**CSE 573 Computer Vision and Image Processing**

**Project 3 Report**

**Name: - Chinmay Swami**

**Person No: - 50290385**

**UB Name: - chinmayp**

# Task 1: - Morphology image processing

In this task we had to process the image by using 2 Morphological algorithms to remove noise. Once the noise was removed we had to perform boundary detection.

The two algorithms used are as follows:-

***Steps:-***

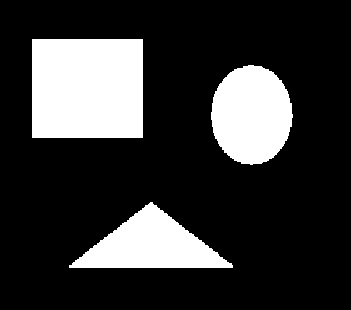
1. Read image
2. Noise Removal
   1. Perform Opening.
      1. Perform Erosion
      2. Perform Dilation
   2. Perform Closing
      1. Perform Dilation
      2. Perform Erosion
   3. Save Output and again read original image
   4. Perform Closing
      1. Perform Dilation
      2. Perform Erosion
   5. Perform Opening.
      1. Perform Erosion
      2. Perform Dilation
3. Boundary Extraction
   1. Image 1
      1. Perform Erosion on output of step 2.b.ii
      2. Subtract output of 3.a from 2.b.ii
   2. Image 2
      1. Perform Erosion on output of step 2.e.ii
      2. Subtract output of 3.a from 2.e.ii

**Q1.b]** Please compare the two results, and indicate are they the same in your report.

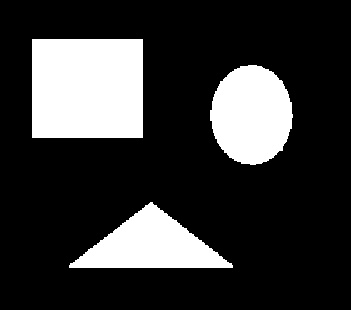
Images obtained from Algorithm1[Opening & then Closing] and Algorithm2[Closing & then Opening] are almost identical however there is one small disparity amongst them. There are some extra protrusions in output of Algorithm2 towards the outside.

## Output

1] Opening and then Closing

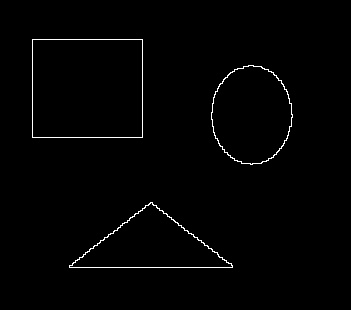


***Figure 1.1:- Performing Opening first and then Closing.***

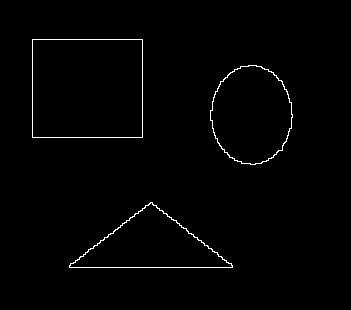


***Figure 1.2:- Performing Closing first and then Opening.***

2] Closing and then Opening



***Figure 1.3:- Boundaries detected based on Figure 1.***

******

***Figure 1.4:- Boundaries detected based on Figure 2.***

## Functions Created and their Discriptions

1] def performErosion(image, mask):- This function is used to perform Erosion on the image.

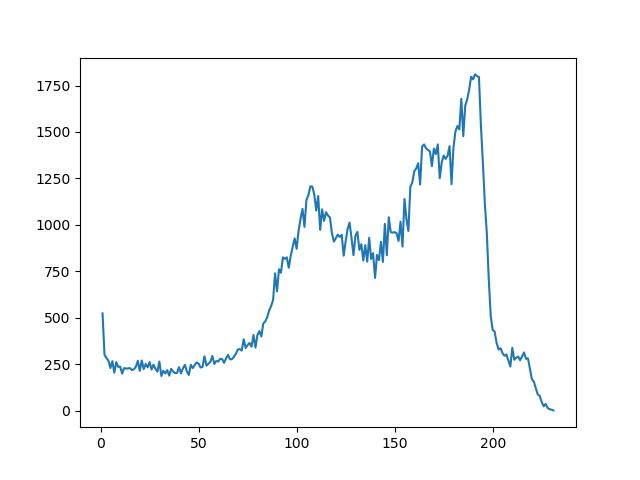
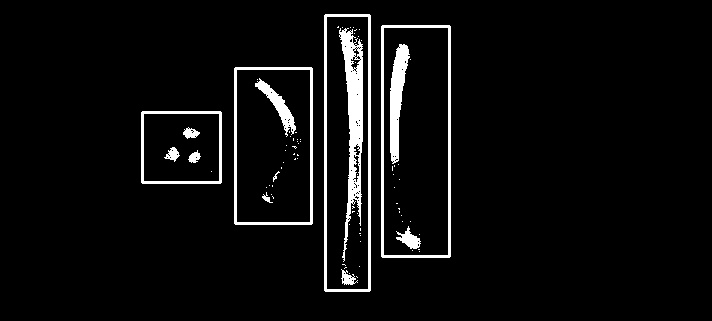
2] def performDilation(image, mask):- This function is used to perform Dilation on the image.

# Task 2: - Image segmentation and point detection

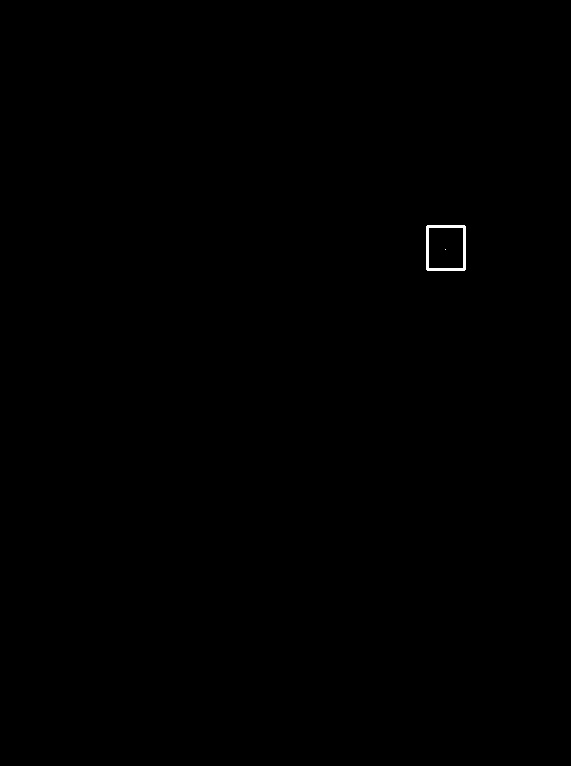
***Steps***:-

1. Detecting the Porosity point
   1. Read Image and create a mask as shown in slide 27 of Image Analysis.pdf.
   2. Apply Laplacian filter on the image read in step 1
   3. Use the formula shown in slide 27 of Image Analysis.pdf to calculate R
   4. Create an array similar to image but with all zeros
   5. If R value of a particular Pixel is greater than 90% of MAX[|R|] then in the array created at step1.d change the value of that pixel to 255.
   6. Print the updated array.
2. Thresholding
   1. Constructing Histogram from scratch: -
      1. Histogram is nothing but unique values on the X axis and frequency of those values on the Y axis hence we first read the image Segment.jpg
      2. Using np. unique function find the unique pixel values present in image and its frequency.
      3. To plt. plot (X, Y) pass list of unique values as X coordinate values and frequency as Y-co-ordinate values.
   2. Create an array similar to image but with all zeros
   3. Based on the threshold value identified if pixel value in the image is greater than or equal to the threshold value then make the pixel value as 255 in the array created in step2. b.
   4. Write the updated array to obtain the final image

## Output



***Figure 2.2:- Segmentation using optimal thresholding value Figure 2.3:- Histogram of image segment.jpg***



***Figure 2.1:- Porosity present in the blade shown by a point within the square***

Porosity Point Co-ordinates: - [249, 445]

Co-Ordinate of points for Object1: - [142,112], [220,112], [142,182], [220,182]

Co-Ordinate of points for Object2: - [235,68], [311,68], [235,223], [311,223]

Co-Ordinate of points for Object3: - [325,15], [369,15], [325,290], [369,290]

Co-Ordinate of points for Object4: - [382,26], [449,26], [382,256], [449,256]

## Functions Created and their Discriptions

1] def detectPoints(image, mask): - Used to generate and save the |R|value

2] def generateFinalImage(imageWithPoints, maxSumofProduct,isItTask2, percentValue): This is used to generate a new image based on the threshold value and the |R| values.

# Task 3: - Hough transform Steps:-

1. Perform Edge detection by using sobel operator. Used the code from project1 task1
2. We used the image generated by Sobel Y for detecting diagonal lines and image generated by sobel X for detecting vertical lines.
3. Thresholding was done to generate a binary image for the images in above step. And some extra thresholding was done to get only vertical lines from Sobel X image.
4. classify as per new cluster centers and again perform 1 more iteration
5. Apply K-means to perform color quantization

## Code

clusterCenters = [[6.2, 3.2], [6.6, 3.7], [6.5, 3.0]]

X = [[5.9, 3.2], [4.6, 2.9], [6.2, 2.8], [4.7, 3.2], [5.5, 4.2], [5.0, 3.0], [4.9, 3.1],[6.7, 3.1],[5.1, 3.8], [6.0, 3.0]]

XDistance = []

# Calculate the distances of points from clusters

XDistance = calculateDistance(clusterCenters, X, XDistance)

XClassified = []

print(len(XDistance), XDistance)

# Classify the points based on distance calculated earlier

XClassified = chooseCenteroid(XDistance,XClassified)

print(XClassified)

# Plot the cluesters

plotClusters(X, clusterCenters, XClassified, 'task3\_iter1\_a.png')

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Task 3.2:- Calculate new cluster centroids \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Calculate new centroids

clusterCenters = caculateNewMean(X,XClassified)

plotClusters(X, clusterCenters, XClassified,'task3\_iter1\_b.png')

XDistance = []

XDistance = calculateDistance(clusterCenters, X, XDistance)

XClassified = []

XClassified = chooseCenteroid(XDistance,XClassified)

print(XClassified)

# Classify the points based on distance calculated earlier

XClassified = chooseCenteroid(XDistance,XClassified)

plotClusters(X, clusterCenters, XClassified,'task3\_iter2\_a.png')

# \*\*\*\*\*\*\*\*\*\*\*\*\* Task 3.3 :- calculate euclidian disctances for new mean and generate clusters \*\*\*\*\*\*\*\*\*\*\*\*

# Calculate new centroids

clusterCenters = caculateNewMean(X,XClassified)

XDistance = []

# Calculate the distances of points from clusters

XDistance = calculateDistance(clusterCenters, X, XDistance)

XClassified = []

# Classify the points based on distance calculated earlier

XClassified = chooseCenteroid(XDistance,XClassified)

print(XClassified)

plotClusters(X,clusterCenters,XClassified,'task3\_iter2\_b.png')

# Task 3.4

kmeanFull()

def kmeanFull():

image = cv2.imread("Images/baboon.jpg")

imgHeight, imgWidth = image.shape[:2]

image = np.asarray(image,dtype=np.float64) / 255

noOfClusters = 3

clusterCenters = []

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Randomly generate cluster centers \*\*\*\*\*\*\*\*\*\*\*\*\*\*

for times in range(noOfClusters):

imageX = random.randint(0, 511)

imageY = random.randint(0, 511)

print(imageX, imageY)

clusterCenters.append(image[imageX][imageY])

# to get rid of the dtype info

clusterCenters = np.array(clusterCenters, dtype=np.float64) / 255

prevClusterCenters = np.asarray(clusterCenters)

print("cluster Centers \n", clusterCenters)

image = image.reshape((image.shape[0] \* image.shape[1], 3))

print("image Shape ", image.shape)

converge = False

iteration = 0

while (converge == False):

print("\*\*\*\*\*\*\*\*\*\*\* Iteration",iteration," \*\*\*\*\*\*\*\*\*\*\*")

iteration +=1

# \*\*\*\*\*\*\*\*\*\*\*\*\*\* Calculating euclidean distances \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

colorDistance = calculateColorDistance(noOfClusters, image, clusterCenters)

# print("colorDistance \n",colorDistance)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Classify the colors to clusters \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ptsClassified = np.zeros((image.shape[0],noOfClusters))

ptsClassified = findCluster(colorDistance, ptsClassified, noOfClusters)

print("cluster element numbers :",np.sum(ptsClassified,axis=0))

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Calculate new Cluster centers \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

clusterCenters = reCalculateMean(ptsClassified, colorDistance, noOfClusters, clusterCenters)

converge = np.array\_equal(prevClusterCenters,clusterCenters)

print("New clusters: \n",clusterCenters)

if iteration == 15:

converge = True

for imgIndex in range(image.shape[0]):

for centers in range(noOfClusters):

if ptsClassified[imgIndex][centers] == 1:

image[imgIndex] = clusterCenters[centers]

print(image)

image = image.reshape((imgHeight,imgWidth,3))

colorDistance = colorDistance.reshape((imgHeight,imgWidth,noOfClusters))

image = image \* 255

image = image.astype(np.uint8)

colorDistance = colorDistance \* 255

colorDistance = colorDistance.astype(np.uint8)

cv2.imwrite("task3\_baboon\_5.jpg", image)

**Functions.py**

clusterCentersColor = ['Red', 'Green', 'Blue']

def calculateDistance(clusterCenters,X,XDistance):

for dataPts in range(len(X)):

XDistanceTemp = []

for centers in range(len(clusterCenters)):

# Center points are X0,Y0 and Datapoints are X1,Y1

d1 = ((clusterCenters[centers][0] - X[dataPts][0])\*\*2)

d2 = ((clusterCenters[centers][1] - X[dataPts][1])\*\*2)

distance = (d1 + d2)\*\*0.5

XDistanceTemp.append(distance)

XDistance.append(XDistanceTemp)

return XDistance

def chooseCenteroid(XDistance,XClassified):

for distances in range(len(XDistance)):

minValIndexLoc = XDistance[distances].index(min(XDistance[distances]))

if minValIndexLoc == 0: # belongs to cluster 1

XClassified.append('Red')

elif minValIndexLoc == 1: # belongs to cluster 2

XClassified.append('Green')

elif minValIndexLoc == 2: # belongs to cluster 3

XClassified.append('Blue')

return XClassified

def caculateNewMean(X,XClassified):

sumRedX = 0

sumRedY = 0

countRed = 0

sumBlueX = 0

sumBlueY = 0

countBlue = 0

sumGreenX = 0

sumGreenY = 0

countGreen = 0

for index in range(len(X)):

if XClassified[index] == 'Red': # belongs to cluster 1

sumRedX += X[index][0]

sumRedY += X[index][1]

countRed += 1

elif XClassified[index] == 'Blue': # belongs to cluster 2

sumBlueX += X[index][0]

sumBlueY += X[index][1]

countBlue += 1

elif XClassified[index] == 'Green': # belongs to cluster 3

sumGreenX += X[index][0]

sumGreenY += X[index][1]

countGreen += 1

newCenters = [[sumRedX/countRed, sumRedY/countRed],

[sumGreenX / countGreen, sumGreenY / countGreen],

[sumBlueX / countBlue, sumBlueY / countBlue]]

return newCenters

def plotClusters(X,clusterCenters,XClassified,fineName):

# Extract X & Y co-ordinates

arrayX = np.asarray(X)

arrayClusterCenters = np.asarray(clusterCenters)

xX = arrayX[:, :1].tolist()

yX = arrayX[:, 1:].tolist()

centerx = arrayClusterCenters[:, :1].tolist()

centery = arrayClusterCenters[:, 1:].tolist()

plt.scatter(centerx, centery, color=clusterCentersColor, marker="o")

for i in range(len(clusterCenters)):

plt.text(centerx[i][0] + 0.005, centery[i][0], s='(' + str(centerx[i][0]) + ',' + str(centery[i][0]) + ')')

for i in range(len(X)):

plt.scatter(xX[i], yX[i], color=XClassified[i], marker="^")

plt.text(xX[i][0] + 0.005, yX[i][0], s='(' + str(xX[i][0]) + ',' + str(yX[i][0]) + ')')

# plt.show()

plt.savefig(fineName)

plt.clf()

def calculateColorDistance(noOfClusters, image, clusterCenters):

colorDistance = np.zeros((image.shape[0], noOfClusters))

print("in function")

print(image.shape[0])

for imgX in range(image.shape[0]):

for centers in range(noOfClusters):

x = image[imgX]

y = clusterCenters[centers]

distance = np.linalg.norm(x-y)

colorDistance[imgX][centers] = distance

return colorDistance

def findCluster(colorDistance, ptsClassified, noOfClusters):

colorDistancecolumns = []

for imgX in range(colorDistance.shape[0]):

for centers in range(noOfClusters):

colorDistancecolumns.append(colorDistance[imgX][centers])

minValIndexLoc = colorDistancecolumns.index(min(colorDistancecolumns))

ptsClassified [imgX][minValIndexLoc] = 1

colorDistancecolumns = []

return ptsClassified

def reCalculateMean(ptsClassified, colorDistance, noOfClusters, clusterCenters):

oldClusterCenters = clusterCenters

clusterCenters = []

newMean = []

print(type(newMean))

clusterElementNo = np.sum(ptsClassified, axis=0)

for cluster in range(noOfClusters):

for imgX in range(colorDistance.shape[0]):

if ptsClassified[imgX][cluster] == 1:

# print(colorDistance[imgX])

newMean.append(colorDistance[imgX])

if not newMean:

newMean.append(oldClusterCenters[cluster])

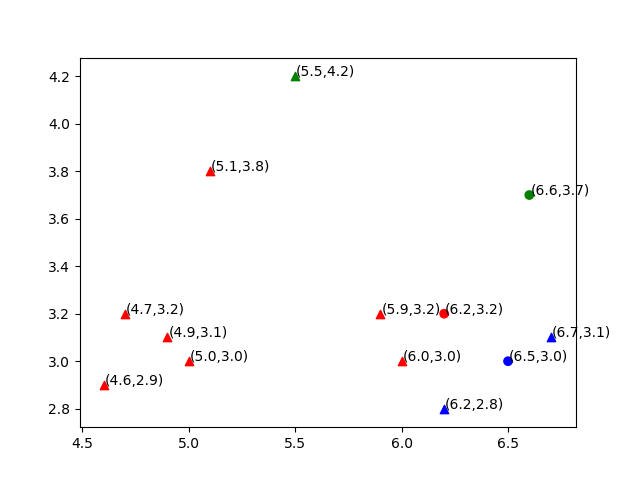
mean = np.mean(newMean,axis = 0)

clusterCenters.append(mean)

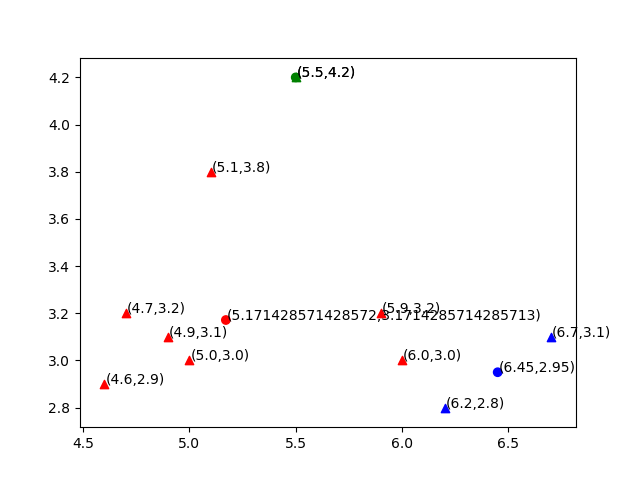
newMean = []

return clusterCenters

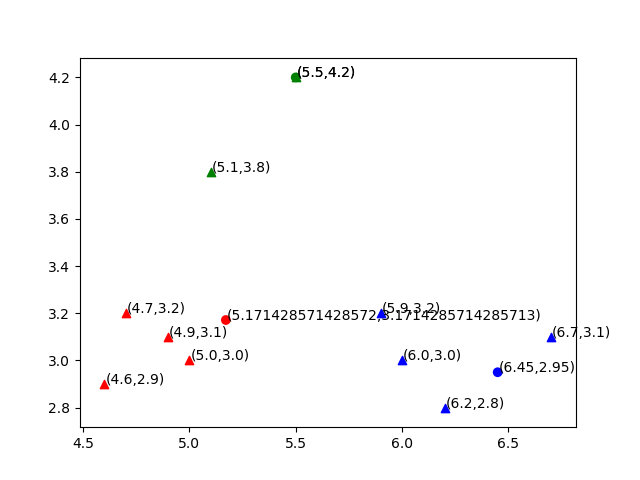
## Code SET A and Bonus part



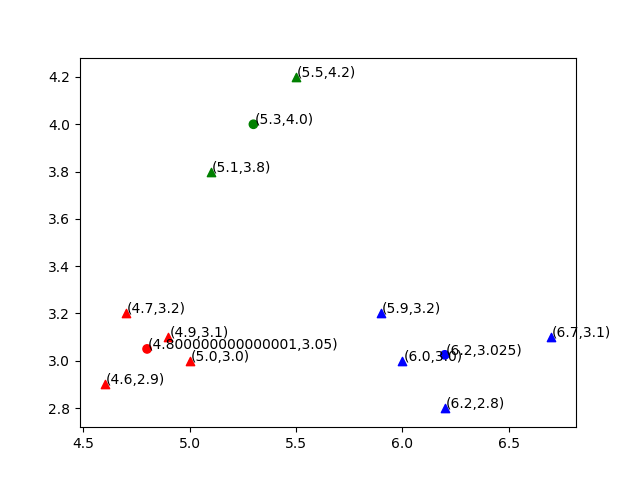
***Figure 3.1:- classified points as per given clusters***

******

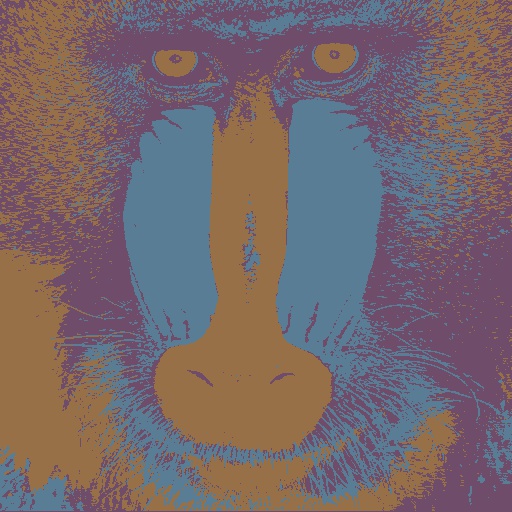
***Figure 3.2:- Recomputed cluster centers***

******

***Figure 3.3:- Reclassify as per new cluster centers***

******

***Figure 3.4:- Recompute cluster centers***

******

***Figure 3.5:- Color Quantization with 3 clusters.***