24-678: Computer Vision for Engineers
Ryan Wu
ID: weihuanw
PS5 Report
Due: Nov 3 2023

This file contains the following:
PS5-1 Binary image processing – detecting blobs, contours, and central axes

- wall1-blobs.png, wall2-blobs.png
- wall1-contours.png, wall2-contours.png
- wall1-crackss.png, wall2-cracks.png
- readme.txt
- source code file(s) (attached to the end)

Findings and discussion:

We are tasked to perform dilation, erosion, contouring, crack detection, and thinning for 2 separate wall images.

In my program, I defined 5 different functions to perform: 1. Loading the given images, 2. Perform dilation and erosion, 3. Contour drawing, 4. Crack detection 5. Thinning the detected crack. Certain parameters, like iterations, and contour length thresholds, were determined by trial and error.

In my dilation and erosions logic, the process iteration was set to 3. In my crack detection logic, I choose the contour's length as the determining factor. For wall1 and wall2 images, the contour length threshold was set at 2000 and 500 respectively, any length that is less than their declared threshold was filtered (used <= since the image was inverted to black background before the operation).

The results were satisfactory and the program performed all the tasks stated above.

PS5-1 wall1 & wall2 blobs images and function used

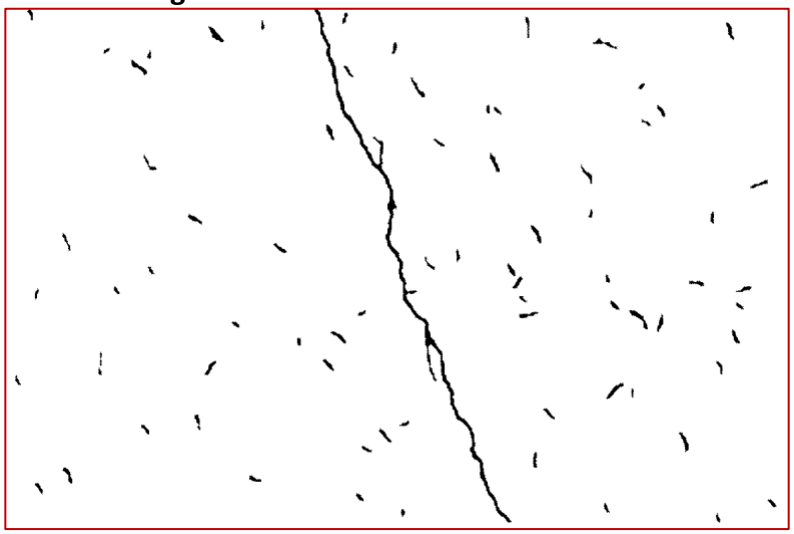

Figure 1. wall1 image with blobs shown.

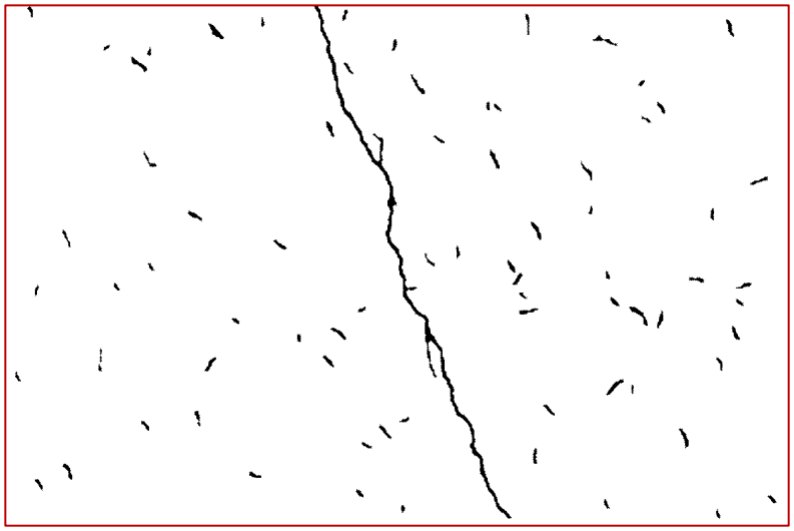

Figure 2. wall2 image with blobs shown.

```
# function for image dilation and erosion
2 usages   ryanwu0521
def dilation_erosion(image, iterations=3):
    kernel = cv2.getStructuringElement(cv2.MORPH_CROSS, (3, 3))
    erode_image = cv2.erode(image, kernel, iterations=iterations)
    dilate_image= cv2.dilate(erode_image, kernel, iterations=iterations)
    return dilate_image
```

Figure 3. The function used for blob detection.

PS5-1 wall1 & wall2 contours images and function used

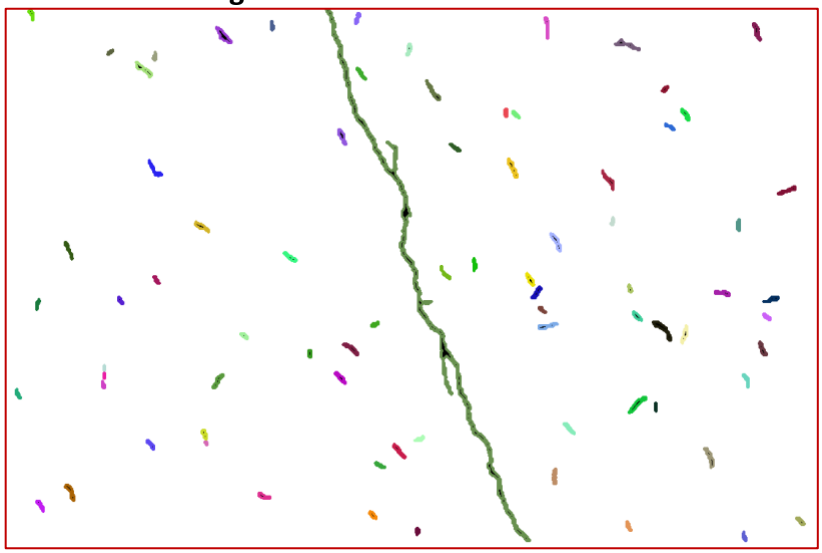

Figure 4. wall1 with random contours shown.

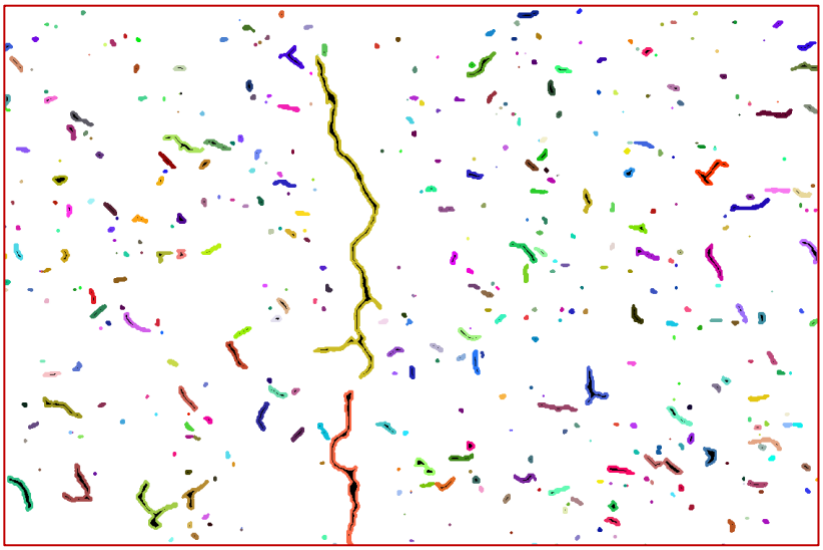

Figure 5. wall2 with random contours shown.

```python
# function for drawing contours
2 usages  ryanwu0521
def blob_contours(image):
    # inverting the image
    inverted_image = cv2.bitwise_not(image)
    # funding contours using the inverted image
    contours, _ = cv2.findContours(inverted_image, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    # converting the image back to BGR
    contour_image = cv2.cvtColor(inverted_image, cv2.COLOR_GRAY2BGR)
    # drawing contours on the image
    for contour in contours:
        color = tuple(np.random.randint(0, 255, 3).tolist())
        cv2.drawContours(contour_image, [contour], -1, color, 2)
    # converting the image back
    contour_image_color = cv2.bitwise_not(contour_image)
    # returning the image with contours
    return contour_image_color
```

Figure 6.The function used for contour drawing.

PS5-1 wall1 & wall2 cracks images

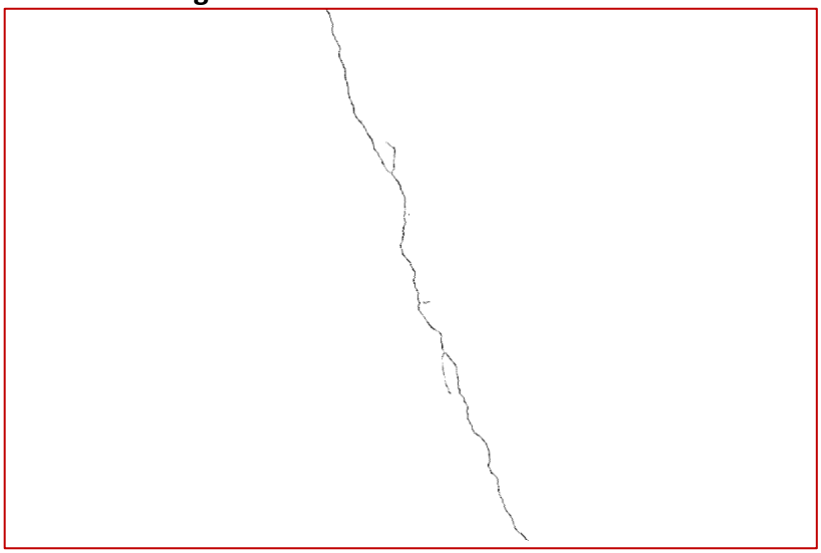

Figure 7. wall1 crack image after thinning.

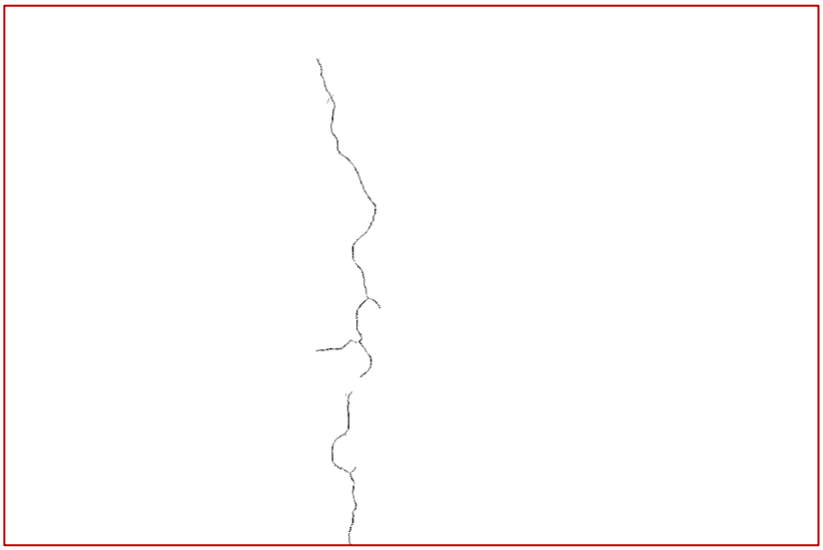

Figure 8. wall2 crack image after thinning

PS5-1 wall1 & wall2 cracks images function used

```python
# function for detecting cracks
2 usages  ryanwu0521
def detect_crack_and_thin(image, contour_length_threshold = 2000):
    # inverting the image
    inverted_image = cv2.bitwise_not(image)
    crack_image = image.copy()
    # Finding contours
    contours, _ = cv2.findContours(inverted_image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    contour_crack = []
    # Filtering contours
    for contour in contours:
        # calculate contour arc length
        arc_length = cv2.arcLength(contour, True)

        # filter contours based on area
        if arc_length <= contour_length_threshold:
            contour_crack.append(contour)

    cv2.drawContours(crack_image, contour_crack, -1, (255, 255, 255), 10)

    # calling the thinning function
    thinned_crack_image = thinning(crack_image)

    return thinned_crack_image
```

Figure 9. The function used for crack detection.

```python
# function for thinning
1 usage  ryanwu0521
def thinning(image):
    # reverse image (black background)
    black_image = cv2.bitwise_not(image)
    # Kernel: 4 neighbor
    k_e = cv2.getStructuringElement(cv2.MORPH_CROSS, (3, 3))
    # Target image
    thin = np.zeros(black_image.shape, dtype=np.uint8)
    # repeat until no white area
    while cv2.countNonZero(black_image) != 0:
        er = cv2.erode(black_image, k_e)
        # OPEN: erosion then dilation (remove noise)
        op = cv2.morphologyEx(er, cv2.MORPH_OPEN, k_e)
        subset = cv2.subtract(er, op)
        thin = cv2.bitwise_or(subset, thin)
        black_image = er.copy()

    # invert the thinned image back to white background
    thinned_image = cv2.bitwise_not(thin)
    return thinned_image
```

Figure 10. The function used for crack thinning.

PS5-1 readme.txt

24-678: Computer Vision for Engineers
Ryan Wu
ID: weihuanw
PS5-1 Binary image processing – detecting blobs, contours, and central axes

Operating system: macOS Ventura 13.5.2
IDE you used to write and run your code: PyCharm 2023.1.4 (Community Edition)
The number of hours you spent to finish this problem: 8 hours.

```python
# 24-678 Computer Vision for Engineers
# Ryan Wu (ID:weihuanw)
# PS05 Binary image processing - detecting blobs, contours, and central axes
# Due 11/3/2023 (Fri) 5 pm

# import necessary packages
import cv2
import numpy as np

# function for loading given images
def load_images(image_path1, image_path2):
    wall1_image = cv2.imread(image_path1, cv2.IMREAD_GRAYSCALE)
    wall2_image = cv2.imread(image_path2, cv2.IMREAD_GRAYSCALE)
    return wall1_image, wall2_image

# function for image dilation and erosion
def dilation_erosion(image, iterations=3):
    kernel = cv2.getStructuringElement(cv2.MORPH_CROSS, (3, 3))
    erode_image = cv2.erode(image, kernel, iterations=iterations)
    dilate_image= cv2.dilate(erode_image, kernel, iterations=iterations)
    return dilate_image

# function for drawing contours
def blob_contours(image):
    # inverting the image
    inverted_image = cv2.bitwise_not(image)
    # funding contours using the inverted image
    contours, _ = cv2.findContours(inverted_image, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    # converting the image back to BGR
    contour_image = cv2.cvtColor(inverted_image, cv2.COLOR_GRAY2BGR)
    # drawing contours on the image
    for contour in contours:
        color = tuple(np.random.randint(0, 255, 3).
```

```
33 tolist())
34         cv2.drawContours(contour_image, [contour
   ], -1, color, 2)
35     # converting the image back
36     contour_image_color = cv2.bitwise_not(
   contour_image)
37     # returning the image with contours
38     return contour_image_color
39
40 # function for detecting cracks
41 def detect_crack_and_thin(image,
   contour_length_threshold = 2000):
42     # inverting the image
43     inverted_image = cv2.bitwise_not(image)
44     crack_image = image.copy()
45     # Finding contours
46     contours, _ = cv2.findContours(inverted_image,
   cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
47     contour_crack = []
48     # Filtering contours
49     for contour in contours:
50         # calculate contour arc length
51         arc_length = cv2.arcLength(contour, True)
52
53         # filter contours based on area
54         if arc_length <= contour_length_threshold:
55             contour_crack.append(contour)
56
57     cv2.drawContours(crack_image, contour_crack, -1
   , (255, 255, 255), 10)
58
59     # calling the thinning function
60     thinned_crack_image = thinning(crack_image)
61
62     return thinned_crack_image
63
64 # function for thinning
65 def thinning(image):
66     # reverse image (black background)
67     black_image = cv2.bitwise_not(image)
68     # Kernel: 4 neighbor
```

```
69      k_e = cv2.getStructuringElement(cv2.
    MORPH_CROSS, (3, 3))
70      # Target image
71      thin = np.zeros(black_image.shape, dtype=np.
    uint8)
72      # repeat until no white area
73      while cv2.countNonZero(black_image) != 0:
74          er = cv2.erode(black_image, k_e)
75          # OPEN: erosion then dilation (remove
    noise)
76          op = cv2.morphologyEx(er, cv2.MORPH_OPEN,
    k_e)
77          subset = cv2.subtract(er, op)
78          thin = cv2.bitwise_or(subset, thin)
79          black_image = er.copy()
80
81      # invert the thinned image back to white
    background
82      thinned_image = cv2.bitwise_not(thin)
83      return thinned_image
84
85  # main function
86  def main():
87      # calling load_images function
88      wall1_image, wall2_image = load_images('wall1.
    png', 'wall2.png')
89      if wall1_image is not None and wall2_image is
    not None:
90          # calling dilation_erosion function
91          wall1_blobed = dilation_erosion(
    wall1_image)
92          wall2_blobed = dilation_erosion(
    wall2_image)
93
94          # calling blob_contours function
95          wall1_contoured = blob_contours(
    wall1_blobed)
96          wall2_contoured = blob_contours(
    wall2_blobed)
97
98          # calling detect_crack function (setting
```

```
 98 contour length threshold to 2000 for wall1 and 500
     for wall2)
 99         wall1_thinned_crack_image =
    detect_crack_and_thin(wall1_blobed,
    contour_length_threshold=2000)
100         wall2_thinned_crack_image =
    detect_crack_and_thin(wall2_blobed,
    contour_length_threshold=500)
101
102         # display and save images
103         # cv2.imshow('Wall 1 Image Blobs',
    wall1_blobed)
104         # cv2.imshow('Wall 2 Image Blobs',
    wall2_blobed)
105         cv2.imwrite("wall1-blobs.png",
    wall1_blobed)
106         cv2.imwrite("wall2-blobs.png",
    wall1_blobed)
107
108         # cv2.imshow('Wall 1 Image Contours',
    wall1_contoured)
109         # cv2.imshow('Wall 2 Image Contours',
    wall2_contoured)
110         cv2.imwrite("wall1-contours.png",
    wall1_contoured)
111         cv2.imwrite("wall2-contours.png",
    wall2_contoured)
112
113         # cv2.imshow('Wall 1 Crack Image',
    wall1_thinned_crack_image)
114         # cv2.imshow('Wall 2 Crack Image',
    wall2_thinned_crack_image)
115         cv2.imwrite("wall1-cracks.png",
    wall1_thinned_crack_image)
116         cv2.imwrite("wall2-cracks.png",
    wall2_thinned_crack_image)
117
118         cv2.waitKey(0)
119         cv2.destroyAllWindows()
120     else:
121         print("Error in loading wall images.")
```

```
122
123 # if __name__ == "__main__":
124 main()
125
```