

Assignment 4 Report

고려대학교 컴퓨터학과
정경륜

1. harris_corner.m 코드 설명

1) Compute R

$$\mathbf{H} = \sum_{x,y} w(x,y) \begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$$

Harris corner detection을 구현하기 위해선 R value를 구해야 한다. R value는 왼쪽의 수식을 통해 구할 수 있다. 먼저 이미지의 x축과 y축에 대해 미분값을 구한 후 왼쪽 수식의 행렬에 들어갈 값을 계산한다. 이후 gaussian filter를 적용하여 H 값을 구현한다. 실제 구현 코드는 아래와 같다.

$$R = \det(\mathbf{H}) - k * \text{trace}^2(\mathbf{H})$$

```

18 % 
19 % ToDo: Compute R
20 %
21
22 sobelX = [-1,0,1; -2,0,2; -1,0,1];
23 sobelY = [1,2,1; 0,0,0; -1,-2,-1];
24
25 Ix = conv2(Img, sobelX, 'same');
26 Iy = conv2(Img, sobelY, 'same');
27
28 h = fspecial('gaussian', 7, 2.0);
29
30 Ix2 = conv2(Ix.^2, h, 'same');
31 Iy2 = conv2(Iy.^2, h, 'same');
32 Ixy = conv2(Ix.*Iy, h, 'same');
33
34 det = Ix2.*Iy2 - Ixy.^2;
35 trace = Ix2 + Iy2;
36
37 %%%%%%
38 % k는 sensitivity factor로 이미지에 따라 적절한 값으로 설정
39 k = 0.04;
40 %%%%%%
41
42 R = det - k*(trace.^2);      % R value 구함

```

먼저 sobel filter를 정의해준뒤 주어진 이미지와 convolution 연산을 통해 미분값을 구해준다. Convolution 연산은 conv2 함수를 사용하였다. 이후 $I_x I_x, I_y I_y, I_x I_y$ 를 앞에서 구한 I_x, I_y 를 통해 구한 후 gaussian filter와 convolution 연산을 진행해 준다. 이후 아래의 수식을 통해 det과 trace를 구한 후 R 값을 구해주게 된다. 여기서 민감도인 k는 0.04로 설정해주었다.

$$\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc \quad \text{trace} \begin{pmatrix} a & b \\ c & d \end{pmatrix} = a + d$$

```

44 %%%%%%
45 % 이미지 전체로 봤을 때 맨 끝의 4개 corners는
46 % 이미지 내의 feature가 아니기 때문에 R value를 0으로 설정
47 [dimX, dimY] = size(R);
48
49 R(1:4, 1:4) = 0;
50 R(1:4, dimY-3:dimY) = 0;
51 R(dimX-3:dimX, 1:4) = 0;
52 R(dimX-3:dimX, dimY-3:dimY) = 0;
%%%%%

```

Image 전체로 봤을 때 양 끝의 네 모서리는 이미지 내의 feature는 아니지만 전체 이미지에서의 corner라서 R 값이 높게 잡힌다. 그래서 이 부분은 0으로 처리하여 이미지 내의 feature의 corner만 detect 할 수 있도록 하였다. 이미지에 따라 주석 처리 할 수도 있다.

2) Visualize R values using jet colormap

```

66 %
67 % ToDo: Visualize R values using jet colormap
68 %
69
70 % R value를 visualize 하고싶다면 true
71 % Corner points를 visualize 하고싶다면 false
72 visualization_R = false;
73
74 if visualization_R
75     imshow(R);
76     colormap("jet");
77     clim("auto");
78     colorbar;
79

```

R value를 jet colormap을 통해 시각화하기 위한 코드이다. 다만 이후 코드에서 코너를 시각화하게 되면 R value 시각화 이미지가 사라진다. 따라서 visualization_R을 toggle 값으로 사용하여 이것이 true면 R value를 시각화 해서 보여주고 false이라면 local maximum points를 수집하여 corner points를 표시하여 시각화 해준다.

3) Threshold R & Collect Local Maximum Points

```
80 %
81 % ToDo: Threshold R & Collect Local Maximum Points
82 %
83 else
84     [dimX, dimY] = size(R);
85     location=[];    % Corner points를 넣을 배열
86     window = 10;    % Local patch의 window size
87     sz = [window window]; % Local patch의 2D size
88
89 %%%%%%%%%%%%%%
90 threshold=0.5e+9; % 이미지 별로 적절한 threshold 값 사용
91 %%%%%%%%%%%%%%
92
93 x_cnt = window;    % Local patch를 구하기 위한 변수
94 y_cnt = window;    % Local patch를 구하기 위한 변수
95
96 for x=1:dimX
97     % Window size에 따른 local patch를 구하는 코드
98     if(x_cnt < window)
99         x_cnt = x_cnt+1;
100        continue
101    end
102    x_cnt=1;
103
104 for y=1:dimY
105     % Window size에 따른 local patch를 구하는 코드
106     if(y_cnt < window)
107         y_cnt = y_cnt+1;
108         continue
109     end
110     y_cnt=1;
```

미리 설정한 threshold 값을 이용해 local maximum points를 찾기 위한 코드이다. 먼저 local maximum을 구할 local patch의 사이즈를 window 변수를 통해 정한다. 그 후 위에서 jet colormap으로 시각화한 R 값을 보고 적절한 threshold를 정한다. 93번 줄부터 110번 줄까지는 for 문을 통해 이미지 전체 pixel을 훑고 가되 local patch들끼리 겹치지 않도록 local patch의 시작점을 체크해주는 코드이다.

```

112 % Boundary index error 처리
113 % Local patch를 구하는 코드
114 if x+(window-1) > dimX
115     local=R(x:dimX, y:y+(window-1));
116 elseif y+(window-1) > dimY
117     local=R(x:x+(window-1), y:dimY);
118 else
119     local=R(x:x+(window-1), y:y+(window-1));
120 end
121
122 % Local maxima
123 [local_maxima, max_index]=max(local(:));
124
125 % 앞서 설정한 threshold 보다 local maxima가 더 높을 때 corner point로 넣음
126 if local_maxima >= threshold
127     [r, c] = ind2sub(sz,max_index);      % 1D index -> 2D index
128     location=[location; y+c-1, x+r-1]; % Push corner point
129 end
130 end
131
132 points= cornerPoints(location);
133
134

```

114번 줄부터 120번 줄까지는 local patch를 구하는 코드이다. 다만 x축과 y축의 끝부분에선 window보다 작은 범위만 남았을 수도 있기 때문에 boundary index error를 처리하는 코드를 넣어주었다. 이후 123번 줄에서 local maximum을 max 함수를 이용해서 구하였다. 이렇게 구한 local maximum이 앞서 정한 threshold보다 크다면 corner points라고 생각하여 location 배열에 넣는다. 다만 넣을 때 max 함수로 구한 max value의 index는 1D로 나오기 때문에 2차원에서의 좌표값을 구하기 위해서 ind2sub 함수를 이용해 local patch 안에서 local maximum의 좌표값을 구한다. 이때 ind2sub 함수에 넣는 값은 local patch의 크기와 index이다. 이후 133번 줄에서 cornerPoints 함수를 이용하여 앞서 구한 corner의 좌표값이 담긴 location 배열을 points에 담게 된다.

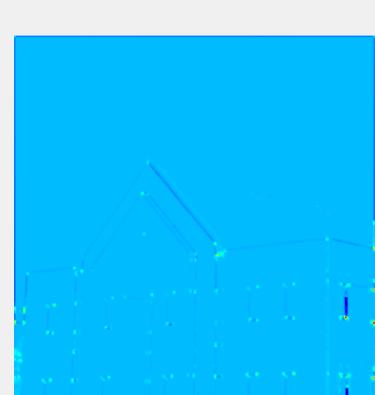
2. Visualize the results - building-600by600.tif

1) Visualizing R values

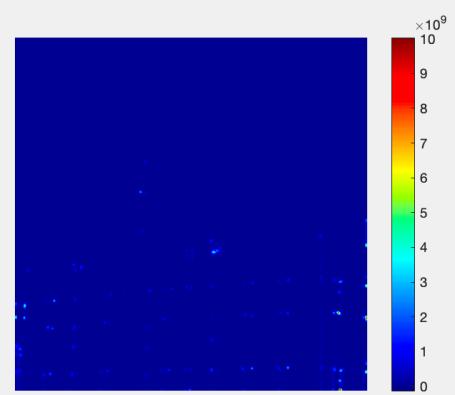
k=0.2



k = 0.04



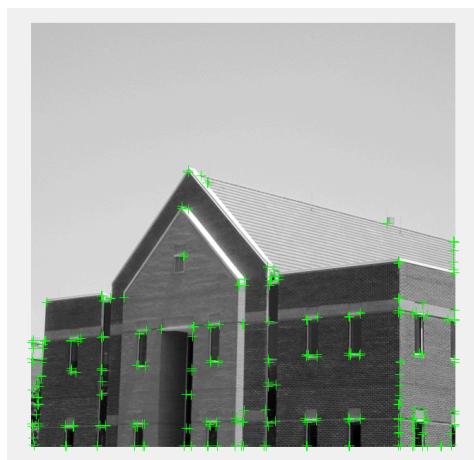
k=0.002



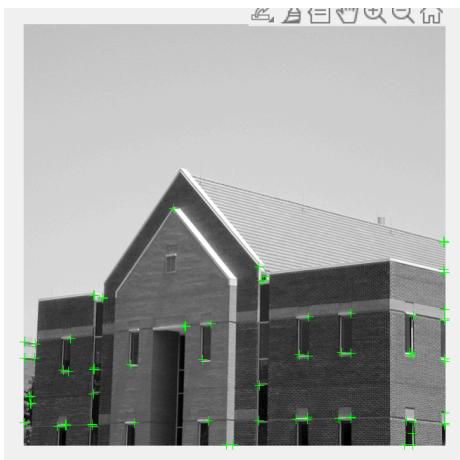
각각 k를 0.2, 0.04, 0.002로 높았을 때 R value를 시각화한 모습이다. K는 sensitivity factor로 높으면 높을수록 더 적은 corner를 탐지하고, 낮으면 낮을 수록 더 많은 corner를 탐지한다. k가 0.2일 때는 확실히 다른 두 이미지 보다 corner에 해당하는 R value가 다른 위치의 R value와 큰 차이를 보이지 않는다. k는 0.04부터 corner의 R value가 다른 위치의 값보다 확연히 더 큰 값을 가지고 있음을 볼 수 있다.

2) Visualizing corner points

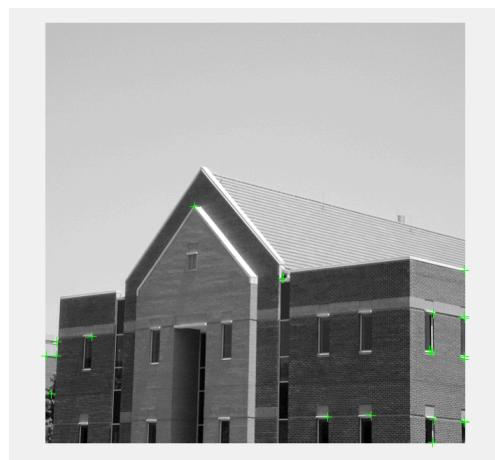
k=0.04
threshold=0.2e+9



k=0.04
threshold=0.7e+9

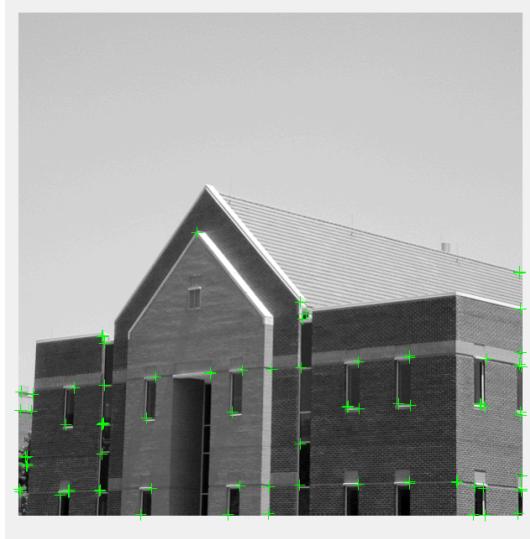


k=0.04
threshold=2e+9

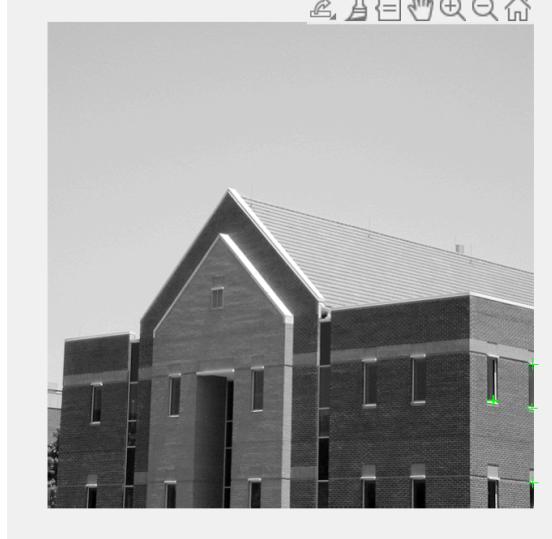


K가 0.04일 때 threshold 값만을 변화시켜 보았다. Local patch의 window 크기는 10으로 고정했다. Threshold 값이 높으면 높을수록 corner로 감지된 points가 줄어 들고 있음을 볼 수 있다. 만약 k가 0.2였을 땐 다음과 같은 결과가 나온다.

$k=0.2$
threshold=0.1e+9



$k=0.2$
threshold=0.7e+9

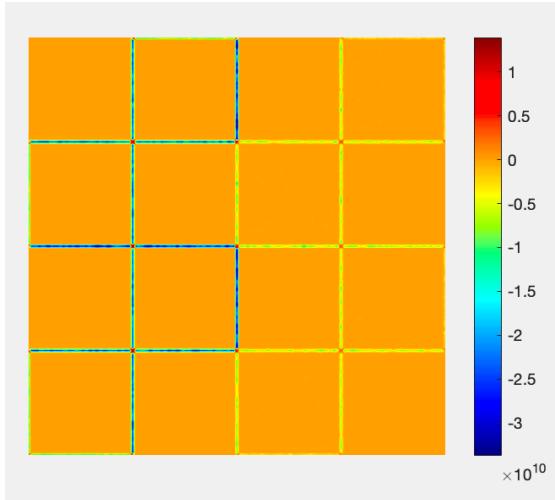


K 를 0.2로 높였기 때문에 이전의 0.04로 설정했을 때보다 탐지되는 corners의 수가 적을 것이다. 이것은 위의 결과를 통해 확인할 수 있다. 같은 위에서 설정한 threshold 값과 같은 값을 주었을 때 $k=0.2$ 일 때가 0.04일 때보다 훨씬 더 적은 corner를 탐지하는 것을 볼 수 있다. 특히 threshold가 0.7e+9인 경우 corner를 거의 탐지하지 못함을 알 수 있다. 따라서 이미지에 따라 적절한 k 값과 threshold를 설정해야 함을 알 수 있다.

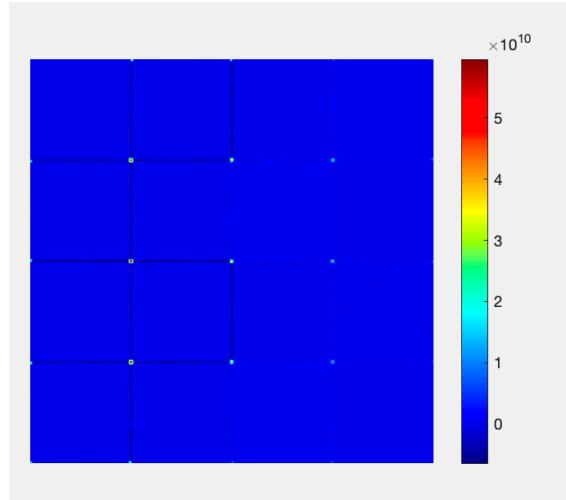
3. Visualize the results - checkerboard-noisy2.tif

1) Visualizing R values

$k=0.2$



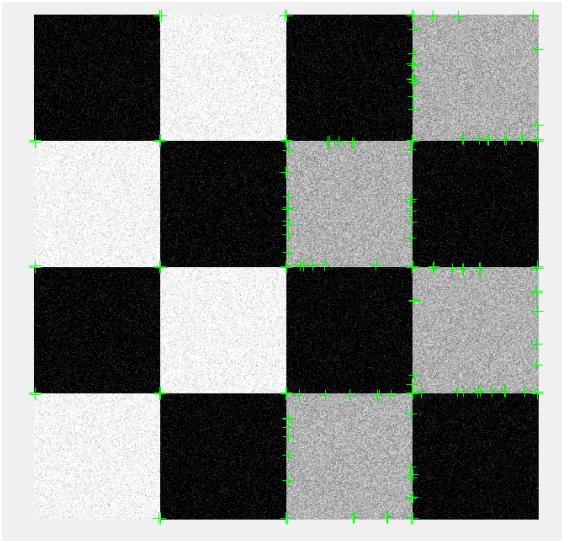
$k=0.04$



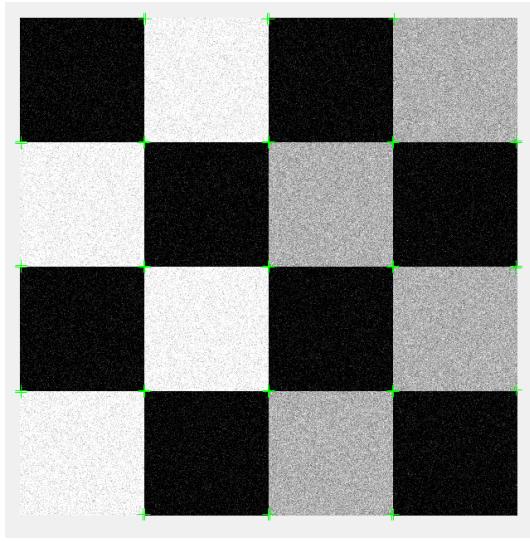
k 를 각각 0.2와 0.04로 놓았을 때 시각화한 R value이다.

2) Visualizing corner points

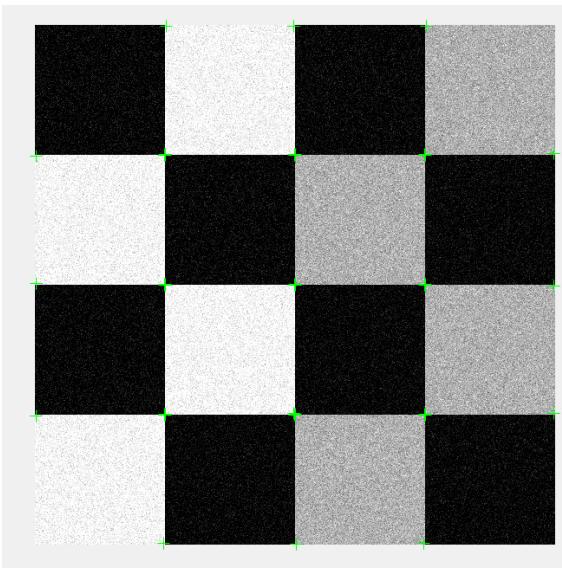
$k=0.04$
threshold= $0.5e+9$



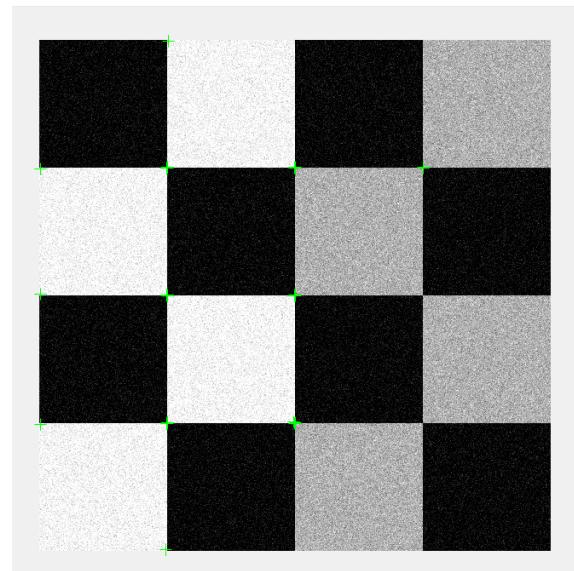
$k=0.04$
threshold= $4e+9$



$k=0.2$
threshold= $0.5e+9$



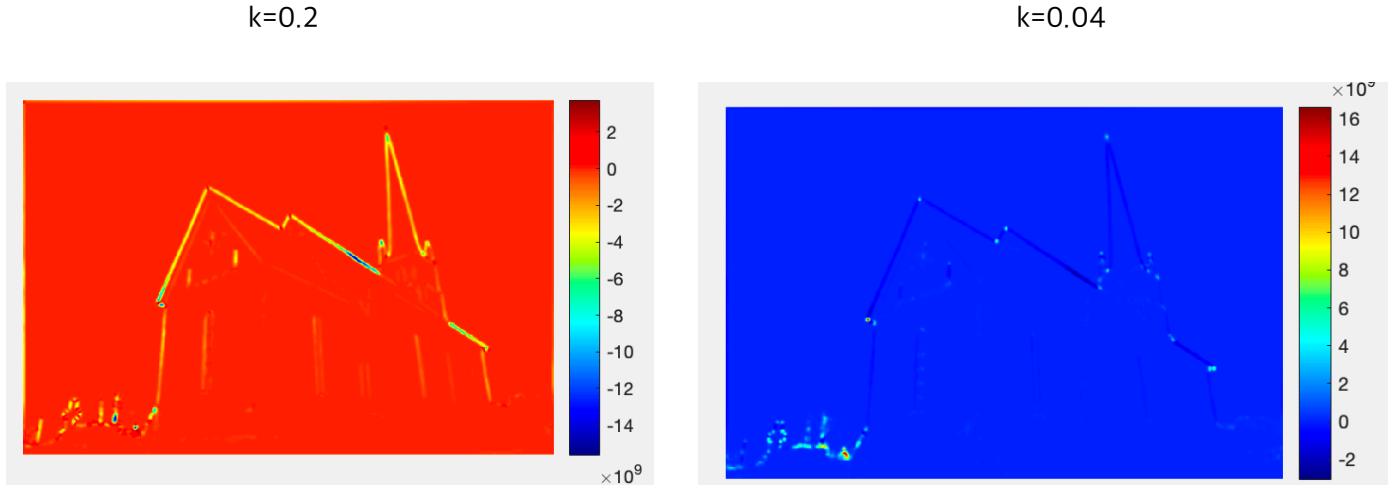
$k=0.2$
threshold= $4e+9$



위의 결과를 보면 k 가 높을수록 threshold가 높을수록 탐지되는 corner의 개수가 줄어듦을 알 수 있다. 그러나 $k=0.04$ 일 때 threshold= $0.5e+9$ 일 때를 보면 threshold가 너무 낮으면 edge까지 corner로 탐지하고 있는 모습을 볼 수 있다. 따라서 적절한 k 와 threshold 값을 설정하는 것이 중요하다. 해당 이미지에선 $k=0.04$, threshold= $4e+9$ 일 때와 $k=0.2$, threshold= $0.5e+9$ 일 때 가장 corner를 잘 탐지하고 있다.

4. Visualize the results - Custom Image

1) Visualizing R values



k=0.04일 때보다 k=0.2일 때 확실히 민감도가 떨어져서 corner의 R value가 edge의 R value와 큰 차이를 보이지 않는 것을 육안으로 확인할 수 있다. 반면 k=0.04일 때 corner 부분만 확연히 다른 R value를 가지고 있음을 알 수 있다.

2) Visualizing corner points



Custom image에서도 마찬가지로 k값이 작을수록, threshold 값이 낮을수록 더 많은 corner를 탐지한다. 또 threshold=3e+9일 때처럼 threshold가 너무 크면 corner를 거의 탐지하지 못하기 때문에 적절한 값을 설정하는 것이 중요하다. 해당 custom image에서는 k=0.04, threshold=0.7e+9일 때 가장 결과가 좋다.

앞선 3가지 이미지에서 테스트한 결과 적절한 threshold와 k값을 설정해야 corner를 잘 탐지할 수 있는 것을 알 수 있다. 이 값들은 R value를 jet colormap으로 시각화 했을 때 결과물을 보면서 어림짐작하여 적절한 값으로 잡아나갈 수 있다. 따라서 corner를 탐지하고 싶은 이미지 별로 적절한 값으로 바꿔나가며 corner detection을 최적화 해야한다. k와 threshold가 낮을수록 더 많은 points를 corner로 탐지한다. 이 점을 유의하면서 설정값을 바꿔나가면 된다.