Introduction to Computer Vision HW03

- Canny Edge Detection -

2021-1학기 컴퓨터비전개론 059분반 201724437 김재현

Before assignments

주어진 과제를 해결하기 전, 이전 과제에 쓰인 함수 일부를 수정하고 새로운 유틸리티 함수를 작성하였습니다.

Line 14~19

```
def gauss1d(sigma):
   …
   d = math.ceil(sigma*6)
   d += ( 1 if (d%2 == 0) else 0)
```

위에 나타난 바와 같이 gauss1d()의 길이 d를 구하는 부분을 수정하였습니다. 우선 sigma * 6의 값을 올림하여 정수를 얻고, 이것이 짝수인지 홀수인지의 여부에 따라 1을 더하거나 더하지 않는 방식으로 'sigma * 6와 같거나 큰 최소의 홀수'를 구하도록 하였습니다.

Line 25~29

```
def gauss2d(sigma):
   …
   return g/np.sum(g)
```

또 gauss2d() 함수에서 얻은 gauss filter의 합이 1.0을 초과하는 문제가 발생하여, 이를 필터 전체의 원소의 합으로 나누어 normalize한 후 반환하도록 수정하였습니다.

Line 56~58

```
def saveImageFromArray(array, filename, mode):
    img = Image.fromarray(array.astype('uint8'))
    img.save(filename, mode)
```

한편 과제 전체에 쓰일 유틸리티 함수인 saveImageFromArray( )도 작성하였습니다. 이 함수는 이미지 배열을 인자로 받아 uint8 배열로 변환한 후, 한 번 더 이미지로 변환시킵니다. 그리고 같이 전달받은 파일명(filename)과 저장방식(mode)을 이용해 이미지를 파일로 저장합니다.

Part 1. Noise reduction

PIL과 numpy를 이용하여 이미지를 열고 grayscale로 바꾼 뒤, numpy 배열로 변환하여 필터를 적용했습니다. 코드와 결과 이미지는 아래와 같습니다.

Line 64~76

```
iguanaImg = Image.open("iguana.bmp")
iguanaImg.show()

iguanaImg = iguanaImg.convert('L')
iguanaImgArr = np.asarray(iguanaImg)
iguanaFImgArr = gaussconvolve2d(iguanaImgArr, 1.6)

iguanaFImg = Image.fromarray(iguanaFImgArr)
iguanaFImg.show()

saveImageFromArray(iguanaFImgArr, 'iguana_filtered.bmp','PNG')
```

>>


그림 1,2 – iguana.bmp, iguana_filtered.bmp

Part 2. Finding the intensity gradient of the image

gaussian filter를 거친 이미지에 sobel filter를 적용하여 이미지의 gradient와 각 픽셀에서의 theta 값을 얻어낼 수 있습니다. gradient를 구할 때는 np.hypot() 함수를 사용한 뒤 gradient의 최댓값으로 나눈 후 255를 곱하여 값들을 0~255 사이로 매핑했습니다. theta를 구할 때는 np.arctan2() 함수를 사용했습니다. 코드와 결과 이미지는 아래와 같습니다.

Line 83~111

```
def sobel_filters(img):
    # Define sobel x, y filter respectively
    sobelX = np.array([[-1., 0., 1.], [-2., 0., 2.], [-1., 0., 1.]])
    sobelY = np.array([[1., 2., 1.], [0., 0., 0.], [-1., -2., -1.]])

    imgArr = np.asarray(img)
    iX = convolve2d(imgArr, sobelX)
    iY = convolve2d(imgArr, sobelY)

    G = np.hypot(iX, iY)
    G = G*(255.0/G.max())

    theta = np.arctan2(iY, iX)

    return (G, theta)
```

```
iguanaG, iguanaTheta = sobel_filters(iguanaFImg)

iguanaGImg = Image.fromarray(iguanaG)
iguanaGImg.show()

saveImageFromArray(iguanaG, 'iguana_gradient.bmp', 'PNG')
```

>>

그림 3 – iguana_gradient.bmp

Part 3. Non-Maximum Suppression

Part 2에서 구한 gradient의 각각의 픽셀에 대하여 그 theta 값을 (0, 45, 90, 135)도 중 하나로 매핑하고, 이 방향에 놓인 이웃 픽셀들과 비교했을 때 최대인 픽셀, 즉 local maximum만을 이미지에 남기는 과정을 거쳐야 합니다.

우선 저는 theta 값을 (0, 45, 90, 135)도가 아니라 (0, 1, 2, 3)에 매핑한다고 하면, round 함수를 이용할 수 있기 때문에 매핑이 더 간단해질 것이라 판단했습니다. 따라서 theta 값을 radian 수치로 두 번 나누고, 이들이 (0, 1, 2, 3) 중 하나가 되도록 round한 뒤 벗어나는 값들을 0으로 맞추어 주었습니다.

이렇게 매핑된 theta 값들을 하나의 index로 보고, 이 theta index에 해당하는 pixel offset 배열을 미리 정의하였습니다. 그 다음 gradient의 각각의 픽셀과 (offset들을 빼거나 더해서 얻어낸) 이웃 픽셀들을 비교하여, local maximum을 만족하는 gradient의 픽셀만을 남겨두었습니다. 해당 코드와 결과 이미지는 아래와 같습니다.

Line 118~164

```
def non_max_suppression(G, theta):

    theta = theta % math.pi
    theta = theta / (math.pi/4)
    theta = theta.round()
    theta[theta == 4] = 0
    theta = theta.astype('uint8')


    base = G.copy()
    res = np.zeros(np.shape(G))

    direction = np.array([[1,0], [1,1], [0,1], [-1, 1]])
```

```
    for i in range(1, res.shape[0]-1):
        for j in range(1, res.shape[1]-1):
            g1 = base[i-direction[theta[i,j],1], j+direction[theta[i,j],0]]
            g2 = base[i+direction[theta[i,j],1], j-direction[theta[i,j],0]]
            if((base[i][j]>=g1) & (base[i][j]>=g2)):
                res[i][j] = base[i][j]

    return res
iguanaNMS = non_max_suppression(iguanaG, iguanaTheta)

iguanaNMSImg = Image.fromarray(iguanaNMS)
iguanaNMSImg.show()

saveImageFromArray(iguanaNMS, 'iguana_NMS.bmp', 'PNG')
```

>>

그림 4 – iguana_NMS.bmp

Part 4. Double threshold

Part 3에서 얻은 NMS를 거친 이미지에 대해 상한값과 하한값을 적용하여 범위를 나누고, 범위에 해당하는 값에 매핑해야 합니다.

우선 픽셀의 최댓값과 최솟값을 통해 diff 값을 구하고, 이를 다시 최솟값에 일정 비율만큼 더하는 방식으로 high threshold 값과 low threshold 값을 얻었습니다. 이 후 numpy array의 Boolean indexing을 이용하여 두 threshold 값 이하, 사이, 이상에 존재하는 값들을 각각 0, 80, 255로 매핑해주었습니다. 해당 코드와 결과 이미지는 아래와 같습니다.

Line 170~191

```
def double_thresholding(img):
    diff = img.max() - img.min()
    Th = img.min() + diff * 0.15
    Tl = img.min() + diff * 0.03

    res = np.zeros(np.shape(img))
    res[img < Tl] = 0
```

```
    res[(Tl <= img) & (img < Th)] = 80
    res[Th <= img] = 255

    return res

iguanaDT = double_thresholding(iguanaNMS)

iguanaDTImg = Image.fromarray(iguanaDT)
iguanaDTImg.show()

saveImageFromArray(iguanaDT, 'iguana_DT.bmp', 'PNG')
```

>>

그림 5 - iguana_DT.bmp

Part 5. Double threshold

Part 4에서 얻은 이미지에서 DFS를 통해 strong-edge 픽셀과 연결된 weak-edge 픽셀을 찾고, 이를 strong-edge로 바꾸어야 합니다.

우선 이미지로부터 strong, weak edge에 대한 그래프를 얻어내야 합니다. 이를 위해 우선 임계값 80 이상인 픽셀들의 위치를 알아냈습니다. 그 다음 각각의 픽셀에 대해, 이 픽셀 배열 내에 이웃한 픽셀이 존재하는지, 즉 그래프에서의 이웃 노드가 존재하는지를 Boolean array 형태로 계산한 후 실제 이웃 픽셀의 위치를 알아냈습니다. 이것을 그래프 딕셔너리에 추가하는 과정을 반복하여 전체 이미지를 그래프로 나타내었습니다. 이후 이 그래프를 DFS 방식으로 순회하되, 최종 이미지와 같은 크기의 배열을 선언한 후 방문한 픽셀의 위치와 같은 곳에는 1, 아닌 곳에는 0을 할당하였습니다. 이 때 최종 배열에서 1의 값을 가지는 곳은 strong-edge라고 할 수 있으므로, 이 배열에 255를 곱하는 방식으로 최종 edge 이미지를 얻어낼 수 있습니다. 해당 코드와 결과 이미지는 아래와 같습니다.

```
                                                                Line 203~269
def getImageGraph(img):
    graph = {}
    edgePixels = np.argwhere(img >= 80)

    for i, pivot in enumerate(edgePixels):
```

```
        rows = edgePixels[:,:1].flatten()
        rowMask = (pivot[0]-1<=rows) & (rows<=(pivot[0]+1))

        cols = edgePixels[:,1:].flatten()
        colMask = (pivot[1]-1<=cols) & (cols<=(pivot[1]+1))

        neighborMask = rowMask & colMask
        neighborMask[i] = False

        neighbors = edgePixels[neighborMask]
        graph[tuple(pivot)] = list(map(tuple, neighbors))
    return graph

def hysteresis(img):

    graph = getImageGraph(img)
    visited = np.zeros(np.shape(img))
    willVisit = []

    for node in graph.keys():
        if(img[node] == 255):
            willVisit.append(node)
            while willVisit:
                dest = willVisit.pop()
                if visited[dest] != 1:
                    visited[dest] = 1
                    willVisit.extend(graph[dest])
    res = visited * 255.0
    return res

iguanaHyst = hysteresis(iguanaDT)
iguanaHystImg = Image.fromarray(iguanaHyst)
iguanaHystImg.show()

saveImageFromArray(iguanaHyst, 'iguana_edge.bmp', 'PNG')
```

>>

그림 6 – iguana_edge.bmp