**Shraddha Dwivedi**

**UB Person Number: 50290266**

**Computer Science and Engineering**

**University at Buffalo**

[**sdwivedi@buffalo.edu**](mailto:sdwivedi@buffalo.edu)

**1. Morphology Image Processing**

import numpy as np

from cv2 import imread, imwrite, cvtColor, COLOR\_GRAY2RGB, putText, FONT\_HERSHEY\_SIMPLEX, LINE\_AA

def displayImage(img, outputName):

imwrite('outputImages/'+outputName+'.jpg', img)

return 1

imgLoc = './noise.jpg'

def extractK():

return np.ones((3, 3), np.uint8)

# Implementing the function for eroison

def erosion(img, kernel):

Len, Wid = img.shape

result = np.zeros((Len-2, Wid-2))

for i in range(1, Len-1):

for j in range(1, Wid-1):

if( (img[i-1,j-1]==255 and kernel[0,0]==1) and (img[i-1,j]==255 and kernel[0,1]==1) and (img[i-1,j+1]==255 and kernel[0,2]==1)

and (img[i,j-1]==255 and kernel[1,0]==1) and (img[i,j]==255 and kernel[1,1]==1) and (img[i,j+1]==255 and kernel[1,2]==1)

and (img[i+1,j-1]==255 and kernel[2,0]==1) and (img[i+1,j]==255 and kernel[2,1]==1) and (img[i+1,j+1]==255 and kernel[2,2]==1)):

result[i-1][j-1] = 255

return result

# Implementing the function for dilation

def dilation(img, kernel):

Len, Wid = img.shape

result = np.zeros((Len-2, Wid-2))

for i in range(1, Len-1):

for j in range(1, Wid-1):

if( (img[i-1,j-1]==255 and kernel[0,0]==1) or (img[i-1,j]==255 and kernel[0,1]==1) or (img[i-1,j+1]==255 and kernel[0,2]==1)

or (img[i,j-1]==255 and kernel[1,0]==1) or (img[i,j]==255 and kernel[1,1]==1) or (img[i,j+1]==255 and kernel[1,2]==1)

or (img[i+1,j-1]==255 and kernel[2,0]==1) or (img[i+1,j]==255 and kernel[2,1]==1) or (img[i+1,j+1]==255 and kernel[2,2]==1)):

result[i-1][j-1] = 255

return result

# Implementing the function for closing in which we used the implemented erosion and dilation

def closing(img,kernel):

img1 = dilation(img, kernel)

result = erosion(img1,kernel)

return result

# Implementing the function for opening in which we used the implemented erosion and dilation

def opening(img, kernel):

img1 = erosion(img, kernel)

result = dilation(img1, kernel)

return result

#Implementing the function for boundary detection

def boundaryDetection(img, kernel):

erodedImg = erosion(img, kernel)

erodedImg = np.pad(erodedImg, (1, 1), 'constant')

result = np.asarray(img - erodedImg)

return result

def main():

kernel = extractK()

img = imread(imgLoc, 0)

opening1 = opening(img, kernel)

res\_noise1 = closing(opening1, kernel)

closing1 = closing(img, kernel)

res\_noise2 = opening(closing1, kernel)

displayImage(res\_noise1, 'res\_noise1.jpg')

displayImage(res\_noise2, 'res\_noise2.jpg')

res\_bound1 = boundaryDetection(res\_noise1, kernel)

res\_bound2 = boundaryDetection(res\_noise2, kernel)

displayImage(res\_bound1, 'res\_bound1.jpg')

displayImage(res\_bound2, 'res\_bound2.jpg')

main()

**Output Results:**

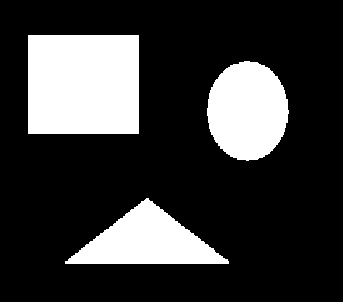
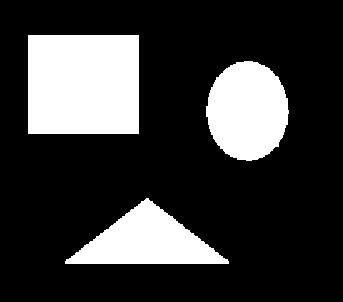
 

Fig 1: res\_noise1.jpg Fig 2: res\_noise2.jpg

In the Figure 1, first I applied open function, then I applied In the Figure 2, first I applied close function, then applied

Close function Open function.

Both the images in Figure 1 and Figure 2 are same.

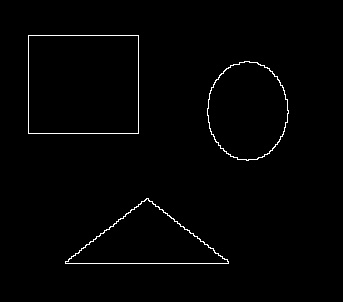
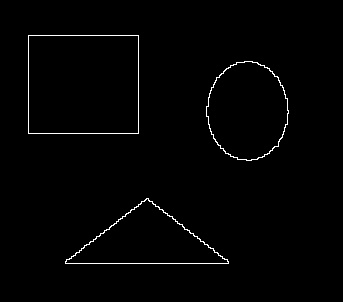
 

Fig 3: res\_bound1.jpg Fig 4: res\_bound2.jpg

**2. Image Segmentation and Point Detection:**

from cv2 import imread, imwrite, cvtColor, COLOR\_GRAY2RGB, putText, FONT\_HERSHEY\_SIMPLEX, LINE\_AA

from Utilities import sharpen, blur

import matplotlib.mlab as mlab

import matplotlib.pyplot as plt

def displayImage(img, outputName):

imwrite('outputImages/'+outputName+'.jpg', img)

return 1

pointimgPath = './point.jpg'

pointimgPath1 = './point\_HQ.jpg'

segmentimg = './segment.jpg'

threshold = [747, 1274, 1815]

kernel1 = np.array(

[[-1,-1,-1],

[-1,8,-1],

[-1,-1,-1]])

kernel = np.array(

[[-1,-1,-1,-1,-1],

[-1,-1,-1,-1,-1],

[-1,-1,8,-1,-1],

[-1,-1,-1,-1,-1],

[-1,-1,-1,-1,-1]])

def main():

# task 2a with original given image

pointimg = imread(pointimgPath, 0)

imgAL, imgAW = pointimg.shape

finalImage = np.zeros((imgAL, imgAW))

count = 0

pts = []

for i in range(1, imgAL-1):

for j in range(1, imgAW-1):

S = pointimg[i-1,j-1] \* kernel1[0,0] + pointimg[i-1,j] \* kernel1[0,1] + pointimg[i-1,j+1] \* kernel1[0,2]

+ pointimg[i,j-1] \* kernel1[1,0] + pointimg[i,j] \* kernel1[1,1] + pointimg[i,j+1] \* kernel1[1,2]

+ pointimg[i+1,j-1] \* kernel1[2,0] + pointimg[i+1,j] \* kernel1[2,1] + pointimg[i+1,j+1] \* kernel1[2,2]

if (abs(S)>threshold[0]):

count = count + 1

finalImage[i,j]=255

pts.append((i, j))

writeImage(finalImage, 'Task2A\_1')

print("Number of points", count)

print("Points detected : "+str(pts))

print()

#Image with better quality for task 2a

pointimg = imread(pointimgPath1, 0)

imgAL, imgAW = pointimg.shape

finalImage = np.zeros((imgAL, imgAW))

count = 0

pts = []

for i in range(2, imgAL-2):

for j in range(2, imgAW-2):

S = pointimg[i-2,j-2] \* kernel[0,0] + pointimg[i-2,j-1] \* kernel[0,1] + pointimg[i-2,j] \* kernel[0,2] + pointimg[i-2,j+1] \* kernel[0,3] + pointimg[i-2,j+2] \* kernel[0,4]

+ pointimg[i-1,j-2] \* kernel[1,0] + pointimg[i-1,j-1] \* kernel[1,1] + pointimg[i-1,j] \* kernel[1,2] + pointimg[i-1,j+1] \* kernel[1,3] + pointimg[i-1,j+2] \* kernel[1,4]

+ pointimg[i,j-2] \* kernel[2,0] + pointimg[i,j-1] \* kernel[2,1] + pointimg[i,j] \* kernel[2,2] + pointimg[i,j+1] \* kernel[2,3] + pointimg[i,j+2] \* kernel[2,4]

+ pointimg[i+1,j-2] \* kernel[3,0] + pointimg[i+1,j-1] \* kernel[3,1] + pointimg[i+1,j] \* kernel[3,2] + pointimg[i+1,j+1] \* kernel[3,3] + pointimg[i+1,j+2] \* kernel[3,4]

+ pointimg[i+2,j-2] \* kernel[4,0] + pointimg[i+2,j-1] \* kernel[4,1] + pointimg[i+2,j] \* kernel[4,2] + pointimg[i+2,j+1] \* kernel[4,3] + pointimg[i+2,j+2] \* kernel[4,4]

if (abs(S)>threshold[1]):

count = count + 1

finalImage[i,j]=255

pts.append((i, j))

displayImage(finalImage, 'Task2A\_2')

print("Number of points", count)

print("Points detected "+str(pts))

print()

# image with image sharpening for task 2a

pointimg = imread(pointimg1, 0)

pointimg = np.pad(pointimg, [(2, 2), (2, 2)], mode='constant', constant\_values=0)

pointimg = sharpen(pointimg)

imgAL, imgAW = pointimg.shape

finalImage = np.zeros((imgAL, imgAW))

count = 0

pts = []

for i in range(2, imgAL-2):

for j in range(2, imgAW-2):

S = pointimg[i-2,j-2] \* kernel[0,0] + pointimg[i-2,j-1] \* kernel[0,1] + pointimg[i-2,j] \* kernel[0,2] + pointimg[i-2,j+1] \* kernel[0,3] + pointimg[i-2,j+2] \* kernel[0,4] + pointimg[i-1,j-2] \* kernel[1,0]

+ pointimg[i-1,j-1] \* kernel[1,1] + pointimg[i-1,j] \* kernel[1,2] + pointimg[i-1,j+1] \* kernel[1,3] + pointimg[i-1,j+2] \* kernel[1,4] + pointimg[i,j-2] \* kernel[2,0] + pointimg[i,j-1] \* kernel[2,1]

+ pointimg[i,j] \* kernel[2,2] + pointimg[i,j+1] \* kernel[2,3] + pointimg[i,j+2] \* kernel[2,4] + pointimg[i+1,j-2] \* kernel[3,0] + pointimg[i+1,j-1] \* kernel[3,1] + pointimg[i+1,j] \* kernel[3,2]

+ pointimg[i+1,j+1] \* kernel[3,3] + pointimg[i+1,j+2] \* kernel[3,4] + pointimg[i+2,j-2] \* kernel[4,0]

+ pointimg[i+2,j-1] \* kernel[4,1] + pointimg[i+2,j] \* kernel[4,2] + pointimg[i+2,j+1] \* kernel[4,3]

+ pointimg[i+2,j+2] \* kernel[4,4]

if (abs(S)>threshold[2]):

count = count + 1

finalImage[i,j]=255

pts.append((i, j))

for pt in pts:

if (pt==(254,447)):

putText(finalImage, str((447,254)), (435,240), FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), lineType=LINE\_AA)

displayImage(finalImage, 'Task2A\_3')

print("Number of points", count)

print("Points detected : "+str(pts))

print()

segmentimg1 = imread(segmentimg, 0)

finalImage = segmentimg1.copy()

segmentimgL, segmentimgW = segmentimg1.shape

unique, counts = np.unique(segmentimg1, return\_counts=True)

fig, ax = plt.subplots()

ax.bar(unique, counts, 1)

plt.xlabel('Pixel Values')

plt.ylabel('Count')

plt.title('Histogram')

fig.savefig('outputImages/Task2B\_Histogram.png')

S\_threshold = 200

for i in range(0, segmentimgL):

for j in range(0, segmentimgW):

if (segmentimg1[i,j]<S\_threshold):

finalImage[i,j]=0

minX=None; maxX=None

minY=None; maxY=None

x=[]

y=[]

for i in range(0, segmentimgL):

for j in range(0, segmentimgW):

if (segmentimg1[i,j]>S\_threshold):

x.append(j)

y.append(i)

x=np.array(x)

y=np.array(y)

minX = np.amin(x)

maxX = np.amax(x)

minY = np.amin(y)

maxY = np.amax(y)

pts = [(minX, minY),(maxX, minY),(maxX, maxY),(minX, maxY)]

print(pts)

finalImage[minY:maxY, minX] = 255

finalImage[minY:maxY, maxX] = 255

finalImage[minY, minX:maxX] = 255

finalImage[maxY, minX:maxX] = 255

putText(finalImage,str((minX, minY)), (minX, minY), FONT\_HERSHEY\_SIMPLEX, 1.0, (255, 255, 255), lineType=LINE\_AA)

putText(finalImage,str((maxX, minY)), (maxX, minY), FONT\_HERSHEY\_SIMPLEX, 1.0, (255, 255, 255), lineType=LINE\_AA)

putText(finalImage,str((maxX, maxY)), (maxX, maxY), FONT\_HERSHEY\_SIMPLEX, 1.0, (255, 255, 255), lineType=LINE\_AA)

putText(finalImage,str((minX, maxY)), (minX, maxY), FONT\_HERSHEY\_SIMPLEX, 1.0, (255, 255, 255), lineType=LINE\_AA)

color = cvtColor(segmentimg1, COLOR\_GRAY2RGB)

color[minY:maxY, minX, 0] = 0; color[minY:maxY, minX, 1] = 0; color[minY:maxY, minX, 2] = 255

color[minY:maxY, maxX, 0] = 0; color[minY:maxY, maxX, 1] = 0; color[minY:maxY, maxX, 2] = 255

color[minY, minX:maxX, 0] = 0; color[minY, minX:maxX, 1] = 0; color[minY, minX:maxX, 2] = 255

color[maxY, minX:maxX, 0] = 0; color[maxY, minX:maxX, 1] = 0; color[maxY, minX:maxX, 2] = 255

putText(color,str((minX, minY)), (minX, minY), FONT\_HERSHEY\_SIMPLEX, 1.0, (255, 255, 255), lineType=LINE\_AA)

putText(color,str((maxX, minY)), (maxX, minY), FONT\_HERSHEY\_SIMPLEX, 1.0, (255, 255, 255), lineType=LINE\_AA)

putText(color,str((maxX, maxY)), (maxX, maxY), FONT\_HERSHEY\_SIMPLEX, 1.0, (255, 255, 255), lineType=LINE\_AA)

putText(color,str((minX, maxY)), (minX, maxY), FONT\_HERSHEY\_SIMPLEX, 1.0, (255, 255, 255), lineType=LINE\_AA)

displayImage(color, 'Task2B\_FinalImage')

main()

**Output Results:**

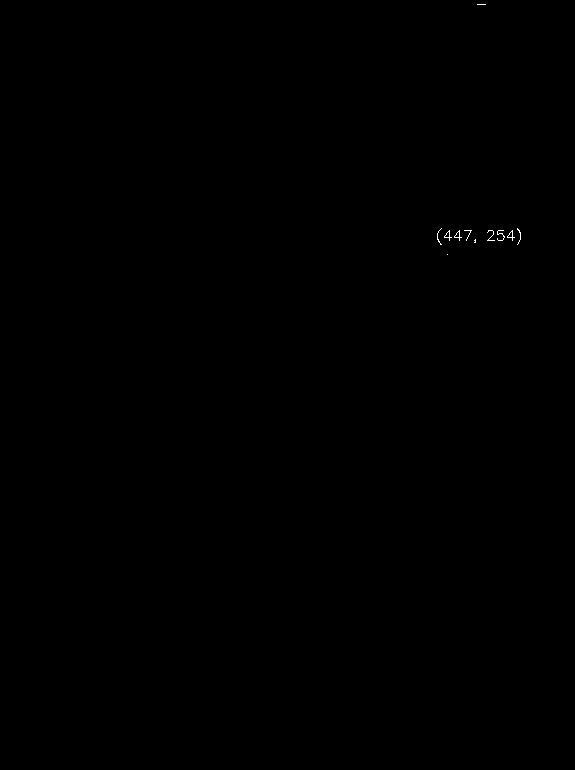


Fig 5: Detecting porosity in Original Image

(For Fig 5) Number of points detected: 45

Points detected : [(8, 319), (9, 320), (10, 321), (16, 329), (16, 330), (19, 334), (19, 335), (27, 345), (31, 349), (31, 350), (35, 353), (74, 353), (74, 354), (74, 355), (75, 354), (75, 355), (97, 354), (97, 355), (109, 351), (109, 352), (109, 353), (109, 354), (109, 355), (145, 284), (148, 346), (157, 351), (157, 352), (158, 352), (158, 353), (158, 354), (158, 355), (159, 353), (159, 354), (159, 355), (160, 354), (160, 355), (161, 355), (165, 353), (165, 354), (166, 340), (188, 355), (193, 355), (198, 355), (199, 355), (206, 355)]

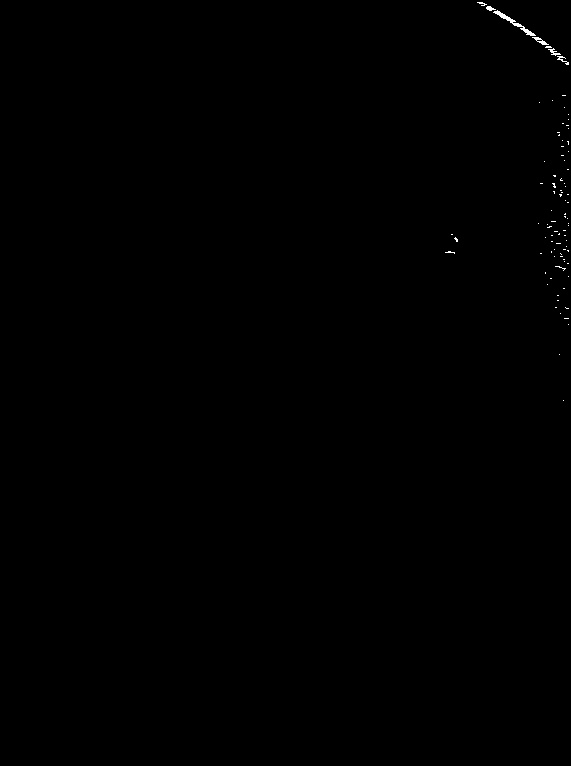


Fig 6: Detecting porosity by enhancing the quality of Original Image

(For Fig 6) Number of points detected: 581

Points detected [(2, 477), (2, 478), (2, 479), (2, 480), (2, 481), (2, 482), (3, 479), (3, 480), (3, 481), (3, 482), (3, 483), (3, 484), (4, 484), (4, 485), (4, 486), (5, 481), (5, 482), (5, 483), (5, 484), (6, 487), (6, 488), (6, 489), (6, 490), (7, 486), (7, 487), (7, 488), (7, 489), (7, 490), (7, 491), (8, 486), (8, 487), (8, 488), (8, 489), (8, 490), (8, 491), (8, 492), (8, 493), (9, 487), (9, 488), (9, 489), (9, 490), (9, 491), (9, 492), (9, 493), (9, 494), (9, 495), (10, 489), (10, 490), (10, 491), (10, 497), (11, 494), (11, 495), (11, 496), (11, 497), (11, 498), (11, 499), (12, 493), (12, 494), (12, 495), (12, 496), (12, 497), (12, 498), (12, 499), (12, 500), (13, 495), (13, 496), (13, 497), (13, 498), (13, 499), (13, 500), (13, 501), (13, 502), (14, 497), (14, 498), (14, 499), (14, 500), (14, 501), (14, 502), (14, 503), (14, 504), (15, 499), (15, 500), (15, 501), (15, 502), (15, 503), (15, 504), (15, 505), (16, 500), (16, 501), (16, 502), (16, 503), (16, 504), (16, 505), (16, 506), (16, 507), (17, 501), (17, 502), (17, 503), (17, 504), (17, 505), (17, 506), (17, 507), (17, 508), (17, 509), (18, 503), (18, 504), (18, 505), (18, 506), (18, 507), (18, 508), (18, 509), (18, 510), (19, 506), (19, 507), (19, 508), (19, 509), (19, 510), (19, 511), (19, 512), (20, 507), (20, 508), (20, 509), (20, 510), (20, 511), (20, 512), (20, 513), (20, 514), (21, 510), (21, 511), (21, 512), (21, 513), (21, 514), (21, 515), (22, 510), (22, 511), (22, 512), (23, 512), (23, 513), (23, 514), (23, 515), (23, 516), (23, 517), (23, 518), (24, 513), (24, 514), (24, 515), (24, 516), (24, 517), (24, 518), (24, 519), (24, 520), (25, 514), (25, 515), (25, 516), (25, 517), (25, 518), (25, 519), (25, 520), (25, 521), (26, 517), (26, 518), (26, 519), (26, 520), (26, 521), (26, 522), (26, 523), (27, 523), (27, 524), (28, 520), (28, 521), (28, 522), (28, 523), (28, 524), (29, 521), (29, 522), (29, 523), (29, 524), (29, 525), (29, 526), (29, 527), (30, 523), (30, 524), (30, 525), (30, 526), (30, 527), (30, 528), (31, 524), (31, 525), (31, 526), (31, 527), (31, 528), (31, 529), (31, 530), (32, 525), (32, 526), (32, 527), (32, 528), (32, 529), (32, 530), (32, 531), (33, 527), (33, 528), (33, 529), (33, 530), (33, 531), (33, 532), (33, 533), (34, 526), (34, 527), (34, 528), (34, 529), (34, 530), (34, 531), (34, 532), (34, 533), (34, 534), (35, 530), (35, 531), (35, 532), (35, 533), (35, 534), (35, 535), (37, 532), (37, 533), (37, 534), (37, 535), (37, 536), (37, 537), (37, 538), (38, 534), (38, 535), (38, 536), (38, 537), (38, 538), (38, 539), (38, 540), (39, 535), (39, 536), (39, 537), (39, 538), (39, 539), (39, 540), (39, 541), (40, 536), (40, 542), (41, 537), (41, 538), (41, 539), (41, 540), (41, 541), (41, 542), (41, 543), (41, 544), (42, 539), (42, 540), (42, 541), (42, 542), (42, 543), (42, 544), (42, 545), (43, 541), (43, 542), (43, 543), (43, 544), (43, 545), (43, 546), (44, 542), (44, 543), (44, 544), (44, 545), (44, 546), (44, 547), (44, 548), (46, 546), (46, 547), (46, 548), (46, 549), (46, 550), (47, 545), (47, 546), (47, 547), (47, 548), (47, 549), (47, 550), (47, 551), (48, 553), (49, 548), (49, 549), (49, 550), (49, 551), (49, 552), (49, 553), (49, 554), (50, 550), (50, 551), (50, 552), (51, 550), (51, 551), (51, 552), (51, 553), (51, 554), (51, 555), (51, 556), (52, 551), (52, 552), (52, 553), (53, 553), (53, 554), (53, 555), (53, 556), (53, 557), (53, 558), (53, 559), (54, 551), (54, 552), (54, 553), (54, 554), (54, 555), (54, 556), (54, 557), (54, 558), (54, 559), (54, 560), (55, 555), (56, 557), (56, 558), (56, 559), (56, 560), (56, 561), (56, 562), (57, 559), (57, 560), (57, 561), (57, 562), (57, 563), (57, 564), (58, 558), (58, 559), (58, 565), (59, 557), (59, 558), (59, 559), (59, 560), (59, 561), (59, 562), (60, 561), (60, 562), (61, 562), (61, 563), (61, 564), (61, 565), (61, 566), (62, 567), (62, 568), (63, 565), (63, 566), (63, 567), (63, 568), (64, 566), (64, 567), (64, 568), (95, 562), (95, 563), (95, 564), (95, 565), (100, 552), (102, 539), (107, 564), (114, 568), (119, 568), (123, 562), (123, 563), (125, 566), (125, 567), (125, 568), (128, 568), (132, 557), (132, 558), (132, 559), (134, 559), (135, 558), (135, 559), (140, 562), (141, 567), (141, 568), (142, 568), (143, 568), (144, 568), (146, 561), (146, 562), (146, 563), (151, 568), (155, 561), (155, 562), (155, 563), (160, 564), (161, 544), (169, 567), (169, 568), (175, 553), (175, 554), (175, 555), (176, 554), (176, 555), (178, 561), (179, 560), (180, 560), (180, 561), (180, 562), (183, 540), (183, 541), (183, 542), (183, 552), (183, 553), (183, 554), (183, 564), (184, 553), (184, 554), (185, 554), (186, 553), (186, 554), (186, 555), (186, 565), (187, 554), (187, 555), (190, 560), (190, 561), (191, 553), (191, 554), (192, 554), (192, 561), (192, 562), (193, 554), (194, 559), (194, 560), (194, 561), (194, 567), (194, 568), (195, 560), (195, 561), (196, 560), (200, 565), (202, 567), (202, 568), (210, 551), (213, 565), (214, 564), (214, 565), (216, 564), (216, 565), (217, 565), (218, 567), (221, 551), (221, 552), (221, 553), (221, 554), (221, 555), (223, 538), (223, 568), (224, 547), (225, 568), (226, 549), (226, 550), (226, 551), (227, 547), (227, 548), (227, 549), (230, 563), (230, 564), (230, 565), (231, 554), (231, 555), (231, 556), (231, 557), (233, 559), (234, 451), (234, 452), (234, 558), (234, 559), (235, 551), (235, 552), (235, 564), (235, 565), (237, 455), (237, 545), (238, 454), (238, 455), (238, 456), (239, 455), (239, 456), (239, 457), (240, 456), (240, 457), (240, 566), (241, 456), (241, 457), (241, 551), (241, 552), (243, 556), (243, 557), (243, 558), (243, 559), (243, 560), (246, 566), (249, 554), (249, 563), (249, 564), (249, 565), (250, 567), (250, 568), (251, 448), (251, 449), (251, 450), (251, 551), (251, 568), (252, 445), (252, 446), (252, 447), (252, 448), (252, 449), (252, 450), (252, 451), (252, 452), (252, 453), (252, 454), (252, 551), (253, 454), (253, 540), (253, 541), (253, 560), (254, 561), (256, 554), (256, 560), (256, 561), (256, 562), (259, 563), (259, 564), (261, 564), (263, 567), (263, 568), (266, 555), (266, 556), (266, 557), (266, 558), (266, 559), (267, 559), (267, 560), (267, 561), (268, 562), (268, 563), (268, 564), (268, 565), (269, 564), (270, 563), (272, 545), (273, 545), (276, 568), (278, 550), (278, 551), (284, 547), (288, 563), (288, 564), (295, 557), (295, 558), (300, 557), (300, 558), (307, 555), (307, 556), (307, 565), (308, 566), (308, 567), (308, 568), (313, 560), (318, 564), (318, 565), (318, 566), (318, 567), (318, 568), (324, 568), (354, 559), (400, 563)]

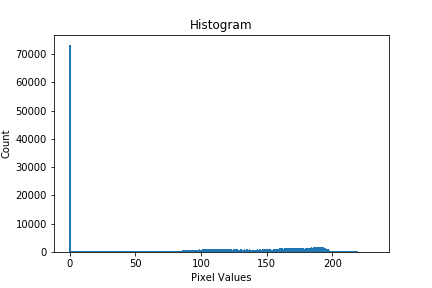


Fig 7: Histogram



Fig 8: Segmented Image

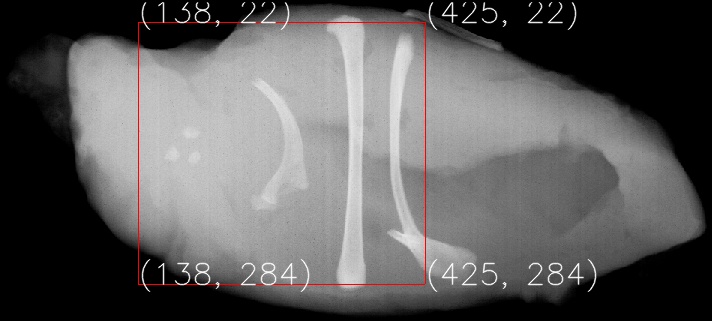


Fig 9: Rectangular bounding box on Segmented Image

**3. Hough Transform:**

import math as math

import numpy as np

from cv2 import imread, imwrite, line as l

from Utilities import sharpen, blur, edgeDetector

def displayImage(img, outputname):

imwrite('outputImages/'+outputname+'.jpg', img)

return 1

imgLoc = './hough.jpg'

rThreshold = 75;gThreshold = 0;bThreshold = 75

thresh = 50

noOfVerticalLines = 25

noOfDiagonalLines = 48

thetaOffset = -90

def threshold(image, t):

img = image.copy()

for i in range(0, len(img)):

for j in range(0, len(img[0])):

if (img[i,j]<t):

img[i,j] = 0

return img

def createImage(img, Len, Wid, DiagonalL=None):

if (DiagonalL==None):

DiagonalL = math.ceil(math.sqrt(Len\*\*2 + Wid\*\*2))

L = 2\*DiagonalL+1

W = 181

Img = np.zeros((L, W))

for i in range(0, Len):

y = (Len-1)-i

for j in range(0, Wid):

x = j

if(img[i,j]>thresh):

for l in range(0, W):

o = math.radians(thetaOffset+l)

p = x\*math.cos(o) + y\*math.sin(o)

k = int( DiagonalL - math.floor(p) )

Img[k, l] = Img[k, l] + 1

return Img

def findLines(baseImage, img, Len, Wid, DiagonalL=None, lineType=None):

print(lineType+" LINES : ")

Img = createImage(img, Len, Wid, DiagonalL)

L, W = Img.shape

if (lineType=='VERTICAL'):

noOfLines = noOfVerticalLines

for j in range(0, W):

if (j<85 or j>95):

Img[:, j] = 0

elif (lineType=='DIAGONAL'):

noOfLines = noOfDiagonalLines

for j in range(0, W):

if (j<=126 or j>=128):

Img[:, j] = 0

lines = []

sortedImg = np.sort(Img.flatten())

threshold\_Top = sortedImg[(L\*W)-noOfLines]

for i in range(0, L):

for j in range(0, W):

if ( Img[i,j]>threshold\_Top ):

lines.append( ( (DiagonalL-i), (thetaOffset+j) ) )

print(" Number of lines detected : "+str(len(lines)))

print(" Co-ordinate Values : "+str(lines))

rawLines = np.zeros((Len, Wid))

for line in lines:

p = line[0]

o = line[1]

o = math.radians(o)

cosO = math.cos(o)

sinO = math.sin(o)

if (lineType=='VERTICAL'):

y1 = 0

x1 = int( (p-y1\*sinO)/cosO )

y2 = (Len-1)

x2 = int( (p-y2\*sinO)/cosO )

l(rawLines, (x1,(Len-1)-y1),(x2,(Len-1)-y2),(255,255,255),2)

l(baseImage, (x1,(Len-1)-y1),(x2,(Len-1)-y2),(0,0,255),2)

else:

y1 = 0

x1 = int( (p-y1\*sinO)/cosO )

y2 = (Len-1)

x2 = int( (p-y2\*sinO)/cosO )

l(rawLines, (x1,(Len-1)-y1),(x2,(Len-1)-y2),(255,255,255),2)

l(baseImage, (x1,(Len-1)-y1),(x2,(Len-1)-y2),(255,0,0),2)

return rawLines, baseImage

def main():

img = imread(imgLoc)

Len, Wid, numOfChannels = img.shape

DiagonalL = math.ceil(math.sqrt(Len\*\*2 + Wid\*\*2))

rImg = thresholdImg(img[:,:,0], rThreshold)

gImg = thresholdImg(img[:,:,1], gThreshold)

bImg = thresholdImg(img[:,:,2], bThreshold)

sobelXRImg, sobelYRImg, EDRedLines = edgeDetector(rImg)

sobelXGImg, sobelYGImg, EDGLines = edgeDetector(gImg)

sobelXBImg, sobelYBImg, EDBlueLines = edgeDetector(bImg)

rawLines\_v, baseImage\_v = findLines(img.copy(), EDRedLines, Len, Wid, DiagonalL, 'VERTICAL')

displayImage(rawLines\_v, 'BW\_red\_line')

displayImage(baseImage\_v, 'red\_line')

rawLines\_s, baseImage\_s = findLines(img.copy(), EDBlueLines, Len, Wid, DiagonalL, 'DIAGONAL')

displayImage(rawLines\_s, 'BW\_blue\_line')

displayImage(baseImage\_s, 'blue\_line')

main()



Fig: Detecting Red Lines in Gray scale Image

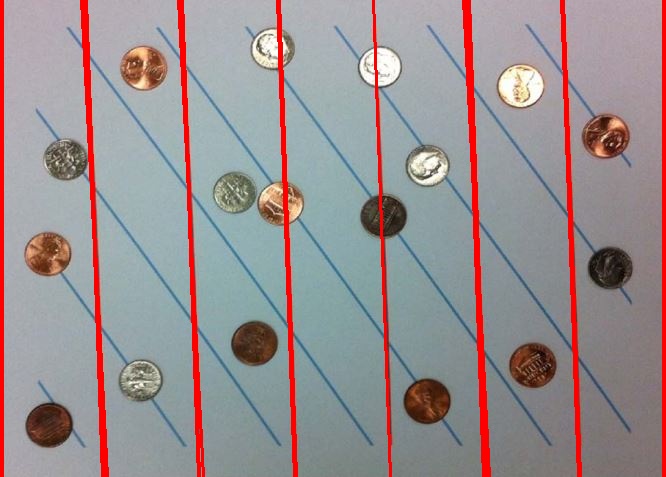


Fig: Detecting Red Lines in Original Image

VERTICALLINES :

**Number of Red lines detected : 24**

Line (p,o) values : [(664, 0), (663, 0), (580, 2), (579, 2), (578, 2), (489, 3), (488, 3), (486, 2), (485, 2), (484, 2), (483, 2), (390, 2), (296, 2), (295, 2), (294, 2), (203, 3), (199, 2), (198, 2), (107, 3), (106, 3), (103, 2), (102, 2), (2, 0), (1, 0)]

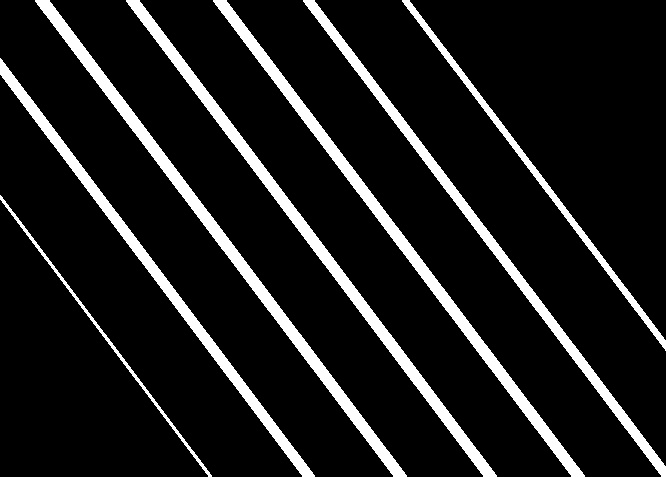


Fig: Detecting blue lines in Gray scale Image

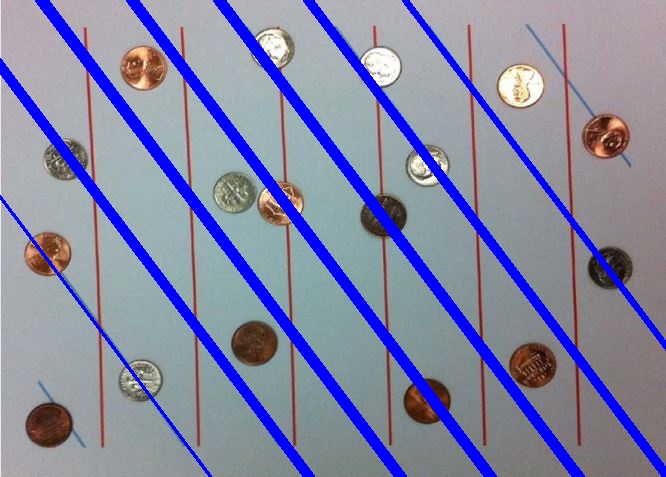


Fig: Detecting Blue Line in Original Image

DIAGONALLINES :

**Number of Blue lines detected : 47**

Line (p,o) values : [(612, 37), (611, 37), (610, 37), (609, 37), (536, 37), (535, 37), (534, 37), (533, 37), (532, 37), (531, 37), (530, 37), (466, 37), (465, 37), (464, 37), (463, 37), (462, 37), (461, 37), (460, 37), (459, 37), (458, 37), (396, 37), (395, 37), (394, 37), (393, 37), (392, 37), (391, 37), (390, 37), (389, 37), (388, 37), (324, 37), (323, 37), (322, 37), (321, 37), (320, 37), (319, 37), (318, 37), (317, 37), (316, 37), (250, 37), (249, 37), (248, 37), (247, 37), (246, 37), (245, 37), (244, 37), (243, 37), (168, 37)]

**Task 3 Bonus Code**

import numpy as np

import math as math

from Utilities import sharpen, blur, edgeDetector

def displayImage(img, outputname):

imwrite('outputImages/'+outputname+'.jpg', img)

return 1

imgLoc = './hough.jpg'

h = 30

thresh = 0

radius = 24

noOfCircle = 80

def imageThreshold(img0, threshold):

img = img0.copy()

for i in range(0, len(img)):

for j in range(0, len(img[0])):

if (img[i,j]<threshold):

img[i,j] = 0

return img

def createImage(img, imgLength, imgWidth):

accImg = np.zeros((imgLength, imgWidth))

for i in range(0, imgLength):

y = i

for j in range(0, imgWidth):

x = j

if(img[i,j]>h):

for l in range(0, 360):

o = math.radians(l)

a = int(x - radius\*math.cos(o))

b = int(y - radius\*math.sin(o))

if (a>=0 and a<imgWidth and b>=0 and b<imgLength):

accImg[b, a] += 1

return accImg

def findCircles(baseImg, img, imgLength, imgWidth):

accImg = createImage(img, imgLength, imgWidth)

accImgLength, accImgWidth = accImg.shape

circleCenters = []

accImgSorted = np.sort(accImg.flatten())

houghTransformLineThreshold\_Top = accImgSorted[(accImgLength\*accImgWidth)-noOfCircle]

for i in range(0, accImgLength):

for j in range(0, accImgWidth):

if ( accImg[i,j]>=houghTransformLineThreshold\_Top ):

circleCenters.append((j,i))

print(" Number of coins detected : "+str(len(circleCenters)))

print(" Co-ordinate values : "+str(circleCenters))

rawImg = np.zeros((imgLength, imgWidth))

for circleCenter in circleCenters:

a = circleCenter[0]

b = circleCenter[1]

center = (a, b)

cir(rawImg, center, radius, (255,255,255), 2)

cir(baseImg, center, radius, (0,0,255), 2)

return rawImg, baseImg

def main():

colorImg = imread(imgLoc)

img = imageThreshold(colorImg[:,:,1], thresh)

imgLength, imgWidth = img.shape

sobelXImg, sobelYImg, edgedetect = edgeDetector(img)

displayImage(edgedetect, "cirlceedgedetection")

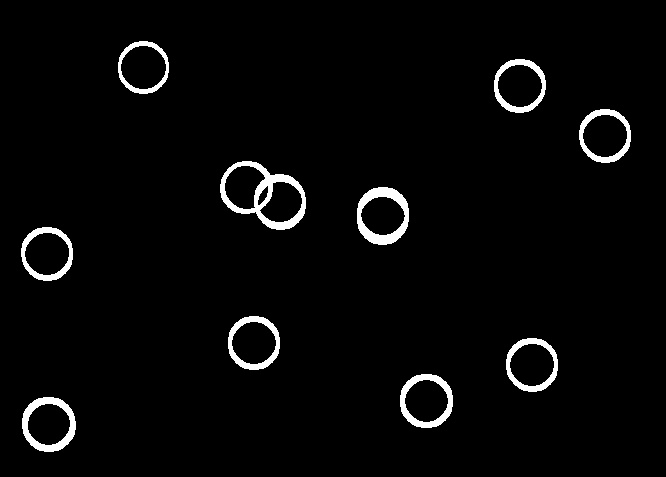
rawCirclesImg, baseCirclesImg = findCircles(colorImg, edgedetect, imgLength, imgWidth)

displayImage(rawCirclesImg, 'coinGrayScale')

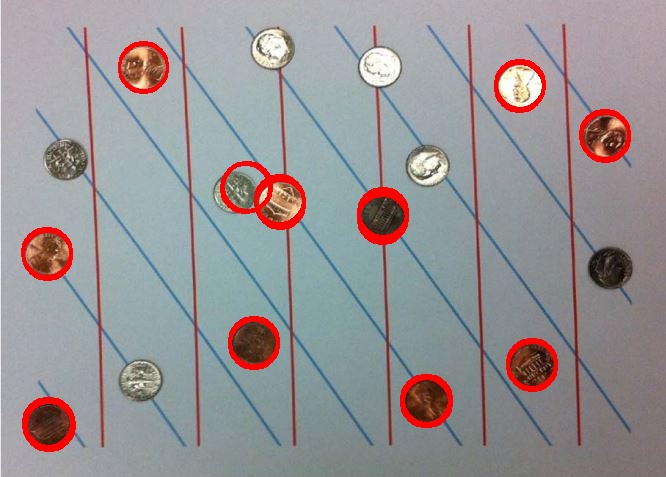
displayImage(baseCirclesImg, 'coin')

main()

**Output:**



**Fig: coins detection in gray scale image**



**Coins detection in original image**

**Number of coins detected : 81**

Co-ordinate values : [(143, 66), (143, 67), (143, 68), (519, 84), (520, 84), (520, 85), (519, 86), (519, 87), (604, 134), (605, 134), (604, 135), (605, 135), (605, 136), (605, 137), (246, 186), (247, 186), (246, 187), (247, 187), (245, 188), (279, 199), (279, 201), (280, 201), (279, 202), (280, 202), (280, 203), (280, 204), (382, 212), (383, 212), (383, 213), (382, 214), (383, 214), (382, 215), (383, 215), (383, 216), (383, 217), (382, 218), (382, 219), (46, 252), (47, 252), (47, 254), (47, 255), (253, 341), (254, 341), (253, 342), (254, 342), (254, 343), (253, 344), (254, 344), (531, 363), (532, 363), (531, 364), (532, 364), (531, 365), (532, 365), (532, 366), (425, 399), (426, 399), (427, 399), (425, 400), (426, 400), (427, 400), (425, 401), (426, 401), (427, 401), (425, 402), (426, 402), (427, 402), (47, 422), (48, 422), (49, 422), (47, 423), (48, 423), (49, 423), (47, 424), (48, 424), (49, 424), (48, 425), (49, 425), (50, 425), (48, 426), (49, 426)]

**References:**

[1] <https://opencv-python-tutroals.readthedocs.io>

[2] <https://techtutorialsx.com>

[3] https://stackoverflow.com/