DHAYANIDHI GUNASEKRAN

UBIT: dhayanid
Person Number: 50290938
Email: dhayanid@buffalo.edu

Morphology and Segmentation

Dhayanidhi Gunasekaran
University at Buffalo
dhayanid@buffalo.edu

Abstract

To perform the Image Morphology, Segmentation and Hough transform for the given set of input images.

1 Image Morphology

To remove the noise using two morphology image processing algorithms and to extract the boundary of the noise reduced image.

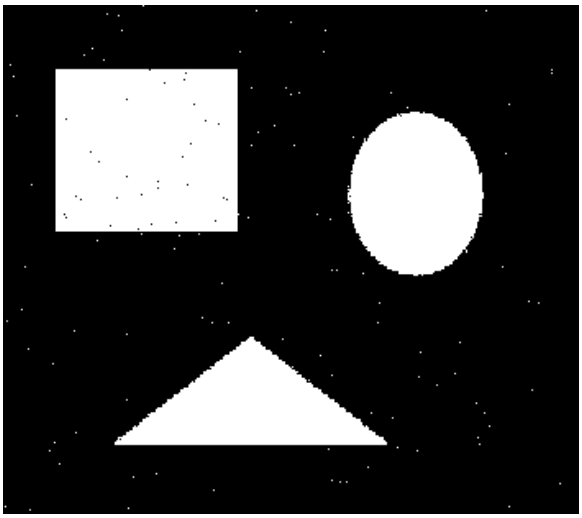

Figure 1: Input Image for Morphology

1.1 Task Objective

To reduce the noise using the image morphology algorithm and to extract the boundary.

1. Opening + Closing
2. Closing + Opening

Opening

$$A \circ B = (A \ominus B) \oplus B$$

Closing

$$A \bullet B = (A \oplus B) \ominus B$$

Dilation

$$A \oplus B = \{z \mid (\hat{B})_z \cap A \neq \varnothing\}$$

Erosion

$$A \ominus B = \{z \mid (B)_z \subseteq A\}$$

Boundary Extraction

$$\beta(A) = A - (A \ominus B)$$

Kernel

[[0 1 0]

[1 1 1]

[0 1 0]]

1.2 Procedure

The implementation is done using python, opencv and numpy libraries.

a) Images are read using imread function.
b) Dilation and Erosion function are implemented.
c) Noise reduction algorithm mentioned above are implemented
d) Boundaries are extracted using the boundary detection algorithm mentioned.
e) Output image is saved.

1.3 Code

```
import cv2
import numpy as np

# funtion to apply eclipse kernel for an image
def multiply_kernel(matrix, row, col):
   matrix[row,col] = 255
   if row-1 >=0:
      matrix[row-1, col]=255

   if row+1 < matrix.shape[0] :
      matrix[row+1,col] = 255

   if col-1 >= 0:
      matrix[row,col-1] = 255

   if col+1 < matrix.shape[1]:
      matrix[row,col+1] = 255

   return matrix
```

```
def dilation(image,kernel):
    row = image.shape[0]
    col = image.shape[1]
    output = np.copy(image)
    for i in range(row):
        for j in range(col):
            if image[i, j] == 255:
                output = multiply_kernel(output,i,j)
    return output

img = cv2.imread('original_imgs/noise.jpg',0)

#kernel = np.ones((3,3),np.uint8)
kernel = [[0,1,0],[1,1,1],[0,1,0]]
kernel =np.array(kernel)

#dilation = dilation(img,kernel)

def invertImage(img):
    row = img.shape[0]
    col = img.shape[1]
    output = np.zeros((row,col))
    for i in range(row):
        for j in range(col):
            if img[i,j] == 255:
                output[i,j] = 0
            elif img[i,j] == 0:
                output[i,j] = 255

    return output

def erosion(img,kernel):
    invertedimg =  invertImage(img)
    dilated = dilation(invertedimg,kernel)
    output = invertImage(dilated)
    return output


eroded_img = erosion(img,kernel)
cv2.imwrite("erosion_self.jpg",eroded_img)

def opening(img,kernel):
    return dilation(erosion(img,kernel),kernel)

def closing(img,kernel):
    return erosion(dilation(img,kernel),kernel)

def boundary(img,kernel):
    return img - erosion(img,kernel)

res_noise1 = opening(img,kernel)
res_noise1 = closing(res_noise1,kernel)
cv2.imwrite("output/res_noise1.jpg",res_noise1)

res_noise2 = closing(img,kernel)
res_noise2 = opening(res_noise2,kernel)
cv2.imwrite("output/res_noise2.jpg",res_noise2)
```

```
bound1 = boundary(res_noise1,kernel)
cv2.imwrite("output/res_bound1.jpg",bound1)

bound2 = boundary(res_noise2,kernel)
cv2.imwrite("output/res_bound2.jpg",bound2)
```

1.4 Output Images

1.4.1 Noise reduced Image

Figure 2: Noise reduced images

1.4.2 Comparing two noise reduced images

When comparing the two noise reduced images there are 99 pixels which differ from each other.

1.4.3 Boundary Extraction

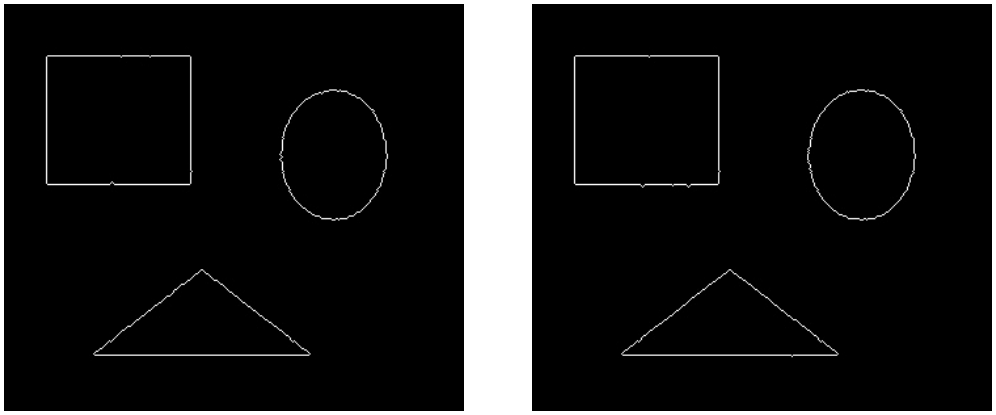

Figure 3: Boundary images

2 Image Segmentation

Image segmentation is a technique which masks the input image with a kernel and a threshold is determined by plotting the histogram. Based on the threshold, the point or segment is detected.

2.1 Task Objective

To mask the input image with a kernel and threshold is determined to separate the point or segment.

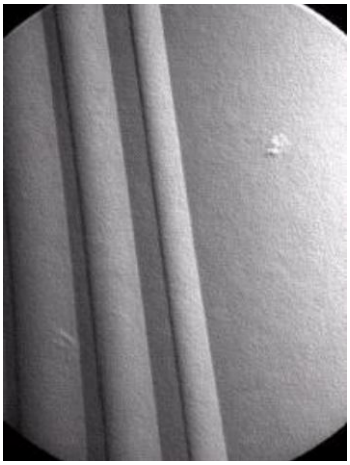

Figure 4: Input image for point detection

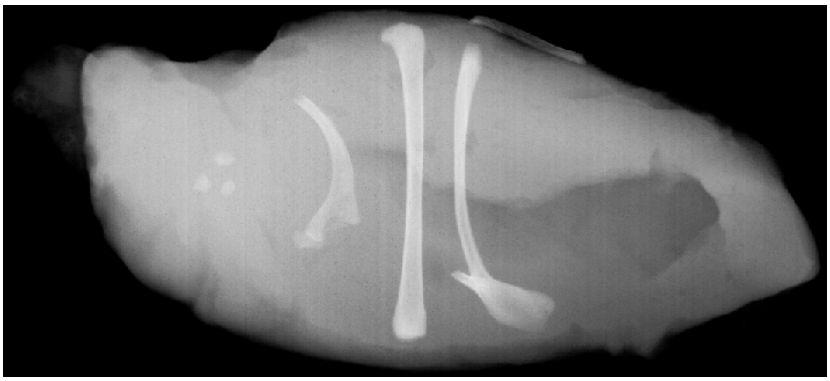

Figure 5: Input image for segmentation

Kernel Used

[[-1 -1 -1]

[-1 8 -1]

[-1 -1 -1]]

2.2 Code

```
# coding: utf-8
# In[1]:
import cv2
import numpy as np
```

```
from matplotlib import pyplot as plt
from collections import Counter

#this function returns the max value for the given matrix
def get_Max(matrix):
    largest_num = matrix[0][0]
    for row_idx, row in enumerate(matrix):
        for col_idx, num in enumerate(row):
            if num > largest_num:
                largest_num = num

    return largest_num

#This function normalises the image matrix to 0-255 scale
def Normalise_Matrix(Matrix):
    row = len(Matrix)
    col = len(Matrix[0])
    MAX_VALUE = get_Max(Matrix)
    for i in range(row):
        for j in range(col):
            Matrix[i][j] = (Matrix[i][j]/MAX_VALUE)*255
    return Matrix

#This function generates a matrix with given row,col size filled with zeros
def initialise_matrix(row, col):
    matrix = [[0 for x in range(col)] for y in range(row)]
    return matrix

# funtion to get 3*3 matrix based on its position
def get_3_cross3(matrix, row, col):
    MAT = initialise_matrix(3, 3)
    if row == 0 or col == 0:
        MAT[0][0] = 0
    else:
        MAT[0][0] = matrix[row-1][col-1]

    if row == 0:
        MAT[0][1] = 0
    else:
        MAT[0][1] = matrix[row-1][col]

    if row == 0 or col == len(matrix[0])-1:
        MAT[0][2] = 0
    else:
        MAT[0][2] = matrix[row-1][col+1]

    if col == 0:
        MAT[1][0] = 0
    else:
        MAT[1][0] = matrix[row][col-1]

    MAT[1][1] = matrix[row][col]

    if col == len(matrix[0])-1:
        MAT[1][2] = 0
    else:
        MAT[1][2] = matrix[row][col+1]
```

```
    if row == len(matrix)-1 or col == 0:
        MAT[2][0] = 0
    else:
        MAT[2][0] = matrix[row+1][col-1]

    if row == len(matrix)-1:
        MAT[2][1] = 0
    else:
        MAT[2][1] = matrix[row+1][col]

    if row == len(matrix)-1 or col == len(matrix[0])-1:
        MAT[2][2] = 0
    else:
        MAT[2][2] = matrix[row+1][col+1]

    return MAT

#this function pads 0 for the edge rows and cols
def generatePatchMatrix(matrix, row, col):
    PATCH_MAT = initialise_matrix(5, 5)
    row_i = row - 2
    for i in range(5):
        col_i = col - 2
        for j in range(5):
            if row_i < 0 or row_i > len(matrix)-1 or col_i < 0 or col_i > len(matrix[0])-1:
                PATCH_MAT[i][j] = 0
            else:
                PATCH_MAT[i][j] = matrix[row_i][col_i]
            col_i = col_i + 1

        if i > 4:
            row_i = row - 2
        else:
            row_i = row_i + 1
    return PATCH_MAT


# this function does element wise multiplication of the given 2 matrices
def elem_wise_operation(kernel, pos,size):
    op = initialise_matrix(size, size)
    op = np.multiply(kernel,pos)
    return op

# this function returns the sum of all the values in a matrix
def sum_of_elems(MAT):
    value = 0
    row = len(MAT)
    col = len(MAT[0])
    for i in range(row):
        for j in range(col):
            value = value + MAT[i][j]
    return value

def mask_Input_Image(image):
    op_mat = initialise_matrix(len(image), len(image[0]))
    for i in range(len(image)):
        for j in range(len(image[0])):
            pos_mat = get_3_cross3(image, i, j)
```

```
            computed_mat = np.multiply(pos_mat, kernel)
            op_mat[i][j] = sum_of_elems(computed_mat)
    return op_mat

# In[290]:

#Color_image = cv2.imread('original_imgs/point.jpg')
img = cv2.imread('original_imgs/turbine-blade.jpg', cv2.IMREAD_GRAYSCALE)
#img = cv2.imread('original_imgs/point.jpg',0)
kernel = [[-1, -1, -1],[-1, 8, -1],[-1, -1, -1]]
kernel = np.array(kernel)
print(kernel)
print(img.shape)

mask_img = mask_Input_Image(img)

cv2.imwrite("masked_image.jpg", np.asarray(mask_img))
print("mask image generated")
histogram_img = np.copy(Normalise_Matrix(mask_img))
arr=[]
for i in range(histogram_img.shape[0]):
    for j in range(histogram_img.shape[1]):
        if histogram_img[i,j]!=0:
            arr.append(histogram_img[i,j])

C = Counter(arr)
x, y = list(C.keys()), list(C.values())
plt.bar(x, y)
plt.show()
#plt.hist(np.asarray(mask_img).ravel(),256,[0,256])
#plt.show()

# In[291]:

Color_image = cv2.imread('original_imgs/turbine-blade.jpg')
output_i = np.copy(histogram_img) * 0.
for i in range(histogram_img.shape[0]):
    for j in range(histogram_img.shape[1]):
        if np.abs(histogram_img[i,j]) > 254 :
            output_i[i,j] = 255.
            print(str(i)+","+str(j))

cv2.circle(Color_image, (445,249), 15, (0,255,0), 3)
cv2.imwrite("output/point_detect.jpg",Color_image)

# In[287]:

original_seg = cv2.imread('original_imgs/segment.jpg')
seg_img = cv2.imread('original_imgs/segment.jpg', cv2.IMREAD_GRAYSCALE)
hist_img = np.copy(seg_img)
array=[]
for i in range(hist_img.shape[0]):
    for j in range(hist_img.shape[1]):
        if hist_img[i,j]!=0:
            array.append(hist_img[i,j])

ounter = Counter(array)
x, y = list(ounter.keys()), list(ounter.values())
```

```
plt.bar(x, y)
plt.show()


# In[288]:

output_img = np.copy(seg_img) * 1.

for i in range(seg_img.shape[0]):
   for j in range(seg_img.shape[1]):
      if seg_img[i,j] < 203: #170.0 and seg_img[i,j]<190.0:
         output_img[i,j] = 0.


# In[292]:

cv2.rectangle(original_seg,(160,125),(210,168),(0,255,0),2)
cv2.rectangle(original_seg,(250,210),(305,76),(0,255,0),2)
cv2.rectangle(original_seg,(330,285),(363,20),(0,255,0),2)
cv2.rectangle(original_seg,(388,255),(425,38),(0,255,0),2)
cv2.imwrite("output/segment_detect.jpg",original_seg)
```

2.3 Output

2.3.1 Histograms

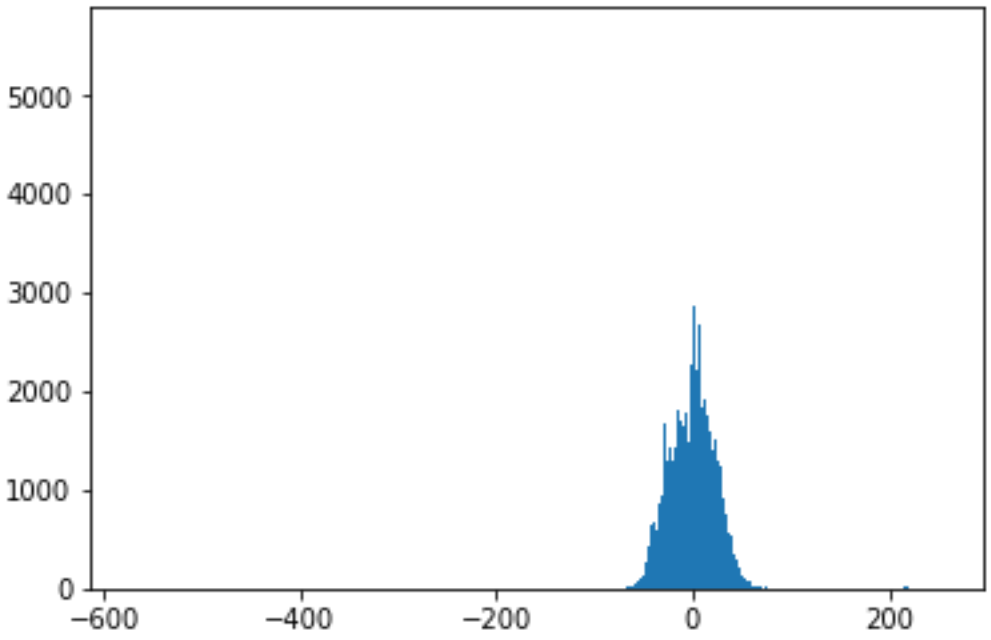

Figure 6: Histogram for Point Detection

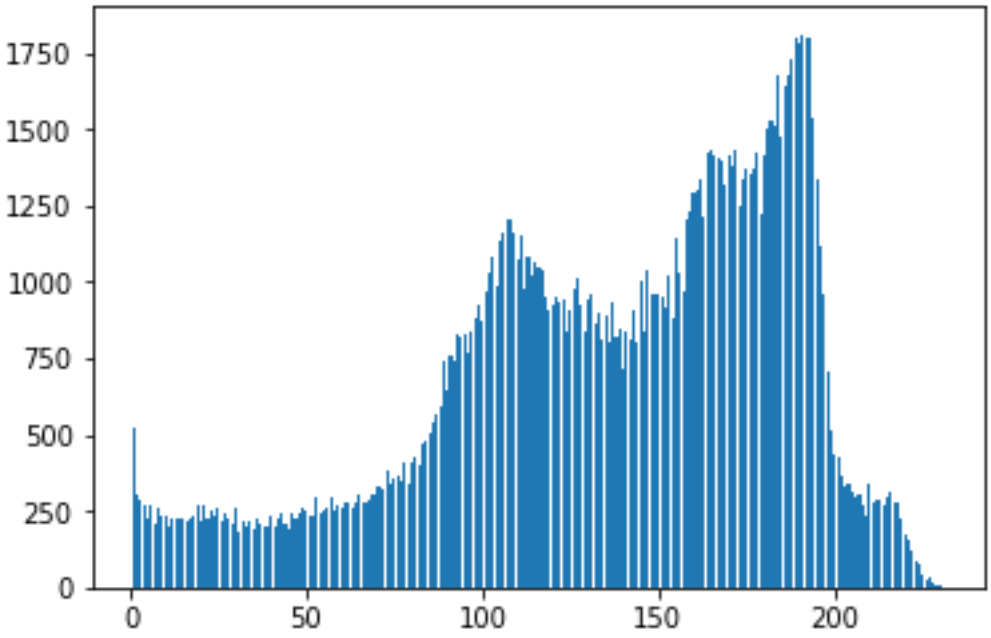

Figure 7: Histogram for Segment detection

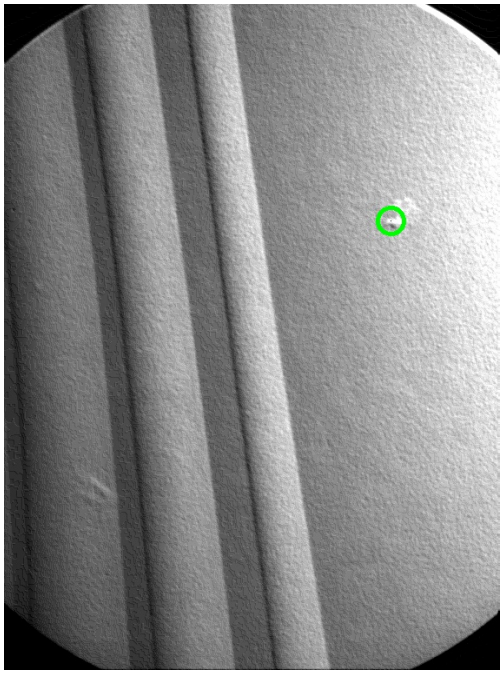
Figure 8: Point detection output

Coordinates of the detected point is (445,249)

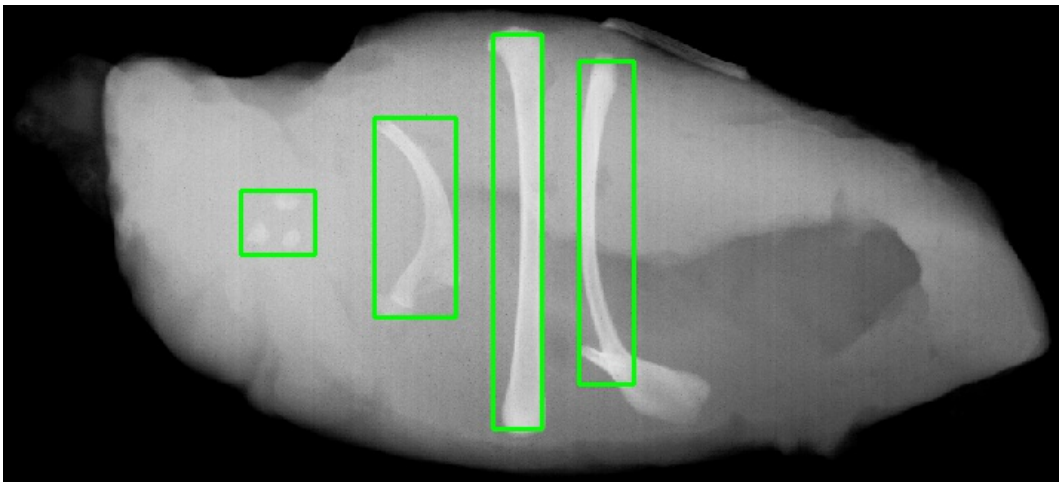

Figure 9: Segment detection output

The coordinates of the bounding boxes from left to right are as follows.

1. (125,160), (125,210), (168,160), (168,210)
2. (76,250), (76,305), (210,250), (210,305)
3. (20,330), (20,363), (285,330), (285,363)
4. (38,388), (38,425), (255,388), (255,425)

3 Hough Transform

3.1 Task Objective

Use Hough transform to detect the lines in the given image.

To get a red lines, blue lines and coins separately using Hough Transform.

1. Convert the image to HSV format and filter the respective lines using the color range.
2. Use the color filtered image and apply edge detection to get the edges.
3. Hough transform is applied for the edge detected image and the output is determined.
4. The same procedure is applied for Blue line and coins as well.

3.2 Code

```
# coding: utf-8

# In[1]:
import cv2
import numpy as np
from matplotlib import pyplot as plt

# In[2]:

def get_Max(matrix):
    largest_num = matrix[0][0]
    for row_idx, row in enumerate(matrix):
        for col_idx, num in enumerate(row):
            if num > largest_num:
                largest_num = num

    return largest_num
```

```
def Normalise_Matrix(Matrix):
    row = len(Matrix)
    col = len(Matrix[0])
    MAX_VALUE = get_Max(Matrix)
    for i in range(row):
        for j in range(col):
            Matrix[i][j] = (Matrix[i][j]/MAX_VALUE)*255

    return Matrix


def initialise_matrix(row, col):
    matrix = [[0 for x in range(col)] for y in range(row)]
    return matrix


def elem_wise_multiple(MAT_A, MAT_B, row, col):
    MAT = initialise_matrix(3, 3)
    for i in range(row):
        for j in range(col):
            MAT[i][j] = MAT_A[i][j] * MAT_B[i][j]
    return MAT

def sum_of_elems(MAT):
    value = 0
    row = len(MAT)
    col = len(MAT[0])
    for i in range(row):
        for j in range(col):
            value = value + MAT[i][j]

    return value

# funtion to get 3*3 matrix based on its position
def get_3_cross3(matrix, row, col):
    MAT = initialise_matrix(3, 3)
    if row == 0 or col == 0:
        MAT[0][0] = 0
    else:
        MAT[0][0] = matrix[row-1][col-1]

    if row == 0:
        MAT[0][1] = 0
    else:
        MAT[0][1] = matrix[row-1][col]

    if row == 0 or col == len(matrix[0])-1:
        MAT[0][2] = 0
    else:
        MAT[0][2] = matrix[row-1][col+1]

    if col == 0:
        MAT[1][0] = 0
    else:
        MAT[1][0] = matrix[row][col-1]

    MAT[1][1] = matrix[row][col]
```

```
    if col == len(matrix[0])-1:
        MAT[1][2] = 0
    else:
        MAT[1][2] = matrix[row][col+1]

    if row == len(matrix)-1 or col == 0:
        MAT[2][0] = 0
    else:
        MAT[2][0] = matrix[row+1][col-1]

    if row == len(matrix)-1:
        MAT[2][1] = 0
    else:
        MAT[2][1] = matrix[row+1][col]

    if row == len(matrix)-1 or col == len(matrix[0])-1:
        MAT[2][2] = 0
    else:
        MAT[2][2] = matrix[row+1][col+1]

    return MAT


# In[3]:


def detectEdges(image):

    INPUT_IMAGE = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

    # cv.namedWindow("Input Image")
    # cv.imshow('Image',inputim)
    # cv.waitKey(0)
    print(INPUT_IMAGE.shape)


    # Initialise the Gx and  Gy matrix
    Gx = [[1, 0, -1],
    [2, 0, -2],
    [1, 0, -1]]
    Gy = [[1, 2, 1],
    [0, 0, 0],
    [-1, -2, -1]]

    rows = len(INPUT_IMAGE)
    cols = len(INPUT_IMAGE[0])
    INPUTXEDGE = initialise_matrix(rows, cols)
    NUM_MAT = initialise_matrix(rows, cols)
    INPUTYEDGE = initialise_matrix(rows, cols)

    # loop through the matrices and calculate Gx * Input
    for i in range(rows):
        for j in range(cols):
            THREE_CROSS_THREE = get_3_cross3(INPUT_IMAGE, i, j)
            OUTPUT = elem_wise_multiple(Gx, THREE_CROSS_THREE, 3, 3)
            INPUTXEDGE[i][j] = sum_of_elems(OUTPUT)
```

```
    for i in range(rows):
        for j in range(cols):
            THREE_CROSS_THREE = get_3_cross3(INPUT_IMAGE, i, j)
            OUTPUT = elem_wise_multiple(Gy, THREE_CROSS_THREE, 3, 3)
            INPUTYEDGE[i][j] = sum_of_elems(OUTPUT)

    OP = initialise_matrix(rows, cols)

    for i in range(rows):
        for j in range(cols):
            OP[i][j] = (((INPUTXEDGE[i][j]**2)+(INPUTYEDGE[i][j]**2))**0.5)

    INPUTXEDGE = Normalise_Matrix(INPUTXEDGE)
    INPUTYEDGE = Normalise_Matrix(INPUTYEDGE)
    OP = Normalise_Matrix(OP)

    INPUTXEDGE = np.asarray(INPUTXEDGE)
    INPUTYEDGE = np.asarray(INPUTYEDGE)
    OP = np.asarray(OP)

    print("edge detected")
    return OP


# In[4]:


def hough_line(img):
     # Rho and Theta ranges
    thetas = np.deg2rad(np.arange(-90.0, 90.0))
    width, height = img.shape
    diag_len = np.ceil(np.sqrt(img.shape[0] * img.shape[0] + img.shape[1] * img.shape[1]))   # max_dist
    # Hough accumulator array of theta vs rho
    accumulator = np.zeros((2 * int(diag_len), len(thetas)))
    rhos=[]
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if img[i,j]!= 0.:
                for x in range(len(thetas)):
                    # Calculate rho. diag_len is added for a positive index
                    rho = round(i * np.cos(thetas[x]) + j * np.sin(thetas[x]))+diag_len
                    rhos.append(rho)
                    accumulator[int(rho), int(x)] += 1

    return accumulator, thetas, rhos


# In[75]:


def minimise_no_rhos(rho,theta):
    rho_op =[]
    theta_op = []
    count= 0
    array = []
    for x in rho:
        count+=1
        if len(rho_op) == 0:
```

```
            rho_op.append(x)
        else:
            if (x - rho[count-1]) < 10:
                array.append(x)
            else:
                rho_op.append(np.median(array))
                theta_op.append(theta[count-1])
                array = []

    return rho_op,theta_op


# In[6]:


img = cv2.imread('original_imgs/hough.jpg')
hsv_img = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
lower_red = np.array([170,110,110])
upper_red = np.array([180,150,150])
mask1 = cv2.inRange(hsv_img, lower_red, upper_red)

output_hsv = hsv_img.copy()
output_hsv[np.where(mask1==0)] = 0

cv2.imwrite("onlyred.jpg",output_hsv)


# In[7]:


edges = detectEdges(output_hsv)
cv2.imwrite("red_edge.jpg",edges)


# In[76]:


accumulators,thetas,rhos = hough_line(edges)
print(accumulators.shape)
# code to get the indices which are above the range
idx = np.argmax(accumulators)
print(idx)
print(accumulators[int(idx/accumulators.shape[1]),int(idx%accumulators.shape[1])])
mask = [accumulators > 150.] [0] * 1.
accum = accumulators * mask
rho = []
theta = []
for i in range(mask.shape[0]):
    for j in range(mask.shape[1]):
        if mask[i,j]==1:
            rho.append(i)
            theta.append(thetas[j])


# In[77]:


print(rho)
```

```
print(theta)


# In[79]:


r,t= minimise_no_rhos(rho,theta)


# In[33]:


rho = [257,258,352,353,445,446,541,542,636,637,734,733]
theta = [-1.53588974175501 ,-1.53588974175501,-1.53588974175501, -1.53588974175501,-
1.53588974175501,-1.53588974175501]
#gray_img = cv2.imread('original_imgs/hough.jpg', cv2.IMREAD_GRAYSCALE)
dup = np.copy(edges)
#gs = gray_img.shape
diag_len = np.ceil(np.sqrt(dup.shape[0] *dup.shape[0]+ dup.shape[1] * dup.shape[1]))
indices = []
for i in range(dup.shape[0]):
    for j in range(dup.shape[1]):
        for t in theta:
            rho_dup = round(i * np.cos(t) + j * np.sin(t))+diag_len
            #print(rho_dup, rho)
            for r in rho:
                if r == rho_dup:
                    indices.append([i, j])


# In[34]:


gray_img = cv2.imread('original_imgs/hough.jpg')
output = np.zeros((gray_img.shape[0],gray_img.shape[1]))

for i in indices:
    gray_img[i[0],i[1]] = (0,255,0)

cv2.imwrite("output/red_line.jpg",gray_img)


# In[49]:


img_b = cv2.imread('original_imgs/hough.jpg')
hsv_img_b = cv2.cvtColor(img_b,cv2.COLOR_BGR2HSV)
lower_blue = np.array([95,60,110],np.uint8)
upper_blue = np.array([110,140,150],np.uint8)
mask1 = cv2.inRange(hsv_img_b, lower_blue, upper_blue)

output_hsv_b = hsv_img_b.copy()
output_hsv_b[np.where(mask1==0)] = 0

cv2.imwrite("onlyblue.jpg",output_hsv_b)


# In[50]:
```

```
edges_b = detectEdges(output_hsv_b)
cv2.imwrite("blue_edge.jpg",edges_b)


# In[62]:


accumulators_b,thetas_b,rhos_b = hough_line(edges_b)
print(accumulators_b.shape)
# code to get the indices which are above the range
idx_b = np.argmax(accumulators_b)
print(idx_b)
print(accumulators_b[int(idx_b/accumulators_b.shape[1]),int(idx_b%accumulators_b.shape[1])])
mask_b = [accumulators_b > 125.] [0] * 1.
accum_b = accumulators_b * mask_b
rho_b = []
theta_b = []
for i in range(mask_b.shape[0]):
    for j in range(mask_b.shape[1]):
        if mask_b[i,j]==1:
            rho_b.append(i)
            theta_b.append(thetas_b[j])


# In[80]:


r_b,t_b= minimise_no_rhos(rho_b,theta_b)


# In[65]:


rho_b = [489,490,564,565,637,638,708,709,779,780,853,854,930,931]
theta_b =[-0.9424777960769379]
#gray_img = cv2.imread('original_imgs/hough.jpg', cv2.IMREAD_GRAYSCALE)
dup_b = np.copy(edges_b)
#gs = gray_img.shape
diag_len_b = np.ceil(np.sqrt(dup_b.shape[0] *dup_b.shape[0]+ dup_b.shape[1] * dup_b.shape[1]))
indices_b = []
for i in range(dup_b.shape[0]):
    for j in range(dup_b.shape[1]):
        for t in theta_b:
            rho_dup = round(i * np.cos(t) + j * np.sin(t))+diag_len_b
            #print(rho_dup, rho_b)
            for r in rho_b:
                if r == rho_dup:
                    indices_b.append([i, j])


# In[66]:

gray_img_b = cv2.imread('original_imgs/hough.jpg')
#output = np.zeros((gray_img_b.shape[0],gray_img_b.shape[1]))

for i in indices_b:
```

```
    gray_img_b[i[0],i[1]] = (0,255,0)

cv2.imwrite("output/blue_line.jpg",gray_img_b)
```

3.3 Output

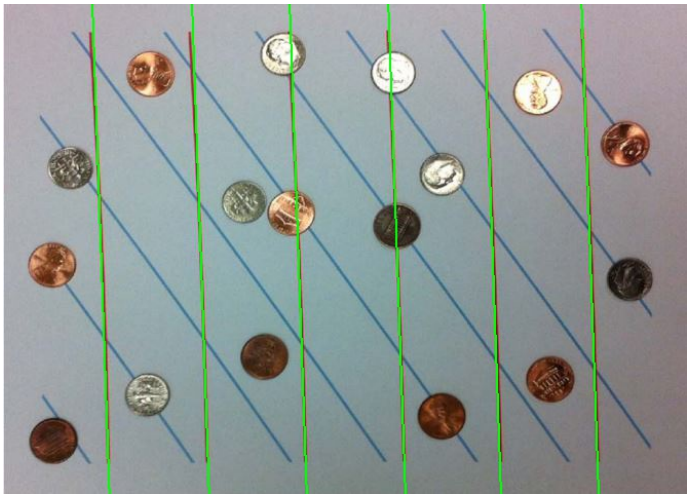

Figure 10: Red Line – 6 line detected

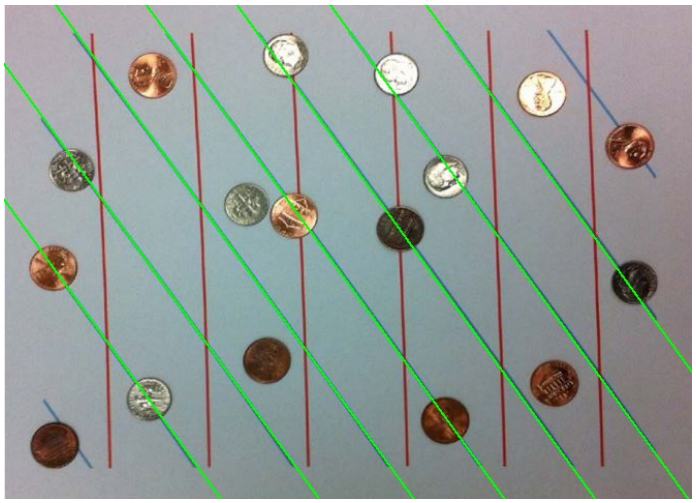

Figure 11: Blue Line – 7 lines detected

4 Conclusion

All the tasks such as Morphology, Segmentation and Hough transform are implemented for the given input images and the respective code and output images are provided in the report.

References

[1] Richard Szeliski (2010), Computer Vision: Algorithms and Applications

[2] https://docs.opencv.org/2.4/doc/tutorials/

[3] https://docs.python.org/3/