전산학특강(영상처리특론)

Assignment 2 Report

고려대학교 컴퓨터학과
정경륜

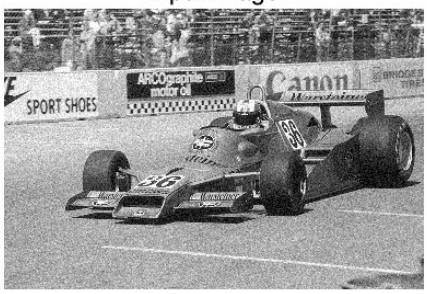
1. myLPF.m 코드 설명

```
26
27      %
28      % ToDo
29      %
30      G = F;
31      D = zeros(PQ(1), PQ(2));
32      H = zeros(PQ(1), PQ(2));
33      centerP = PQ(1)/2;
34      centerQ = PQ(2)/2;
35
36      D0=50;
37      n=2;
38
39      for x=1:PQ(1)
40          for y=1:PQ(2)
41              D(x,y)= sqrt((x-centerP)^2 + (y-centerQ)^2);
42              H(x,y)=1/(1+(D(x,y)/D0)^(2*n));
43          end
44      end
45      G= G.*H;
46
47      figure,imshow(log(1+abs((G))), []);
48
```

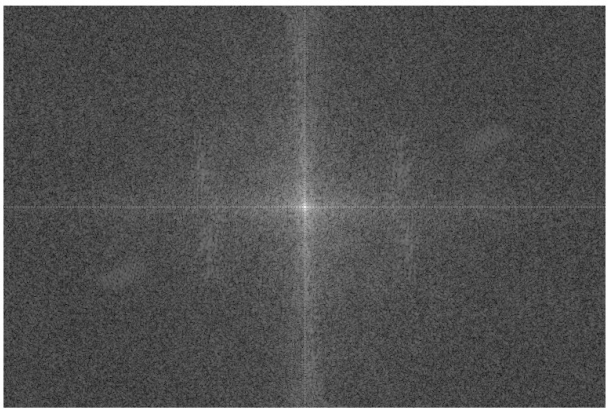
myLPF의 핵심 코드는 위와 같다. 우선 중앙 픽셀과의 거리를 구하기 위해 중앙값의 좌표를 33, 34번 라인을 통해 구해준다. 또한 cut-off frequency인 D0와 차수인 degrees를 이미지에 따라 적절하게 조절한다. 주어진 이미지인 “racing-noisy.png” 에선 cut-off frequency인 D0는 50, degrees인 n은 2를 사용하였다. 그 후 이제 filter의 값을 계산하게 되는데 41번 라인에서 픽셀과 중심 픽셀 간 거리를 구하고 42번 라인에서 교과서에서 제시한 butterworth lowpass filter의 수식과 동일하게 코드를 작성해주었다. 그 후 45번 라인에서 fourier 변환을 거친 값과 앞서 구한 필터를 곱하여 masking 된 fourier 값을 구하고 이후 코드에서 fourier 역변환을 통해 결과 이미지를 구하게 된다.

2. LPF 결과

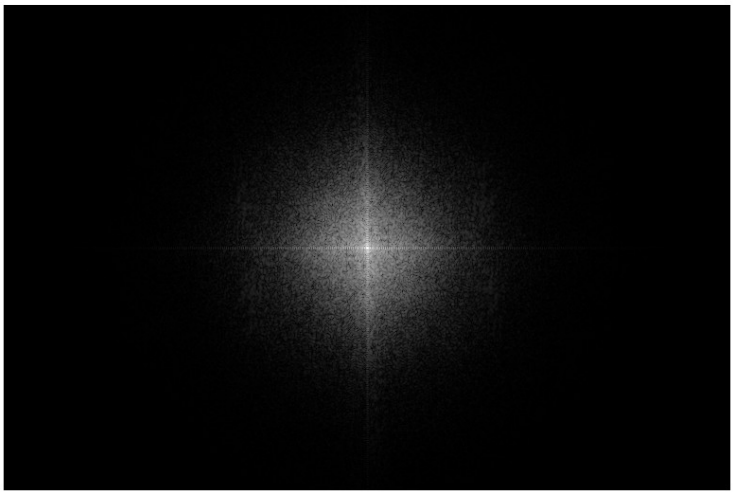
Input Image



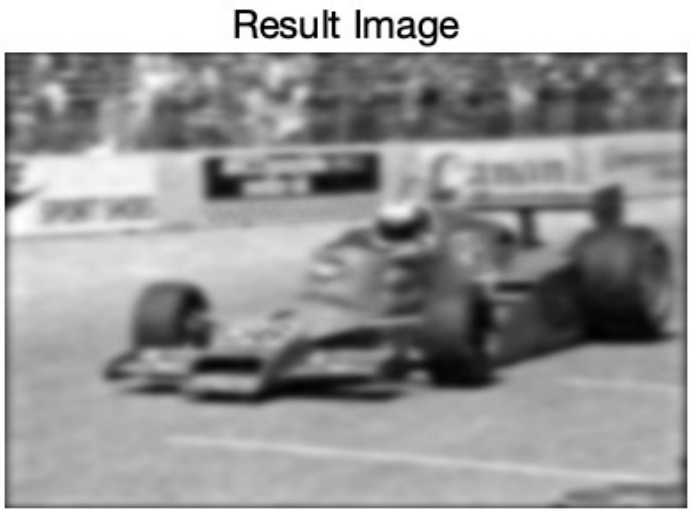
처음 원본 이미지는 위와 같다.



원본 이미지에 대한 fourier 값은 다음과 같다.

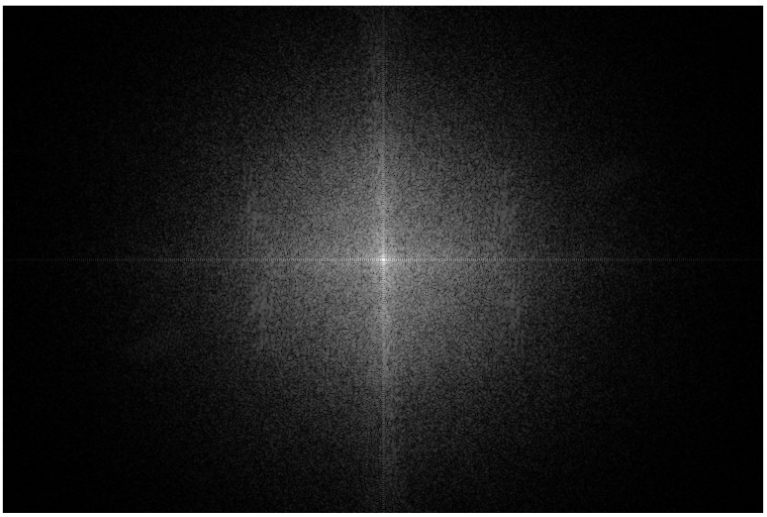


이에 butterworth low pass filter 거치고 난 뒤는 다음과 같다. Cut-off frequency는 50, degrees는 2를 사용하였다.

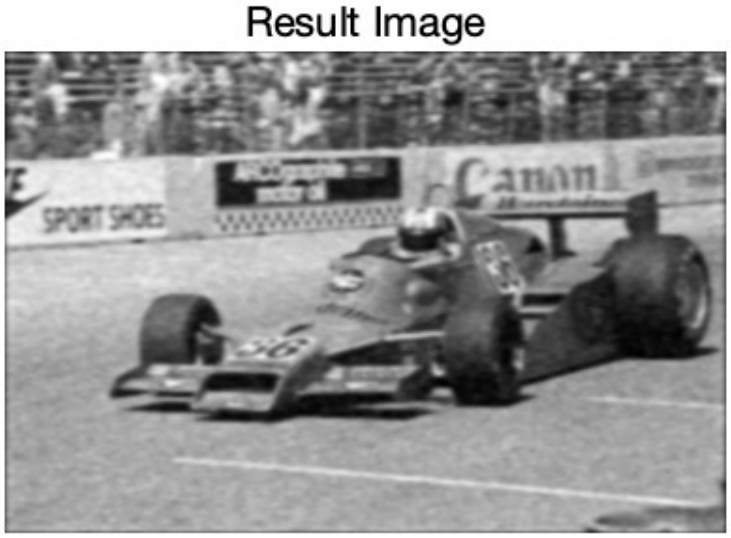


이후 다시 fourier 역변환을 하고 나면 다음과 같다. 이미지의 세부적인 디테일은 날아가고 상대적으로 저주파 영역의 모습만 남은 것을 볼 수 있다.

1) 만일 cut-off frequency를 높이면…

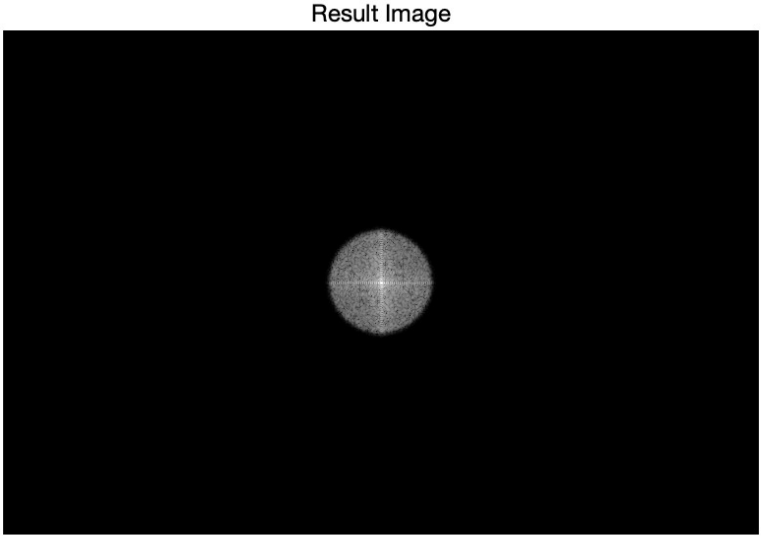


Cut-off frequency는 100, degrees는 2로 설정하였다. Cut-off frequency가 커짐에 따라 pass 되는 영역이 더 넓어진 것을 확인할 수 있다.



Cut-off frequency가 커짐에 따라 cut-off frequency가 작았을 때보다 blurring 현상이 많이 줄어든 것을 확인할 수 있다. 그 이유는 cut-off frequency가 blurring의 정도를 결정하기 때문이다.

2) 만일 order를 높이게 되면…



Cut-off frequency는 50, degrees를 30으로 높였다.

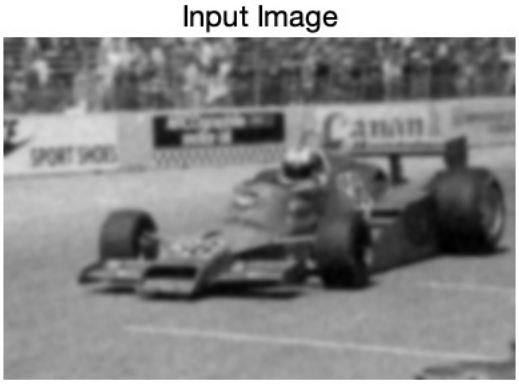
Degrees를 높이게 되면 결과에서 ringing 현상이 심해지는 것을 볼 수 있다. 그 이유는 degrees term이 ringing을 조절하게 되는데 degrees가 낮으면 gaussian에 가까워 지고, 높으면 sinc 함수에 가까워지기 때문이다.

3. myHBF.m 코드 설명

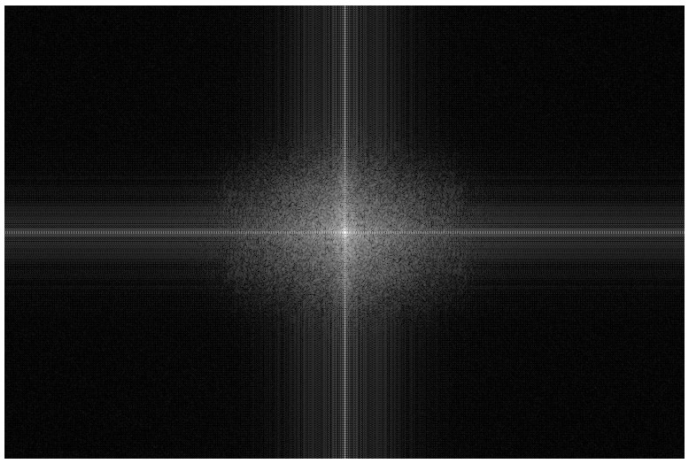
```
28      %
29      % ToDo
30      %
31      G = F;
32
33      D = zeros(PQ(1), PQ(2));
34      H = zeros(PQ(1), PQ(2));
35      centerP = PQ(1)/2;
36      centerQ = PQ(2)/2;
37
38      D0=50;
39      n=2;
40      k=7;
41
42      for x=1:PQ(1)
43          for y=1:PQ(2)
44              D(x,y)= sqrt((x-centerP)^2 + (y-centerQ)^2);
45              H(x,y)=1-(1/(1+(D(x,y)/D0)^(2*n)));
46          end
47      end
48      G= (1 + k*H).*G;
49
50      figure,imshow(log(1+abs((G))), []);
51
```

myHBF의 핵심 코드는 위와 같다. LPF와 거의 유사한데 우선 중앙 픽셀과의 거리를 구하기 위해 중앙값의 좌표를 35, 36번 라인을 통해 구해준다. 또한 cut-off frequency인 D0와 degrees인 n을 이미지에 따라 적절하게 조절한다. 해당 이미지인 “racing-blur.png” 에선 cut-off frequency인 D0는 50, degrees인 n은 2를 사용하였다. 그 후 이제 filter의 값을 계산하게 되는데 44라인에서 픽셀과 중심 픽셀 간 거리를 구하고 45번 라인에서 교과서에서 제시한 High Boosting Filter의 수식과 동일하게 코드를 작성해주었다. HBF = 1 - LPF와 같다. 그 후 48번 라인에서 fourier 변환을 거친 값과 앞서 구한 필터를 곱하여 masking 된 fourier 값을 구하고 이후 fourier 역변환을 통해 결과 이미지를 구하게 된다. 이때 k값을 통해서 HBF의 강도를 조절할 수 있는데 해당 이미지에선 k=7을 사용하였다.

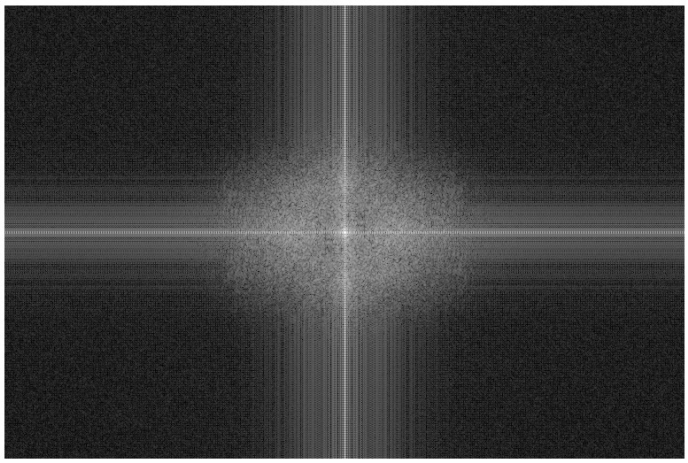
4. HBF 결과



Input image는 blurring이 많이 된 모습을 볼 수 있다.



원본 이미지에 대한 fourier 값은 다음과 같다.



이에 high boosting filter 거치고 난 뒤는 다음과 같다. 이때 cut-off frequency는 50, degrees는 2를 사용하였다.

이후 다시 fourier 역변환을 하고 나면 다음과 같다. 처음에 blurring 처리 되어 흐리게 보였던 글씨의 디테일이 살아난 모습을 볼 수 있다.

1) 만약 k 값을 낮추게 되면…

Boosting weight인 k값을 낮추게 되면 그만큼 unsharp masking의 정도가 낮아져서 왼쪽과 같이 위의 결과보다 unsharp 한 모습을 살펴볼 수 있다.

2) 만약 k 값을 높이게 되면…

반대로 k값을 높이게 되면 그만큼 unsharp masking 정도가 높아져 맨 처음 결과보다 강하게 sharp 처리된 모습을 살펴볼 수 있다.

5. myNotch.m 코드 설명

```
1   input_file_names = {'cat-halftone.png', 'text-sineshade.tif', 'saturn-rings-sinusoidal-interf.tif'};
2
3   for file_idx=1:3
4       input=imread(string(input_file_names(file_idx)));
```

주어진 이미지 3개가 모두 주어진 상황에서도 돌아갈 수 있도록 앞부분 코드를 약간 수정하였다.

```
39      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
40      % cat-halfone notches...
41      if file_idx==1
42          D0=30;
43          n=4;
44          notch_x=[212, -212, -259, 0, 266, -266, 44, -44, 211, 211];
45          notch_y=[319, 319, 0, 387, 387, 387, 320, 320, 63, -63];
46
47      % text-sineshade notches...
48      elseif file_idx==2
49          D0=5;
50          n=4;
51          notch_x=[0];
52          notch_y=[14.5];
53
54      % saturn-rings-sinusoidal-interf notches...
55      % 마지막 이미지는 상당한 시간이 소요됩니다.
56      elseif file_idx==3
57          D0=3;
58          n=2;
59          idx = 1;
60          for i=45:6776
61              notch_x(idx)=i;
62              notch_y(idx)=0;
63              idx = idx +1;
64          end
65      end
66      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

각 이미지 별로 cut-off frequency인 D0, degrees 를 나타내는 n, 그리고 notch 쌍들이 다르기 때문에 이미지 별로 알맞은 변수들을 할당해 주었다.

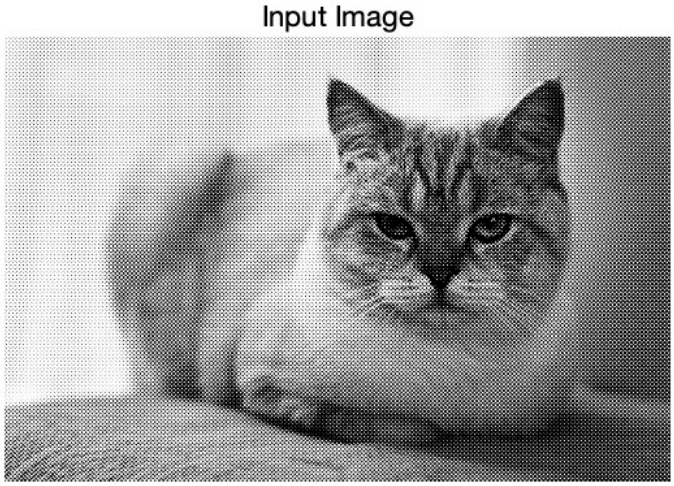
첫 번째, 두 번째 이미지는 빠르게 작동하나 **세 번째 이미지는 대칭 선을 그려야 하기 때문에 상당한 연산을 필요로 한다. 따라서 세 번째 이미지 연산을 할 때는 수 십분 소요된다.**

```
68      Q = length(notch_x);
69
70      for x=1:PQ(1)
71          for y=1:PQ(2)
72              for k=1:Q
73                  u=notch_x(k);
74                  v=notch_y(k);
75                  D_1= sqrt((x-centerP-u)^2 + (y-centerQ-v)^2);
76                  D_2= sqrt((x-centerP+u)^2 + (y-centerQ+v)^2);
77                  H1=1/(1+(D0/D_1)^(2*n));
78                  H2=1/(1+(D0/D_2)^(2*n));
79                  if k == 1
80                      H(x,y)=H1*H2;
81                  else
82                      H(x,y)=H(x,y)*H1*H2;
83                  end
84              end
85          end
86      end
87
88      G=G.*H;
89
90      figure,imshow(log(1+abs((G))), []);
91
```

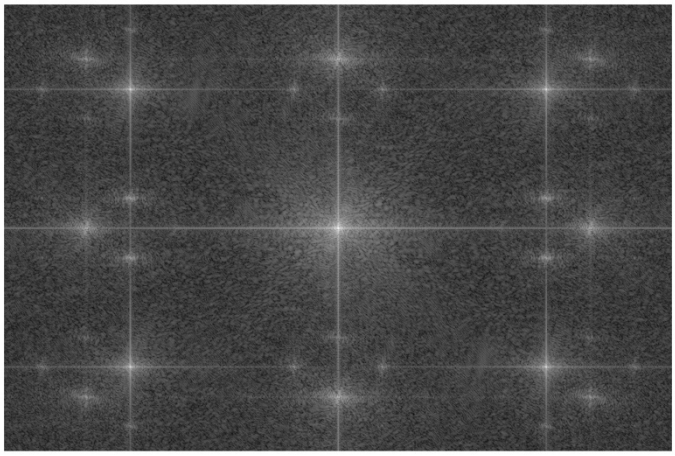
myNotch.m의 핵심 알고리즘은 다음과 같다. 75, 76번 라인에서 notch 쌍들을 중심으로 하는 거리를 구하고 이를 이용해 77번부터 82번 라인까지 교과서에 제시한 수식에 맞춰서 계산한다. 이후 88번 라인에서 fourier 변환을 거친 값과 앞서 구한 필터를 곱하여 masking 된 fourier 값을 구하고 이후 fourier 역변환을 통해 결과 이미지를 구하게 된다.

6. Notch Filter 결과

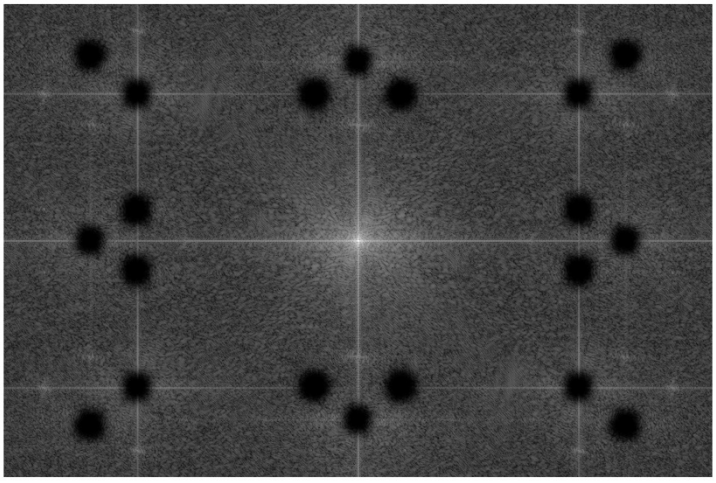
1) cat-halftone.png



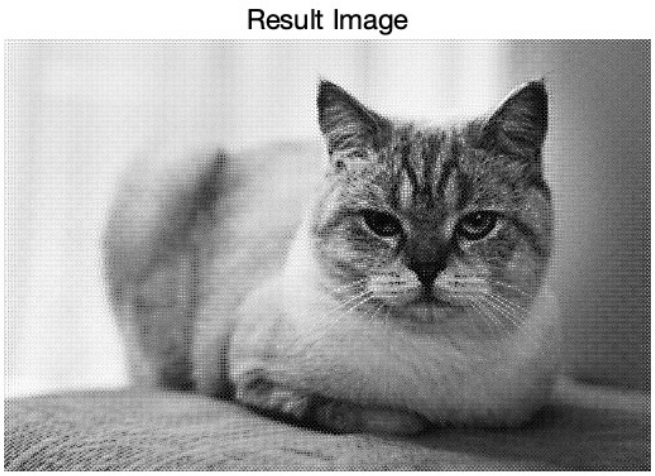
Foreground 로 고양이 이미지가 있지만 background noise 로 checkerboard 패턴이 나타난다.



원본 이미지에 대한 fourier 값은 다음과 같다. 여러 개의 notch 쌍들이 존재하는 것을 볼 수 있다.

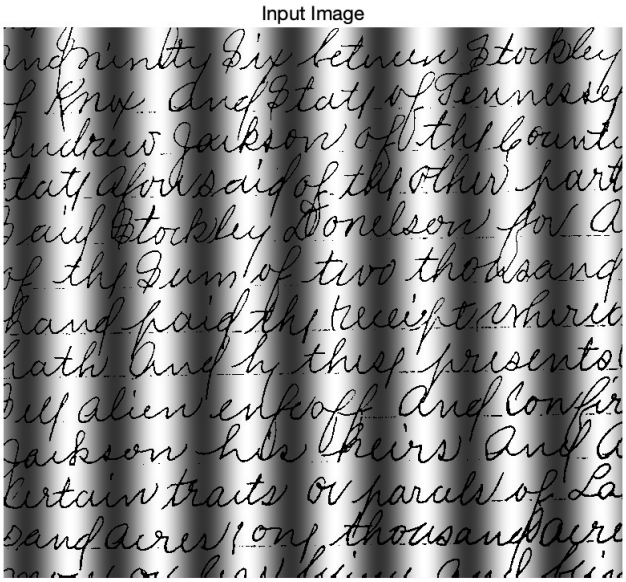


해당 notch 쌍들에 대해 masking 처리를 해주었다. 이때 사용한 cut-off frequency 는 모두 동일하게 30 을 적용하였고 degrees 는 4 를 사용하였다.

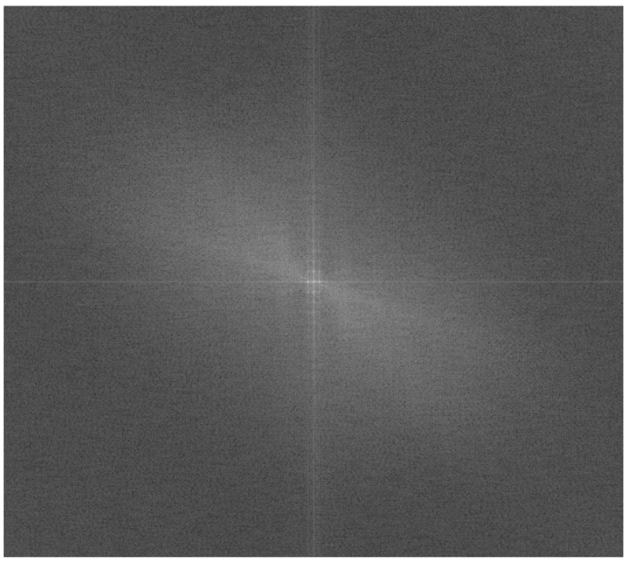


이전에 존재하였던 checkerboard 패턴이 많이 제거된 모습을 확인할 수 있다.

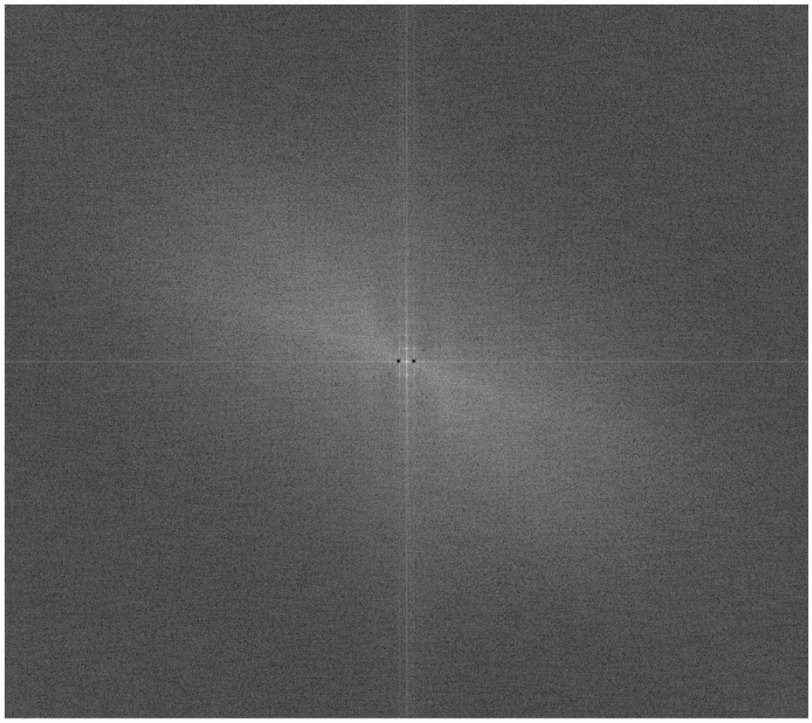
2) text-sineshade.tif



Input image 에 세로로 긴 검은 노이즈가 있어 글씨를 확인하는데 방해가 되는 것을 확인할 수 있다.

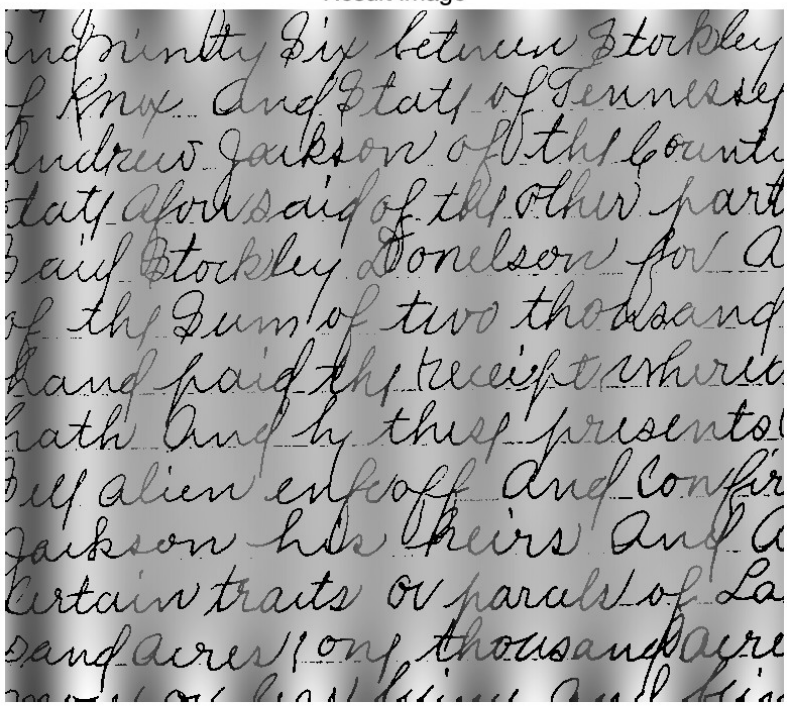


원본 이미지에 대한 fourier 값은 다음과 같다.



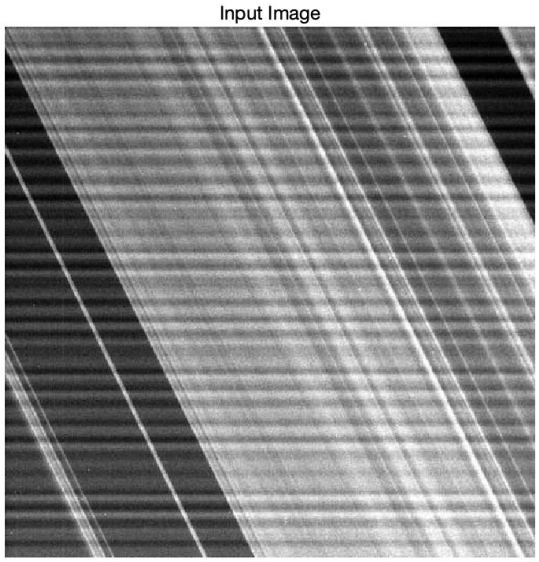
Input image 에서 세로로 나타나는 노이즈를 제거하기 위해 가운데 가로로 대칭되어 점으로 나타나 있는 notch 쌍을 제거해준다. 이때 사용한 cut-off frequency 는 5, degrees 는 4 를 사용하였다. (자세히 보면 중앙에 가로로 대칭된 마스킹이 보입니다.)

Result Image

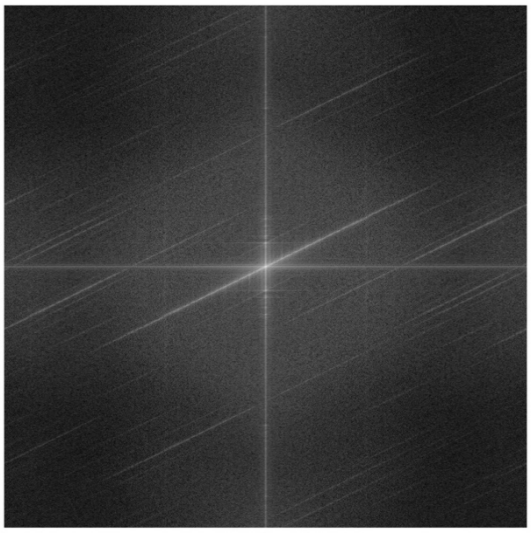


이전 input 보단 훨씬 깔끔해진 모습을 볼 수 있다. 물론 왼쪽 끝부분의 검은 세로줄을 완벽하게 제거하긴 어렵지만 나머지 부분에선 많이 제거되었고 글씨를 알아보는데 훨씬 용이해졌다.

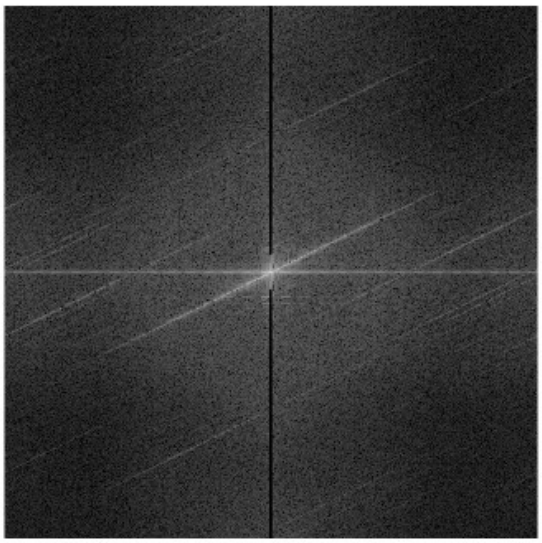
3) saturn-rings-sinusoidal-interf.tif



Input image를 보면 가로로 줄지어 나타나는 background noise가 존재하는 것을 볼 수 있다.

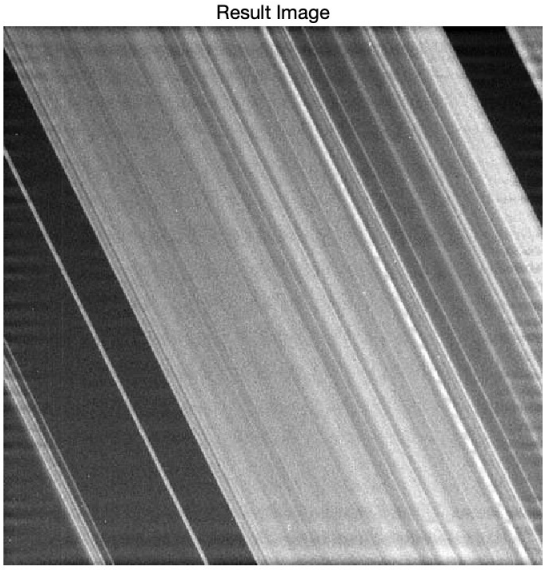


원본 이미지에 대한 fourier 값은 다음과 같다. 중앙을 기준으로 가로, 세로로 긴 흰 선이 존재하는 것을 알 수 있다.



Input image 에서 가로로 나타나는 노이즈를 제거하기 위해 fourier 에서 세로로 긴 대칭 선을 제거해준다. Cut-off frequency 는 3, degrees 는 2 를 적용하였다. 다만 앞서 언급했듯이 해당 결과를 도출하기 위해서 수 십분 가량의 상당한 연산 시간이 필요하다.

Result Image



Input image와 비교해 봤을 때 가로로 길게 존재하던 노이즈 선들은 많이 제거된 모습을 볼 수 있다.