

Fundamentals of Biomedical Image Processing HW 3

B12508026 戴偉璿

November 12, 2025

1 Theoretical Questions

To detect a 1-pixel break in a binary line, we can use a directional filter such as $[1 \ -2 \ 1]$. When applied to a continuous line $(1 \ 1 \ 1)$, the filter output is zero; when applied to the edge of the break $(1 \ 1 \ 0)$, the output is negative (-1) ; when applied to a break $(1 \ 0 \ 1)$, the output becomes positive (2) . Therefore, pixels with nonzero responses indicate line breaks.

Here are the filters that can detect the 1-pixel break in different directions:

- Vertical:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

- Horizontal:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

- +45 degree:

$$A = \begin{bmatrix} 0 & 0 & 1 \\ 0 & -2 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

- -45 degree:

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2 Programming Exercises

1. I utilized BFS algorithm to label all connected components in the binary image, and calculated the area of each components. With these areas, I sorted them in descending order and identified the largest gap between consecutive areas to determine a threshold. Components with areas below this threshold were removed from the original image. The resulting image, with small connected components removed, is shown in Figure 1. The components below the threshold are highlighted in gray.

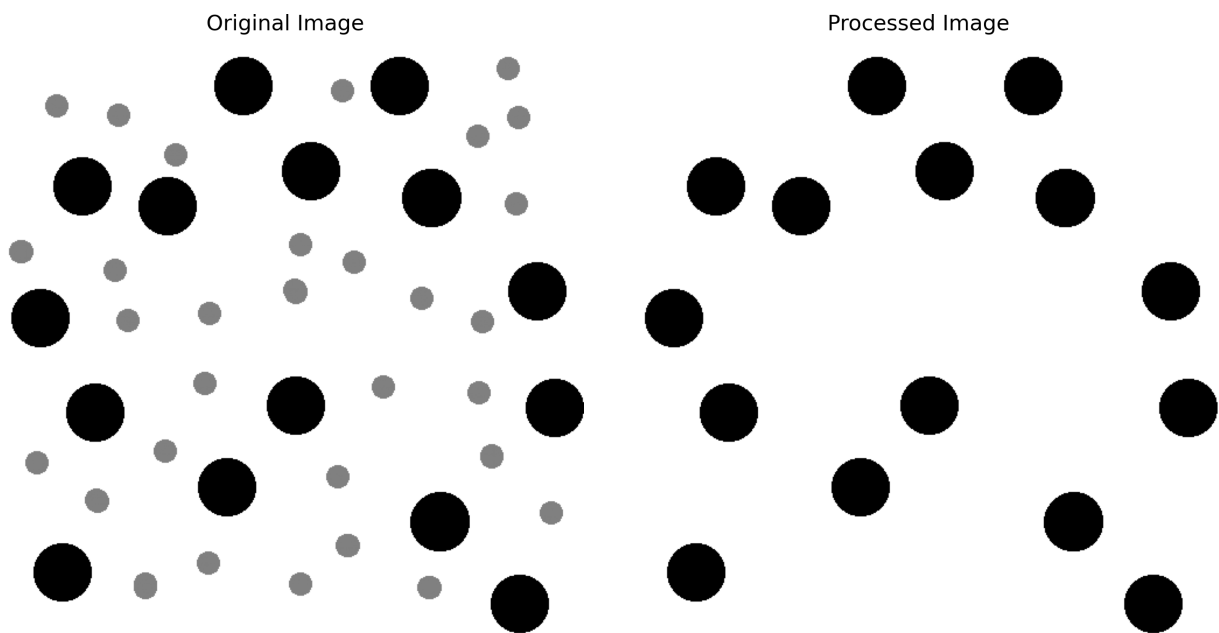


Figure 1: Result of removing small connected components

Following is the code used for this task:

```

1  import os
2  import cv2
3  import numpy as np
4  import matplotlib
5  matplotlib.use('Agg')
6  from matplotlib import pyplot as plt
7
8  os.makedirs('result/p1', exist_ok=True)
9
10 # Read image
11 img_ori = cv2.imread('../src/Fig1.tif', 0)
12 if img_ori is None:
13     raise ValueError("Image not found or unable to load.")
14
15 img = (img_ori < 128).astype(np.uint8) * 255 # Binarize
16 print(img.shape)
17
18 H, W = img.shape
19 dirs = [(-1, -1), (-1, 0), (-1, 1),

```

```

20         ( 0, -1),          ( 0, 1),
21         ( 1, -1), ( 1,  0), ( 1, 1)]
22 visited = np.zeros((H, W), dtype=bool)
23 label_map = np.zeros((H, W), dtype=int)
24 areas = []
25 label = 0
26
27 # BFS for connected components
28 for i in range(H):
29     for j in range(W):
30         if img[i, j] and visited[i, j] == 0:
31             label += 1
32             area = 0
33             stack = [(i, j)]
34             while stack:
35                 x, y = stack.pop()
36                 if visited[x, y]:
37                     continue
38                 visited[x, y] = 1
39                 label_map[x, y] = label
40                 area += 1
41                 for dx, dy in dirs:
42                     nx, ny = x + dx, y + dy
43                     if 0 <= nx < H and 0 <= ny < W:
44                         if img[nx, ny] and not visited[nx, ny]:
45                             stack.append((nx, ny))
46             areas.append(area)
47
48 print(f'Total connected components: {label}')
49
50 # Find the threshold to remove small components
51 areas_sorted = sorted(areas, reverse=True)
52 max_dist = 0
53 max_thresh = 0
54
55 for i in range(label-1):
56     dist = abs(areas_sorted[i] - areas_sorted[i+1])
57     if dist > max_dist:
58         max_dist = dist
59     max_thresh = (areas_sorted[i] + areas_sorted[i+1]) / 2
60
61 removed = np.zeros(label, dtype=bool)
62 for i in range(label):
63     removed[i] = areas[i] < max_thresh
64
65 # Create new image with small components removed
66 img_new = img_ori.copy()
67 pre_remove = img_ori.copy()
68
69 # generate a binary mask for removed components
70 mask = (label_map > 0) & removed[label_map - 1]
71 img_new[mask] = 255
72 pre_remove[mask] = 128
73
74
75 plt.figure(figsize=(10, 5))
76 plt.subplot(1, 2, 1)
77 plt.title('Original Image')
78 plt.imshow(pre_remove, cmap='gray')

```

```
79 plt.axis('off')
80
81 plt.subplot(1, 2, 2)
82 plt.title('Processed Image')
83 plt.imshow(img_new, cmap='gray')
84 plt.axis('off')
85 plt.tight_layout()
86 plt.savefig('result/p1/1.png', dpi=300)
```

2. (a) Set a threshold to binarize the image. Grey level above the threshold is set to 255, otherwise set to 0. The binarized image with threshold 128 is shown in Figure 2.

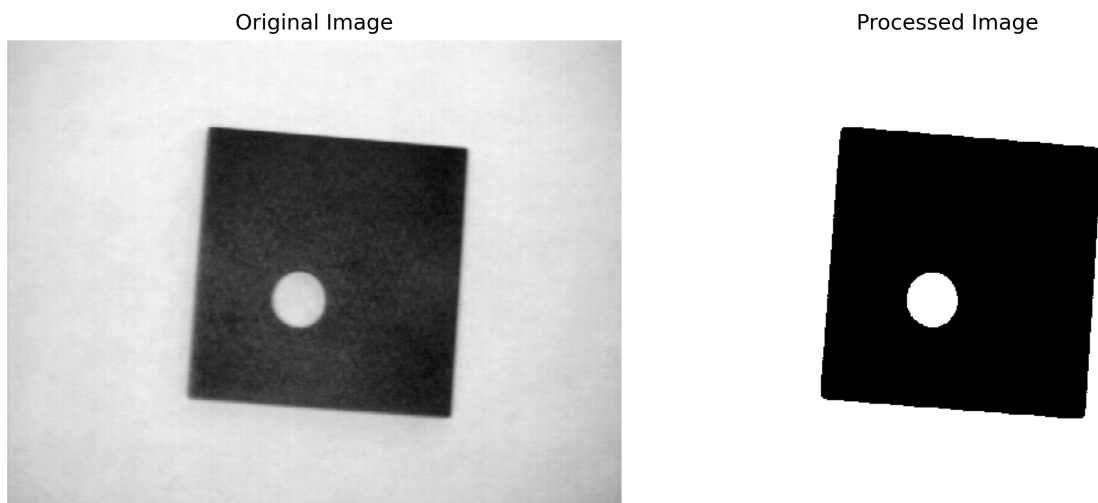


Figure 2: Binarized image with threshold 128

Following is the code used for this task:

```
1 import os
2 import cv2
3 import numpy as np
4 import matplotlib
5 matplotlib.use('Agg')
6 from matplotlib import pyplot as plt
7
8 os.makedirs('result/p2', exist_ok=True)
9
10 # Read image
11 img_ori = cv2.imread('../src/Fig2.gif', 0)
12 if img_ori is None:
13     raise ValueError("Image not found or unable to load.")
14
15 # Input threshold
16 thresh = input("Enter threshold value (0-255): ")
17 try:
18     thresh = int(thresh)
19     if not (0 <= thresh <= 255):
```

```

20         raise ValueError
21     except ValueError:
22         raise ValueError("Invalid threshold value. Please enter an integer between 0 and 255.")
23
24     # Binarize image based on threshold
25     img = np.zeros_like(img_ori, dtype=np.uint8)
26     img[img_ori >= thresh] = 255
27
28     plt.figure(figsize=(10, 5))
29     plt.subplot(1, 2, 1)
30     plt.title('Original Image')
31     plt.imshow(img_ori, cmap='gray')
32     plt.axis('off')
33
34     plt.subplot(1, 2, 2)
35     plt.title('Processed Image')
36     plt.imshow(img, cmap='gray')
37     plt.axis('off')
38     plt.tight_layout()
39     plt.savefig('result/p2/2a.png', dpi=300)

```

- (b) Apply erosion to the binarized image to get the eroded image. Then, subtract the eroded image from the original binary image to get the edge-detected image. The detected edge image is shown in Figure 3.

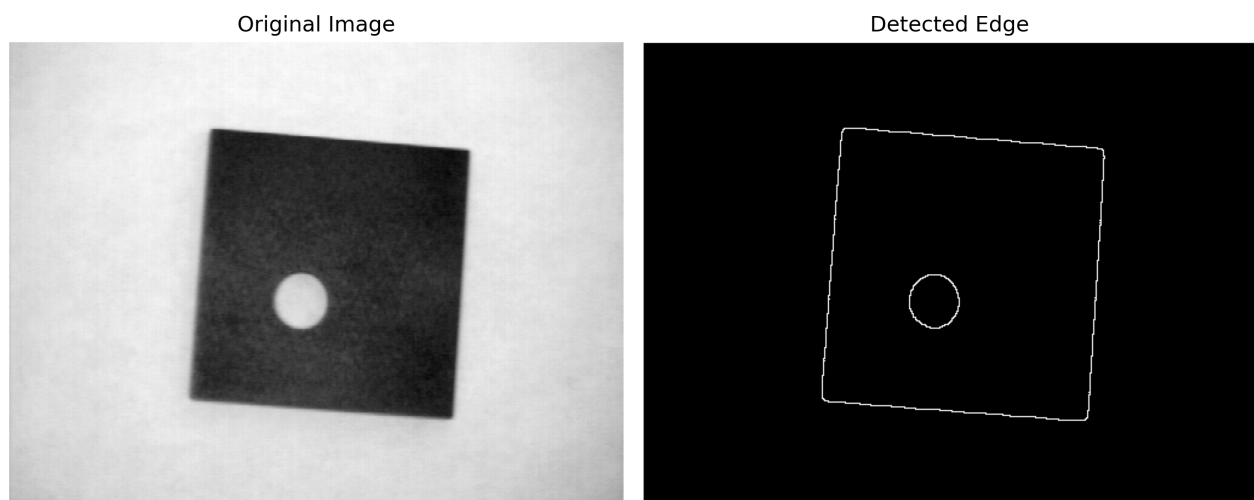


Figure 3: Detected edge image

Following is the code used for this task:

```

1  import os
2  import cv2
3  import numpy as np
4  import matplotlib
5  matplotlib.use('Agg')
6  from matplotlib import pyplot as plt

```

```
7
8 os.makedirs('result/p2', exist_ok=True)
9
10 # Read image
11 img_ori = cv2.imread('../src/Fig2.gif', 0)
12 if img_ori is None:
13     raise ValueError("Image not found or unable to load.")
14
15 thresh = 128
16 img = np.zeros_like(img_ori, dtype=np.uint8)
17 img[img_ori >= thresh] = 255
18
19 # Erosion
20 eroded = np.zeros_like(img, dtype=np.uint8)
21 H, W = img.shape
22 kernel = np.ones((3, 3), np.uint8)
23 for i in range(1, H-1):
24     for j in range(1, W-1):
25         region = img[i-1:i+2, j-1:j+2]
26         if np.all(region == 255):
27             eroded[i, j] = 255
28
29 edge = img - eroded
30
31 plt.figure(figsize=(10, 5))
32 plt.subplot(1, 2, 1)
33 plt.title('Original Image')
34 plt.imshow(img_ori, cmap='gray')
35 plt.axis('off')
36
37 plt.subplot(1, 2, 2)
38 plt.title('Detected Edge')
39 plt.imshow(edge, cmap='gray')
40 plt.axis('off')
41 plt.tight_layout()
42 plt.savefig('result/p2/2b.png', dpi=300)
```

- (c) Utilize the edge image created in part (b) to perform Hough Transform for line detection. In purpose of reduce the computational load, I set the range of the circle radius r from 10 to 30, the step size r_{step} to 2, and the number of angular bins θ_{res} to 60. The detected lines are then superimposed on the original image, as shown in Figure 4.

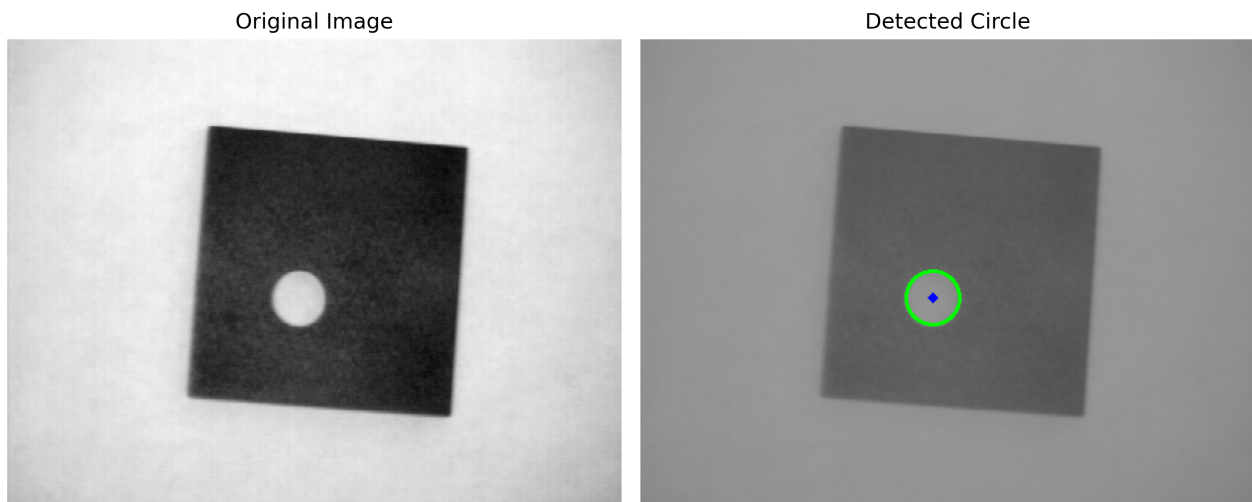


Figure 4: Detected circle on original image

Following is the code used for this task:

```

1  import os
2  import cv2
3  import numpy as np
4  import matplotlib
5  matplotlib.use('Agg')
6  from matplotlib import pyplot as plt
7  from math import cos, sin, pi
8
9  os.makedirs('result/p2', exist_ok=True)
10
11 # Read image
12 img_ori = cv2.imread('../src/fig2.gif', 0)
13 if img_ori is None:
14     raise ValueError("Image not found or unable to load.")
15
16 # Get the edge of the image
17 thresh = 128
18 img = np.zeros_like(img_ori, dtype=np.uint8)
19 img[img_ori >= thresh] = 255
20
21 eroded = np.zeros_like(img, dtype=np.uint8)
22 H, W = img.shape
23 kernel = np.ones((3, 3), np.uint8)
24 for i in range(1, H-1):
25     for j in range(1, W-1):

```

```

26         region = img[i-1:i+2, j-1:j+2]
27         if np.all(region == 255):
28             eroded[i, j] = 255
29
30 edge = img - eroded
31
32 # Set the Hough Transform parameters
33 min_r, max_r, r_step, theta_res = 10, 30, 2, 60
34 acc = np.zeros((H, W, (max_r - min_r)//r_step), dtype=np.uint16)
35
36 # Accumulate votes in the Hough space
37 ys, xs = np.nonzero(edge)
38 for x, y in zip(xs, ys):
39     for r_idx, r in enumerate(range(min_r, max_r, r_step)):
40         for t in range(theta_res):
41             theta = 2 * pi * t / theta_res
42             a = int(x - r * cos(theta))
43             b = int(y - r * sin(theta))
44             if 0 <= a < W and 0 <= b < H:
45                 acc[b, a, r_idx] += 1
46
47 b, a, r_idx = np.unravel_index(np.argmax(acc), acc.shape)
48 r = min_r + r_idx * r_step
49
50 print(f"Detected circle: center=({a},{b}), radius={r}")
51
52 result = cv2.cvtColor(img_ori, cv2.COLOR_GRAY2BGR)
53 cv2.circle(result, (int(a), int(b)), int(r), (0, 255, 0), 2)
54 cv2.circle(result, (int(a), int(b)), 2, (0, 0, 255), 3)
55
56 plt.figure(figsize=(10, 5))
57 plt.subplot(1, 2, 1)
58 plt.title('Original Image')
59 plt.imshow(img_ori, cmap='gray')
60 plt.axis('off')
61
62 plt.subplot(1, 2, 2)
63 plt.title('Detected Circle')
64 plt.imshow(result)
65 plt.axis('off')
66 plt.tight_layout()
67 plt.savefig('result/p2/2c.png', dpi=300)

```
