

Answer



Lena.bmp

Robert's Operator
Thresholds:25:Prewitt's Edge Detector
Thresholds:100Sobel's Edge Detector
Thresholds:100:Frei and Chen's Gradient Operator
Thresholds:100Kirsch's Compass Operator
Thresholds:300Robinson's Compass Operator
Thresholds:100Nevatia-Babu 5x5 Operator
Thresholds:15000

Description& Algorithm

Implement following edge detectors with thresholds :

(a) Robert's Operator: 25

Roberts operators: two 2x2 masks to calculate gradient

$$f'(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x+\Delta x) - f(x)}{\Delta x}$$

$$f'(x) \approx f(x+1) - f(x)$$

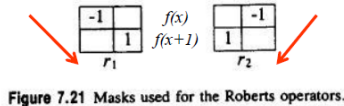


Figure 7.21 Masks used for the Roberts operators.

gradient magnitude: $\sqrt{r_1^2 + r_2^2}$

Where r_1, r_2 are values from first, second masks respectively.

(b) Prewitt's Edge Detector: 100

Prewitt edge detector: two 3x3 masks in row, column directions.

$$\theta = \arctan(p_1/p_2)$$

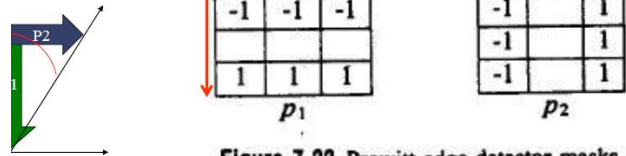


Figure 7.22 Prewitt edge detector masks.

Gradient magnitude: $g = \sqrt{p_1^2 + p_2^2}$

Gradient direction: $\theta = \arctan(p_1/p_2)$ clockwise w.r.t. column axis

Where p_1, p_2 are values from first, second masks respectively.

(c) Sobel's Edge Detector: 100

Sobel edge detector: two 3x3 masks in row, column directions.

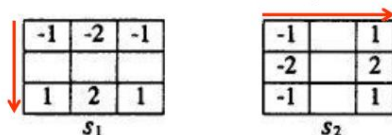


Figure 7.23 Sobel edge detector masks.

Gradient magnitude: $g = \sqrt{s_1^2 + s_2^2}$

Gradient direction: $\theta = \arctan(s_1/s_2)$ clockwise w.r.t. column axis

Where s_1, s_2 are values from first, second masks respectively.

(d) Frei and Chen's Gradient Operator: 100

Frei and Chen edge detector: two in a set of nine orthogonal masks (3x3).

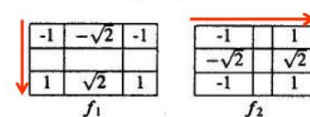


Figure 7.24 Frei and Chen gradient masks.

Gradient magnitude: $g = \sqrt{f_1^2 + f_2^2}$

Gradient direction: $\theta = \arctan(f_1/f_2)$ clockwise w.r.t. column axis

Where f_1, f_2 are values from first, second masks respectively.

(e) Kirsch's Compass Operator: 300

Kirsch edge detector : set of eight compass template edge masks.

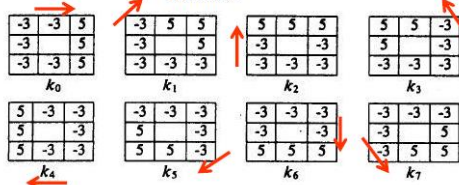


Figure 7.25 Kirsch compass masks.

Gradient magnitude: $g = \max_{n,n=0,\dots,7} k_n$

Gradient direction: $\theta = 45^\circ \arg\max k_n$



compass

(f) Robinson's Compass Operator: 100

Robinson edge detector : compass template mask set with only 0, $\pm 1, \pm 2$.

Simplify mask coefficients (Kirsch)

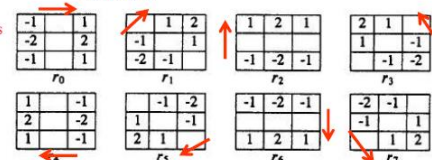


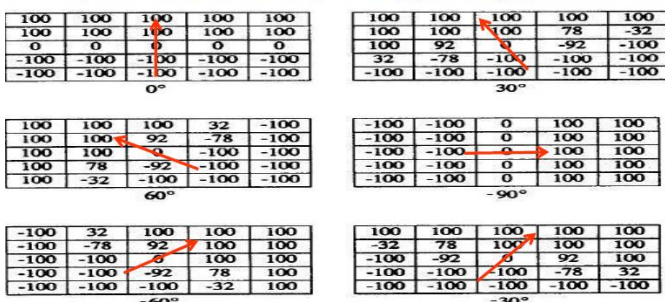
Figure 7.26 Robinson compass masks.

Done by only four masks since negation of each mask is also a mask.

Gradient magnitude and direction same as Kirsch operator.

(g) Nevatia-Babu 5x5 Operator: 15000

Nevatia and Babu: set of six 5x5 compass template masks.



Code description

本次作業使用 python3.7 · IDE 使用 Spyder

(a) Main function:

1. 先建立好每一種會用到 neighbor(2*2,3*3,5*5)
2. 將 lena 當作 input 給每一個 operator
3. 將每個 output 存起來

```

215 if __name__ == '__main__':
216     neighbor_2 = createNeighbor(0, 2)
217     neighbor_3 = createNeighbor(-1, 2)
218     neighbor_5 = createNeighbor(-2, 3)
219
220     lena = cv2.imread('lena.bmp', cv2.IMREAD_GRAYSCALE)
221
222     robert = robert_operator(lena, 25)
223     cv2.imwrite('Robert.bmp', robert)
224
225     prewitt = prewitt_operator(lena, 100)
226     cv2.imwrite('Prewitt.bmp', prewitt)
227
228     sobel = sobel_operator(lena, 100)
229     cv2.imwrite('Sobel.bmp', sobel)
230
231     frei_chen = frei_chen_operator(lena, 100)
232     cv2.imwrite('Frei and Chen.bmp', frei_chen)
233
234     kirsch = kirsch_operator(lena, 300)
235     cv2.imwrite('Kirsch.bmp', kirsch)
236
237     robinson = robinson_operator(lena, 100)
238     cv2.imwrite('Robinson.bmp', robinson)
239
240     nevatia_babu = nevatia_babu_operator(lena, 15000)
241     cv2.imwrite('Nevatia-Babu.bmp', nevatia_babu)
242

```

(b) createNeighbor:

給定上下屆，就可以取出區塊的位置標號

```

13 def createNeighbor(start, end):
14     x = list(range(start, end))
15     return list(itertools.product(x, repeat=2))
16

```

(c) Robert:

1. 每個 pixel 的 neighbor 對應 2*2 的 mask
2. 乘上 mask 上的值並加總
3. 最後 mask 的值分別平方開根號
4. 得 gradient magnitude
5. if magnitude < threshold → 改成 255

```

18 def robert_operator(img, threshold):
19     r, c = img.shape
20     robert_img = np.zeros(img.shape, dtype=np.uint8)
21     r1_mask = [-1, 0, 0, 1]
22     r2_mask = [0, -1, 1, 0]
23     for i in range(r):
24         for j in range(c):
25             candidate = []
26             for m, n in neighbor_2:
27                 if 0 <= i+m < r and 0 <= j+n < c:
28                     candidate.append(img[i+m][j+n])
29             else:
30                 candidate.append(0)
31             r1 = 0
32             r2 = 0
33             for k in range(len(candidate)):
34                 r1 += int(candidate[k]) * r1_mask[k]
35                 r2 += int(candidate[k]) * r2_mask[k]
36             value = math.sqrt((r1**2) + (r2**2))
37             if (value < threshold):
38                 robert_img[i][j] = 255
39     return robert_img

```

(d) Prewitt、Sobel、Frei and Chen:

1. 不同於 Robert' s operator 的地方在於這三個是用 3*3 的 mask 而 Robert' s operator 是用 2*2 的 mask
2. 這三個 operator 彼此之間 mask 權重不同，可以得到不同的結果。

```

90 def frei_chen_operator(img, threshold):
91     r, c = img.shape
92     frei_chen_img = np.zeros(img.shape, dtype=np.uint8)
93     number = math.sqrt(2)
94     f1_mask = [-1, -number, -1,
95                0, 0, 0,
96                1, number, 1]
97     f2_mask = [-1, 0, 1,
98                -number, 0, number,
99                -1, 0, 1]
100
101     for i in range(r):
102         for j in range(c):
103             candidate = []
104             for m, n in neighbor_3:
105                 if 0 <= i+m < r and 0 <= j+n < c:
106                     candidate.append(img[i+m][j+n])
107             else:
108                 candidate.append(0)
109             f1 = 0
110             f2 = 0
111             for k in range(len(candidate)):
112                 f1 += int(candidate[k]) * f1_mask[k]
113                 f2 += int(candidate[k]) * f2_mask[k]
114             value = math.sqrt((f1**2) + (f2**2))
115             if (value < threshold):
116                 frei_chen_img[i][j] = 255
117     return frei_chen_img

```

```

42 def prewitt_operator(img, threshold):
43     r, c = img.shape
44     prewitt_img = np.zeros(img.shape, dtype=np.uint8)
45     p1_mask = [-1, -1, -1, 0, 0, 0, 1, 1, 1]
46     p2_mask = [-1, 0, 1, -1, 0, 1, -1, 0, 1]
47     for i in range(r):
48         for j in range(c):
49             candidate = []
50             for m, n in neighbor_3:
51                 if 0 <= i+m < r and 0 <= j+n < c:
52                     candidate.append(img[i+m][j+n])
53             else:
54                 candidate.append(0)
55             p1 = 0
56             p2 = 0
57             for k in range(len(candidate)):
58                 p1 += int(candidate[k]) * p1_mask[k]
59                 p2 += int(candidate[k]) * p2_mask[k]
60             value = math.sqrt((p1**2) + (p2**2))
61             if (value < threshold):
62                 prewitt_img[i][j] = 255
63     return prewitt_img

```

```

66 def sobel_operator(img, threshold):
67     r, c = img.shape
68     sobel_img = np.zeros(img.shape, dtype=np.uint8)
69     s1_mask = [-1, -2, -1, 0, 0, 0, 1, 2, 1]
70     s2_mask = [-1, 0, 1, -2, 0, 2, -1, 0, 1]
71     for i in range(r):
72         for j in range(c):
73             candidate = []
74             for m, n in neighbor_3:
75                 if 0 <= i+m < r and 0 <= j+n < c:
76                     candidate.append(img[i+m][j+n])
77             else:
78                 candidate.append(0)
79             s1 = 0
80             s2 = 0
81             for k in range(len(candidate)):
82                 s1 += int(candidate[k]) * s1_mask[k]
83                 s2 += int(candidate[k]) * s2_mask[k]
84             value = math.sqrt((s1**2) + (s2**2))
85             if (value < threshold):
86                 sobel_img[i][j] = 255
87     return sobel_img

```


(e) Kirsch、Robinson:

1. 用 3*3 的 mask 來做 edge detection
2. 用 8 個 3*3 的 mask，每個 mask 罩在每個 pixel 上都會有 gradient magnitude
3. 取 max(gradient magnitude) · 和 threshold 做比較來決定 pixel 值
4. <threshold 設 255

```

120 def kirsch_operator(img, threshold):
121     r, c = img.shape
122     kirsch_img = np.zeros(img.shape, dtype=np.uint8)
123     masks = [(-3, -3, 5, -3, 0, 5, -3, -3, 5),
124              (-3, 5, 5, -3, 0, 5, -3, -3, -3),
125              (5, 5, 5, -3, 0, -3, -3, -3, -3),
126              (5, 5, -3, 5, 0, -3, -3, -3, -3),
127              (5, -3, -3, 5, 0, -3, 5, -3, -3),
128              (-3, -3, -3, 5, 0, -3, 5, 5, -3),
129              (-3, -3, -3, -3, 0, -3, 5, 5, 5),
130              (-3, -3, -3, -3, 0, 5, -3, 5, 5)]
131     for i in range(r):
132         for j in range(c):
133             value = 0
134             for mask in masks:
135                 candidate = []
136                 for m, n in neighbor_3:
137                     if 0 <= i+m < r and 0 <= j+n < c:
138                         candidate.append(img[i+m][j+n])
139                 else:
140                     candidate.append(0)
141                 g = 0
142                 for k in range(len(candidate)):
143                     g += int(candidate[k]) * mask[k]
144                 value = max(value, g)
145             if (value < threshold):
146                 kirsch_img[i][j] = 255
147     return kirsch_img
148
149

```

```

151 def robinson_operator(img, threshold):
152     r, c = img.shape
153     robinson_img = np.zeros(img.shape, dtype=np.uint8)
154     masks = [(-1, 0, 1, -2, 0, 2, -1, 0, 1),
155              (0, 1, 2, -1, 0, 1, -2, -1, 0),
156              (1, 2, 1, 0, 0, 0, -1, -2, -1),
157              (2, 1, 0, 1, 0, -1, 0, -1, -2),
158              (1, 0, -1, 2, 0, -2, 1, 0, -1),
159              (0, -1, -2, 1, 0, -1, 2, 1, 0),
160              (-1, -2, -1, 0, 0, 0, 1, 2, 1),
161              (-2, -1, 0, -1, 0, 1, 0, 1, 2)]
162     for i in range(r):
163         for j in range(c):
164             value = 0
165             for mask in masks:
166                 candidate = []
167                 for m, n in neighbor_3:
168                     if 0 <= i+m < r and 0 <= j+n < c:
169                         candidate.append(img[i+m][j+n])
170                 else:
171                     candidate.append(0)
172                 g = 0
173                 for k in range(len(candidate)):
174                     g += int(candidate[k]) * mask[k]
175                 value = max(value, g)
176             if (value < threshold):
177                 robinson_img[i][j] = 255
178     return robinson_img
179

```

(f) Nevatia and Babu:

1. 用 6 個 5*5 的 mask 找出哪個 mask 對該 pixel 值有最大的 gradient magnitude
2. 拿該 magnitude 去和 threshold 做比較，
3. < threshold 設 255

```

182 def nevatia_babu_operator(img, threshold):
183     r, c = img.shape
184     nevatia_babu_img = np.zeros(img.shape, dtype=np.uint8)
185     masks = [(100, 100, 100, 100, 100, 100, 100, 100, 0, 0, 0, 0, 0, -100, -100, -100, -100, -100, -100, -100, -100, -100),
186              (100, 100, 100, 100, 100, 100, 100, 100, 70, -32, 180, 0, 0, -100, -100, -100, -100, -100, -100, -100, -100),
187              (0, -100, 32, -70, -100, -100, -100, -100, -100, -100, -100, -100, -100, 100, 100, 100, 100, 100, 100, 100),
188              (100, 100, 100, 32, -100, 100, 100, 92, -70, -100, 100, 100, 0, -100, -100, 100, 100, 100, 100, 100),
189              (100, -100, 100, 70, -32, -100, -100, 100, -32, -100, -100, -100, -100, 100, 100, 100, 100, 100, 100, 100),
190              (-100, -100, 0, 100, 100, -100, -100, -100, 0, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100),
191              (0, 100, 100, -100, -100, -100, 0, 100, 100, -100, -100, -100, 0, 100, 100, 100, 100, 100, 100, 100),
192              (-100, 32, 100, 100, 100, -100, -70, 70, 100, 100, -100, -100, 0, 100, 100, 100, 100, 100, 100, 100),
193              (100, 100, -100, -100, -70, 70, 100, -100, -100, -100, -100, -32, 100),
194              (100, 100, 100, 100, 100, -32, 70, 100, 100, 100, -100, -70, 0, 92, 100, -100, -100, -100, -70, 32, -100, -100, -100, -100)]
195     for i in range(r):
196         for j in range(c):
197             value = 0
198             for mask in masks:
199                 candidate = []
200                 for m, n in neighbor_5:
201                     if 0 <= i+m < r and 0 <= j+n < c:
202                         candidate.append(img[i+m][j+n])
203                 else:
204                     candidate.append(0)
205                 g = 0
206                 for k in range(len(candidate)):
207                     g += int(candidate[k]) * mask[k]
208                 value = max(value, g)
209             if (value < threshold):
210                 nevatia_babu_img[i][j] = 255
211     return nevatia_babu_img
212

```