

Computer Vision HW9

R06922075 翁瑋

- Write programs to generate the following gradient magnitude images and choose proper thresholds to get the binary edge images: Generate additive white Gaussian noise
 1. Roberts operator
 2. Prewitt edge detector
 3. Sobel edge detector
 4. Frei and Chen gradient operator
 5. Kirsch compass operator
 6. Robinson compass operator
 7. Nevatia-Babu 5X5 operator

```
23 def Robert(im, threshold):
24     pixels = im.load()
25     im_robert = Image.new('L', im.size, 'black')
26     masks = [[[-1,0],[0,1]], [[0,-1],[1,0]]]
27
28     for i in range(im.size[0]):
29         for j in range(im.size[0]):
30             r_sq = []
31             for num_mask in range(len(masks)):
32                 r = 0
33                 for m in range(2):
34                     for n in range(2):
35                         r += getpixel(im,i+m,j+n)*masks[num_mask][m][n]
36                 r_sq.append(r**2)
37             result = 0 if np.sqrt(sum(r_sq)) > threshold else 255
38             im_robert.putpixel((i,j), result)
39
40     return im_robert
```

Robert Operator

```
42 def Prewitt(im, threshold):
43     pixels = im.load()
44     im_prewitt = Image.new('L', im.size, 'black')
45     masks = [[[-1,-1,-1],[0,0,0],[1,1,1]], [[-1,0,1],[-1,0,1],[-1,0,1]]]
46
47     size = len(masks[0])
48
49     for i in range(im.size[0]):
50         for j in range(im.size[0]):
51             r_sq = []
52             for num_mask in range(len(masks)):
53                 r = 0
54                 for m in range(3):
55                     for n in range(3):
56                         r += getpixel(im,i+n,j+m)*masks[num_mask][m][n]
57                 r_sq.append(r**2)
58             result = 0 if np.sqrt(sum(r_sq)) > threshold else 255
59             im_prewitt.putpixel((i,j), result)
60
61     return im_prewitt
```

Prewitt Operator

```

63 def Sobel(im, threshold):
64     pixels = im.load()
65     im_sobel = Image.new('L', im.size, 'black')
66     masks = [[[-1,-2,-1],[0,0,0],[1,2,1]] , [[-1,0,1],[-2,0,2],[-1,0,1]]]
67
68     size = len(masks[0])
69
70     for i in range(im.size[0]) :
71         for j in range(im.size[0]) :
72             r_sqt = []
73             for num_mask in range(len(masks)) :
74                 r = 0
75                 for m in range(-(size//2),size//2+1) :
76                     for n in range(-(size//2),size//2+1) :
77                         r += getpixel(im,i+n,j+m)*masks[num_mask][m][n]
78                 r_sqt.append(r**2)
79             result = 0 if np.sqrt(sum(r_sqt)) > threshold else 255
80             im_sobel.putpixel((i,j) , result)
81
82     return im_sobel

```

Sobel Operator

```

84 def Frei_and_Chien(im, threshold):
85     pixels = im.load()
86     im_fac = Image.new('L', im.size, 'black')
87     masks = [[[-1,-np.sqrt(2),-1],[0,0,0],[1,np.sqrt(2),1]] , [[-1,0,1],[-np.sqrt(2),0,np.sqrt(2)],[-1,0,1]]]
88
89     size = len(masks[0])
90
91     for i in range(im.size[0]) :
92         for j in range(im.size[0]) :
93             r_sqt = []
94             for num_mask in range(len(masks)) :
95                 r = 0
96                 for m in range(-(size//2),size//2+1) :
97                     for n in range(-(size//2),size//2+1) :
98                         r += float(getpixel(im,i+n,j+m))*masks[num_mask][m][n]
99                 r_sqt.append(r**2)
100             result = 0 if np.sqrt(sum(r_sqt)) > threshold else 255
101             im_fac.putpixel((i,j) , result)
102
103     return im_fac

```

Frei and Chen Operator

```

106 def Kirsch(im, threshold):
107     pixels = im.load()
108     im_kirsch = Image.new('L', im.size, 'black')
109     masks = [[[-3,-3,5],[-3,0,5],[-3,-3,5]] , [[-3,5,5],[-3,0,5],[-3,-3,-3]] ,
110             [[5,5,5],[-3,0,-3],[-3,-3,-3]] , [[5,5,-3],[5,0,-3],[-3,-3,-3]] ,
111             [[5,-3,-3],[5,0,-3],[5,-3,-3]] , [[-3,-3,-3],[5,0,-3],[5,5,-3]] ,
112             [[-3,-3,-3],[-3,0,-3],[5,5,5]] , [[-3,-3,-3],[-3,0,5],[-3,5,5]]]
113
114     size = len(masks[0])
115
116     for i in range(im.size[0]) :
117         for j in range(im.size[0]) :
118             r_sqt = []
119             for num_mask in range(len(masks)) :
120                 r = 0
121                 for m in range(-(size//2),size//2+1) :
122                     for n in range(-(size//2),size//2+1) :
123                         r += getpixel(im,i+n,j+m)*masks[num_mask][m][n]
124                 r_sqt.append(r)
125             result = 0 if max(r_sqt) > threshold else 255
126             im_kirsch.putpixel((i,j) , result)
127
128     return im_kirsch

```

Kirsch Operator

```

130 def Robinson(im, threshold):
131     pixels = im.load()
132     im_robinson = Image.new('L', im.size, 'black')
133     masks = [[[-1,0,1],[-2,0,2],[-1,0,1]] , [[0,1,2],[-1,0,1],[-2,-1,0]] ,
134             [[1,2,1],[0,0,0],[-1,-2,-1]] , [[2,1,0],[1,0,-1],[0,-1,-2]] ,
135             [[1,0,-1],[2,0,-2],[1,0,-1]] , [[0,-1,-2],[1,0,-1],[2,1,0]] ,
136             [[-1,-2,-1],[0,0,0],[1,2,1]] , [[-2,-1,0],[-1,0,1],[0,1,2]]]
137
138     size = len(masks[0])
139
140     for i in range(im.size[0]) :
141         for j in range(im.size[0]) :
142             r_sqt = []
143             for num_mask in range(len(masks)) :
144                 r = 0
145                 for m in range(-(size//2),size//2+1) :
146                     for n in range(-(size//2),size//2+1) :
147                         r += getpixel(im,i+n,j+m)*masks[num_mask][m][n]
148                 r_sqt.append(r)
149             result = 0 if max(r_sqt) > threshold else 255
150             im_robinson.putpixel((i,j) , result)
151
152     return im_robinson

```

Robinson Operator

```

154 def Nevatia_and_Babu(im, threshold):
155     pixels = im.load()
156     im_nab = Image.new('L', im.size, 'black')
157     masks = [[ [ 100, 100, 100, 100, 100] , [ 100, 100, 100, 100, 100] , [ 0, 0, 0, 0, 0] , [-100,
158             [ 100, 100, 100, 100, 100] , [ 100, 100, 100, 78, -32] , [ 100, 92, 0, -92, -100] , [ 32,
159             [ 100, 100, 100, 32, -100] , [ 100, 100, 92, -78, -100] , [ 100, 100, 0, -100, -100] , [ 100,
160             [[-100,-100, 0, 100, 100] , [-100,-100, 0, 100, 100] , [-100,-100, 0, 100, 100] , [-100,
161             [[-100, 32, 100, 100, 100] , [-100, -78, 92, 100, 100] , [-100,-100, 0, 100, 100] , [-100,
162             [[ 100, 100, 100, 100, 100] , [ -32, 78, 100, 100, 100] , [-100, -92, 0, 92, 100] , [-100,
163
164     size = len(masks[0])
165
166     for i in range(im.size[0]) :
167         for j in range(im.size[0]) :
168             r_sqt = []
169             for num_mask in range(len(masks)) :
170                 r = 0
171                 for m in range(-(size//2),size//2+1) :
172                     for n in range(-(size//2),size//2+1) :
173                         r += getpixel(im,i+n,j+m)*masks[num_mask][m][n]
174                 r_sqt.append(r)
175             result = 0 if max(r_sqt) > threshold else 255
176             im_nab.putpixel((i,j) , result)
177
178     return im_nab

```

Nevatia and Babu Operator

Result :

Original	Roberts operator(12)
	
Prewitt edge detector(24)	Sobel edge detector(38)
	

Frei and Chen gradient operator(30)



Kirsch compass operator(135)



Robinson compass operator(43)



Nevatia-Babu 5X5 operator(12500)

