컴퓨터 비전 1차 과제

2016-12500

박기범

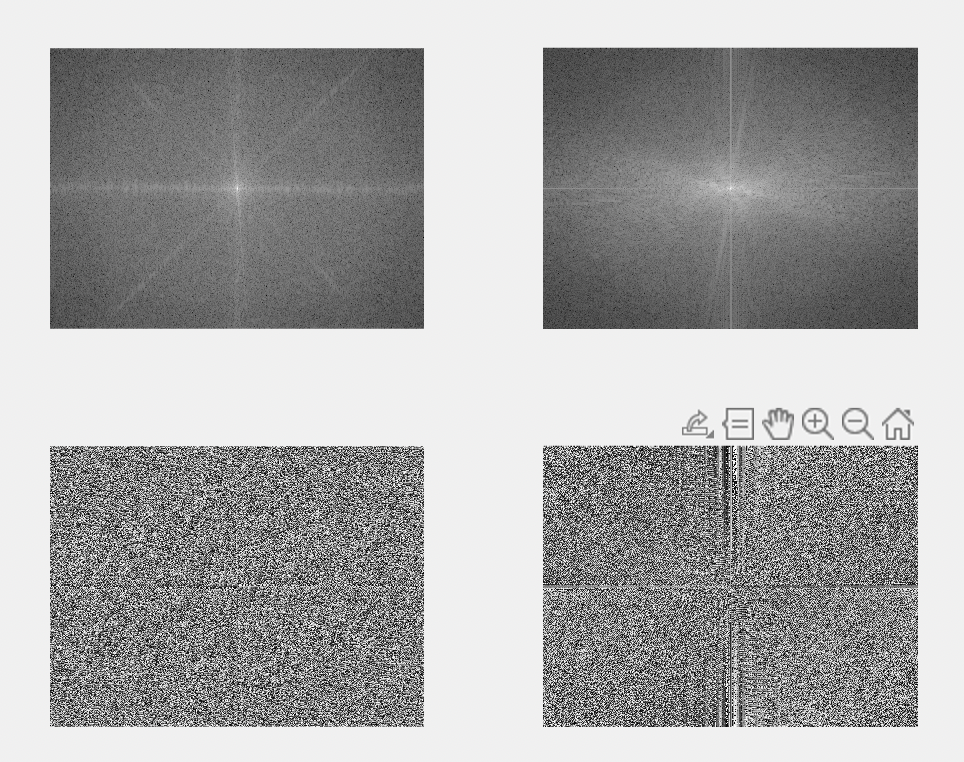
1. **2D Fourier Transform of Images**

대부분 문제에서 제시한대로 코드를 시행 하였다. Imread 로 이미지를 불러온 후에

텍스트, 실외, 포유류, 얼룩말이(가) 표시된 사진

자동 생성된 설명

imresize로 사이즈를 매치 시킨 후 fft를 시행 하는데 해석하기 쉽기 위해 fftshift를 통하여 0 center로 이동시켜서 비교를 진행한다. 또한 magnitude의 경우 log스케일로 바꾸어 더 비교를 쉽게 한다.



좌측 위부터 시계 방향으로 1의 mag 2의 mag 2의 phase 1의 phase

이제 phase와 magnitude를 섞은 이미지를 복구하기 위해

c2 = (abs(b1)) .\* (cos(angle(b2)) + 1i \* sin(angle(b2))); 후에 ifft 함수로 역 푸리에 변환을 진행해주었다.

텍스트, 실외, 잔디이(가) 표시된 사진

자동 생성된 설명

좌측이 1의 mag 2의 phase 우측이 1의 Phase 2의 mag

보이는 데로 실루엣의 경우는 phase를 따라가지만 그럼에도 이미지에 반복되는 노이즈가 생성 됩니다. 실루엣이 유지되는 이유는 테두리 근처에서 대부분의 신호들이 step function같은 함수와 비슷한 형태를 가지기에 같은 phase를 가지게 되기 때문입니다.(phase congrugency) 그러나 나머지 부분에선 서로의 신호 크기가 왜곡이 되어있으므로, 반복되는 노이즈가 많이 생성되게 됩니다.(원래의 크기 였다면 이 노이즈들은 필요 없는 부분에서 상쇄가 되는 방향등으로 보이지 않았을 것입니다.) 즉 엣지의 위치는 대체로 유지가 되지만 모든 신호의 크기가 바뀌므로 원하지 않은 주기적인 신호가 나타나게 됩니다.

2. **Perspective Image Transforms**

filter\_1 = [1.6322 , 0 ,0 ;0.2120, 1.6336, 0.0013 ;-101.9757, -0.6322 , 1];

와 같이 2차원 필터를 설정 해준 후에

Ssgcale이란 함수를 만들었습니다. 이 함수의 목적은 모서리 네점을 미리 변환해보고 그들의 좌표를 보고 변환후의 이미지의 크기 를 미리 가늠하고 음수 좌표를 잘 처리해주기 위해 mx my값들을 받습니다.

for i =1:375

for j = 1:500

a = transpose(filter\_1)\*transpose([j,i,1]);

u = round(a(1)/a(3));

v = round(a(2)/a(3));

fromscratch\_1( v + 1 - my1,u + 1 - mx1 ) = a1(i,j);

end

end

그리고 위와같은 코드로 변환 좌표를 한점씩 찍습니다.

그렇지만 이렇게만 할 시에는 좌표들이 늘어나는 부분에서 빈부분이 발생하므로 interpolation 이 필요합니다. 다행히 이 문제에서는 크게 많이 늘리지 않았으므로 주위 8점중 하나는 정보를 지니고 있습니다. 따라서 제가 구현한 simpleinterpolation 함수는

0인점을 찾아서 주위의 non-zero 값들의 평균을 그 픽셀의 값으로 합니다.

3. **Camera Calibration**

a)

텍스트이(가) 표시된 사진

자동 생성된 설명

b) 우선 A를 구하기 위해

for i = 1:20

u = A\_before(i,1);

v = A\_before(i,2);

x = A\_before(i,3);

y = A\_before(i,4);

z = A\_before(i,5);

tmp1 = [x,y,z,1,0,0,0,0,-u\*x,-u\*y,-u\*z,-u ];

tmp2 = [0,0,0,0,x,y,z,1,-v\*x,-v\*y,-v\*z,-v ];

A = [A;tmp1;tmp2];

end

와 같이 원래 A에 해당하는 행렬을 생성해줍니다.(A\_before은 단순히 좌표들을 행렬로 만든 것입니다.)

그 후에 svd함수로 V행렬을 구하면 문제에서 말한 대로 V의 마지막 column의 행렬을 뽑아내면

0.0031 0.0001 -0.0004 -0.9789

0.0003 0.0006 -0.0028 -0.2041

0.0000 0.0000 -0.0000 -0.0013

이 될 것입니다.

SVD 과정에서 위 크기의 벡터는 알아서 1로 조정이 됩니다.

c)

0.0031 0.0001 -0.0004 -0.9789

0.0003 0.0006 -0.0028 -0.2041

0.0000 0.0000 -0.0000 -0.0013

그 후에

for i = 1:20

u = A\_before(i,1);

v = A\_before(i,2);

x = A\_before(i,3);

y = A\_before(i,4);

z = A\_before(i,5);

tmp1 = [x,y,z,1,0,0,0,0,-u\*x,-u\*y,-u\*z ];

tmp2 = [0,0,0,0,x,y,z,1,-v\*x,-v\*y,-v\*z ];

A = [A;tmp1;tmp2];

b = [b;u;v];

end

와 같은 방법으로 m34가 1이라고 가정한 후에

Ap=b를 만족하는 행렬을 구하기 위해 \를 썼습니다. (문제의 설명과 다르게 시행한 이유는 매트랩 오류로 정확도 경고 인하여 대체 하였습니다.) 그러자 결과가

-2.3326 -0.1100 0.3374 736.6739

-0.2311 -0.4795 2.0872 153.6278

-0.0013 -0.0021 0.0005 1.0000

가 되고 이를 정규화 하면

-0.0031 -0.0001 0.0004 0.9789

-0.0003 -0.0006 0.0028 0.2041

-0.0000 -0.0000 0.0000 0.0013

와 같이 나왔는데 부호만 다름을 알 수 있다. 이는 단순히 이 방법의 경우에는 1로 고정을 통해 부호가 고정이 되므로 나머지는 그대로 실수 배가 되어 -1배가 된 것으로 보입니다.