Fundamentals of Biomedical Image Processing HW 3

B12508026 戴偉璿

November 11, 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.

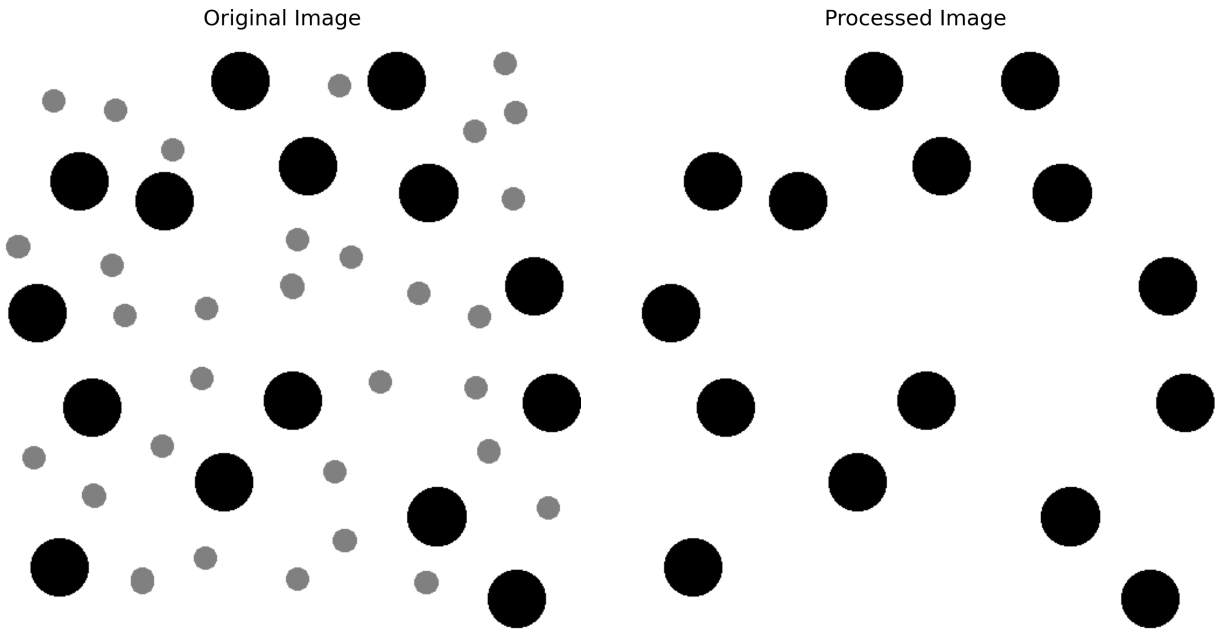

Figure 1: Result of removing small connected components

Following is the code used for this task:

```python
import os
import cv2
import numpy as np
import matplotlib
matplotlib.use('Agg')
from matplotlib import pyplot as plt

os.makedirs('result/p1', exist_ok=True)

# Read image
img_ori = cv2.imread('../src/Fig1.tif', 0)
if img_ori is None:
    raise ValueError("Image not found or unable to load.")

img = (img_ori < 128).astype(np.uint8) * 255  # Binarize
print(img.shape)

H, W = img.shape
dirs = [(-1, -1), (-1,  0), (-1, 1),
```

```
        ( 0, -1),           ( 0, 1),
        ( 1, -1), ( 1,  0), ( 1, 1)]
visited = np.zeros((H, W), dtype=bool)
label_map = np.zeros((H, W), dtype=int)
areas = []
label = 0

# BFS for connected components
for i in range(H):
    for j in range(W):
        if img[i, j] and visited[i, j] == 0:
            label += 1
            area = 0
            stack = [(i, j)]
            while stack:
                x, y = stack.pop()
                if visited[x, y]:
                    continue
                visited[x, y] = 1
                label_map[x, y] = label
                area += 1
                for dx, dy in dirs:
                    nx, ny = x + dx, y + dy
                    if 0 <= nx < H and 0 <= ny < W:
                        if img[nx, ny] and not visited[nx, ny]:
                            stack.append((nx, ny))
            areas.append(area)

print(f'Total connected components: {label}')

# Find the threshold to remove small components
areas_sorted = sorted(areas, reverse=True)
max_dist = 0
max_thresh = 0

for i in range(label-1):
    dist = abs(areas_sorted[i] - areas_sorted[i+1])
    if dist > max_dist:
        max_dist = dist
        max_thresh = (areas_sorted[i] + areas_sorted[i+1]) / 2

removed = np.zeros(label, dtype=bool)
for i in range(label):
    removed[i] = areas[i] < max_thresh

# Create new image with small components removed
img_new = img_ori.copy()
pre_remove = img_ori.copy()

# generate a binary mask for removed components
mask = (label_map > 0) & removed[label_map - 1]
img_new[mask] = 255
pre_remove[mask] = 128


plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(pre_remove, cmap='gray')
```

```
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Processed Image')
plt.imshow(img_new, cmap='gray')
plt.axis('off')
plt.tight_layout()
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.

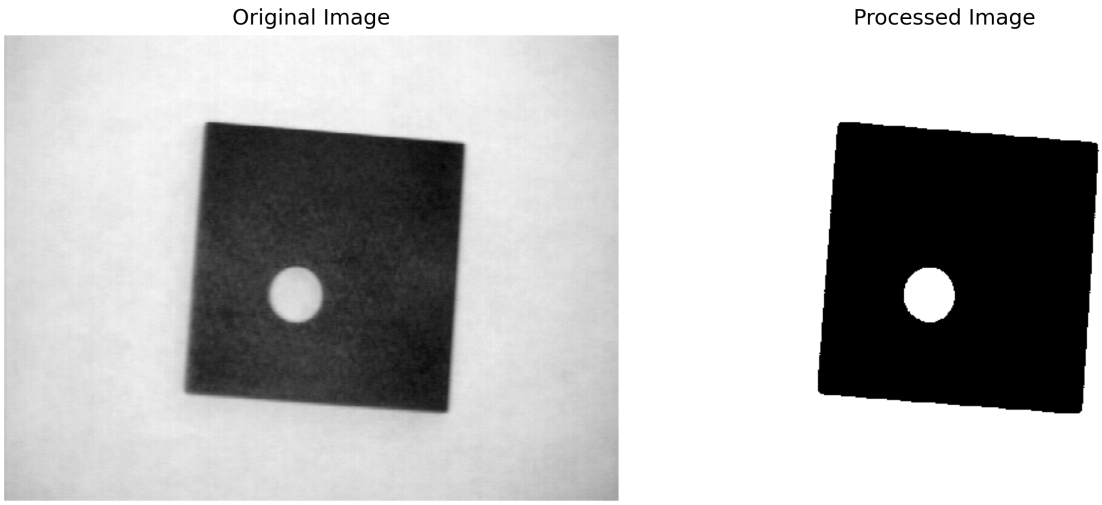

Figure 2: Binarized image with threshold 128

Following is the code used for this task:

```
import os
import cv2
import numpy as np
import matplotlib
matplotlib.use('Agg')
from matplotlib import pyplot as plt

os.makedirs('result/p2', exist_ok=True)

# Read image
img_ori = cv2.imread('../src/Fig2.gif', 0)
if img_ori is None:
    raise ValueError("Image not found or unable to load.")

# Input threshold
thresh = input("Enter threshold value (0-255): ")
try:
    thresh = int(thresh)
    if not (0 <= thresh <= 255):
```

```
        raise ValueError
except ValueError:
    raise ValueError("Invalid threshold value. Please enter an integer between 0 and 255.")

# Binarize image based on threshold
img = np.zeros_like(img_ori, dtype=np.uint8)
img[img_ori >= thresh] = 255

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(img_ori, cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Processed Image')
plt.imshow(img, cmap='gray')
plt.axis('off')
plt.tight_layout()
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.

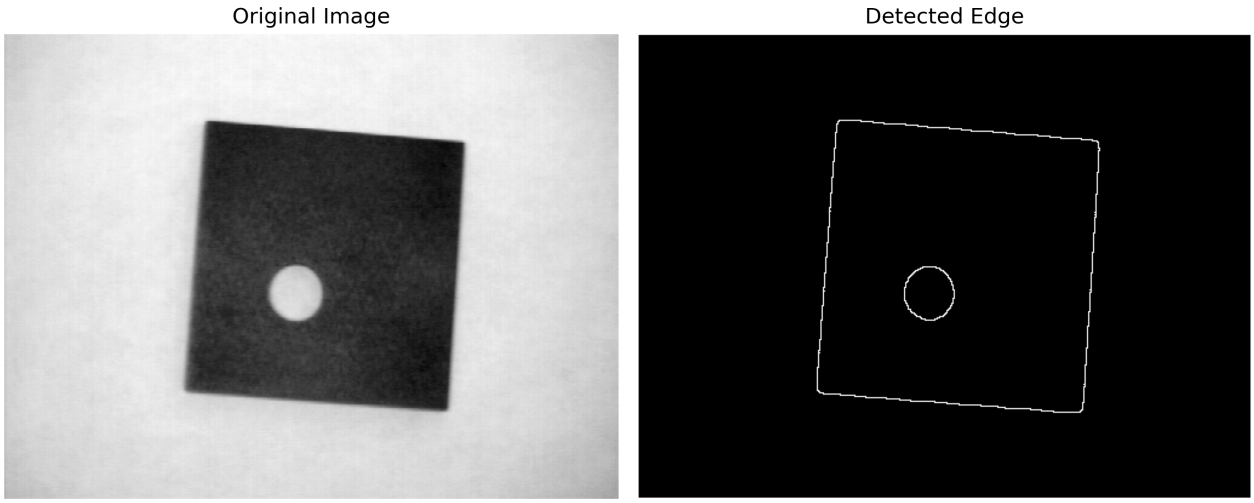

Figure 3: Detected edge image

Following is the code used for this task:

```
import os
import cv2
import numpy as np
import matplotlib
matplotlib.use('Agg')
from matplotlib import pyplot as plt
```

```python

os.makedirs('result/p2', exist_ok=True)

# Read image
img_ori = cv2.imread('../src/Fig2.gif', 0)
if img_ori is None:
    raise ValueError("Image not found or unable to load.")

thresh = 128
img = np.zeros_like(img_ori, dtype=np.uint8)
img[img_ori >= thresh] = 255

# Erosion
eroded = np.zeros_like(img, dtype=np.uint8)
H, W = img.shape
kernel = np.ones((3, 3), np.uint8)
for i in range(1, H-1):
    for j in range(1, W-1):
        region = img[i-1:i+2, j-1:j+2]
        if np.all(region == 255):
            eroded[i, j] = 255

edge = img - eroded

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(img_ori, cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Detected Edge')
plt.imshow(edge, cmap='gray')
plt.axis('off')
plt.tight_layout()
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 sizer_{step} to 2, and the number of angular binsθ_{res} to 60. The detected lines are then superimposed on the original image, as shown in Figure 4.

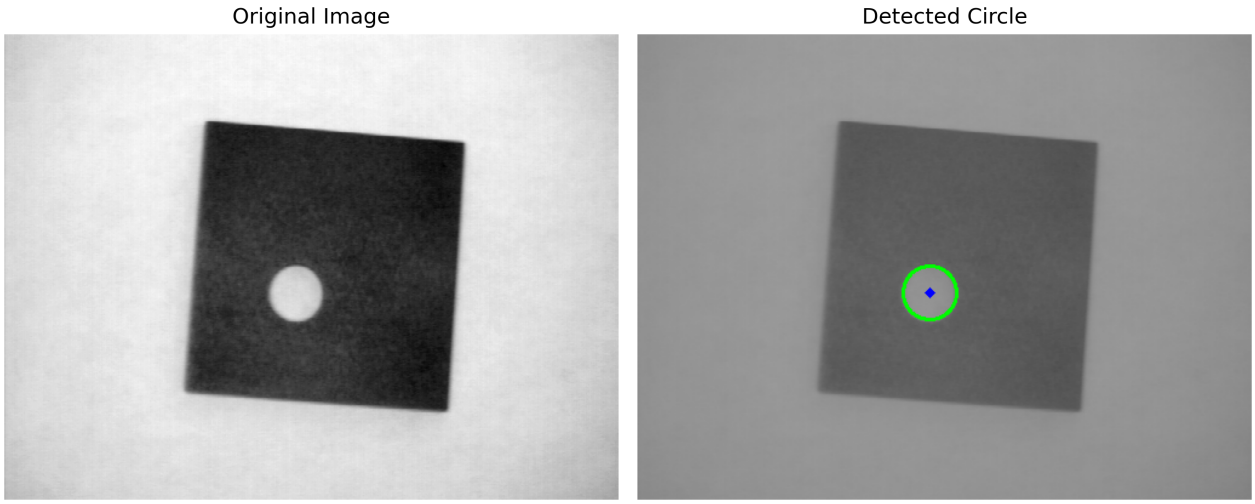

Figure 4: Detected circle on original image

Following is the code used for this task:

```python
import os
import cv2
import numpy as np
import matplotlib
matplotlib.use('Agg')
from matplotlib import pyplot as plt
from math import cos, sin, pi

os.makedirs('result/p2', exist_ok=True)

# Read image
img_ori = cv2.imread('../src/Fig2.gif', 0)
if img_ori is None:
    raise ValueError("Image not found or unable to load.")

# Get the eroded image
thresh = 128
img = np.zeros_like(img_ori, dtype=np.uint8)
img[img_ori >= thresh] = 255

eroded = np.zeros_like(img, dtype=np.uint8)
H, W = img.shape
kernel = np.ones((3, 3), np.uint8)
for i in range(1, H-1):
    for j in range(1, W-1):
```

```
        region = img[i-1:i+2, j-1:j+2]
        if np.all(region == 255):
            eroded[i, j] = 255

edge = img - eroded

# Set the Hough Transform parameters
min_r, max_r, r_step, theta_res = 10, 30, 2, 60
acc = np.zeros((H, W, (max_r - min_r)//r_step), dtype=np.uint16)

# Accumulate votes in the Hough space
ys, xs = np.nonzero(edge)
for x, y in zip(xs, ys):
    for r_idx, r in enumerate(range(min_r, max_r, r_step)):
        for t in range(theta_res):
            theta = 2 * pi * t / theta_res
            a = int(x - r * cos(theta))
            b = int(y - r * sin(theta))
            if 0 <= a < W and 0 <= b < H:
                acc[b, a, r_idx] += 1

b, a, r_idx = np.unravel_index(np.argmax(acc), acc.shape)
r = min_r + r_idx * r_step

print(f"Detected circle: center=({a},{b}), radius={r}")

result = cv2.cvtColor(img_ori, cv2.COLOR_GRAY2BGR)
cv2.circle(result, (int(a), int(b)), int(r), (0, 255, 0), 2)
cv2.circle(result, (int(a), int(b)), 2, (0, 0, 255), 3)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(img_ori, cmap='gray')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Detected Circle')
plt.imshow(result)
plt.axis('off')
plt.tight_layout()
plt.savefig('result/p2/2c.png', dpi=300)
```