CSE573

Project 3

Yunchen Yang

UBIT: yuncheny

Person number: 50202449

Project 3

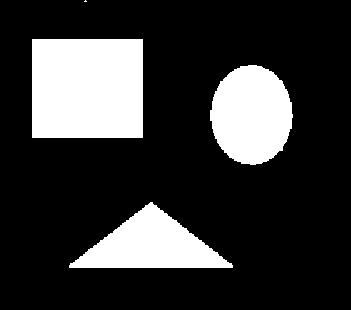
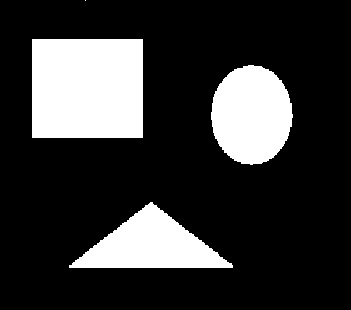
Yunchen Yang

50202449

**Task1**

The opening and closing algorisms are included in the scripts (thecorresponding code file name is “***Project3\_Morphology\_imgpro.py”***). Assuming the original image as “object A”, the processing mask (3×3) as “object B”, two denoising morphology image processing methods, closing followed by opening (Method 1: (A●B)○B) and opening followed by closing (Method 2: (A○B)●B), were used.

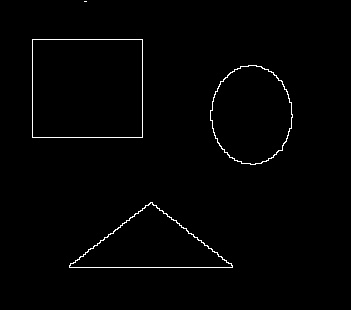
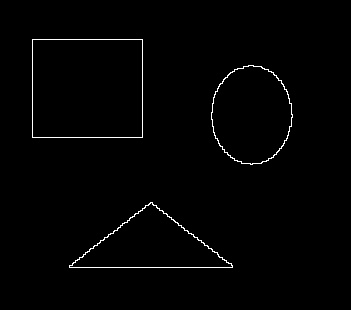
1. Result images generated from two methods were shown below (**Fig 1a, b**).

**a.** Method 1: (A●B)○B **b.** Method 2: (A○B)●B

**Figure 1. Denoising result images**

1. Two denoising methods did not generate same results. For the denoising process on the rectangle objects. The performances of two methods are similar, where the borderlines are quite smooth. However, for the oval objects, which has many outward noises, opening followed by closing (method 2) showed better denoising performance. For the triangle object, which has some tiny inward grooves on the right side, closing followed by opening (method 1) showed better denoising performance. The reason causing the different denosing performance is opening being able to flat the outward peninsular shapes while closing being able to fill the inward corners.
2. Result images of extracted boundaries were shown below (**Fig 2a, b**)

**a.** Method 1: (A●B)○B **b.** Method 2: (A○B)●B

**Figure 2. Boundary extraction result images**

**Source code:**

1. **import** cv2
2. **import** numpy as np
3. **import** copy
4. **import** sys
6. **def** round\_shape(dim):
7. **if** dim%2==0:
8. **print**("dimension of matrix must be odd")
9. **return** None
10. mat=np.zeros([dim,dim])
11. radius=dim/2
12. **for** x **in** range(0,dim):
13. **for** y **in** range(0,dim):
14. **if** pow(((x-(dim-1)/2)\*(x-(dim-1)/2)+(y-(dim-1)/2)\*(y-(dim-1)/2)),0.5)<radius:
15. mat[x,y]=255
16. mat=mat.astype('uint64')
17. **return** mat
19. **def** dilation(img,knl,ori\_pos):
20. img\_pro=copy.deepcopy(img)
21. size\_img=img.shape
22. size\_knl=knl.shape
23. x\_decenter1=ori\_pos[0]
24. x\_decenter2=size\_knl[0]-1-ori\_pos[0]
25. y\_decenter1=ori\_pos[1]
26. y\_decenter2=size\_knl[1]-1-ori\_pos[1]
27. knl\_match=np.zeros(knl.shape).astype('uint64')
28. **for** x **in** range(x\_decenter1,size\_img[0]-x\_decenter2):
29. **for** y **in** range(y\_decenter1,size\_img[1]-y\_decenter2):
30. **for** i **in** range(0,size\_knl[0]):
31. **for** j **in** range(0,size\_knl[1]):
32. knl\_match[i,j]=img[x-x\_decenter1+i,y-y\_decenter1+j]
33. **if** (knl\_match+knl>255).any():
34. img\_pro[x,y]=255
35. **return** img\_pro
37. **def** erosion(img,knl,ori\_pos):
38. img\_pro=copy.deepcopy(img)
39. size\_img=img.shape
40. size\_knl=knl.shape
41. x\_decenter1=ori\_pos[0]
42. x\_decenter2=size\_knl[0]-1-ori\_pos[0]
43. y\_decenter1=ori\_pos[1]
44. y\_decenter2=size\_knl[1]-1-ori\_pos[1]
45. knl\_match=np.zeros(knl.shape).astype('uint64')
46. xlist=np.array(np.where(img==255))[0,:]
47. ylist=np.array(np.where(img==255))[1,:]
48. **for** n **in** range(0,len(xlist)):
49. x=xlist[n]
50. y=ylist[n]
51. **if** x<x\_decenter1 **or** y<y\_decenter1 **or** x+x\_decenter2>size\_img[0]-1 **or** y+y\_decenter2>size\_img[1]-1:
52. **continue**
53. **else**:
54. **for** i **in** range(0,size\_knl[0]):
55. **for** j **in** range(0,size\_knl[1]):
56. knl\_match[i,j]=img[x-x\_decenter1+i,y-y\_decenter1+j]
57. knl\_match=knl\_match\*knl\*(1/255)
58. **if** (knl\_match==knl).all():
59. img\_pro[x,y]=255
60. **else**:
61. img\_pro[x,y]=0
62. **return** img\_pro
64. **def** opening(img,knl,ori\_pos):
65. img\_1=erosion(img,knl,ori\_pos)
66. img\_opening=dilation(img\_1,knl,ori\_pos)
67. **return** img\_opening
69. **def** closing(img,knl,ori\_pos):
70. img\_1=dilation(img,knl,ori\_pos)
71. img\_closing=erosion(img\_1,knl,ori\_pos)
72. **return** img\_closing
74. path=sys.path[0]+'\\'
75. img\_no=cv2.imread(path+"original\_imgs\\noise.jpg",0)
76. knl\_dim=3
77. ori=[int((knl\_dim-1)/2),int((knl\_dim-1)/2)]
78. knl\_round=round\_shape(knl\_dim)
80. res\_noise1=opening(closing(img\_no,knl\_round,ori),knl\_round,ori)
81. res\_noise2=closing(opening(img\_no,knl\_round,ori),knl\_round,ori)
82. res\_bound1=res\_noise1-erosion(res\_noise1,knl\_round,ori)
83. res\_bound2=res\_noise2-erosion(res\_noise2,knl\_round,ori)
85. cv2.imwrite(path+"result\_imgs\\res\_noise1.jpg",res\_noise1)
86. cv2.imwrite(path+"result\_imgs\\res\_noise2.jpg",res\_noise2)
87. cv2.imwrite(path+"result\_imgs\\res\_bound1.jpg",res\_bound1)
88. cv2.imwrite(path+"result\_imgs\\res\_bound2.jpg",res\_bound2)

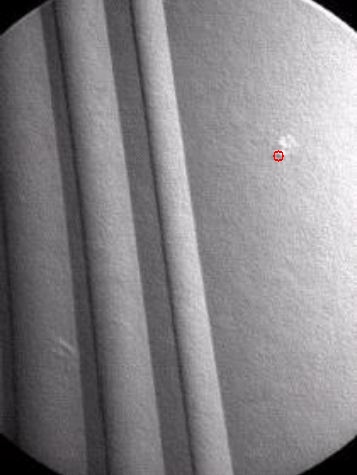
**Task 2**

1. A 5×5 point-detecting mask (**Table 1**) was used to detect the points on the image. The detected points were circled in red and shown in the images below (**Fig 3**). The coordinates of the detected images ([row, column]) are [155, 278] and [156, 278].

Thecorresponding code file name is “***Project3\_*** ***pointdetecting.py”***.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | 24 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |

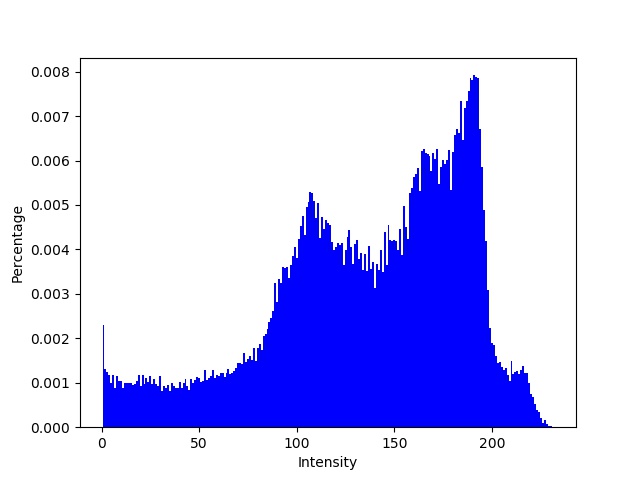
**Table 1. 5×5 point-detecting mask**

****

**Figure 3. Point-detecting result image**

**Source code:**

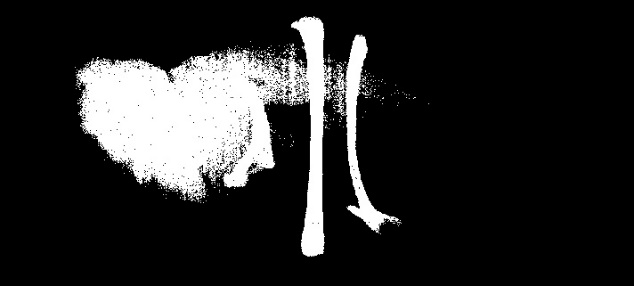
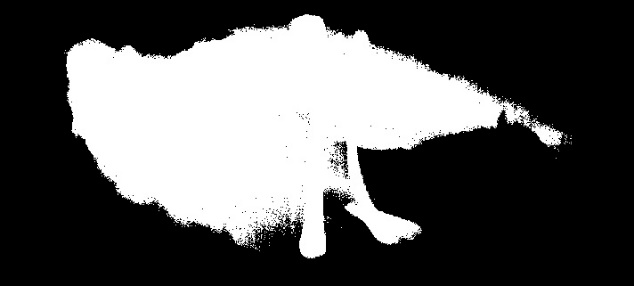
1. **import** cv2
2. **import** numpy as np
3. **import** copy
4. **import** sys
6. **def** point\_detecting(img\_input,filter\_kernel):
7. size\_img\_input=img\_input.shape
8. kernel\_size=filter\_kernel.shape
9. decenter=int((kernel\_size[1]-1)/2)
10. new\_img=np.zeros(img\_input.shape)
11. **for** i **in** range(0,size\_img\_input[0]):
12. **for** j **in** range(0,size\_img\_input[1]):
13. H=np.zeros((kernel\_size[0],kernel\_size[1])).astype('int')
14. G=np.zeros((kernel\_size[0],kernel\_size[1])).astype('int')
15. **for** n **in** range(-decenter,decenter+1):
16. **for** m **in** range(-decenter,decenter+1):
17. **if** i+n **in** range(0,size\_img\_input[0]) **and** j+m **in** range(0,size\_img\_input[1]):
18. H[decenter+n,decenter+m]=img\_input[i+n,j+m]
19. **for** x **in** range(-decenter,decenter+1):
20. **for** y **in** range(-decenter,decenter+1):
21. G[decenter+x,decenter+y]=H[decenter+x,decenter+y]\*filter\_kernel[decenter+x,decenter+y]
22. new\_img[i,j]=sum(sum(G))
23. **else**:
24. new\_img[i,j]=0
25. **return** new\_img
27. **def** thresholding(img\_mat,T):
28. img\_t=np.zeros(img\_mat.shape).astype('uint8')
29. **for** x **in** range(0,img\_mat.shape[0]):
30. **for** y **in** range(0,img\_mat.shape[1]):
31. **if** abs(img\_mat[x,y])>=T:
32. img\_t[x,y]=255
33. **else**:
34. **continue**
35. **return** img\_t
37. path=sys.path[0]+'\\'
38. img=cv2.imread(path+"original\_imgs\\point.jpg",0)
39. img\_color=cv2.imread(path+"original\_imgs\\point.jpg")
40. point\_knl=np.array([[-1,-1,-1,-1,-1],[-1,-1,-1,-1,-1],[-1,-1,24,-1,-1],[-1,-1,-1,-1,-1],[-1,-1,-1,-1,-1]])
41. img\_mask=point\_detecting(img,point\_knl)
42. ratio=(np.max(img\_mask)-np.min(img\_mask))/255
44. img\_thre=thresholding(img\_mask,700)
45. img\_draw=copy.deepcopy(img\_color)
46. point\_coor=np.where(img\_thre==255)
47. **for** num **in** range(len(point\_coor[0])):
48. **if** point\_coor[0][num]>40 **and** point\_coor[0][num]<430 **and** point\_coor[1][num]>5 **and** point\_coor[1][num]<352:
49. img\_draw=cv2.circle(img\_draw,(point\_coor[1][num],point\_coor[0][num]),5,(0,0,255),1)
50. **print**([point\_coor[0][num],point\_coor[1][num]])
51. cv2.imwrite(path+"result\_imgs\\detected\_points.jpg",img\_draw)
52. To Choose an optimal threshold to segment the object from background, a histogram was plotted firstly. As the figure (**Fig 4**)shown, four thresholds were initially selected, 145, 180, 195, 210. The corresponding code file name is “***Project3\_segment\_histplot.py***”.



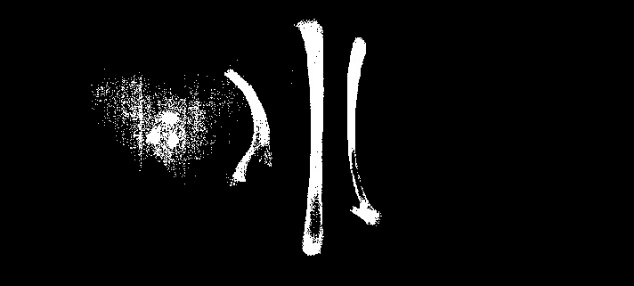
**Figure 4. Histogram of image**

Then the images are segmented using the four selected thresholds. As the figure (**Fig 5 a-d**) shown, Threshold 145 is separating the filet out of the background. Threshold 180 is able to separate the bones and the left shading area of the filet. Threshold 195 and 210 could gradually screen out the shading are of the filet but will also remove some parts of the bones. In this case, the adaptive local thresholding method was used. The image was separated into 8×15 local areas. 195 was selected as the global threshold. For each area, only the pixels showing 13% higher intensities than the average intensity level of the local area will be considered as the real signal. The segmented outcome become much better, where nearly only bones were selected (**Fig 6**). Then the small noises were removed via the denoising morphology image processing method. The bone segmentation was accomplished and shown below (**Fig 7**).

The four corner points of the bounding box ([row, column]) are [14, 249]; [14, 435]; [295, 249] and [295, 435]. The corresponding code file name is “***Project3\_segment.py***”.

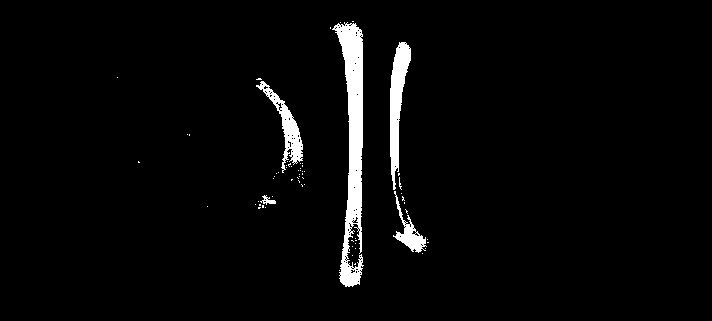


**a.** Threshold: 145  **b.** Threshold: 180

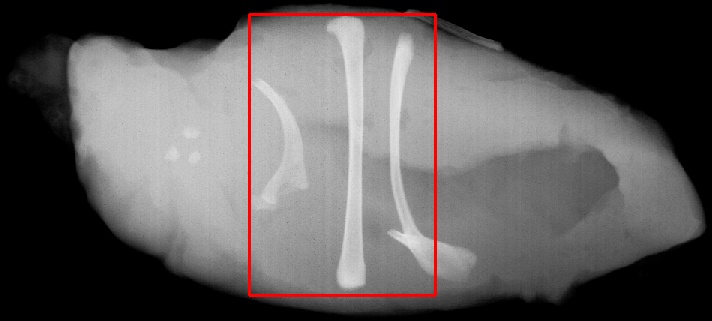


**c.** Threshold: 195  **d.** Threshold: 210

**Figure 5. Images segmented by different thresholds**



**Figure 6. Images segmented by adaptive local thresholding method**

****

**Figure 7. The segmentation result image**

**Source code:**

**Histogram plotting**

1. **import** cv2
2. **import** sys
3. **import** matplotlib.pyplot as plt
5. path=sys.path[0]+'/'
6. img=cv2.imread(path+"original\_imgs/segment.jpg",0)
8. img\_line=tuple(img.reshape(1,-1).astype('int')[0])
9. hist={}
10. **for** n **in** img\_line:
11. hist[n]=hist.get(n,0)+1
12. **del** hist[0]
13. l=len(img\_line)
14. **for** n **in** hist.keys():
15. plt.bar(n,(hist[n]/l),width=1,color='b')
16. plt.xlabel('Intensity')
17. plt.ylabel('Percentage')
18. plt.savefig(path+'result\_imgs\\segment\_histogram.jpg')

**Segmentation**

1. **import** cv2
2. **import** numpy as np
3. **import** sys
4. **import** copy
6. **def** thresholding\_op(img\_mat,T,R,S1,S2):
7. img\_x=img\_mat.shape[0]
8. img\_y=img\_mat.shape[1]
9. gap\_x=int(img\_x/S1)
10. gap\_y=int(img\_y/S2)
11. img\_t=np.zeros(img\_mat.shape).astype('uint8')
12. **for** i **in** range(0,S1):
13. **for** j **in** range(0,S2):
14. **if** i<S1-1 **and** j<S2-1:
15. ind\_x=range(i\*gap\_x,(i+1)\*gap\_x)
16. ind\_y=range(j\*gap\_y,(j+1)\*gap\_y)
17. **elif** i==S1-1 **and** j<S2-1:
18. ind\_x=range(i\*gap\_x,img\_x)
19. ind\_y=range(j\*gap\_y,(j+1)\*gap\_y)
20. **elif** i<S1-1 **and** j==S2-1:
21. ind\_x=range(i\*gap\_x,(i+1)\*gap\_x)
22. ind\_y=range(j\*gap\_y,img\_y)
23. **elif** i==S1-1 **and** j==S2-1:
24. ind\_x=range(i\*gap\_x,img\_x)
25. ind\_y=range(j\*gap\_y,img\_y)
26. point=[]
27. **for** x **in** ind\_x:
28. **for** y **in** ind\_y:
29. point.append(np.array([x,y,img\_mat[x,y]]))
30. ave=np.mean(np.array(list(filter((**lambda** x:x[2]<T),point)))[:,2])
31. point\_select=list(filter((**lambda** x:x[2]>=T),point))
32. **if** point\_select!=None:
33. **for** n **in** range(0,len(point\_select)):
34. **if** point\_select[n][2]>=ave\*(1+R):
35. img\_t[point\_select[n][0],point\_select[n][1]]=255
36. **else**:
37. **continue**
38. **else**:
39. **continue**
40. **return** img\_t
42. **def** thresholding(img\_mat,T):
43. img\_t=np.zeros(img\_mat.shape).astype('uint8')
44. **for** x **in** range(0,img\_mat.shape[0]):
45. **for** y **in** range(0,img\_mat.shape[1]):
46. **if** img\_mat[x,y]>=T:
47. img\_t[x,y]=255
48. **else**:
49. **continue**
50. **return** img\_t
52. **def** round\_shape(dim):
53. **if** dim%2==0:
54. **print**("dimension of matrix must be odd")
55. **return** None
56. mat=np.zeros([dim,dim])
57. radius=dim/2
58. **for** x **in** range(0,dim):
59. **for** y **in** range(0,dim):
60. **if** pow(((x-(dim-1)/2)\*(x-(dim-1)/2)+(y-(dim-1)/2)\*(y-(dim-1)/2)),0.5)<radius:
61. mat[x,y]=255
62. mat=mat.astype('uint64')
63. **return** mat
65. **def** dilation(img,knl,ori\_pos):
66. img\_pro=copy.deepcopy(img)
67. size\_img=img.shape
68. size\_knl=knl.shape
69. x\_decenter1=ori\_pos[0]
70. x\_decenter2=size\_knl[0]-1-ori\_pos[0]
71. y\_decenter1=ori\_pos[1]
72. y\_decenter2=size\_knl[1]-1-ori\_pos[1]
73. knl\_match=np.zeros(knl.shape).astype('uint64')
74. **for** x **in** range(x\_decenter1,size\_img[0]-x\_decenter2):
75. **for** y **in** range(y\_decenter1,size\_img[1]-y\_decenter2):
76. **for** i **in** range(0,size\_knl[0]):
77. **for** j **in** range(0,size\_knl[1]):
78. knl\_match[i,j]=img[x-x\_decenter1+i,y-y\_decenter1+j]
79. **if** (knl\_match+knl>255).any():
80. img\_pro[x,y]=255
81. **return** img\_pro
83. **def** erosion(img,knl,ori\_pos):
84. img\_pro=copy.deepcopy(img)
85. size\_img=img.shape
86. size\_knl=knl.shape
87. x\_decenter1=ori\_pos[0]
88. x\_decenter2=size\_knl[0]-1-ori\_pos[0]
89. y\_decenter1=ori\_pos[1]
90. y\_decenter2=size\_knl[1]-1-ori\_pos[1]
91. knl\_match=np.zeros(knl.shape).astype('uint64')
92. xlist=np.array(np.where(img==255))[0,:]
93. ylist=np.array(np.where(img==255))[1,:]
94. **for** n **in** range(0,len(xlist)):
95. x=xlist[n]
96. y=ylist[n]
97. **if** x<x\_decenter1 **or** y<y\_decenter1 **or** x+x\_decenter2>size\_img[0]-1 **or** y+y\_decenter2>size\_img[1]-1:
98. **continue**
99. **else**:
100. **for** i **in** range(0,size\_knl[0]):
101. **for** j **in** range(0,size\_knl[1]):
102. knl\_match[i,j]=img[x-x\_decenter1+i,y-y\_decenter1+j]
103. knl\_match=knl\_match\*knl\*(1/255)
104. **if** (knl\_match==knl).all():
105. img\_pro[x,y]=255
106. **else**:
107. img\_pro[x,y]=0
108. **return** img\_pro
110. **def** opening(img,knl,ori\_pos):
111. img\_1=erosion(img,knl,ori\_pos)
112. img\_opening=dilation(img\_1,knl,ori\_pos)
113. **return** img\_opening
115. **def** closing(img,knl,ori\_pos):
116. img\_1=dilation(img,knl,ori\_pos)
117. img\_closing=erosion(img\_1,knl,ori\_pos)
118. **return** img\_closing
120. path=sys.path[0]+'/'
121. img=cv2.imread(path+"original\_imgs/segment.jpg",0)
122. img\_color=cv2.imread(path+"original\_imgs/segment.jpg")
123. img\_thre145=thresholding(img,145)
124. img\_thre180=thresholding(img,180)
125. img\_thre195=thresholding(img,195)
126. img\_thre210=thresholding(img,210)
127. img\_thre\_op=thresholding\_op(img,195,0.13,8,15)
129. knl=round\_shape(3)
130. img\_threopop=closing(opening(img\_thre\_op,knl,[1,1]),knl,[1,1])

133. x\_ind=np.where(img\_threopop==255)[0]
134. y\_ind=np.where(img\_threopop==255)[1]
135. x\_min=min(x\_ind)-10
136. y\_min=min(y\_ind)-10
137. x\_max=max(x\_ind)+10
138. y\_max=max(y\_ind)+10
139. img\_seg=cv2.rectangle(img\_color,(y\_min,x\_min),(y\_max,x\_max),(0,0,255),2)
141. cv2.imwrite(path+"result\_imgs\\img\_thre145.jpg",img\_thre145)
142. cv2.imwrite(path+"result\_imgs\\img\_thre180.jpg",img\_thre180)
143. cv2.imwrite(path+"result\_imgs\\img\_thre195.jpg",img\_thre195)
144. cv2.imwrite(path+"result\_imgs\\img\_thre210.jpg",img\_thre210)
146. cv2.imwrite(path+"result\_imgs\\img\_adaptive\_thresholding.jpg",img\_thre\_op)
147. **print**([x\_min,y\_min])
148. **print**([x\_min,y\_max])
149. **print**([x\_max,y\_min])
150. **print**([x\_max,y\_max])
151. cv2.imwrite(path+"result\_imgs\\img\_segmentation.jpg",img\_seg)

**Task3**

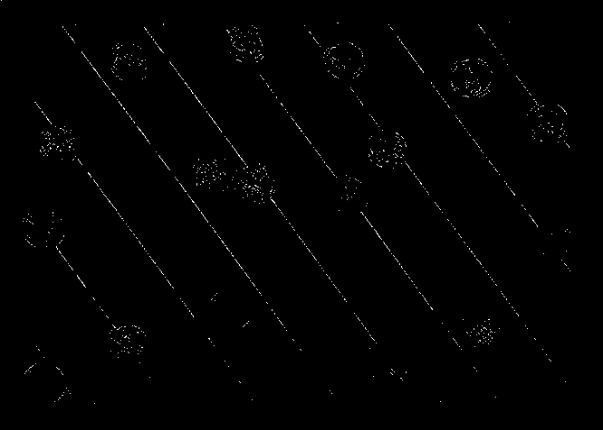
In order to separate the red lines and blue lines for independent line detection methods, a vertical and a -45° line-detecting masks were generated (**Table 2 a, b**). As the figure shown (**Fig 8 a, b**), the vertical blue lines and the inclined red lines are recognized by different masks and shown in different maps. Two maps will be used for the detection of red line and blue line respectively, via Hough Transform.

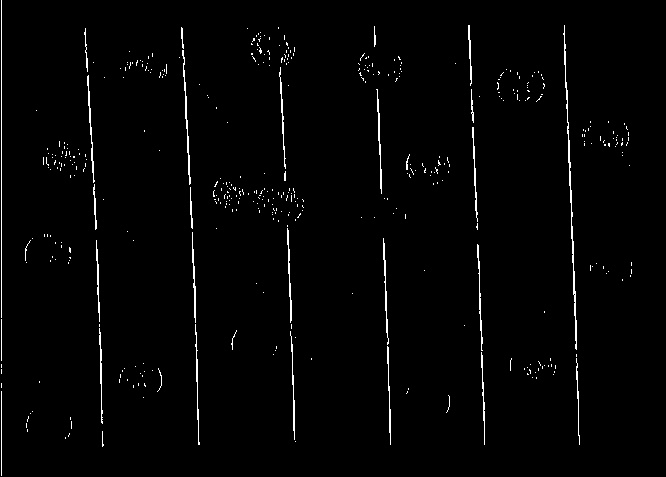
|  |  |  |
| --- | --- | --- |
| -1 | 2 | -1 |
| -1 | 2 | -1 |
| -1 | 2 | -1 |

**Table 2a. Vertical line-detecting mask**

|  |  |  |
| --- | --- | --- |
| 2 | -1 | -1 |
| -1 | 2 | -1 |
| -1 | -1 | 2 |

**Table 2b. -45° line-detecting mask**





**a.** Vertical binary map **b.** -45° binary map

**Figure 8. Binary maps generated from the mask**

* 1. The red lines recognized by Hough Transform was shown below (**Fig 9**). The detected lines were shown in green. Totally 7 red lines were detected. All the 6 red lines were dteected. One false-positively detected line localized at the left border of the image. In the original image, there is a thin white line showing at the left border of the images, which is the reason of the false-positively detected result.



**Figure 9. Detected red lines**

* 1. The blue lines recognized by Hough Transform was shown below (**Fig 11**). Totally 8 red lines were detected. The detected lines were shown in green. A short line got missed during the detecting process. The short length of the line and the relatively larger portion of overlapping between the coin and the line are the major reason causing the missed detection.



**Figure 11. Detected red lines**

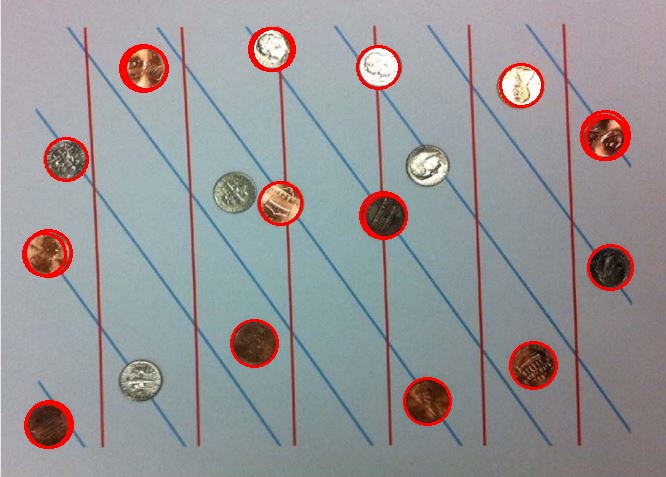
For the line detection, thecorresponding code file name is “***Project3\_houghtrans.py***”.

**Source code:**

1. **import** cv2
2. **import** numpy as np
3. **import** sys
4. **import** math
5. **import** copy
7. **def** line\_detecting(img\_input,filter\_kernel):
8. size\_img\_input=img\_input.shape
9. kernel\_size=filter\_kernel.shape
10. decenter=int((kernel\_size[1]-1)/2)
11. new\_img=np.zeros(img\_input.shape)
12. **for** i **in** range(0,size\_img\_input[0]):
13. **for** j **in** range(0,size\_img\_input[1]):
14. H=np.zeros((kernel\_size[0],kernel\_size[1])).astype('int')
15. G=np.zeros((kernel\_size[0],kernel\_size[1])).astype('int')
16. **for** n **in** range(-decenter,decenter+1):
17. **for** m **in** range(-decenter,decenter+1):
18. **if** i+n **in** range(0,size\_img\_input[0]) **and** j+m **in** range(0,size\_img\_input[1]):
19. H[decenter+n,decenter+m]=img\_input[i+n,j+m]
20. **for** x **in** range(-decenter,decenter+1):
21. **for** y **in** range(-decenter,decenter+1):
22. G[decenter+x,decenter+y]=H[decenter+x,decenter+y]\*filter\_kernel[decenter+x,decenter+y]
23. new\_img[i,j]=sum(sum(G))
24. **else**:
25. new\_img[i,j]=0
26. **return** new\_img
28. **def** thresholding(img\_mat,T,direction):
29. img\_t=np.zeros(img\_mat.shape).astype('uint8')
30. **if** direction==1:
31. **for** x **in** range(0,img\_mat.shape[0]):
32. **for** y **in** range(0,img\_mat.shape[1]):
33. **if** abs(img\_mat[x,y])>=T:
34. img\_t[x,y]=255
35. **else**:
36. **continue**
37. **elif** direction==-1:
38. **for** x **in** range(0,img\_mat.shape[0]):
39. **for** y **in** range(0,img\_mat.shape[1]):
40. **if** abs(img\_mat[x,y])<=T:
41. img\_t[x,y]=255
42. **else**:
43. **continue**
44. **else**:
45. **for** x **in** range(0,img\_mat.shape[0]):
46. **for** y **in** range(0,img\_mat.shape[1]):
47. **if** abs(img\_mat[x,y])<=T[1] **and** abs(img\_mat[x,y])>=T[0]:
48. img\_t[x,y]=255
49. **else**:
50. **continue**
51. **return** img\_t
53. **def** score2map(img\_score):
54. ratio=(np.max(img\_score)-np.min(img\_score))/255
55. img\_map=((img\_score-np.min(img\_score))/ratio).astype('uint8')
56. **return** img\_map
58. **def** hough\_func(theta,x,y):
59. X=x+1
60. Y=y+1
61. rho=math.cos(theta)\*X+math.sin(theta)\*Y
62. **return** rho
64. **def** hough\_votemap(bimap,step):
65. coor=np.where(bimap==255)
66. coor\_x=coor[0]
67. coor\_y=coor[1]
68. num=len(coor\_x)
69. A=int(math.sqrt(bimap.shape[0]\*bimap.shape[0]+bimap.shape[1]\*bimap.shape[1]))
70. vote=np.zeros([A\*2,step]).astype('int')
71. **for** n **in** range(1,step):
72. **for** m **in** range(0,num):
73. r=hough\_func((math.pi/2)\*(3\*n/step-1),coor\_x[m],coor\_y[m])
74. vote[int(r+A),n]=vote[int(r+A),n]+1
75. **return** vote
77. **def** hough\_draw(img,bimap,votemap,T,t\_rho,t\_theta,step):
78. votemap\_noedge=votemap[1:votemap.shape[0]-1,1:votemap.shape[1]-1]
79. pl\_coor=np.where(votemap\_noedge>T)
80. img\_draw=copy.deepcopy(img)
81. A=int(math.sqrt(pow(img.shape[0],2)+pow(img.shape[1],2)))
82. parameter=[]
83. **for** cnt **in** range(len(pl\_coor[0])):
84. x=pl\_coor[0][cnt]+1
85. y=pl\_coor[1][cnt]+1
86. localmat=votemap[x-1:x+2,y-1:y+2]
87. **if** votemap[x,y]==np.max(localmat):
88. rho=x+0.5-A
89. theta=(math.pi/2)\*(3\*y/step-1)
90. parameter.append([rho,theta,votemap[x,y]])
91. **else**:
92. **continue**
93. selected=[]
94. **for** num **in** range(0,len(parameter)):
95. **if** len(selected)==0:
96. selected.append(parameter[num])
97. **else**:
98. **for** cnt **in** range(0,len(selected)):
99. **if** parameter[num][0]>selected[cnt][0]-t\_rho **and** parameter[num][0]<selected[cnt][0]+t\_rho \
100. **and** parameter[num][1]>selected[cnt][1]-t\_theta **and** parameter[num][1]<selected[cnt][1]+t\_theta:
101. **if** parameter[num][2]>selected[cnt][2]:
102. selected[cnt]=parameter[num]
103. **break**
104. **else**:
105. selected.append(parameter[num])
106. **for** dn **in** range(0,len(selected)):
107. rho=selected[dn][0]
108. theta=selected[dn][1]
109. **if** math.sin(theta)==0:
110. X1=int(rho/math.cos(theta))
111. X2=X1
112. Y1=1
113. Y2=img.shape[1]+1
114. **else**:
115. X1=1
116. X2=img.shape[0]+1
117. Y1=int((rho-math.cos(theta)\*X1)/math.sin(theta))
118. Y2=int((rho-math.cos(theta)\*X2)/math.sin(theta))
119. img\_draw=cv2.line(img\_draw,(Y1-1,X1-1),(Y2-1,X2-1),(0,255,0),2)
120. **return** img\_draw

123. path=sys.path[0]+'\\'
124. img=cv2.imread(path+"original\_imgs\\hough.jpg",0)
125. img\_color=cv2.imread(path+"original\_imgs\\hough.jpg")
126. knl\_neg45=np.array([[2,-1,-1],[-1,2,-1],[-1,-1,2]])
127. knl\_vertical=np.array([[-1,2,-1],[-1,2,-1],[-1,2,-1]])
128. score\_neg45=line\_detecting(img,knl\_neg45)
129. score\_vertical=line\_detecting(img,knl\_vertical)
130. map\_neg45=score2map(score\_neg45)
131. map\_vertical=score2map(score\_vertical)
132. bimap\_neg45=thresholding(map\_neg45,[70,80],0)
133. bimap\_vertical=thresholding(map\_vertical,[50,70],0)
135. votemap\_neg45=hough\_votemap(bimap\_neg45,540)
136. img\_neg45=hough\_draw(img\_color,bimap\_neg45,votemap\_neg45,30,15,math.pi/30,540)
137. cv2.imwrite(path+"result\_imgs\\blue\_lines.jpg",img\_neg45)
139. votemap\_vertical=hough\_votemap(bimap\_vertical,300)
140. img\_vertical=hough\_draw(img\_color,bimap\_vertical,votemap\_vertical,180,15,math.pi/30,300)
141. cv2.imwrite(path+"result\_imgs\\red\_lines.jpg",img\_vertical)
143. cv2.imwrite(path+"result\_imgs\\bimap\_blue\_lines.jpg",bimap\_neg45)
144. cv2.imwrite(path+"result\_imgs\\bimap\_red\_lines.jpg",bimap\_vertical)
     1. To detect the coins in the images, Hough Transform was also used to detect the round shape edges. The radius range scanned by the algorism was set as [20, 30) pixels. The detected round circles were labeled in red. As the figure shown (**Fig 12**), totally 14 coins have been detected. 3 coins were missed. The reason of the missed detection are the irregular or uncomplete shape of the detected edges of those coins due to the shading effect, and the relatively more complicate inner structures.

The corresponding code file name is “***Project3\_houghtrans\_round.py***”.



**Figure 12. Detected coins**

**Source codes**

1. **import** cv2
2. **import** numpy as np
3. **import** sys
4. **import** math
5. **import** copy
7. **def** round\_hough(bimap,R1,R2):
8. coor=np.where(bimap==255)
9. x=bimap.shape[0]
10. y=bimap.shape[1]
11. vote=np.zeros([x,y,(R2-R1)]).astype('int')
12. **for** n **in** range(0,x):
13. **for** m **in** range(0,y):
14. **for** cnt **in** range(0,len(coor[0])):
15. x1=coor[0][cnt]
16. y1=coor[1][cnt]
17. r=math.sqrt(pow((n-x1),2)+pow((m-y1),2))
18. **if** round(r)<R2 **and** round(r)>=R1:
19. vote[n,m,(round(r)-R1)]=vote[n,m,(round(r)-R1)]+1
20. **else**:
21. **continue**
22. **return** vote
24. **def** hough\_draw\_round(img,votemap,R1,R2,T,t\_a,t\_b,t\_r):
25. votemap\_noedge=votemap[1:votemap.shape[0]-1,1:votemap.shape[1]-1,1:votemap.shape[2]-1]
26. pl\_coor=np.where(votemap\_noedge>T)
27. img\_draw=copy.deepcopy(img)
28. parameter=[]
29. **for** cnt **in** range(0,len(pl\_coor[2])):
30. x=pl\_coor[0][cnt]+1
31. y=pl\_coor[1][cnt]+1
32. z=pl\_coor[2][cnt]+1
33. localmat=votemap[x-1:x+2,y-1:y+2,z-1:z+2]
34. **if** votemap[x,y,z]==np.max(localmat):
35. parameter.append([x,y,z+R1])
36. **else**:
37. **continue**
38. selected=[]
39. **for** num **in** range(0,len(parameter)):
40. **if** len(selected)==0:
41. selected.append(parameter[num])
42. **else**:
43. **for** cnt **in** range(0,len(selected)):
44. **if** parameter[num][0]>selected[cnt][0]-t\_a **and** parameter[num][0]<selected[cnt][0]+t\_a \
45. **and** parameter[num][1]>selected[cnt][1]-t\_b **and** parameter[num][1]<selected[cnt][1]+t\_b\
46. **and** parameter[num][2]>selected[cnt][1]-t\_r **and** parameter[num][1]<selected[cnt][1]+t\_r:
47. **if** parameter[num][2]>selected[cnt][2]:
48. selected[cnt]=parameter[num]
49. **break**
50. **else**:
51. selected.append(parameter[num])
52. **for** dn **in** range(0,len(selected)):
53. img\_draw=cv2.circle(img\_draw,(selected[dn][1],selected[dn][0]),selected[dn][2],(0,0,255),2)
54. **return** img\_draw
56. path=sys.path[0]+'\\'
57. img=cv2.imread(path+"original\_imgs\\hough.jpg",0)
58. img\_color=cv2.imread(path+"original\_imgs\\hough.jpg")
59. img\_blur=cv2.GaussianBlur(img,(7,7),1)
60. map\_edge=cv2.Canny(img\_blur,100,250)
62. votemap\_edge=round\_hough(map\_edge,20,30)
63. img\_d=hough\_draw\_round(img\_color,votemap\_edge,20,30,50,5,5,5)
64. cv2.imwrite(path+"result\_imgs\\coin.jpg",img\_d)