Computer Vision I

Homework 9 - General Edge Detection B04902090 資工四 施長元

Usage of the full code:

`python3 main.py [Image_path]`

After the code exit, output file will be in the directory where you execute the code.

Environment: Python3.7 on Windows Linux Subsystem (Ubuntu 18.04.1)

Contents:

You are to implement:

1. Robert's operator, threshold = 12

Figure 7.21 Masks used for the Roberts operators.

```python
def robert(img_o, thres):
  print("Robert's operator, threshold = %d" % thres)
  img_t = np.zeros(shape, dtype=np.int16)
  for i in range(shape[0]):
    for j in range(shape[1]):
      r = (-img_o[i+2, j+2] + img_o[i+3, j+3]) ** 2
      r += (-img_o[i+2, j+3] + img_o[i+3, j+2]) ** 2
      img_t[i, j] = 0 if r ** 0.5 > thres else 1
  return img_t
```

2. Prewitt's Edge Detector, threshold = 24

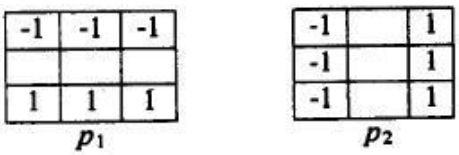

Figure 7.22 Prewitt edge detector masks.

```
def prewitt(img_o, thres):
  print("Prewitt's Edge Detector, threshold = %d" % thres)
  img_t = np.zeros(shape, dtype=np.int16)
  for i in range(shape[0]):
    for j in range(shape[1]):
      p = (-np.sum(img_o[i+2, j+2:j+5]) + np.sum(img_o[i+4, j+2:j+5])) ** 2
      p += (-np.sum(img_o[i+2:i+5, j+2]) + np.sum(img_o[i+2:i+5, j+4])) ** 2
      img_t[i, j] = 0 if p ** 0.5 > thres else 1
  return img_t
```

3. Sobel's Edge Detector, threshold = 38

-1	-2	-1
1	2	1

s_1

-1		1
-2		2
-1		1

s_2

Figure 7.23 Sobel edge detector masks.

```
def sobel(img_o, thres):
  print("Sobel's Edge Detector, threshold = %d" % thres)
  img_t = np.zeros(shape, dtype=np.int16)
  for i in range(shape[0]):
    for j in range(shape[1]):
      s = (-np.sum(img_o[i+2, j+2:j+5]) - img_o[i+2, j+3] +
          np.sum(img_o[i+4, j+2:j+5]) + img_o[i+4, j+3]) ** 2
      s += (-np.sum(img_o[i+2:i+5, j+2]) - img_o[i+3, j+2] +
          np.sum(img_o[i+2:i+5, j+4]) + img_o[i+3, j+4]) ** 2
      img_t[i, j] = 0 if s ** 0.5 > thres else 1
  return img_t
```

4. Frei and Chen's Gradient Operator, threshold = 30

-1	$-\sqrt{2}$	-1
1	$\sqrt{2}$	1

f_1

-1		1
$-\sqrt{2}$		$\sqrt{2}$
-1		1

f_2

Figure 7.24 Frei and Chen gradient masks.

```
def FandC(img_o, thres):
  print("Frei and Chen's Gradient Operator, threshold = %d" % thres)
  img_t = np.zeros(shape, dtype=np.int16)
  for i in range(shape[0]):
    for j in range(shape[1]):
      f = (img_o[i+4, j+2] - img_o[i+2, j+2] + (2**0.5)*(img_o[i+4, j+3] -
          img_o[i+2, j+3]) + img_o[i+4, j+4] - img_o[i+2, j+4]) ** 2
      f += (img_o[i+2, j+4] - img_o[i+2, j+2] + (2**0.5)*(img_o[i+3, j+4] -
          img_o[i+3, j+2]) + img_o[i+4, j+4] - img_o[i+4, j+2]) ** 2
      img_t[i, j] = 0 if f ** 0.5 > thres else 1
  return img_t
```

5. Kirsch's Compass Operator, threshold = 135

-3	-3	5
-3		5
-3	-3	5

k_0

-3	5	5
-3		5
-3	-3	-3

k_1

5	5	5
-3		-3
-3	-3	-3

k_2

5	5	-3
5		-3
-3	-3	-3

k_3

5	-3	-3
5		-3
5	-3	-3

k_4

-3	-3	-3
5		-3
5	5	-3

k_5

-3	-3	-3
-3		-3
5	5	5

k_6

-3	-3	-3
-3		5
-3	5	5

k_7

Figure 7.25 Kirsch compass masks.

```
def Kirsch(img_o, thres):
  print("Kirsch's Compass Operator, threshold = %d" % thres)
  img_t = np.zeros(shape, dtype=np.int16)
  for i in range(shape[0]):
    for j in range(shape[1]):
      k = np.zeros(8, dtype=np.int16)
      k[0] = -3*(np.sum(img_o[i+1:i+4, j+1]) + img_o[i+1, j+2] + img_o[i+3, j+2]) + \
              5*np.sum(img_o[i+1:i+4, j+3])
      k[1] = -3*(np.sum(img_o[i+1:i+4, j+1]) + img_o[i+3, j+2] + img_o[i+3, j+3]) + \
              5*(img_o[i+1, j+2] + img_o[i+1, j+3] + img_o[i+2, j+3])
      k[2] = -3*(np.sum(img_o[i+3, j+1:j+4]) + img_o[i+2, j+1] + img_o[i+2, j+3]) + \
              5*np.sum(img_o[i+1, j+1:j+4])
      k[3] = -3*(np.sum(img_o[i+1:i+4, j+3]) + img_o[i+3, j+2] + img_o[i+3, j+1]) + \
              5*(img_o[i+1, j+1] + img_o[i+2, j+1] + img_o[i+1, j+2])
      k[4] = -3*(np.sum(img_o[i+1:i+4, j+3]) + img_o[i+1, j+2] + img_o[i+3, j+2]) + \
              5*np.sum(img_o[i+1:i+4, j+1])
      k[5] = -3*(np.sum(img_o[i+1:i+4, j+3]) + img_o[i+1, j+1] + img_o[i+1, j+2]) + \
              5*(img_o[i+2, j+1] + img_o[i+3, j+1] + img_o[i+3, j+2])
      k[6] = -3*(np.sum(img_o[i+1, j+1:j+4]) + img_o[i+2, j+1] + img_o[i+2, j+3]) + \
              5*np.sum(img_o[i+3, j+1:j+4])
      k[7] = -3*(np.sum(img_o[i+1:i+4, j+1]) + img_o[i+1, j+2] + img_o[i+1, j+3]) + \
              5*(img_o[i+2, j+3] + img_o[i+3, j+3] + img_o[i+3, j+2])
      img_t[i, j] = 0 if np.max(k) > thres else 1
  return img_t
```

6. Robinson's Compass Operator, threshold = 43

-1		1
-2		2
-1		1

r_0

	1	2
-1		1
-2	-1	

r_1

1	2	1
-1	-2	-1

r_2

2	1	
1		-1
	-1	-2

r_3

1		-1
2		-2
1		-1

r_4

	-1	-2
1		-1
2	1	

r_5

-1	-2	-1
1	2	1

r_6

-2	-1	
-1		1
	1	2

r_7

Figure 7.26 Robinson compass masks.

```
def robinson(img_o, thres):
  print("Robinson's Compass Operator, threshold = %d" % thres)
  img_t = np.zeros(shape, dtype=np.int16)
  for i in range(shape[0]):
    for j in range(shape[1]):
      r = np.zeros(8, dtype=np.int16)
      r[0] = -1*(img_o[i+1, j+1] + img_o[i+3, j+1]) + img_o[i+1, j+3] + img_o[i+3, j+3] + 2*img_o[i+2, j+3] - 2*img_o[i+2, j+1]
      r[1] = -1*(img_o[i+2, j+1] + img_o[i+3, j+2]) + img_o[i+1, j+2] + img_o[i+2, j+3] + 2*img_o[i+1, j+3] - 2*img_o[i+3, j+1]
      r[2] = -1*(img_o[i+3, j+1] + img_o[i+3, j+3]) + img_o[i+1, j+1] + img_o[i+1, j+3] + 2*img_o[i+1, j+2] - 2*img_o[i+3, j+2]
      r[3] = -1*(img_o[i+3, j+2] + img_o[i+2, j+3]) + img_o[i+2, j+1] + img_o[i+1, j+2] + 2*img_o[i+1, j+1] - 2*img_o[i+3, j+3]
      r[4], r[5], r[6], r[7] = -r[0], -r[1], -r[2], -r[3]
      img_t[i, j] = 0 if np.max(r) > thres else 1
  return img_t
```

7. Nevatia-Babu 5x5 Operator, threshold = 12500

100	100	100	100	100
100	100	100	100	100
0	0	0	0	0
-100	-100	-100	-100	-100
-100	-100	-100	-100	-100

0°

100	100	100	100	100
100	100	100	78	-32
100	92	0	-92	-100
32	-78	-100	-100	-100
-100	-100	-100	-100	-100

30°

100	100	100	32	-100
100	100	92	-78	-100
100	100	0	-100	-100
100	78	-92	-100	-100
100	-32	-100	-100	-100

60°

-100	-100	0	100	100
-100	-100	0	100	100
-100	-100	0	100	100
-100	-100	0	100	100
-100	-100	0	100	100

-90°

-100	32	100	100	100
-100	-78	92	100	100
-100	-100	0	100	100
-100	-100	-92	78	100
-100	-100	-100	-32	100

-60°

100	100	100	100	100
-32	78	100	100	100
-100	-92	0	92	100
-100	-100	-100	-78	32
-100	-100	-100	-100	-100

-30°

Figure 7.27 Nevatia-Babu 5×5 compass template masks.

```
def nevatia(img_o, thres):
  print("Nevatia-Babu 5x5 Operator, threshold = %d" % thres)
  img_t = np.zeros(shape, dtype=np.int16)
  for i in range(shape[0]):
    for j in range(shape[1]):
      n = np.zeros(6, dtype=np.int16)
      n[0] = 100*(np.sum(img_o[i:i+2, j:j+5])) + -100*(np.sum(img_o[i+3:i+5, j:j+5]))   # 0

      n[1] = 100*(np.sum(img_o[i, j:j+5]) + np.sum(img_o[i+1, j:j+3]) + img_o[i+2, j]) \
           + 78*img_o[i+1, j+3] - 32*img_o[i+1, j+4] + 92*img_o[i+2, j+1] - 92*img_o[i+2, j+3] + 32*img_o[i+3, j] - 78*img_o[i+3, j+1] \
           - 100*(np.sum(img_o[i+4, j:j+5]) + np.sum(img_o[i+3, j+2:j+5]) + img_o[i+2, j+4])   # 30

      n[2] = 100*(np.sum(img_o[i:i+5, j]) + np.sum(img_o[i:i+3, j+1]) + img_o[i, j+2]) \
           + 78*img_o[i+3, j+1] - 32*img_o[i+4, j+1] + 92*img_o[i+1, j+2] - 92*img_o[i+3, j+2] + 32*img_o[i, j+3] - 78*img_o[i+1, j+3] \
           - 100*(np.sum(img_o[i:i+5, j+4]) + np.sum(img_o[i+2:i+5, j+3]) + img_o[i+4, j+2])   # 60

      n[3] = 100*(np.sum(img_o[i:i+5, j+3:j+5])) + -100*(np.sum(img_o[i:i+5, j:j+2]))   # -90

      n[4] = 100*(np.sum(img_o[i:i+5, j+4]) + np.sum(img_o[i:i+3, j+3]) + img_o[i, j+2]) \
           + 78*img_o[i+3, j+3] - 32*img_o[i+4, j+3] + 92*img_o[i+1, j+2] - 92*img_o[i+3, j+2] + 32*img_o[i, j+1] - 78*img_o[i+1, j+1] \
           - 100*(np.sum(img_o[i:i+5, j]) + np.sum(img_o[i+2:i+5, j+1]) + img_o[i+4, j+2])   # -60

      n[5] = 100*(np.sum(img_o[i, j:j+5]) + np.sum(img_o[i+1, j+2:j+5]) + img_o[i+2, j+4]) \
           + 78*img_o[i+1, j+1] - 32*img_o[i+1, j] - 92*img_o[i+2, j+1] + 92*img_o[i+2, j+3] + 32*img_o[i+3, j+4] - 78*img_o[i+3, j+3] \
           - 100*(np.sum(img_o[i+4, j:j+5]) + np.sum(img_o[i+3, j:j+3]) + img_o[i+2, j])   # -30

      img_t[i, j] = 0 if np.max(n) > thres else 1
  return img_t
```