Image Processing Lab

Subject Code: MCALE231

A Practical Journal Submitted in Fulfillment of the Degree of

MASTER

IN

COMPUTER APPLICATION

Year 2022-2023

By

(Ravishankar Jaiswal)

(172047)

Semester- 2 Under the Guidance of

Prof. Dnyaneshwar Doere

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

PCP Center
[Satish Pradhan Dyanasadhana College, Thane]

Institute of Distance and Open Learning,

Vidyanagari, Kalina, Santacruz (E) -400098

CERTIFICATE

This to certify that, **Ravishankar Jaiswal** appearing **Master in Computer Application (Semester 2) (172047):** has satisfactory completed the prescribed practical of **MCALE231- Image Processing Lab** as laid down by the University of Mumbai for the academic year 2022-23

Teacher in charge Examiners Coordinator
IDOL, MCA
University of Mumbai

Date: -
Place: -

INDEX

Practical No 1: Programs for image enhancement using spatial domain filters. 4

Program for Average spatial Filter. 4

Practical No 2: To Find DFT/FFT forward and inverse transform of image. 5

Practical No 3: To find DCT forward and inverse transform of image. 6

Practical No 4:Morphological operational: Dilation, Erosion, Opening, Closing. 7

Erosion: 7

Dilation: 7

Opening: 8

Closing: 8

Practical No 5: The detection of discontinuities – Point, Line and Edge detections, Hough transform, Thresholding, Region based segmentation chain codes. 9

Point: 9

Line and Edge detections: 9

Region-Based Segmentation: 10

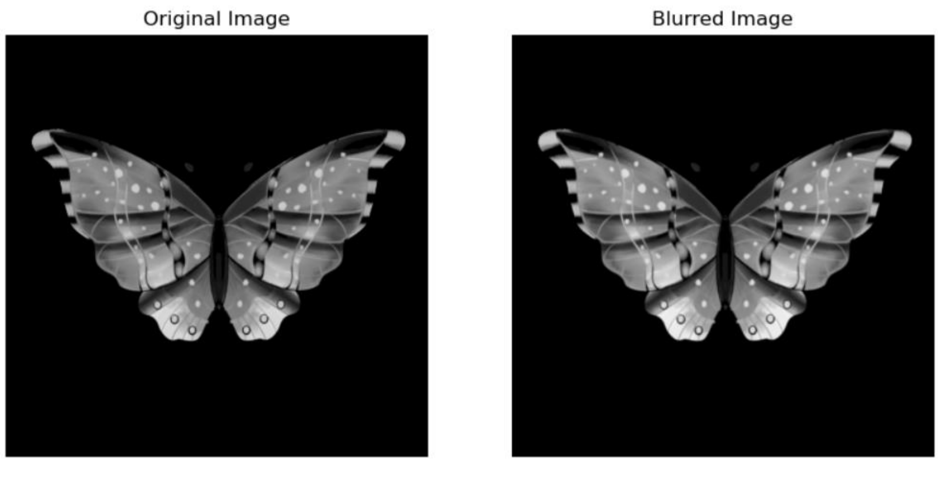
Hough transform: 13

Practical No 1: Programs for image enhancement using spatial domain filters.

Program for Average spatial Filter.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('C:\\Users\\SSING386\\Downloads\\butterfly.png', 0)
if img is not None:
   m, n = img.shape
   mask = np.ones([3, 3], dtype=int)
   mask = mask / 9
   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)
   plt.figure(figsize=(10, 5))
   plt.subplot(1, 2, 1)
   plt.imshow(img, cmap='gray')
   plt.title('Original Image')
   plt.axis('off')
   plt.subplot(1, 2, 2)
   plt.imshow(img_new, cmap='gray')
   plt.title('Blurred Image')
   plt.axis('off')
   plt.show()
else:
   print("Error: Image not loaded.")
```

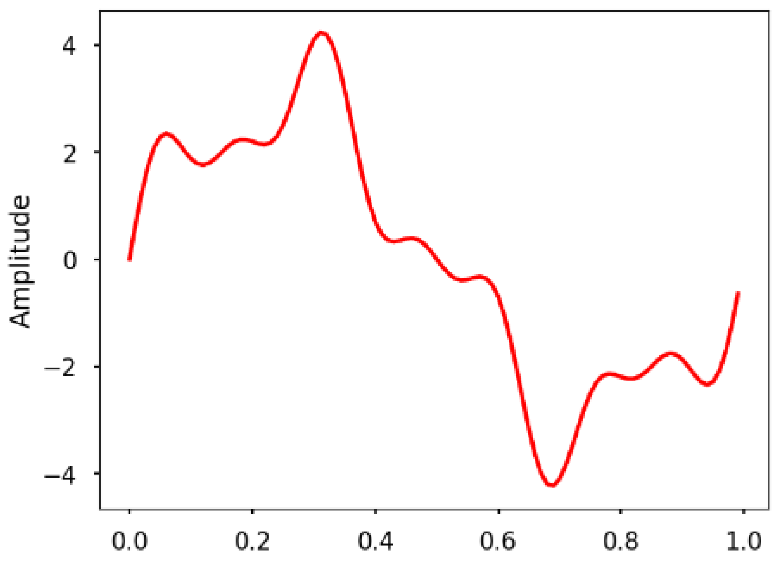
OUTPUT



Practical No 2: To Find DFT/FFT forward and inverse transform of image.

```
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('seaborn-poster')
%matplotlib inline
# sampling rate
sr = 100
# sampling interval
ts = 1.0/sr
t = np.arange(0,1,ts)
freq = 1.
x = 3*np.sin(2*np.pi*freq*t)
freq = 4
x += np.sin(2*np.pi*freq*t)
freq = 7
x += 0.5* np.sin(2*np.pi*freq*t)
plt.figure(figsize = (8, 6))
plt.plot(t, x, 'r')
plt.ylabel('Amplitude')
plt.show()
```

OUTPUT



Practical No 3: To find DCT forward and inverse transform of image.

```
import math
pi = 3.142857
m = 8
n = 8
def dctTransform(matrix):
        dct = []
        for i in range(m):
                dct.append([None for _ in range(n)])
        for i in range(m):
                for j in range(n):
                        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 = 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()
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)
```

OUTPUT:

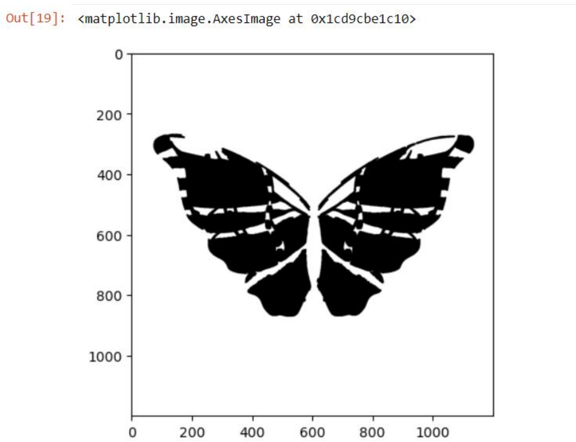
```
2039.9999999999995      -1.1681078033445202     1.1910101606541048      -1.2306043961129725     1.2892204000077006      -1.3705
578971374432    1.480259193374122       -1.626904189710917
-1.1681078033447012     0.000668860706007024    -0.0006819746385176018  0.0007046463715241202   -0.0007382100046555706  0.00078
47840071448786  -0.0008475991738947641  0.000931568372211089
1.1910101606542707      -0.00068197463848918    0.0006953456876459541   -0.0007184619311288998  0.0007526836253646252   -0.0008
00170775129061  0.0008642175194708557   -0.0009498330491961582
-1.2306043961130728     0.0007046463715241202   -0.0007184619311786378  0.0007423466567360038   -0.0007777060254028356  0.00082
67718496846044  -0.0008929477797785523  0.000981409533308053
1.2892204000077108      -0.0007382100047408358  0.0007526836253433089   -0.0007777060253886248  0.0008147496273664956   -0.0008
661525492072997 0.0009354805634451679   -0.0010281559167868437
-1.3705578971374133     0.0007847840071093515   -0.000800170775129061  0.0008267718496810517   -0.0008661525492108524  0.00092
07985046231215  -0.000994500454542191   0.0010930227377894397
1.4802591933741172      -0.0008475991738450261  0.0008642175194708557   -0.000892947779782105   0.0009354805634309571   -0.0009
94500454552849  0.0010741016076369903   -0.0011805097468604586
-1.6269041897109473     0.0009315683721826673   -0.0009498330491766183  0.0009814095332867367   -0.0010281559167744092  0.00109
30227377778934  -0.0011805097468409187  0.0012974594325925182
```

Practical No 4:Morphological operational: Dilation, Erosion, Opening, Closing.

Erosion:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('C:\\Users\\SSING386\\Downloads\\butterfly.png', 0)
binr = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
kernel = np.ones((5, 5), np.uint8)
invert = cv2.bitwise_not(binr)
erosion = cv2.erode(invert, kernel,iterations=1)
plt.imshow(erosion, cmap='gray')
```

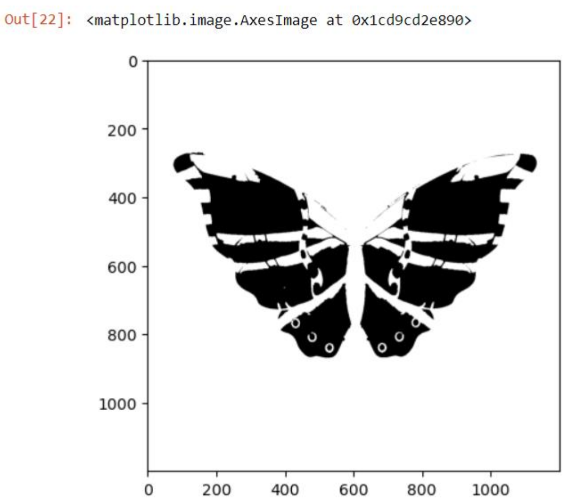
Output:



Dilation:

```
import cv2
img = cv2.imread('C:\\Users\\SSING386\\Downloads\\butterfly.png', 0)
binr = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
kernel = np.ones((3, 3), np.uint8)
invert = cv2.bitwise_not(binr)
dilation = cv2.dilate(invert, kernel, iterations=1)
plt.imshow(dilation, cmap='gray')
```

OUTPUT:

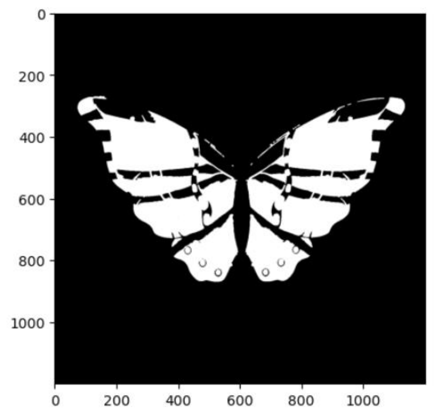


Opening:

```
import cv2
img = cv2.imread('C:\\Users\\SSING386\\Downloads\\butterfly.png', 0)
binr = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
kernel = np.ones((3, 3), np.uint8)
invert = cv2.bitwise_not(binr)
dilation = cv2.dilate(invert, kernel, iterations=1)
plt.imshow(dilation, cmap='gray')
binr = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
kernel = np.ones((3, 3), np.uint8)
opening = cv2.morphologyEx(binr, cv2.MORPH_OPEN, kernel, iterations=1)
plt.imshow(opening, cmap='gray')
```

OUTPUT:

Out[23]: <matplotlib.image.AxesImage at 0x1cd9de16090>

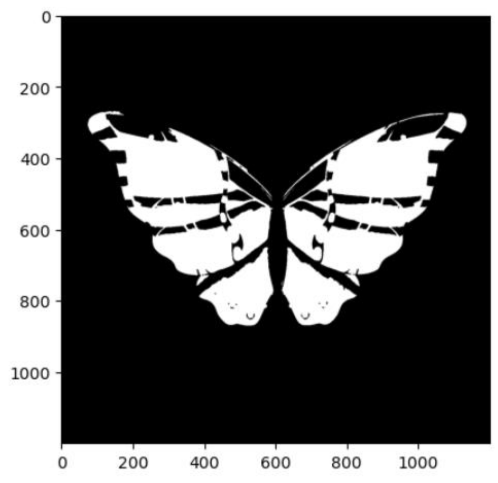


Closing:

```
import cv2
img = cv2.imread('C:\\Users\\SSING386\\Downloads\\butterfly.png', 0)
binr = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
kernel = np.ones((3, 3), np.uint8)
closing = cv2.morphologyEx(binr, cv2.MORPH_CLOSE, kernel, iterations=1)
plt.imshow(closing, cmap='gray')
```

OUTPUT:

Out[24]: <matplotlib.image.AxesImage at 0x1cd9edba490>

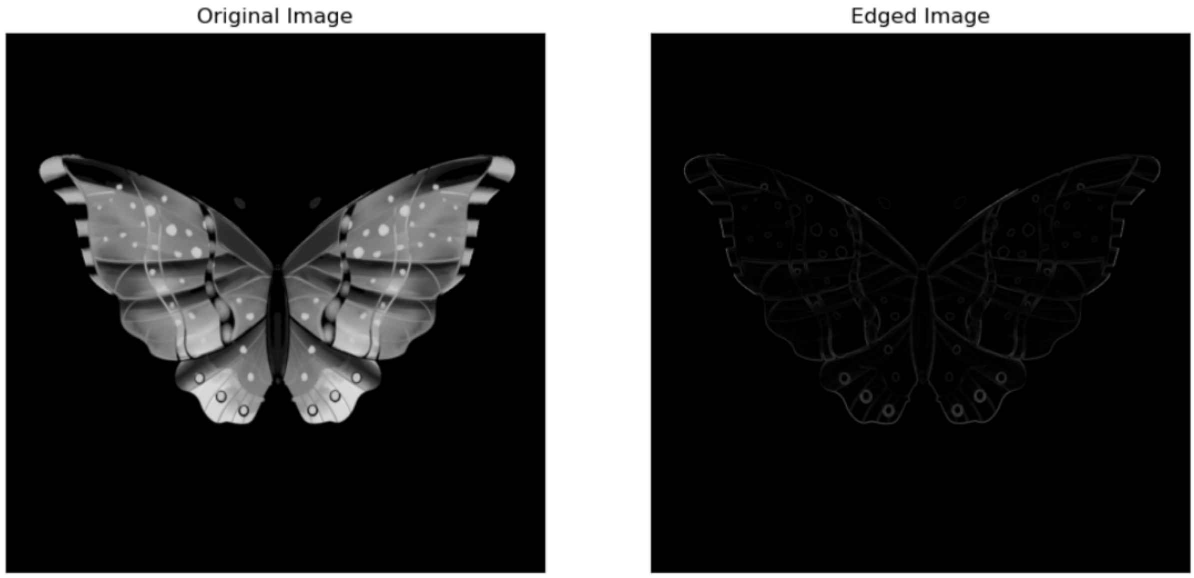


Practical No 5: The detection of discontinuities – Point, Line and Edge detections, Hough transform, Thresholding, Region based segmentation chain codes.

Point:

```
import cv2
import numpy as np
from scipy import ndimage
import matplotlib.pyplot as plt
roberts_cross_v = np.array([[1, 0],[0, -1]])
roberts_cross_h = np.array([[0, 1],[-1, 0]])
img = cv2.imread("C:\\Users\\SSING386\\Downloads\\butterfly.png", 0).astype('float64')
img /= 255.0
vertical = ndimage.convolve(img, roberts_cross_v)
horizontal = ndimage.convolve(img, roberts_cross_h)
edged_img = np.sqrt(np.square(horizontal) + np.square(vertical))
edged_img *= 255
cv2.imwrite("output.jpg", edged_img)
plt.subplot(121), plt.imshow(img, cmap='gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(255), plt.imshow(edged_img, cmap='gray')
plt.title('Edged Image'), plt.xticks([]), plt.yticks([])
plt.show()
```

Output:



Line and Edge detections:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
I = cv2.imread('C:\\Users\\SSING386\\Downloads\\butterfly.png', 0).astype(float)
In = I.copy()
mask1 = np.array([[1, 0, -1], [1, 0, -1], [1, 0, -1]])
mask2 = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]])
mask3 = np.array([[0, -1, -1], [1, 0, -1], [1, 1, 0]])
mask4 = np.array([[1, 1, 0], [1, 0, -1], [0, -1, -1]])
```

```
for i in range(1, I.shape[0]-1):
    for j in range(1, I.shape[1]-1):
        neighbour_matrix1 = mask1 * In[i-1:i+2, j-1:j+2]
        avg_value1 = np.sum(neighbour_matrix1)
        neighbour_matrix2 = mask2 * In[i-1:i+2, j-1:j+2]
        avg_value2 = np.sum(neighbour_matrix2)
        neighbour_matrix3 = mask3 * In[i-1:i+2, j-1:j+2]
        avg_value3 = np.sum(neighbour_matrix3)
        neighbour_matrix4 = mask4 * In[i-1:i+2, j-1:j+2]
        avg_value4 = np.sum(neighbour_matrix4)
        I[i, j] = max([avg_value1, avg_value2, avg_value3, avg_value4])
plt.imshow(I, cmap='gray')
plt.show()
```

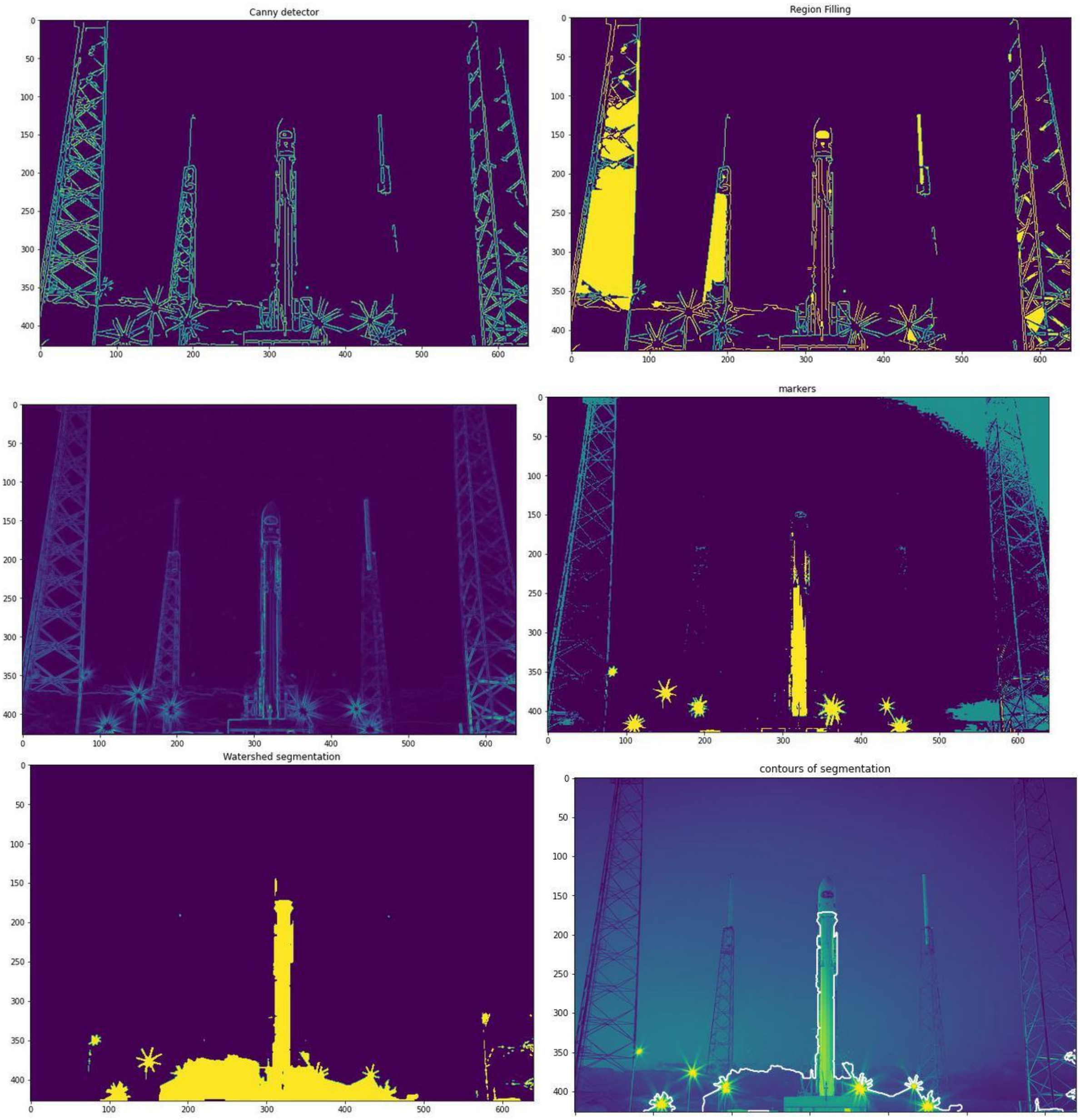
OUTPUT:

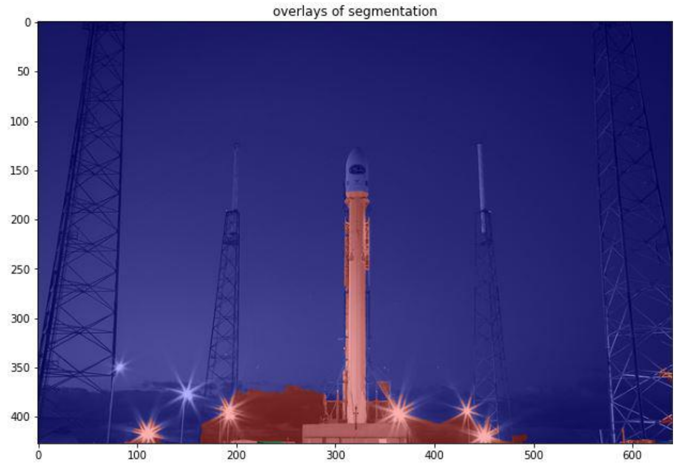
Region-Based Segmentation:

```
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
rocket = data.rocket()
rocket_wh = rgb2gray(rocket)
edges = canny(rocket_wh)
plt.imshow(edges, interpolation='gaussian')
plt.title('Canny detector')
fill_im = nd.binary_fill_holes(edges)
plt.imshow(fill_im)
plt.title('Region Filling')
elevation_map = sobel(rocket_wh)
plt.imshow(elevation_map)
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')
segmentation = morphology.watershed(elevation_map, markers)
plt.imshow(segmentation)
plt.title('Watershed segmentation')
segmentation = nd.binary_fill_holes(segmentation - 1)
label_rock, _ = nd.label(segmentation)
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)
```

Output:

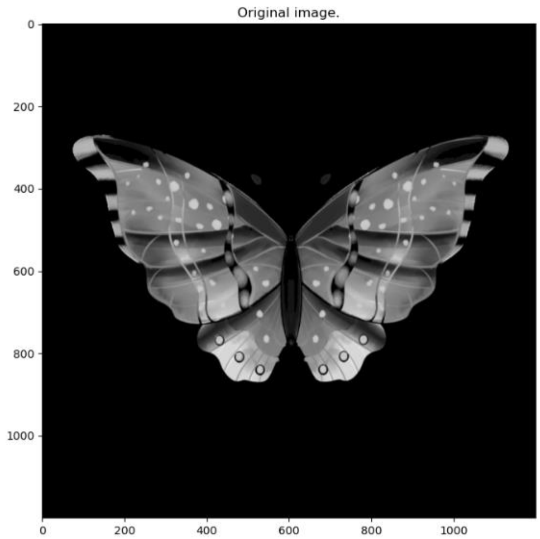


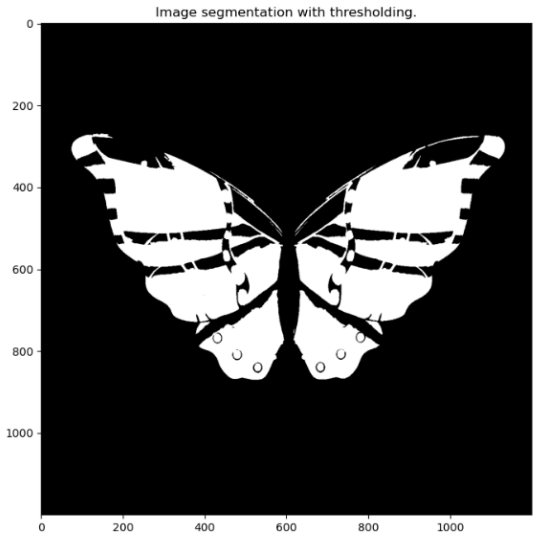


Thresholding:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread("C:\\Users\\SSING386\\Downloads\\butterfly.png", 0)
plt.figure(1)
plt.imshow(image, cmap='gray')
plt.title("Original image.")
plt.show()
counts, x = np.histogram(image.flatten(), bins=16)
thresh = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]
plt.figure(2)
plt.imshow(thresh, cmap='gray')
plt.title("Image segmentation with thresholding.")
plt.show()
```

Output:





Hough transform:

```
import cv2
import numpy as np
image = cv2.imread('C:\\Users\\SSING386\\Downloads\\butterfly.png')
gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,50,150,apertureSize=3)
lines_list =[]
lines = cv2.HoughLinesP(edges, 1, np.pi/180, threshold=100, minLineLength=5,maxLineGap=10 )
for points in lines:
        x1,y1,x2,y2=points[0]
        cv2.line(image,(x1,y1),(x2,y2),(0,255,0),2)
        lines_list.append([(x1,y1),(x2,y2)])
cv2.imwrite('detectedLines.png',image)
```

Output:

```
Out[5]: True
```