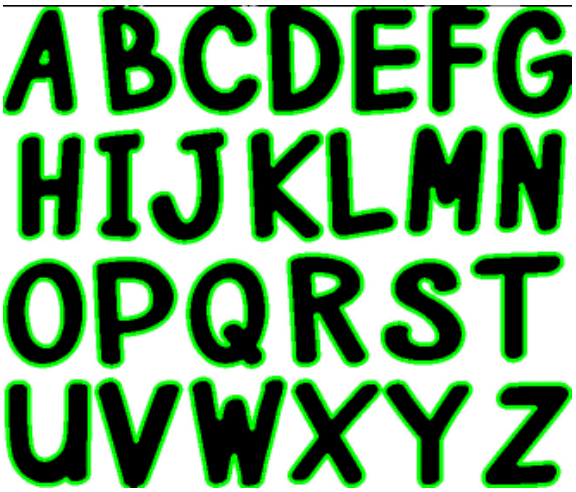
```
threshold2 =
cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_
BINARY,13,5)
cv2_imshow(threshold2)
```


```
import cv2
image = cv2.imread('1.jpg')
#cv2.imshow(image)
image_edges = cv2.Canny(image,100,200)
cv2_imshow(image_edges)
```


```
img = cv2.imread('1.jpg')
#Converting the image into gray-scale
img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
cv2_imshow(img)
#Finding edges of the image
edge_image = cv2.Canny(img,250,200)
cv2_imshow(edge_image)
```



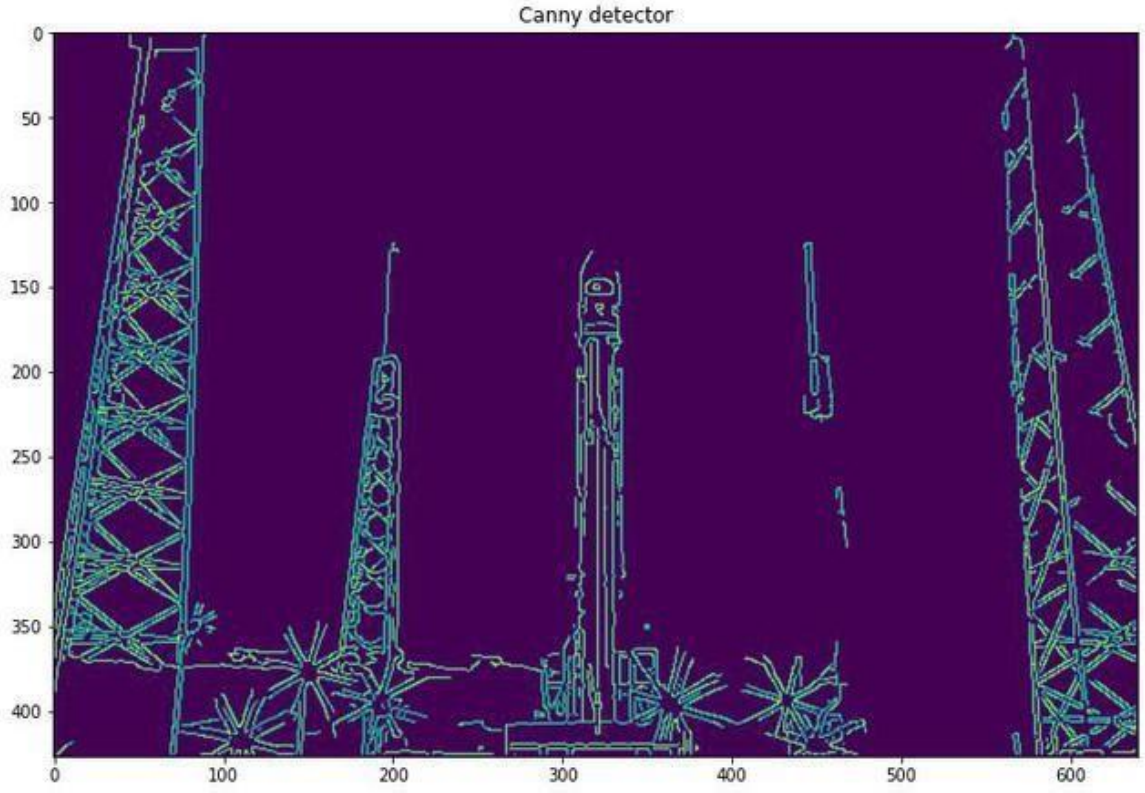
```
#Loading the image
img = cv2.imread('1.jpg')
#Converting the image into gray-scale
img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
cv2_imshow(img)
#Output
edge_image = cv2.Canny(img,250,200)
#showing Edged image
cv2_imshow(edge_image)
# Finding all the lines in an image based on given parameters
contours, hierarchy = cv2.findContours(edge_image,
    cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
#Reverting the original image back to BGR so we can draw in colors
img = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)
#parameter -1 specifies that we want to draw all the contours
cv2.drawContours(img, contours, -1, (0, 255, 0), 3)
cv2_imshow(img)
```

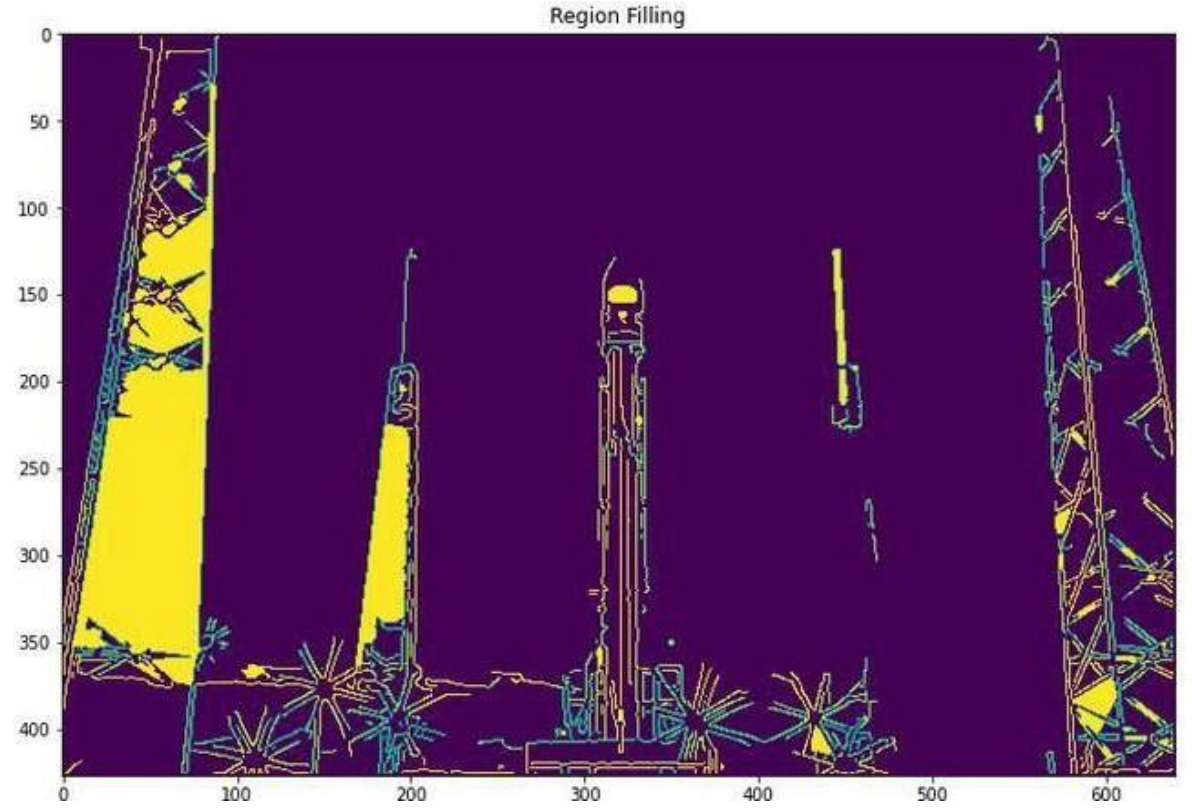


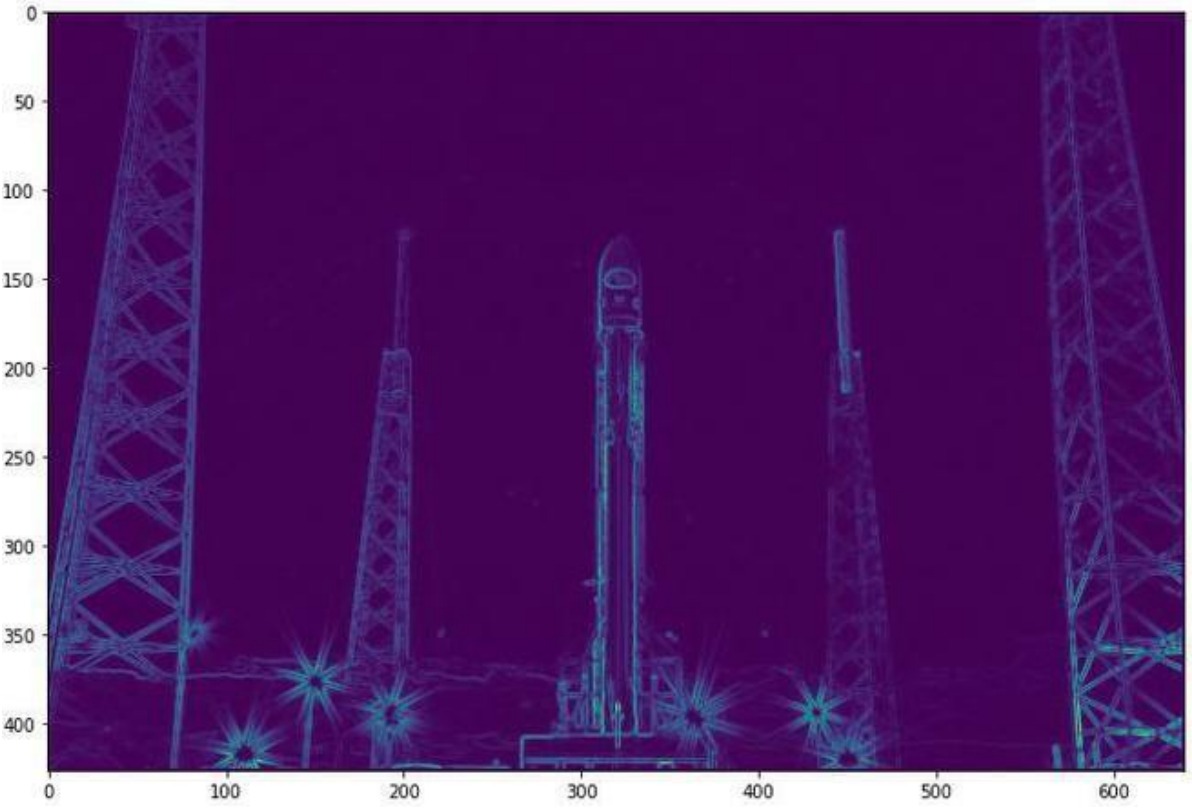

```
# code
import numpy as np
import matplotlib.pyplot as plt
from skimage.feature import canny
from skimage import data,morphology
from skimage.color import rgb2gray
import scipy.ndimage as nd
plt.rcParams["figure.figsize"] = (12,8)
%matplotlib inline

# load images and convert grayscale
rocket = data.rocket()
rocket_wh = rgb2gray(rocket)

# apply edge segmentation
# plot canny edge detection
edges = canny(rocket_wh)
plt.imshow(edges, interpolation='gaussian')
plt.title('Canny detector')

# fill regions to perform edge segmentation
```

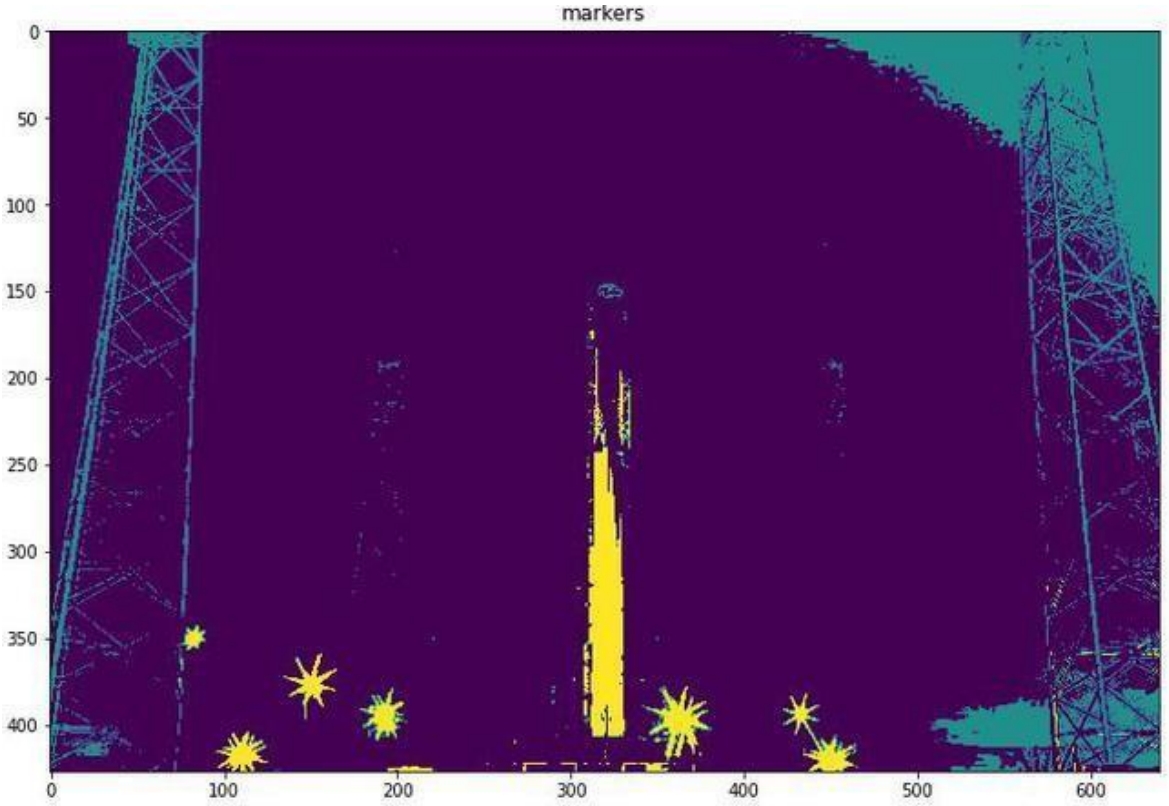
```
fill_im = nd.binary_fill_holes(edges)
plt.imshow(fill_im)
plt.title('Region Filling')

# Region Segmentation
# First we print the elevation map
elevation_map = sobel(rocket_wh)
plt.imshow(elevation_map)

# Since, the contrast difference is not much. Anyways we will perform it
markers = np.zeros_like(rocket_wh)
markers[rocket_wh < 0.1171875] = 1 # 30/255
markers[rocket_wh > 0.5859375] = 2 # 150/255

plt.imshow(markers)
plt.title('markers')

# Perform watershed region segmentation
segmentation = morphology.watershed(elevation_map, markers)

plt.imshow(segmentation)
plt.title('Watershed segmentation')

# plot overlays and contour
segmentation = nd.binary_fill_holes(segmentation - 1)
label_rock, _ = nd.label(segmentation)
# overlay image with different labels
image_label_overlay = label2rgb(label_rock, image=rocket_wh)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(24, 16), sharey=True)
```

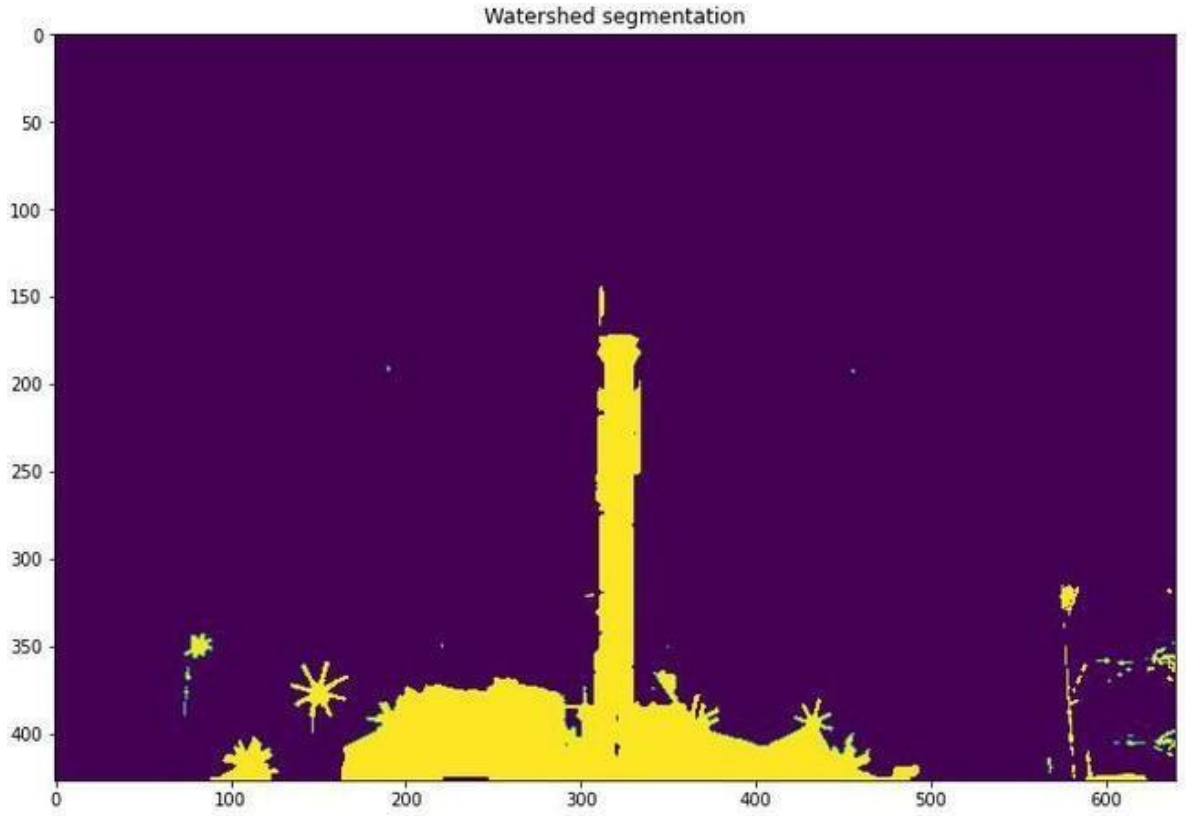
```
ax1.imshow(rocket_wh)
ax1.contour(segmentation, [0.8], linewidths=1.8, colors='w')
ax2.imshow(image_label_overlay)

fig.subplots_adjust(**margins)
```

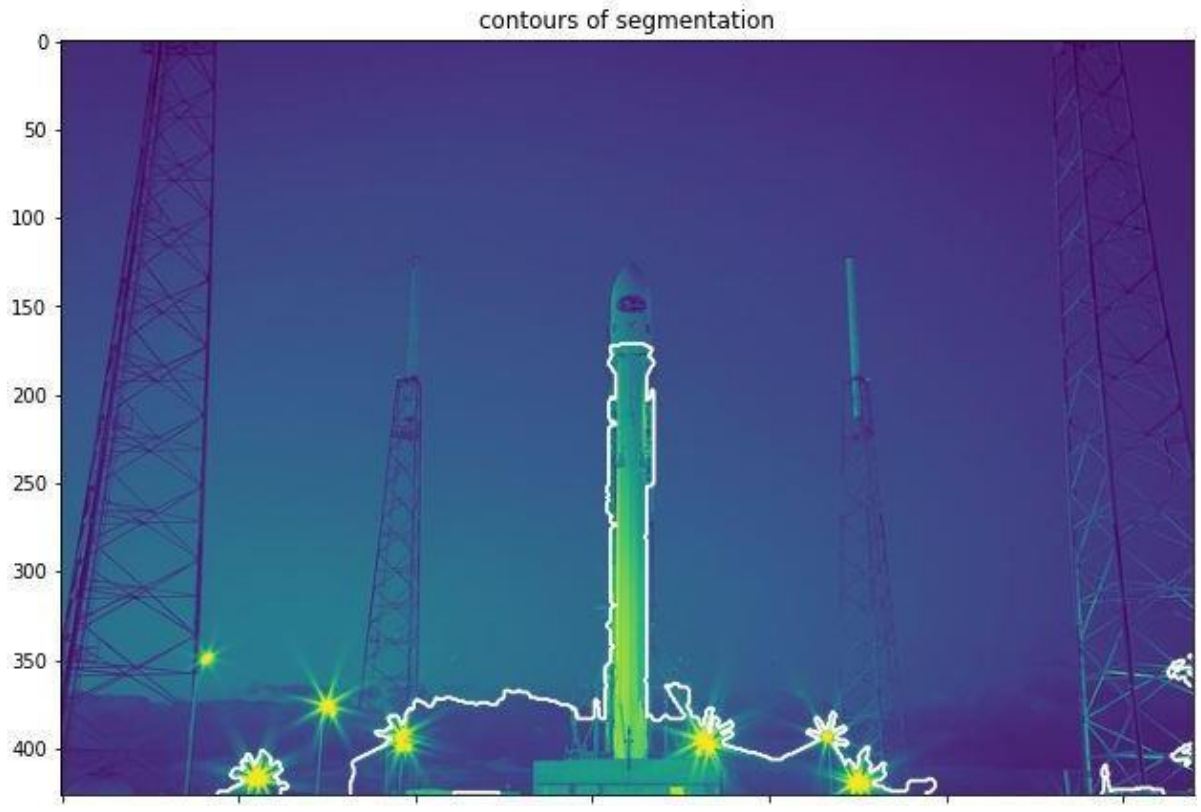


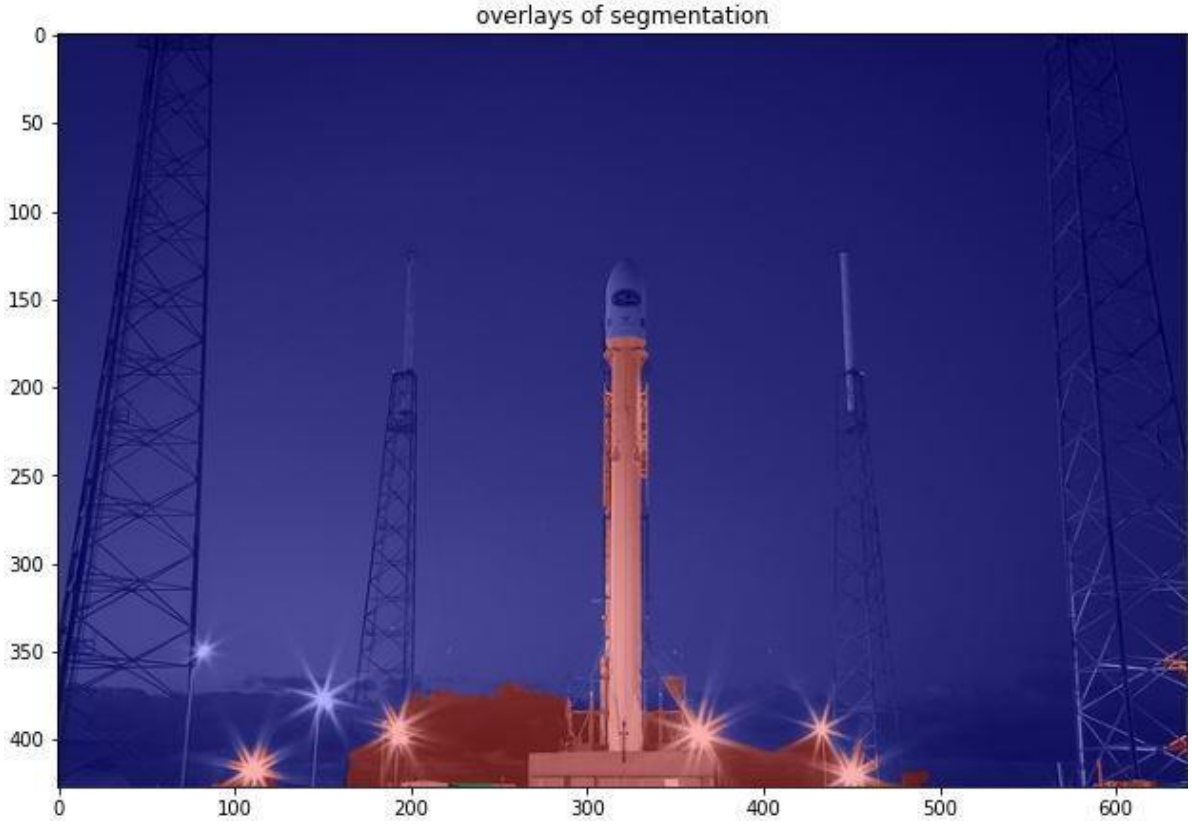

Region Filling
0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600


0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600


markers
0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600


Watershed segmentation
0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600


contours of segmentation
0
50
100
150
200
250
300
350
400


overlays of segmentation
0
50
100
150
200
250
300
350
400
0
100
200
300
400
500
600