컴퓨터그래픽스 5주차 과제 보고서 2
201300995 이상건 물리학과

구현 내용
가우시안 필터, DoG, sobel 이용해서 엣지 검출하기다.

이유(구현 방법)
행렬로 계산하면 이미지인줄 알았는데 아니여서 다시 작성한다.
일단 전체 이미지에 IxIx, IxIy, IyIy에 가우시안 곱한 걸 생성해놓고 좌표값만 가져다 쓸 수 있게만 만들면 된다고 했었다. 그래서 ppt를 보고 아래 식을 참고했다.

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

w(x,y)가 필터부분인데 여기선 가우시안 필터를 사용한다. [Ix^2, … , Iy^2] 부분이 cv2.sobel과 my_DoG로 구하는 부분이다.
코드를 보면서 설명하는게 낫겠다.

```
def my_HCD(src, method, blockSize, ksize, sigma1, sigma2, k):
    '''
    :param src: 원본 이미지
    :param method : "HARRIS" : harris 방법 사용, "K&T" : Kanade & Tomasi 방법 사용
    :param blockSize: Corner를 검출할 때 고려할 주변 픽셀영역(Window 크기)
    :param ksize: DoG kernel size
    :param sigma1 : DoG에서 사용할 Sigma
    :param sigma2 : Covariance matrix에 Gaussian을 적용할 때 사용할 Sigma
    :param k: 경험적 상수 0.004~0.006
    :return: Corner response
    '''
    y, x = len(src), len(src[0])

    R = np.zeros(src.shape)  # Corner response를 받을 matrix 미리 생성

    #DoG, 배포해 드린 파일의 함수를 사용하세요.
    # gradX = my_DoG(src, ksize, sigma1, gx=1, boundary = 2)
    # gradY = my_DoG(src, ksize, sigma1, gx=0, boundary = 2)
    #Sobel, cv2.Sobel 함수 이용하시면 됩니다.
    gradX = cv2.Sobel(src, cv2.CV_32F, dx = 1, dy = 0, ksize = ksize)
    gradY = cv2.Sobel(src, cv2.CV_32F, dx = 0, dy = 1, ksize = ksize)

    #Covariance matrix 계산
    IxIx = np.multiply(gradX,gradX)
    IxIy = np.multiply(gradX,gradY)
    IyIy = np.multiply(gradY,gradY)
    IxIxGaussian = cv2.GaussianBlur(IxIx,(blockSize,blockSize),sigma2)
    IxIyGaussian = cv2.GaussianBlur(IxIy, (blockSize, blockSize), sigma2)
    IyIyGaussian = cv2.GaussianBlur(IyIy, (blockSize, blockSize), sigma2)
```

이미지단위로 일단 계산부터 하라고 해서 계산부터 했다. #Covariance matrix 밑에가 그 부분인데 sobel필터로 한 것에 가우시안 까지 곱했다.

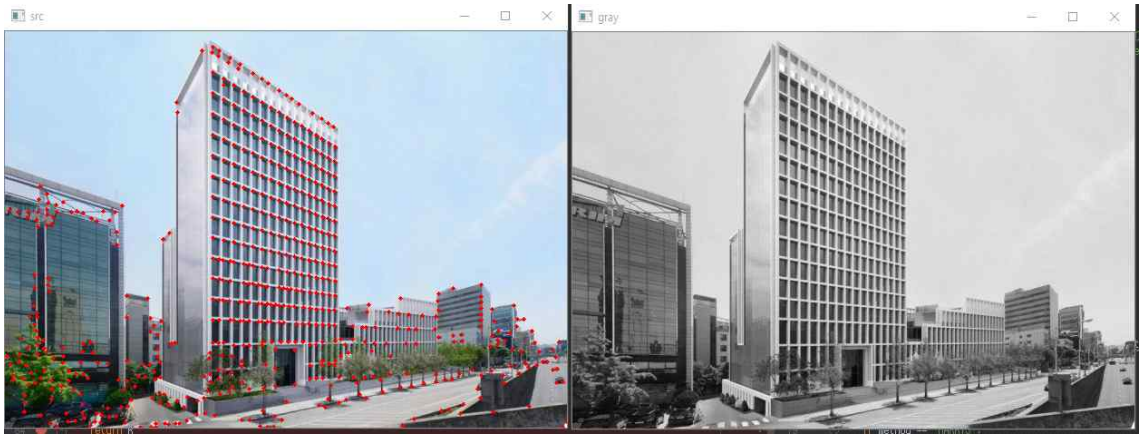
```
    M = np.zeros((2,2))
    # harris 방법
    if method == "HARRIS":
        for i in range(y):
            for j in range(x):
                #Harris 방법으로 R을 계산하세요.
                M[0,0] = np.sum(IxIxGaussian[i,j])
                M[0,1] = M[1,0] = np.sum(IxIyGaussian[i,j])
                M[1,1] = np.sum(IyIyGaussian[i,j])
                lam = np.linalg.eigvals(M)
                det = lam[0] * lam[1]
                tr = lam[0] + lam[1]
                R[i,j] = det - k * (tr ** 2)


    # Kanade & Tomasi 방법
    elif method == "K&T":
        for i in range(y):
            for j in range(x):
                #Kanade & Tomasi 방법으로 R을 계산하세요.
                M[0,0] = np.sum(IxIxGaussian[i,j])
                M[0,1] = M[1,0] = np.sum(IxIyGaussian[i,j])
                M[1,1] = np.sum(IyIyGaussian[i,j])
                lam = np.linalg.eigvals(M)
                R[i,j] = np.min(lam)



    return R
```

본격적으로 계산하는 부분이다. 여기서 M을 정의하고 각각의 좌표 I,j를 구할 때 x,y 좌표값을 넣도록 했다. 왜냐하면 (2,2)의 크기를 가지는 M은 좌표값에 따라 다르기 때문이다. 그래서 해당 값을 빼서 쓰라는건 이런 의미인 것 같다.
그래서 R행렬의 값을 구하는 방법은 ppt와 my_HCD_pixel.py를 참조해서 작성했다.



결과도 잘 나온다.

느낀 점
내가 공부를 안하는 것 같다.

과제 난이도
어렵다.