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

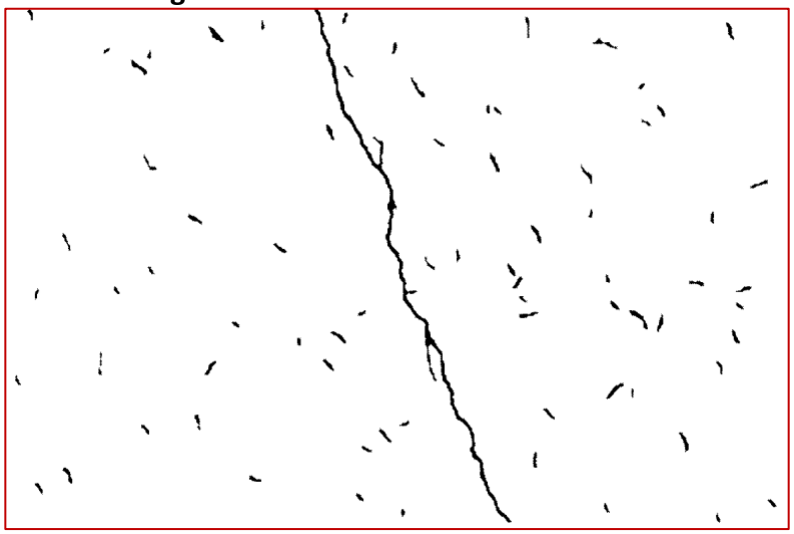

Figure 1. wall1 image with blobs shown.

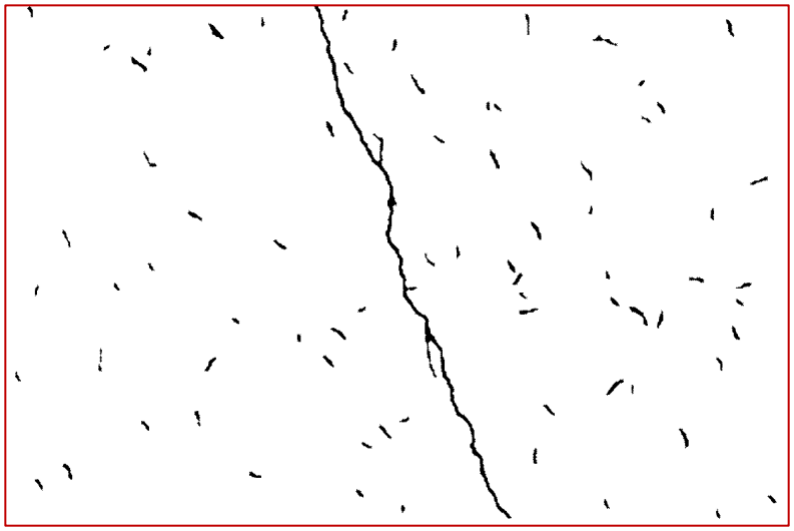

Figure 2. wall2 image with blobs shown.

```
# 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

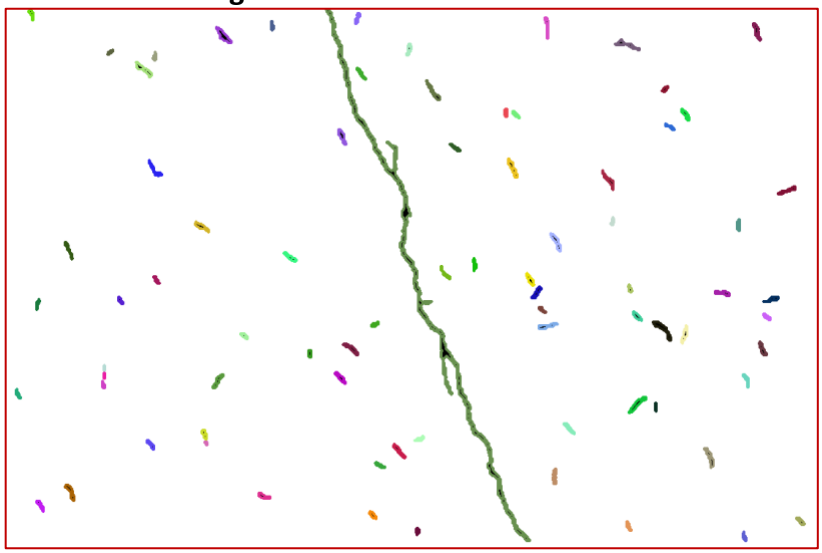

Figure 4. wall1 with random contours shown.

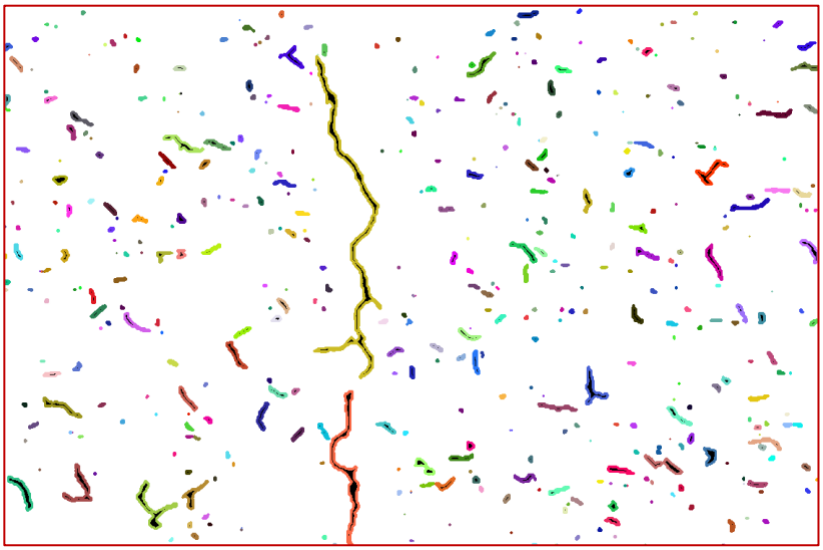

Figure 5. wall2 with random contours shown.

```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

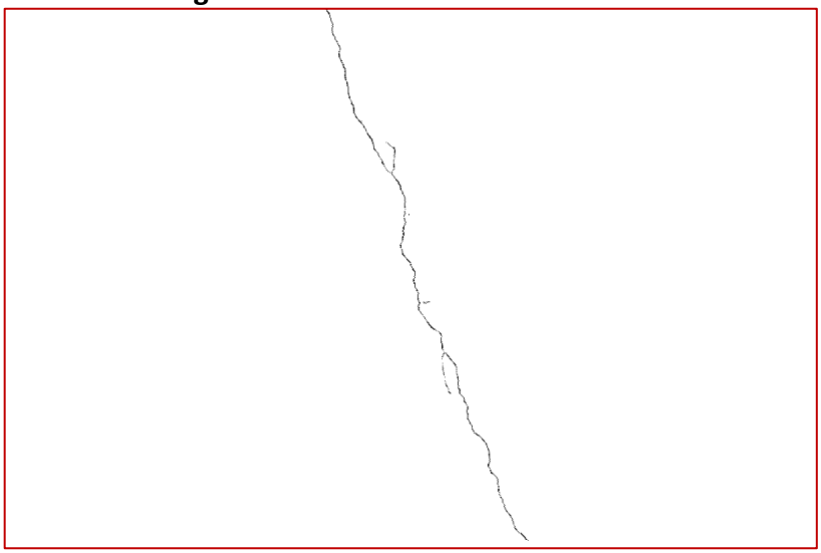

Figure 7. wall1 crack image after thinning.

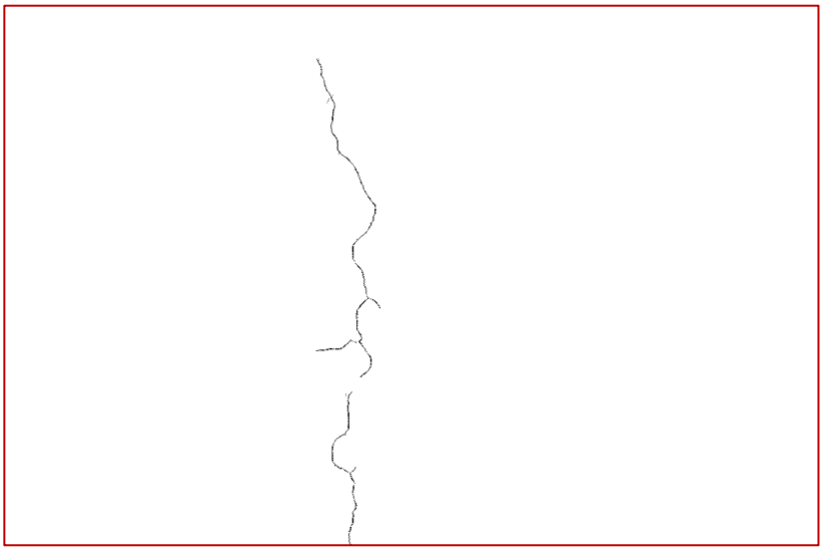

Figure 8. wall2 crack image after thinning

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.