**Assignment 1 Canny Edge Detector**

Jiaying Li

Jl10919

**(i)File name of your source code:** canny\_edge\_detector.py

**(ii)Instructions on how to run your program and instructions on how to compile your program if your program requires compilation.**

(a) open canny\_edge\_detector.py

(b) enter the path of test image in main() : line227.

(c) set t1:

chose one formula to set the threshold in double\_thresholding():

method1: maxima \* 0.1 – line181

method2: median \* 0.66 – line183

method3: mean \* 0.66 – line185

**OR**

manually set t1 in main():line 229

(d) run canny\_edge\_detector.py

This code is written in python2.7 and it needs cv2 and numpy library support. Please ensure these libraries have been installed. And the program reads .bmp file.

The result will be shown:

gaussian\_output.bmp – image after doing gaussian smoothing

sobel\_xout.bmp – image showing the horizontal gradient responses after doing sobel operation

sobel\_yout.bmp -- image showing the vertical gradient responses after doing sobel operation

magnitude.bmp – image showing magnitude

nms.bmp – image showing result after doing non maxima suppression

threshold.bmp – binary edge image after doing double thresholding

**(iii) Output image results (1) to (5) for all test images.**

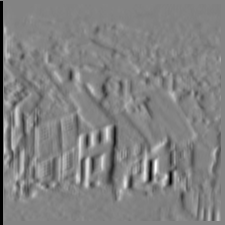
Houses-225.bmp:

*(1) Gaussian smoothing*

图片包含 户外, 天空, 照片

描述已自动生成

*(2) horizontal gradient responses :*



*vertical gradient responses:*

图片包含 户外, 海滩, 水, 海洋

描述已自动生成

*(3) gradient magnitude image*

图片包含 黑色

描述已自动生成

*(4) gradient magnitude image after non-maxima suppression.*

*图片包含 文字, 黑板

描述已自动生成*

*(5) Binary edge map using double thresholding.*

*(here, T1 = 28 , T1=maxima\*0.1)*

图片包含 文字

描述已自动生成

Zebra-crossing-1.bmp:

*(1) Gaussian smoothing*

图片包含 户外, 树, 天空, 道路

描述已自动生成

*(2) horizontal gradient responses :*

图片包含 户外, 自然, 天空

描述已自动生成

*vertical gradient responses:*

*图片包含 户外, 自然, 道路

描述已自动生成*

*(3) gradient magnitude image*

图片包含 户外, 道路

描述已自动生成

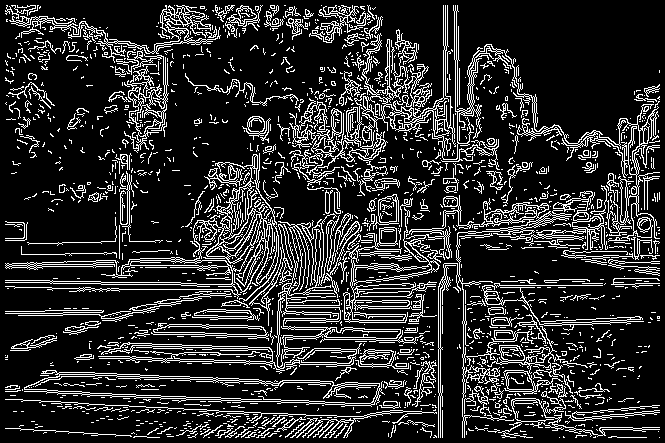
*(4) gradient magnitude image after non-maxima suppression.*

*图片包含 深色, 黑色, 照片

描述已自动生成*

*(5) Binary edge map using double thresholding.*

*(here, T1 = 12 ,T1=mean\*0.66)*



**(v) The source code of your program.**

1. **import** cv2
2. **import** numpy as np
4. # normalize the image matrix to visualize
5. **def** normalization(img):
7. min\_val = np.min(img.ravel())
8. max\_val = np.max(img.ravel())
9. output = (img.astype('float') - min\_val) / (max\_val - min\_val) \* 255
11. **return** output

14. # read image
15. **def** read\_image(path):
16. rawimage = cv2.imread(path)
17. # turn image into grayscale image
18. rawimage = cv2.cvtColor(rawimage, cv2.COLOR\_BGR2GRAY)
19. height, width = rawimage.shape
21. **return** rawimage, height, width

24. # do Gaussian filtering
25. **def** guassian\_smoothing(rawimage, height, width):
26. # initialize Gaussian operator
27. gaussian = np.array([
28. [1, 1, 2, 2, 2, 1, 1],
29. [1, 2, 2, 4, 2, 2, 1],
30. [2, 2, 4, 8, 4, 2, 2],
31. [2, 4, 8, 16, 8, 4, 2],
32. [2, 2, 4, 8, 4, 2, 2],
33. [1, 2, 2, 4, 2, 2, 1],
34. [1, 1, 2, 2, 2, 1, 1]
35. ])
37. # initial gussian filtering output (removing the border)
38. gaussian\_out = np.zeros([height, width])
40. # do cross-correlation operation
41. res = 0
42. **for** i **in** range(3, height - 3):
43. **for** j **in** range(3, width - 3):
44. res = 0.0
45. **for** m **in** range(7):
46. **for** n **in** range(7):
47. res += rawimage[i + m - 3, j + n - 3] \* gaussian[m, n]
48. gaussian\_out[i, j] = round(res / 140)
50. # save the output image
51. cv2.imwrite("./gaussian\_output.bmp", gaussian\_out)
52. ga\_height, ga\_width = gaussian\_out.shape
54. **return** gaussian\_out, ga\_height, ga\_width

57. # do sobel operation
58. **def** sobel\_operation(gaussian\_out, ga\_height, ga\_width):
59. # horizontal sobel operator
60. sobel\_operator\_x = np.array([
61. [-1, 0, 1],
62. [-2, 0, 2],
63. [-1, 0, 1]
64. ])
66. # vertical sobel operator
67. sobel\_operator\_y = np.array([
68. [1, 2, 1],
69. [0, 0, 0],
70. [-1, -2, -1]
71. ])
73. # initialize sobel-operation output
74. sobel\_xout = np.zeros([ga\_height , ga\_width], dtype=float)
75. sobel\_yout = np.zeros([ga\_height , ga\_width], dtype=float)
77. # do cross-correlation operation
78. resx = 0
79. resy = 0
80. **for** i **in** range(4, ga\_height - 4):
81. **for** j **in** range(4, ga\_width - 4):
82. resx = 0.0
83. resy = 0.0
84. **for** m **in** range(3):
85. **for** n **in** range(3):
86. resx += gaussian\_out[i + m - 1, j + n - 1] \* sobel\_operator\_x[m, n]
87. resy += gaussian\_out[i + m - 1, j + n - 1] \* sobel\_operator\_y[m, n]
88. sobel\_xout[i, j] = resx
89. sobel\_yout[i, j] = resy
91. # normalize the gradient
92. # visualize it and save it
93. sobel\_xout\_v = normalization(sobel\_xout)
94. sobel\_yout\_v = normalization(sobel\_yout)
96. cv2.imwrite("./sobel\_xout.bmp", sobel\_xout\_v)
97. cv2.imwrite("./sobel\_yout.bmp", sobel\_yout\_v)
99. **return** sobel\_xout, sobel\_yout
101. # calculate the magnitude
102. **def** magnitude(sobel\_xout, sobel\_yout):
103. #so\_height, so\_width = sobel\_yout.shape
104. magnitude = np.sqrt(sobel\_xout \*\* 2 + sobel\_yout \*\* 2)
105. # normorlize the magnitude
106. # visulize it and save it
107. magnitude\_v = normalization(magnitude)
108. cv2.imwrite("./magnitude.bmp", magnitude\_v)
110. **return** magnitude


114. **def** gradient\_angle(sobel\_xout, sobel\_yout):
115. # compute the angle of gradient (the output is in the range of [-pi,pi])
116. angle = np.arctan2(sobel\_yout, sobel\_xout)
118. **return** angle

121. # do non\_maxima\_suppression
122. **def** non\_maxima\_suppression(angle, magnitude):
123. # quantize angle of the gradient to sector
124. an\_height, an\_width = angle.shape
125. **for** i **in** range(an\_height):
126. **for** j **in** range(an\_width):
127. **if** (- np.pi / 8) <= angle[i][j] < (np.pi / 8) **or** (7 \* np.pi / 8) <= angle[i][j] < (9 \* np.pi / 8) **or** (
128. -9 \* np.pi / 8) <= angle[i][j] < (-7 \* np.pi / 8):
129. angle[i][j] = 0
130. **elif** (np.pi / 8) <= angle[i][j] < (3 \* np.pi / 8) **or** (-7 \* np.pi / 8) <= angle[i][j] < (-5 \* np.pi / 8):
131. angle[i][j] = 1
132. **elif** (3 \* np.pi / 8) <= angle[i][j] < (5 \* np.pi / 8) **or** (-5 \* np.pi / 8) <= angle[i][j] < (-3 \* np.pi / 8):
133. angle[i][j] = 2
134. **elif** (5 \* np.pi / 8) <= angle[i][j] < (7 \* np.pi / 8) **or** (-3 \* np.pi / 8) <= angle[i][j] < (- np.pi / 8):
135. angle[i][j] = 3
137. # do non-maxima suppression (nms)
138. nms\_out = magnitude
139. **for** i **in** range(5, an\_height - 5):
140. **for** j **in** range(5, an\_width - 5):
141. **if** angle[i][j] == 0:
142. **if** magnitude[i][j] <= magnitude[i][j - 1] **or** magnitude[i][j] <= magnitude[i][j + 1]:
143. nms\_out[i][j] = 0
144. **elif** angle[i][j] == 1:
145. **if** magnitude[i][j] <= magnitude[i - 1][j + 1] **or** magnitude[i][j] <= magnitude[i + 1][j - 1]:
146. nms\_out[i][j] = 0
147. **elif** angle[i][j] == 2:
148. **if** magnitude[i][j] <= magnitude[i - 1][j] **or** magnitude[i][j] <= magnitude[i + 1][j]:
149. nms\_out[i][j] = 0
150. **elif** angle[i][j] == 3:
151. **if** magnitude[i][j] <= magnitude[i - 1][j - 1] **or** magnitude[i][j] <= magnitude[i + 1][j + 1]:
152. nms\_out[i][j] = 0
154. # handle the edge cases in nms
155. nms\_height, nms\_width = nms\_out.shape
156. nms\_out[4][4], nms\_out[-5][-5], nms\_out[4][-5], nms\_out[-5][4] = 0, 0, 0, 0
157. **for** j **in** range(nms\_width):
158. **if** nms\_out[4][j] != 0:
159. nms\_out[4][j] = 0
160. **if** nms\_out[-5][j] != 0:
161. nms\_out[-5][j] = 0
162. **for** i **in** range(nms\_height):
163. **if** nms\_out[i][4] != 2:
164. nms\_out[i][4] = 0
165. **if** nms\_out[i][-5] != 2:
166. nms\_out[i][-5] = 0
168. # normorlize the magnitude after nms
169. # visulize it and save it
170. nms\_out\_v = normalization(nms\_out)
171. cv2.imwrite("./nms.bmp", nms\_out\_v)
173. **return** nms\_out, angle

176. # do double thresholding
177. **def** double\_thresholding(nms\_out, angle, t1):
178. # initialize threshold 1 and threshold 2
179. # set t1 method
180. # method1: maxima \* 0.1
181. #t1 = np.max(nms\_out) \* 0.1
182. # method2: median \* 0.66
183. #t1 = np.median(nms\_out) \* 0.66
184. # method3: mean \* 0.66
185. #t1 = np.mean(nms\_out) \* 0.66
186. # method4: mannually
187. #t1 = 12
188. **print** t1
189. t2 = 2 \* t1
190. nms\_height, nms\_width = nms\_out.shape
191. # initialize double-thresholding output
192. threshold\_out = np.ones([nms\_height, nms\_width])
194. # assign the region according to t1,t2
195. **for** i **in** range(nms\_height):
196. **for** j **in** range(nms\_width):
197. **if** nms\_out[i][j] < t1:
198. threshold\_out[i][j] = 0
199. **elif** nms\_out[i][j] > t2:
200. threshold\_out[i][j] = 255
201. **else**:
202. **continue**
204. # assign the region of those in range[t1,t2]
205. **for** i **in** range(1, nms\_height - 1):
206. **for** j **in** range(1, nms\_width - 1):
207. **if** threshold\_out[i][j] == 1:
208. find\_flag = 0
209. **for** ni **in** range(-1, 2):
210. **for** nj **in** range(-1, 2):
211. **if** nms\_out[i + ni][j + nj] > t2 **and** abs(angle[i + ni][j + nj] - angle[i][j]) <= np.pi / 4:
212. threshold\_out[i][j] = 255
213. find\_flag = 1
214. **break**
215. **if** find\_flag == 1:
216. **break**
217. **if** find\_flag == 0:
218. threshold\_out[i][j] = 0
220. # visulize it and save it
221. cv2.imwrite("./threshold.bmp", threshold\_out)
223. **return** threshold\_out
225. **if** \_\_name\_\_ == '\_\_main\_\_':
226. # enter image path
227. path = '/Users/JesLee/Desktop/cv/canny\_edge\_detector/jl10919-project1/test\_image/Zebra-crossing-1.bmp'
228. #set threshold 1 manually or use one formula in the double\_thresholding function
229. t1 = 12
230. # to present all data while printing a matrix
231. np.set\_printoptions(threshold=np.inf)
232. #red image
233. rawimage, height, width = read\_image(path)
234. #do gaussian smoothing
235. gaussian\_out, ga\_height, ga\_width = guassian\_smoothing(rawimage, height, width)
236. #do sobel operation
237. sobel\_xout, sobel\_yout = sobel\_operation(gaussian\_out, ga\_height, ga\_width)
238. #get magnitude
239. magnitude = magnitude(sobel\_xout, sobel\_yout)
240. #get angle
241. angle = gradient\_angle(sobel\_xout, sobel\_yout)
242. #do non-maxima suppression
243. nms\_out, angle = non\_maxima\_suppression(angle, magnitude)
244. #do double thresholding
245. result = double\_thresholding(nms\_out, angle, t1)