**電腦視覺**

**Homework 4**

**Contents:**

1. Description
2. Methodology
3. Code fragments
4. Images

Description

This is the report for the fourth homework for Computer Vision 2019. It is completed in full.

The environment is Windows, with the code (for all parts) being written in Python 3 with the cv2 module.

Methodology

For these parts, “oct kernel” refers to the 3-5-5-5-3 kernel; J and K refer to the kernels in (e).

1. Dilation
   1. The set of white pixels is collected from the binary image, and operated on the oct kernel using the method on the slides. Every pixel in range of [0, 511] x [0, 511] is retained.
   2. The resulting image is saved to **dilated.bmp**.
2. Erosion
   1. The set of white pixels is collected from the binary image.
   2. Each pixel in [-5, 513] x [-5, 513] that also satisfies the method’s requirements is added to the new image.
   3. The resulting image is saved to **eroded.bmp**.
3. Opening
   1. The set of white pixels is collected from the eroded image, and operated on the oct kernel using the method on the slides. Every pixel in range of [0, 511] x [0, 511] is retained.
   2. The resulting image is saved to **opened.bmp**.
4. Closing
   1. The set of white pixels is collected from the dilated image, and operated on the oct kernel using the method on the slides. Every pixel in range of [0, 511] x [0, 511] is retained.
   2. The resulting image is saved to **closed.bmp**.
5. Hit-or-Miss
   1. The J and K kernels are used. The set of white pixels and its complement (i.e. **1** – A) are collected and operated on based on the slides. Pixels in range of [0, 511] x [0, 511] are retained.
   2. The resulting image is saved to **hit\_or\_miss.bmp**.

Code Fragments

The fragment only shows the operations for Dilation and Erosion (of which the other parts are derived from):

1. **def** dilate(image, kernel):
2. A = np.argwhere(image == WHITE)
3. A\_dil\_B = set()
4. **for** a **in** A:
5. **for** b **in** kernel:
6. new\_point = a + b
7. **if** 0 <= new\_point[0] < image.shape[0] **and** 0 <= new\_point[1] < image.shape[1]:
8. A\_dil\_B.add((new\_point[0], new\_point[1]))
10. new\_image = fill\_img(image.shape[0], image.shape[1], 0, A\_dil\_B)
11. **return** new\_image
13. **def** erode(image, kernel):
14. A = np.argwhere(image == WHITE)
15. A = set(tuple(map(tuple, A)))
16. A\_erode\_B = set()
18. **for** r **in** range(-5, 513):
19. **for** c **in** range(-5, 513):
20. # check if (r, c) + b is in A for every b in kernel
21. good = True
22. **for** b **in** kernel:
23. **if** (b[0]+r, b[1]+c) **not** **in** A: good = False; **break**
24. **if** good: A\_erode\_B.add((r,c))
26. new\_image = fill\_img(image.shape[0], image.shape[1], 0, A\_erode\_B)
27. **return** new\_image

Images

The images are shown in the following order: Dilation, Erosion, Opening, Closing, and Hit-or-Miss (the latter which is magnified).





