HW 03 – REPORT

소속 : 정보컴퓨터공학부

학번 : 201824633

이름 : 김유진

1. 서론

Canny Edge Detection



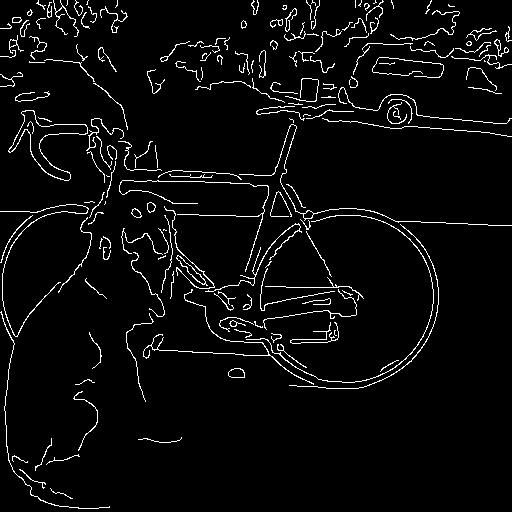
1. Filter Image with derivative of Gaussian
   1. 가우시안 블러를 이용해서 이미지를 흐리게 만든다.
2. Find magnitude and orientation of gradient
   1. gradient와 theta를 구한다. 이를 이용해 초기 edge를 찾는다.



1. Non-maximum suppression
   1. 2번에서 얻은 edge를 보다 sharp하게 만들어준다.



1. Thresholding and linking
   1. threshold 값보다 큰 픽셀만 출력을 해준다. 이때, threshold를 2개로 설정해 더 높은 threshold 값을 high, 낮은 threshold 값을 low라고 해준다. high보다 높을때와 high와 low 사이일 때를 분류해 linking을 해준다.



2. 본론

2-1. Noise reduction

1. load image
2. image to grayscale
3. gaussconvolve2d(array, sigma) sigma = 1.6
4. PIL -> show()

|  |
| --- |
| ######### 1. Noise reduction  # Load iguana.bmp  iguana = Image.open('[HW03] Canny Edge Detection/iguana.bmp')  # Convert it to Greyscale  grey\_iguana = iguana.convert('L')  # to numpy array  grey\_iguana\_array = np.asarray(grey\_iguana, dtype=np.float32)  # blurs the image using gaussconvolved2d(array,sigma)  new\_iguana\_array = gaussconvolve2d(grey\_iguana\_array, 1.6)  # use PIL to show both the original and filtered images.  new\_iguana = Image.fromarray(new\_iguana\_array)  iguana.show()  new\_iguana.show() |

파충류, 도마뱀, 실외, 녹색이(가) 표시된 사진

자동 생성된 설명파충류, 도마뱀이(가) 표시된 사진

자동 생성된 설명

<좌> original <우> 변형된 이미지

2-2. Finding the intensity gradient of the image : sobel\_filter(img)

1. sobel filter 적용. => convolve2d
2. gradient 및 theta 구하기

|  |
| --- |
| def sobel\_filters(img):    # apply Sobel filter in the x, y direction  filter\_x = [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]  filter\_y = [[1, 2, 1], [0, 0, 0], [-1, -2, -1]]  # apply the convolve2d function to obtain the intensity x, y value  Ix = convolve2d(img, filter\_x)  Iy = convolve2d(img, filter\_y)  # Formulate Gradient and Theta  # G: Magnitude of gradient at each pixel in img.  # theta: Direction of gradient at each pixel in img.  G = np.hypot(Ix, Iy)  theta = np.arctan2(Iy, Ix)  # mapping value => 0 ~ 255  G\_max = np.max(G)  G\_min = np.min(G)  G = G / (G\_max - G\_min) \* 255    return (G, theta) |

텍스트, 실내이(가) 표시된 사진

자동 생성된 설명

sobel filter을 적용한 이미지

2-3. non maximum suppression

1. for thin edge
2. (0, 45, 90, 135)
3. theta : radian -> degree 변환

|  |
| --- |
| def non\_max\_suppression(G, theta):  # theta is radians. radian -> degree  theta = np.rad2deg(theta)  # res.shape = G.shape  # if pixel is less than neighbor, that pixel is 0. => zeros  res = np.zeros(G.shape)  for i in range(1, G.shape[0]-1):  for j in range(1, G.shape[1]-1):  # angle standard = [0, 45, 90, 135] => 0 <= \_\_ < 180  angle = theta[i][j]  if angle >= 180:  angle -= 180  if angle < 0:  angle += 180  # 중간 지점을 기준으로 angle 값이 가리키는 방향에 있는 픽셀 2개를 기준으로 더 큰 값이 있는지 없는지 비교한다.  num1, num2 = 0, 0  if 22.5 <= angle < 67.5: # 45  num1 = G[i-1][j+1]  num2 = G[i+1][j-1]  elif 67.5 <= angle < 112.5: # 90  num1 = G[i-1][j]  num2 = G[i+1][j]  elif 112.5 <= angle < 157.5: # 135  num1 = G[i-1][j-1]  num2 = G[i+1][j+1]  else: # 0 : 157.5 ~ 180 / 0 ~ 22.5  num1 = G[i][j-1]  num2 = G[i][j+1]    # num1 보다 크고 num2 보다 큰 경우 G[i][j]를 res[i][j]에 넣어준다.  if (G[i][j] > num1 and G[i][j] > num2):  res[i][j] = G[i][j]  return res |

텍스트, 칠판이(가) 표시된 사진

자동 생성된 설명

non maximum suppression 적용 이미지

2-4. Double threshold

1. strong, weak and non-relevant
2. diff = max(image) – min(image)
3. T\_high = min(image) + diff \* 0.15
4. T\_low = min(image) + diff \* 0.03

|  |
| --- |
| def double\_thresholding(img):    # use the expressions to determine threshold values  diff = np.max(img) - np.min(img)  T\_high = np.min(img) + diff \* 0.15  T\_low = np.min(img) + diff \* 0.03  # default = no-relationship  res = np.zeros(img.shape)  # strong : 175 + 80 = 255  res += np.where(img > T\_high, 175, 0)  # weak  res += np.where(T\_low < img, 80, 0)    return res |

텍스트, 그물, 옥외설치물이(가) 표시된 사진

자동 생성된 설명

double threshold 적용된 이미지

2-5. Edge Tracking by hysteresis

1. strong line에 연결된 weak edge를 strong으로 바꿔주고 그렇지않으면 edge에서 탈락시킨다.
2. DFS를 사용해야한다.

|  |
| --- |
| def hysteresis(img):    strong = 255  weak = 80  # use dfs on all strong edges to obtain connected weak edges  def dfs(i, j):  # neighbor 8 pixels  for a in range(i-1, i+2):  for b in range(j-1, j+2):  # weak인 경우, strong으로 바꿔주고 그 점을 기준으로 또 다시 dfs  if img[a][b] == weak:  img[a][b] = strong  dfs(a, b)  # 종료 조건 설정. strong인 경우 탐색 중지  elif img[a][b] == strong:  continue    # weak edge -> strong [link]  for i in range(1, img.shape[0] - 1):  for j in range(1, img.shape[1] - 1):  if img[i][j] == strong:  dfs(i, j)  # strong인 경우만 strong value 그대로 복사, 아니면 0으로  res = np.where(img == strong, strong, 0)  return res |



최종결과 이미지

3. 결론

- 단계에 따른 이미지

|  |  |
| --- | --- |
| Original | 파충류, 도마뱀, 실외, 녹색이(가) 표시된 사진  자동 생성된 설명 |
| 1 | 파충류, 도마뱀이(가) 표시된 사진  자동 생성된 설명 | |
| 2 | 텍스트, 실내이(가) 표시된 사진  자동 생성된 설명 | |
| 3 | 텍스트, 칠판이(가) 표시된 사진  자동 생성된 설명 | |
| 4 | 텍스트, 그물, 옥외설치물이(가) 표시된 사진  자동 생성된 설명 | |
| 5 |  | |