

Special Lecture for Computer Science – COSE490

Digital Image Processing :: Frequency Domain Filters

Due date : 2022-10-16

고려대학교 컴퓨터학과 2017320108

고재영

개발 환경 : Matlab Desktop

과제 만기일 : 2022-10-16

제출 날짜 : 2022-10-15

최종 제출: 2022-10-15

재제출 사유: none

[0] – 과제 설명

- You need to write a short report explaining your implementation and the results. Submit the report (pdf) along with myLPF.m, myHBF.m, and myNotch.mfiles via blackboard.

- Frequency domain filtering에 관련하여 Low pass butterworth filter, High boost butterworth filter, 그리고 Notch filter를 matlab 내장함수를 사용하지 않고 수업 내용으로 배운 원리에 기반하여 직접 구현을 한다.

[1] – myLPF

첫 번째로, myLPF는 Low-pass Butterworth filter에 관한 부분이다. Frequency domain에서 frequency rectangle의 중앙으로부터 거리로 정의되는 $D(u, v)$ 와 함께, cutoff frequency의 ' D_0 '와 degree order의 ' n '으로 두 가지 parameter로 filter function $H(u, v)$ 를 결정한다.

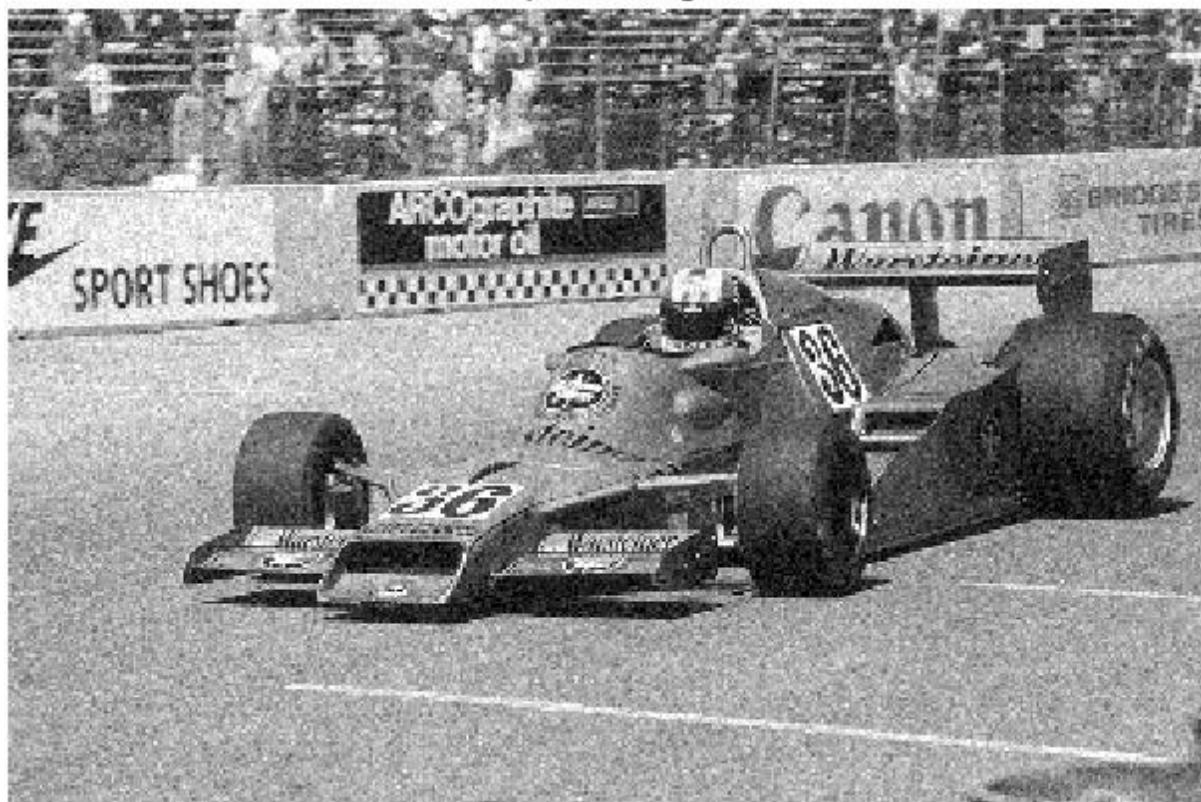
```
% -----  
%  
% Creating Frequency filter and apply - High pass filter  
%  
  
% initialize H(u, v)  
D = zeros(PQ(1), PQ(2));  
H = zeros(PQ(1), PQ(2));  
% cutoff frequency D0 and order n  
D0 = 128;  
n = 2;  
%  
for u = 1:PQ(1)  
    for v = 1:PQ(2)  
        D(u, v) = sqrt((u - PQ(1) / 2).^2 + (v - PQ(2) / 2).^2);  
        H(u, v) = 1 / (1 + (D(u, v) / D0).^(2 * n));  
    end  
end  
  
F = H .* F;  
  
%  
% ToDo  
%  
G = F;  
  
figure, imshow(log(1+abs((G))), []);  
% -----
```

정의 그대로 구현한 모습이다. 이 때, input image의 Discrete Fourier Transform인 F 에 대해, filter function H 를 piecewise multiplication 한다. 2-D convolution으로 인해 frequency domain에선 multiplication을 해야하기 때문이다.

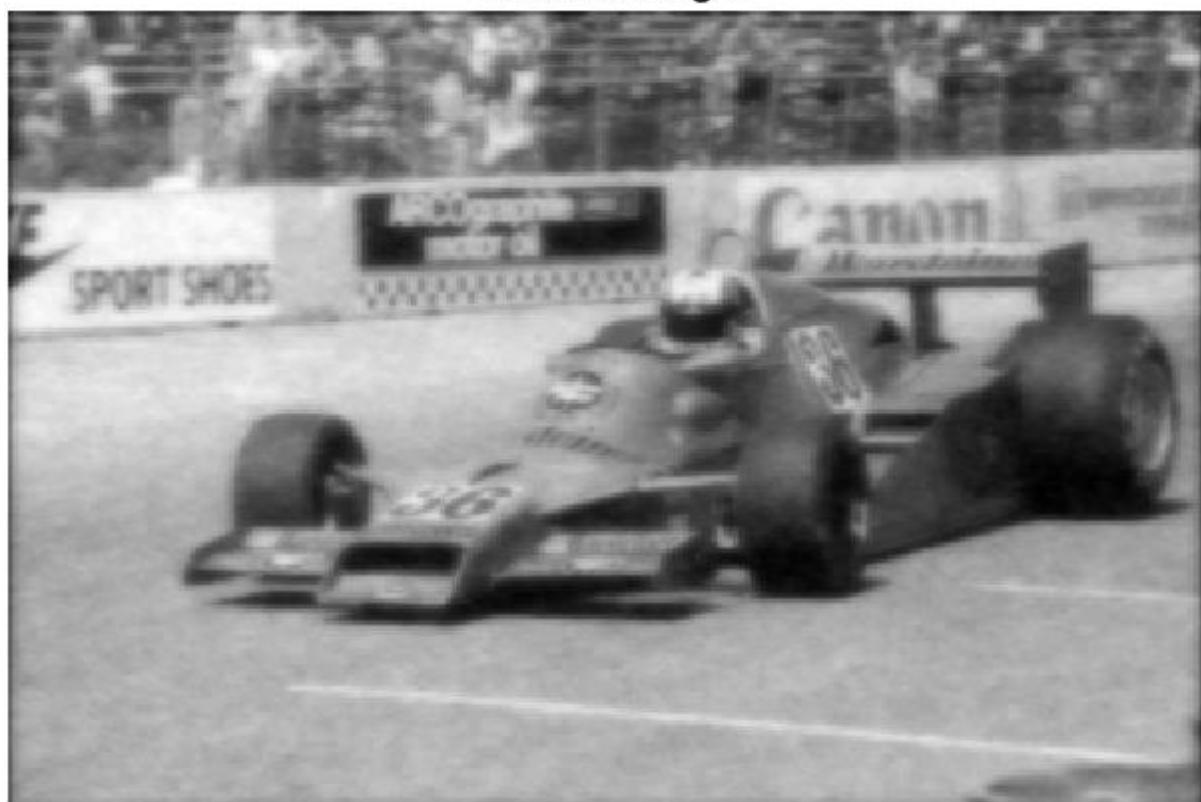
그리고 다음은 두 parameter D_0 와 n 의 값을 달리한 결과이다.

- (1) Degree order

Input Image



Result Image



Result Image



Result Image



Result Image



5개의 이미지는 공통적으로 $D_0 = 64$ 일 때, 순서대로 원본이미지, $n = 1, 2, 5, 20$ 일때의 결과이다.

원본과 비교하자면, Low pass filter를 거친 이후 blur한 이미지가 형성되면서 noise reduction의 효과를 볼 수 있음을 알 수 있다. 하지만, $n = 5, n = 20$ 인 이미지를 나머지 두 이미지와 비교해보면 알 수 있듯이, 차수 n 이 커짐에 따라 점점 sinc function처럼 가까워져 Ideal Low-pass filter가 겪는 ringing artifact가 심해진다. N 이 작을 땐 Gaussian에 가까워진다.

(2) Cutoff frequency

Result Image



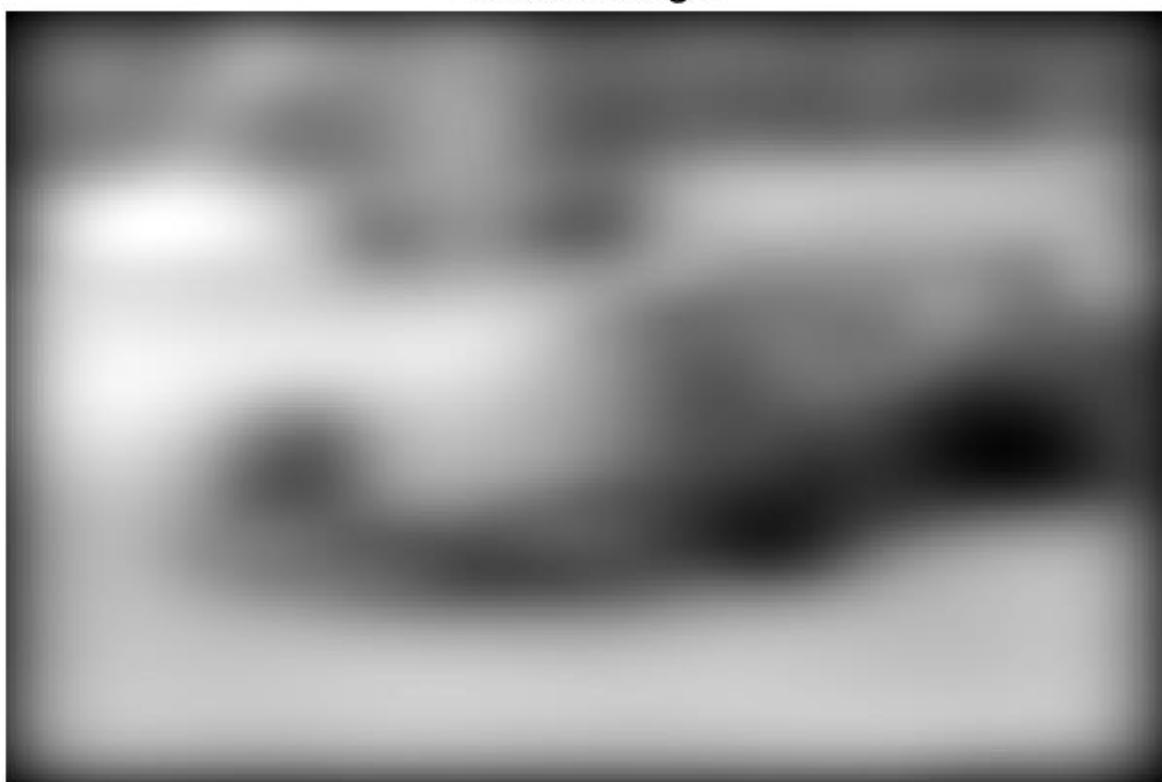
Result Image



Result Image



Result Image



위 4개의 이미지는 차수 $n = 2$ 로 통일했을 때, 순서대로 $D_0 = 128, 64, 30, 10$ 일 때의 결과이다.
Cut off Frequency D_0 가 작을수록 더 심한 blurring을 일으키는 것을 알 수 있다.

[2] – myHBF

두 번째로, myHBF는 Low-pass filter의 reverse operation으로 이해할 수 있다. 다만 한 가지 생각할 것은 앞서서 low pass filter의 두 parameter인 cutoff frequency D0와 degree order n 뿐만 아니라, boosting weight k에 대해서도 생각해야 한다.

```
% -----  
%  
% Creating Frequency filter and apply - High pass filter  
%  
  
% initialize H(u, v)  
D = zeros(PQ(1), PQ(2));  
Hlp = zeros(PQ(1), PQ(2));  
Hhp = zeros(PQ(1), PQ(2));  
Boost = zeros(PQ(1), PQ(2));  
  
% cutoff frequency D0 and order n and boosting weight k  
D0 = 50;  
n = 2;  
k = 10;  
%  
for u = 1:PQ(1)  
    for v = 1:PQ(2)  
        D(u, v) = sqrt((u - PQ(1) / 2).^2 + (v - PQ(2) / 2).^2);  
        Hlp(u, v) = 1 / (1 + (D(u, v) / D0).^(2 * n));  
        Hhp(u, v) = 1 - Hlp(u, v);  
        Boost(u, v) = 1 + k * Hhp(u, v);  
    end  
end  
  
F = Boost .* F;  
  
%  
% ToDo  
%  
G = F;  
  
figure, imshow(log(1+abs((G))), []);  
% -----
```

구현은 low pass filter 때와 일맥상통하기 때문에 코드에 대해선 별다른 코멘트는 생략한다.

아래와 같은 원본 이미지를 가질 때,

Input Image



(1) Cutoff frequency

Result Image



Result Image



Result Image



Result Image



위 4개 이미지는 $n = 2$, $k = 1$ 로 통일할 때, cutoff frequency $D_0 = 200, 64, 20, 10$ 로 달리한 모습이다. 원본과 비교하면 high boost 효과로 더 sharpen해진 것을 알 수 있고, D_0 가 작아질수록 sharpness 정도가 더 강해짐을 알 수 있다.

(2) Order

Result Image



Result Image



Result Image



Result Image



위 4개 이미지는 $D_0 = 100$, $k = 1$ 로 통일할 때, degree order $n = 2, 5, 20, 50$ 으로 달리한 모습이다.

Low pass filter 때와 동일하게, 차수 n이 커질수록 ringing artifact가 심해지는 것을 볼 수 있다.

(3) Weight k

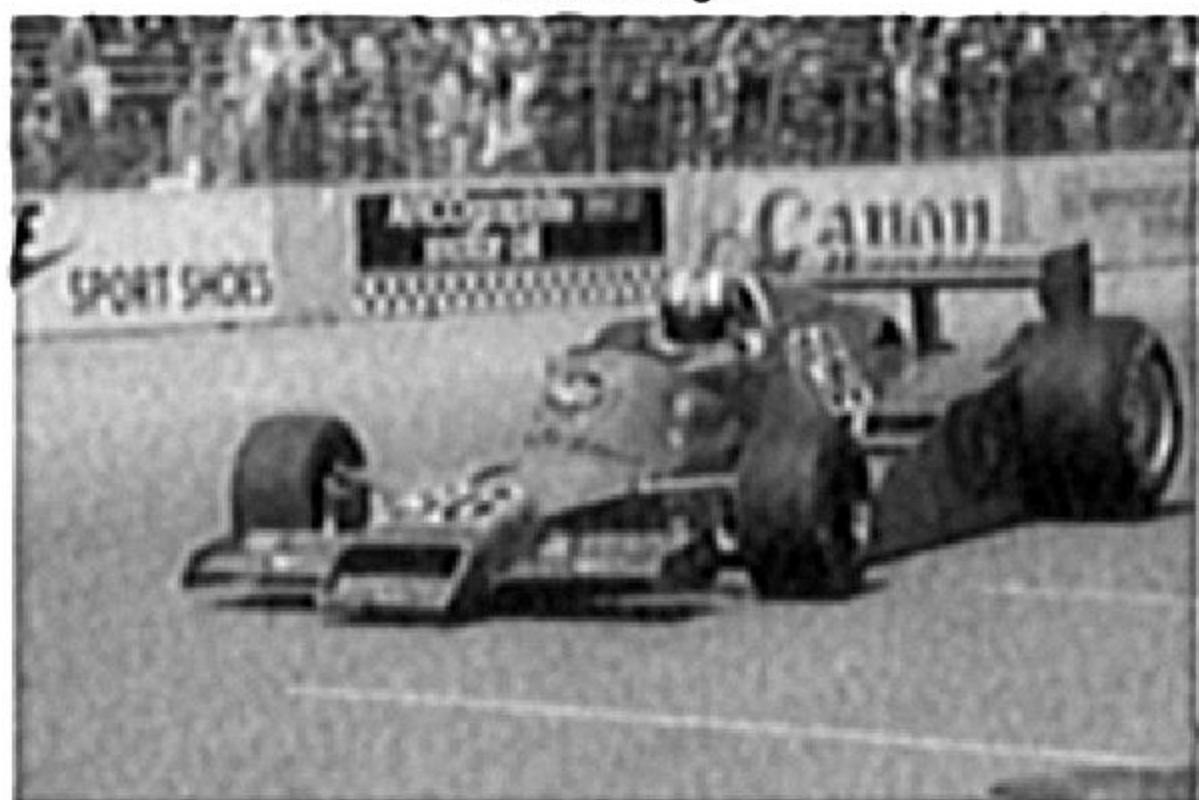
Result Image



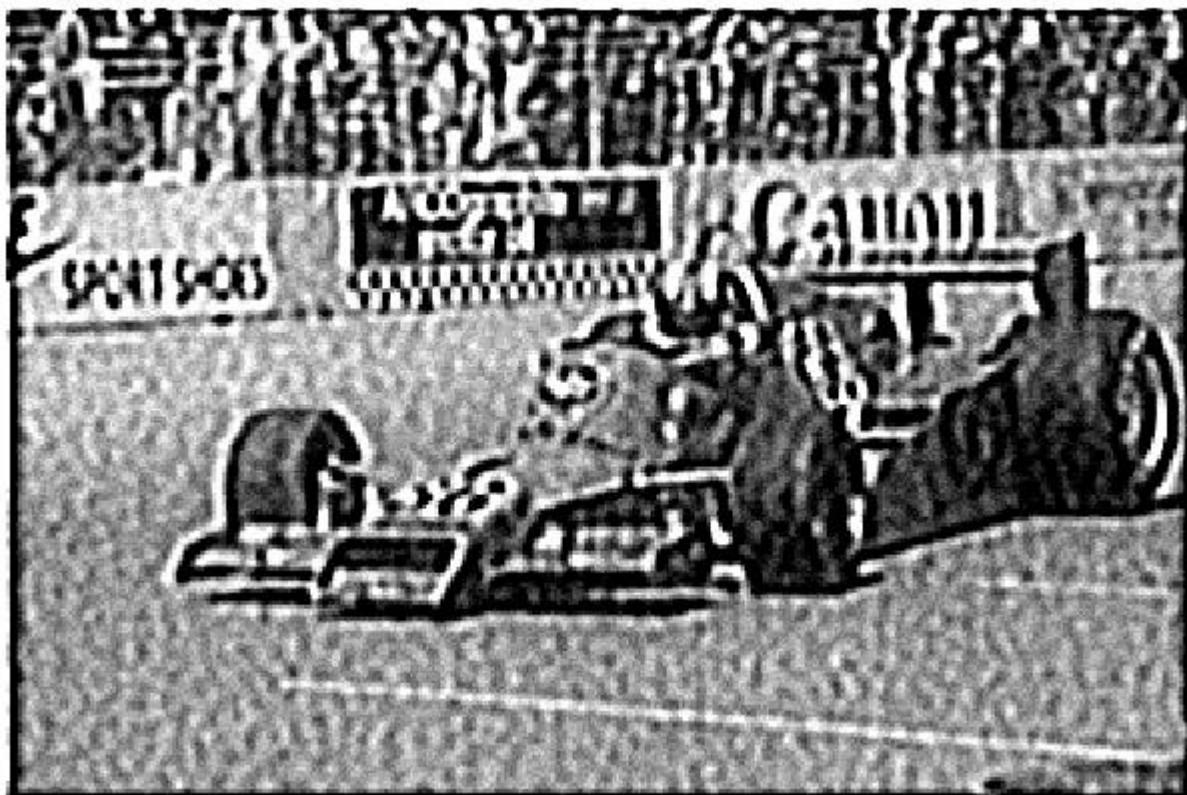
Result Image



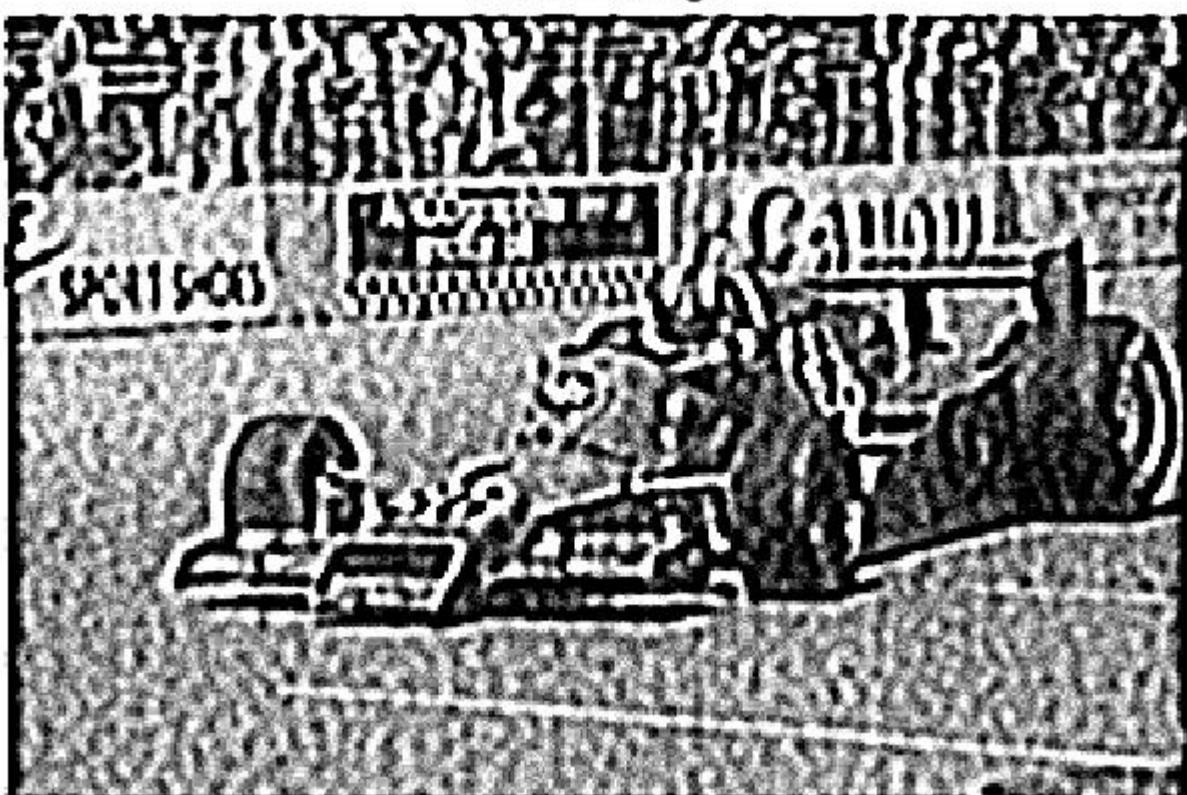
Result Image



Result Image



Result Image



위 5개 이미지는 $D_0 = 100$, $n = 2$ 로 통일할 때, weight $k = 0.001, 1, 10, 50, 100$ 으로 달리한 모습이다. K 가 커짐에 따라, 식을 보면 알 수 있듯이 원래 이미지에 unsharp mask를 더 큰 가중치로 더하기 때문에 극단적으로 edge가 강조된 이미지를 얻을 수 있다.

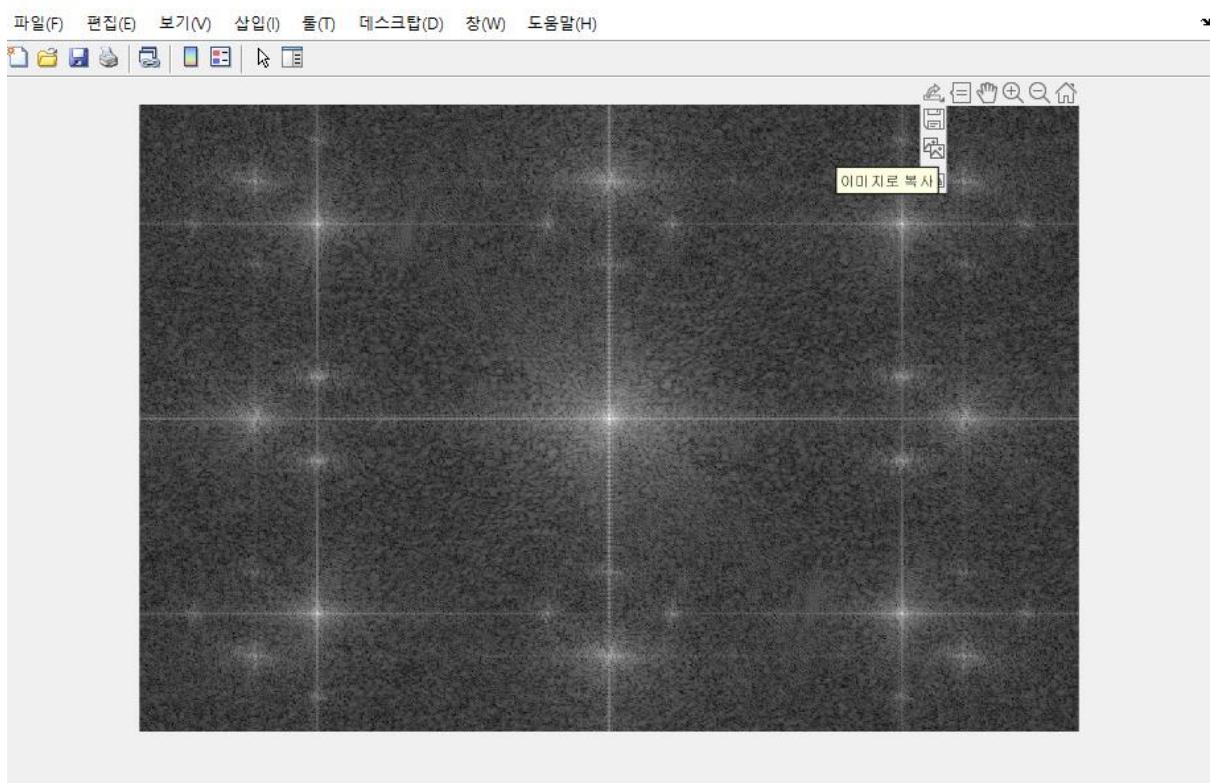
[3] – myNotch

마지막으로, myNotch 는 periodic한 background noise, 즉 일정하게 반복되는 패턴의 노이즈를 제거하는데 굉장히 유용하게 쓰이는 Notch filter에 관련한다. DFT (이산 푸리에 변환)의 특성상 원점 대칭인 것을 이용하여 origin에 대해 symmetric하게 위치한 notch pair에 대해서, 각각을 중심으로 갖는 High-pass filter의 곱으로 표현된다. 결국에 각각의 이미지에 따라 보이는 frequency domain의 형태에 따라 notch를 manually 적용하는, 사용자의 상호작용이 중요한 필터로 볼 수 있다.

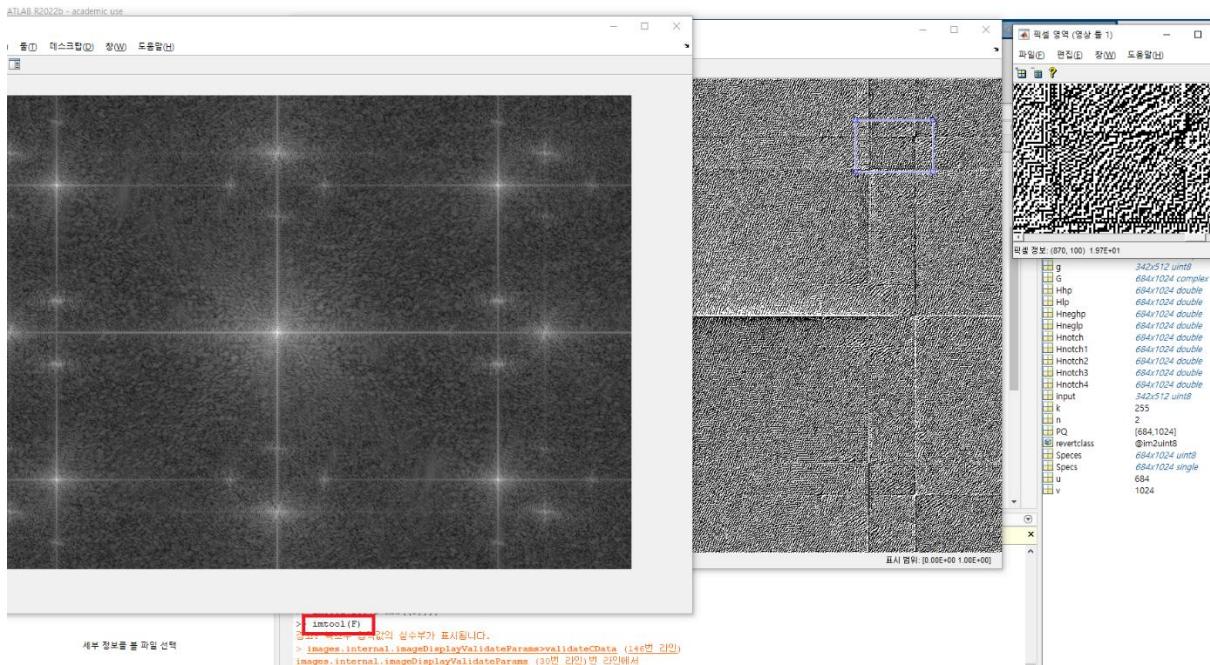
해당 과제의 경우 3가지의 이미지 cat-halftone.png, text-sineshade.tif, saturn- rings-sinusoidal-interf.tif 에 대해서 각각에 마땅한 notch filtering이 필요하다.

Notch 위치 점을 처리하는 이슈가 3번 과제에 가장 큰 이슈로 볼 수 있다. 필자의 경우, 처음에는 아래의 이미지처럼 frequency domain에서 회색조로 나타난 figure 자체 이미지를 복사하고 photoshop-like 한 툴을 이용해서 직접 notch를 삽입하여 해당 이미지를 처리하는 것이 어떨까 생각를 했으나 해당 방식으로 할 경우 여러 단점이 있지만 무엇보다 다운로드된 이미지의 해상도가 원본과도 달라지는 점이 주요했다.

Figure 2



그래서 필자가 선택한 방법은, matlab에 존재하는 영상뷰어 imtool을 이용해서 직접 notch 지점으로 설정할 위치 근방의 픽셀값을 알아내는 방식이었다.



하지만 여러 이미지에 따라 조금씩 달라지기 때문에 필자는 우선 cat-halftone.png 이미지에 대해선 아래와 같은 코드를 이용했다.

```
% -----
%
% Creating Frequency filter and apply - High pass filter
%
%
% initialize H(u, v)
D = zeros(PQ(1), PQ(2));
Dneg = zeros(PQ(1), PQ(2));

Hlp = zeros(PQ(1), PQ(2));
Hhp = zeros(PQ(1), PQ(2));
Hneglp = zeros(PQ(1), PQ(2));
Hneghp = zeros(PQ(1), PQ(2));

Hnotch = zeros(PQ(1), PQ(2));

D0 = 100;
n = 2;
k = 2;
% notch - origin
uk = (PQ(1) / 2)*(3/5);
vk = (PQ(2) / 2)*(5/8);
%
for u = 1:PQ(1)
    for v = 1:PQ(2)
        D(u, v) = sqrt((u - PQ(1) / 2 - uk).^2 + (v - PQ(2) / 2 - vk).^2);
        Dneg(u, v) = sqrt((u - PQ(1) / 2 + uk).^2 + (v - PQ(2) / 2 + vk).^2);

        Hlp(u, v) = 1 / (1 + (D(u, v) / D0).^(2 * n));
        Hhp(u, v) = 1 - Hlp(u, v);
        Hneglp(u, v) = 1 / (1 + (Dneg(u, v) / D0).^(2 * n));
        Hneghp(u, v) = 1 - Hneglp(u, v);
    end
end

Hnotch = Hhp .* Hneghp;

for u = 1:PQ(1)
    for v = 1:PQ(2)
        D(u, v) = sqrt((u - PQ(1) / 2 + uk).^2 + (v - PQ(2) / 2 - vk).^2);
        Dneg(u, v) = sqrt((u - PQ(1) / 2 - uk).^2 + (v - PQ(2) / 2 + vk).^2);

        Hlp(u, v) = 1 / (1 + (D(u, v) / D0).^(2 * n));
        Hhp(u, v) = 1 - Hlp(u, v);
        Hneglp(u, v) = 1 / (1 + (Dneg(u, v) / D0).^(2 * n));
        Hneghp(u, v) = 1 - Hneglp(u, v);
    end
end

Hnotch = Hnotch .* Hhp .* Hneghp;
```

```

F = Hnotch .* F;

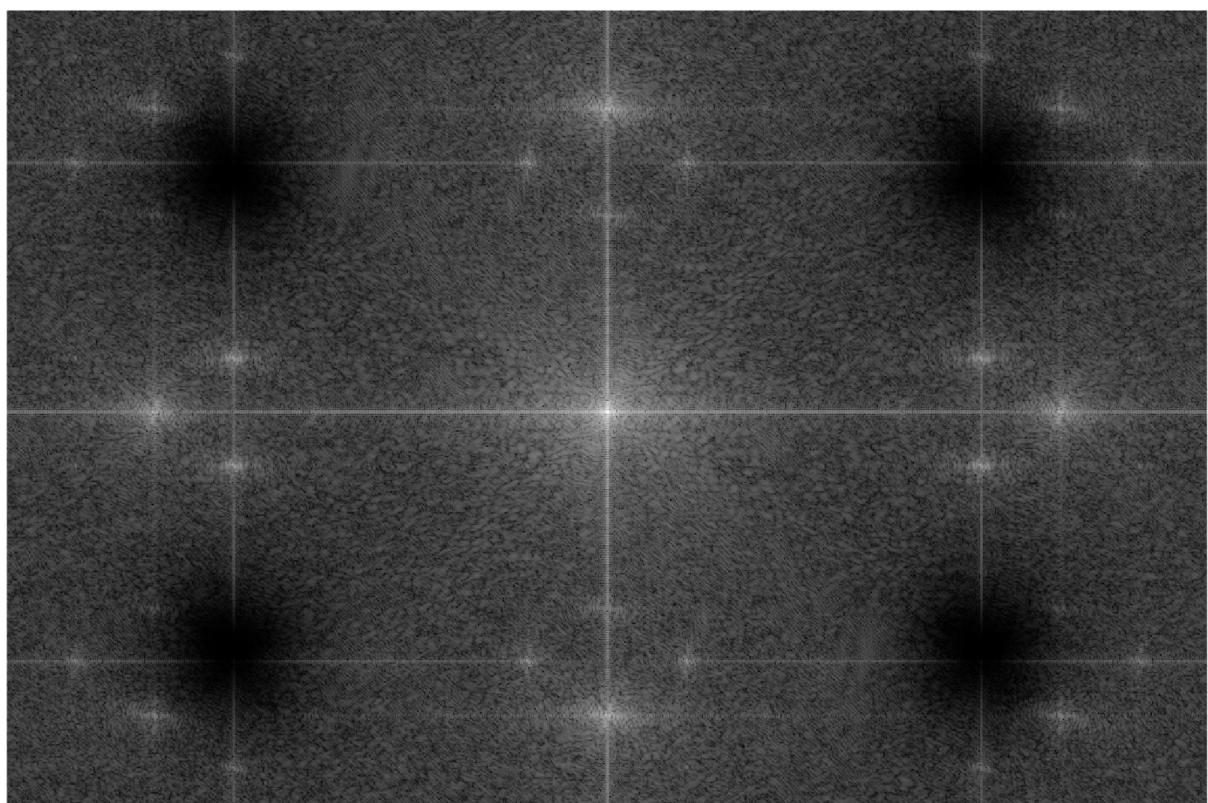
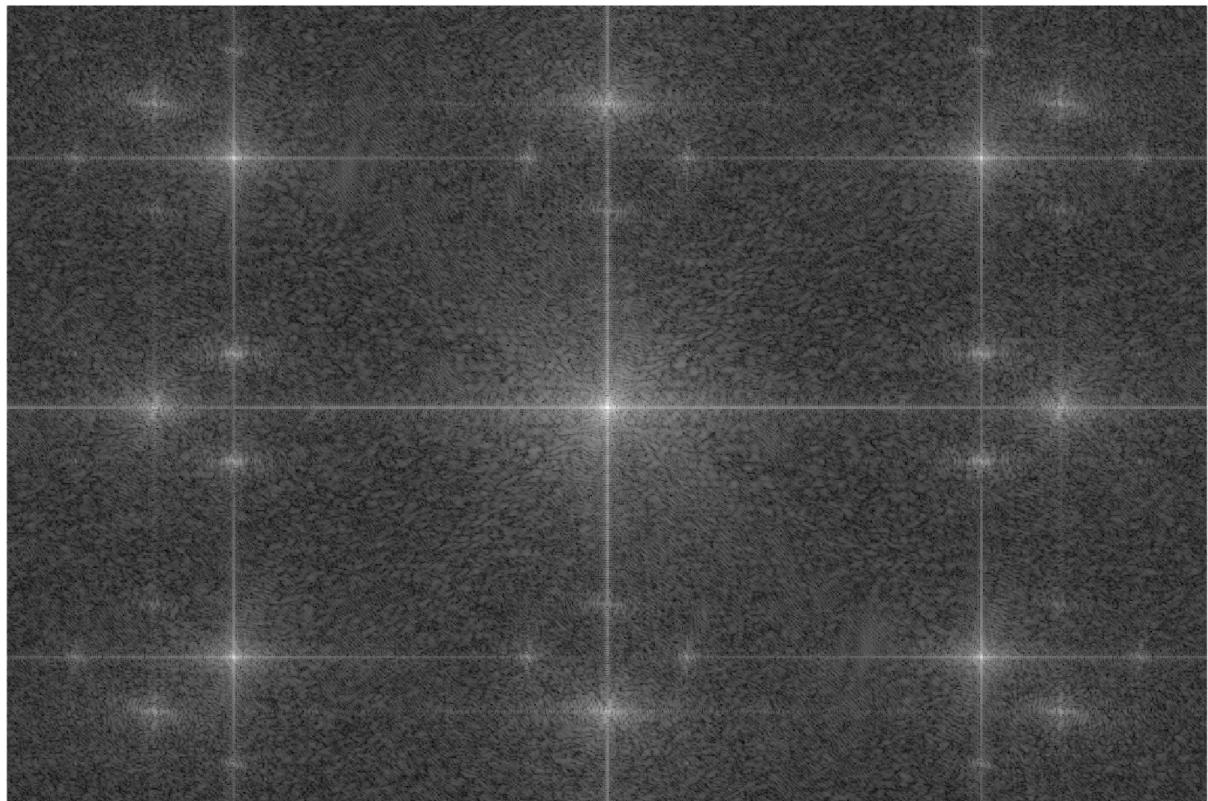
%
% ToDo
%
G = F;
figure, imshow(log(1+abs((G))), []);
%
```

Notch 지점 (u_k, v_k)를 필자는 그림판을 이용해 대충 중심으로부터 떨어진 비율을 짐작했기 때문에, u_k 와 파의 표현도 전체 이미지의 가로, 세로에 대한 비율값으로 적용시켰다.

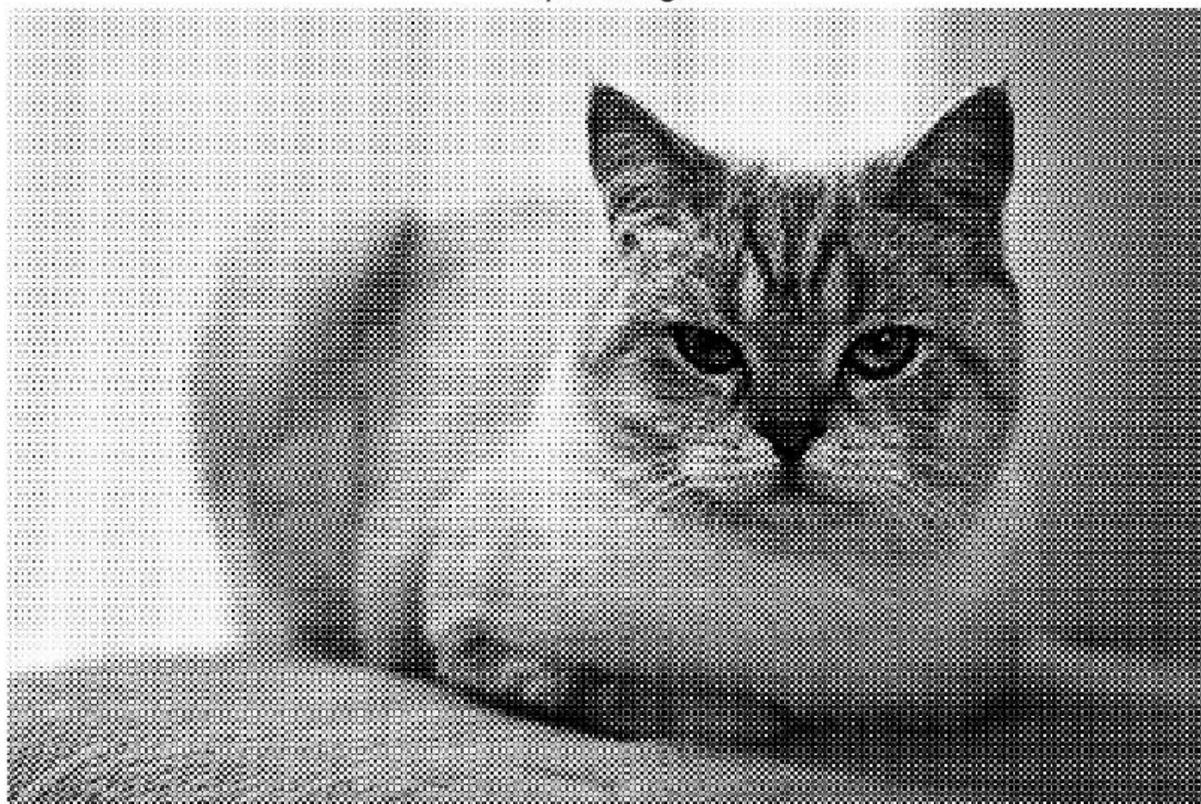
해당 notch에 따라 상응하는 원점대칭 notch가 존재하므로 $D(u, v)$ 와 $D_{neg}(u, v)$ 쌍으로 구하며, 그에 따라 high-pass filter를 앞선 2번 과제의 방식으로 butterworth high-pass filter 값을 구했다. 그리고 H_{notch} 는 해당 high-pass filter들의 곱으로 적용시켰다.

(1) cat-halftone.png 이미지

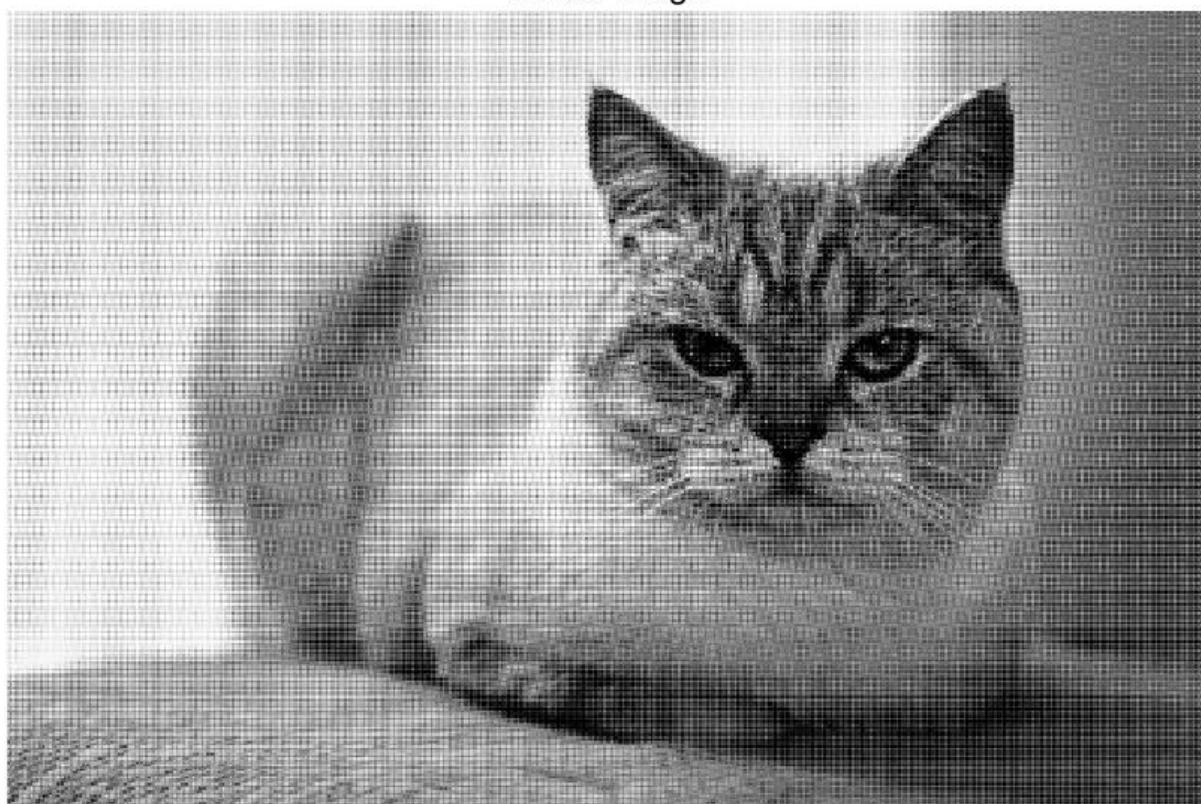
코드에 대한 설명의 경우 위에서 이미 설명했으므로 생략한다. 아래 4개 이미지는 각각 frequency domain의 notch 적용 전후 이미지와 고양이 이미지의 원본과 필터링 적용된 이후 이미지이다.



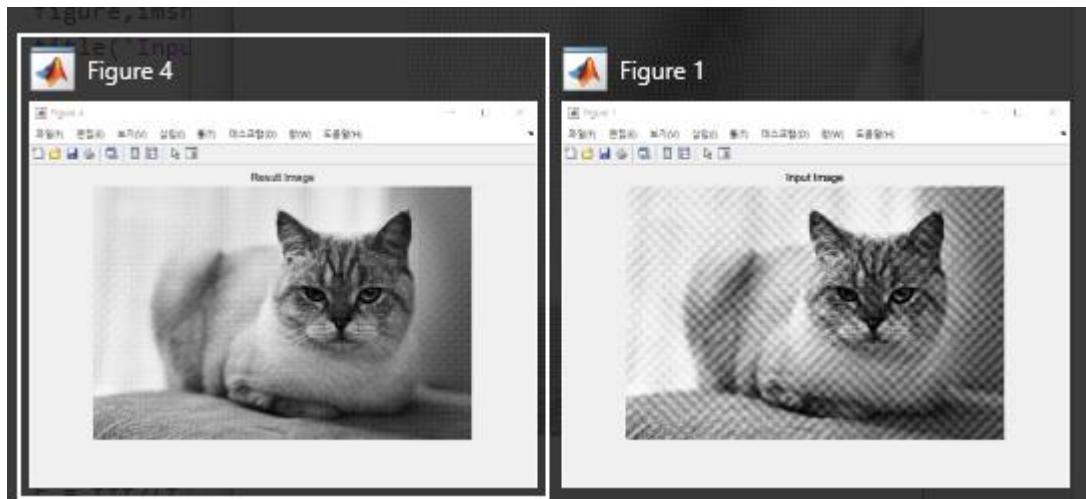
Input Image



Result Image



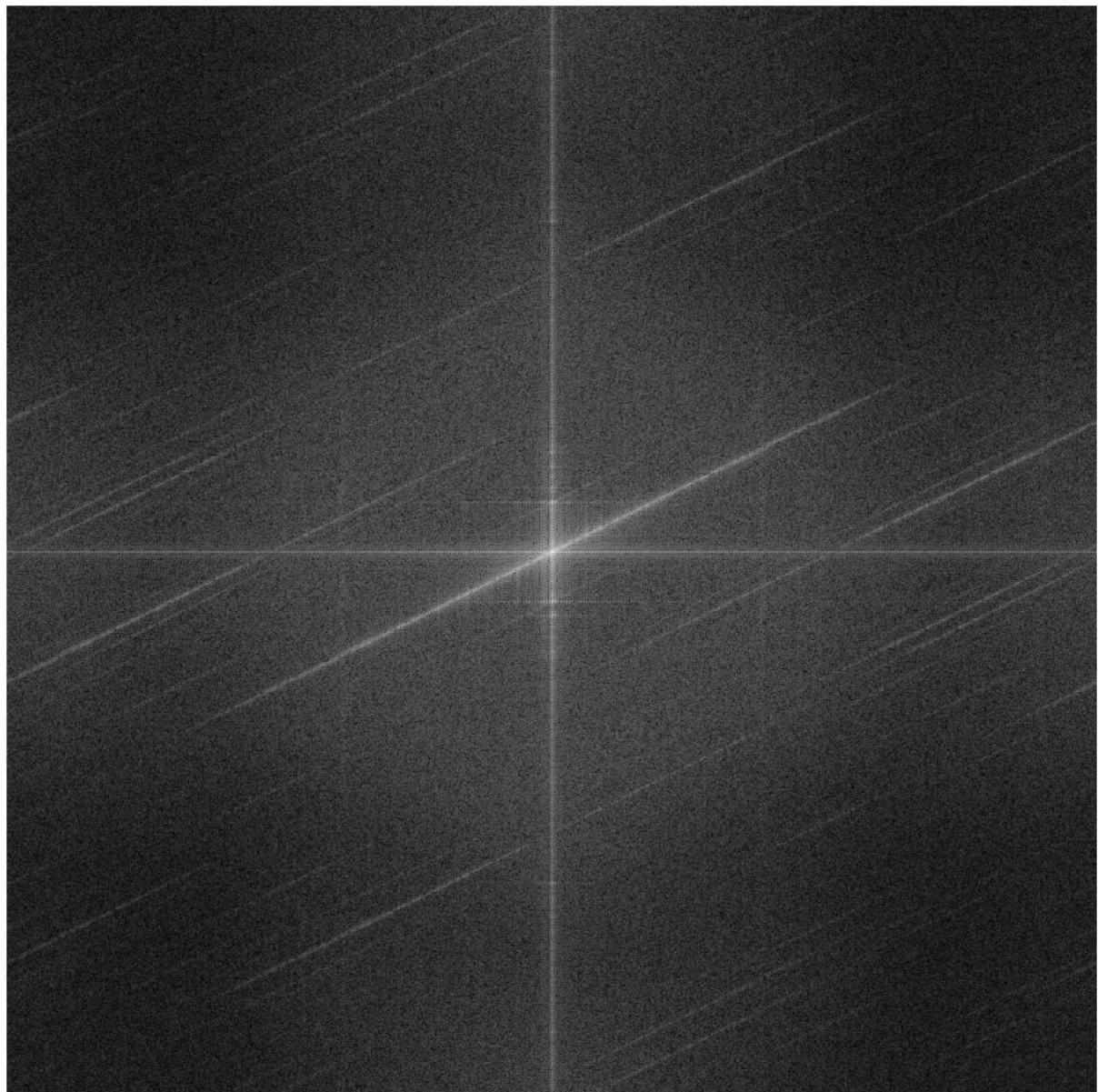
필자는 frequency domain에서 크게 두드러지는 4개의 밝은 부분에 집중했기 때문에, 해당 위치들을 2개 쌍의 notch pairs로 설정하여 필터링을 적용했다. 그 결과로 체커보드와 유사한 패턴의 노이즈가 완화된 모습이다. 노이즈를 완화한 것을 좀 더 쉽게 현저한 차이를 알기 위해 아래 alt-tab을 누른 상태에서 찍은 스크린샷을 추가했다. 확연하게 차이가 보이는 모습이다.

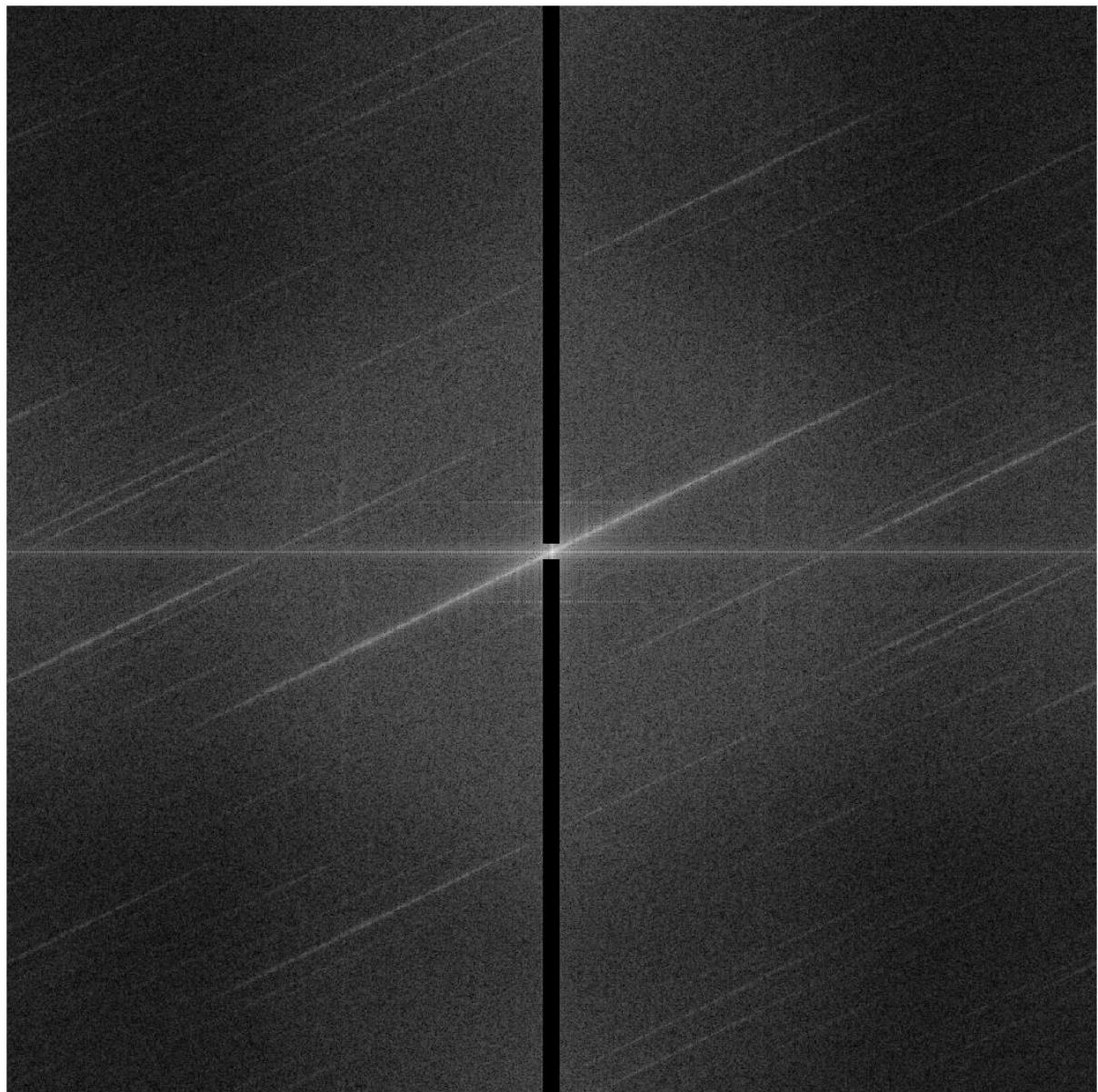


(2) Saturn-rings-sinusodial-interf 이미지

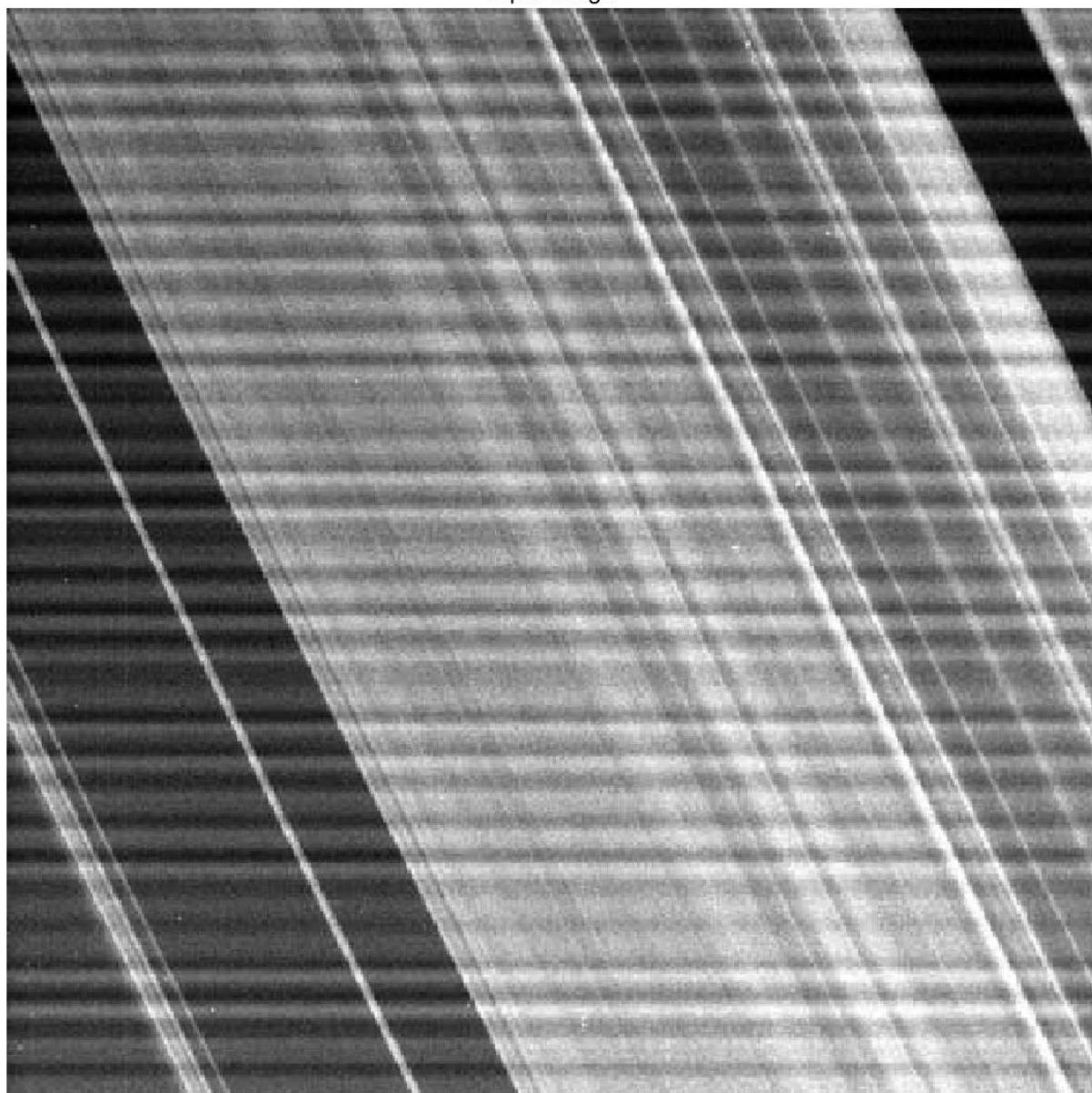
아래에는 해당 이미지에 대한 노치 필터링을 위한 코드와, frequency domain 전후 이미지, 그리고 이미지의 원본과 처리된 결과이미지 들이다.

```
%% Case 3 :: for 'saturn-rings-sinusoidal-interf.tif'
%% narrow rectangle
H1 = ones(PQ(1), PQ(2));
H1neg = ones(PQ(1), PQ(2));
for i = 1:PQ(1)/2 - 10
    for j = PQ(2)/2 - 10: PQ(2)/2 + 10
        H1(i, j) = 0;
        H1neg(PQ(1)-i, PQ(2)-j) = 0;
    end
end
Hnotch = H1 .* H1neg;
F = Hnotch .* F;
%
% ToDo
%
G = F;
figure, imshow(log(1+abs((G))), []);
```

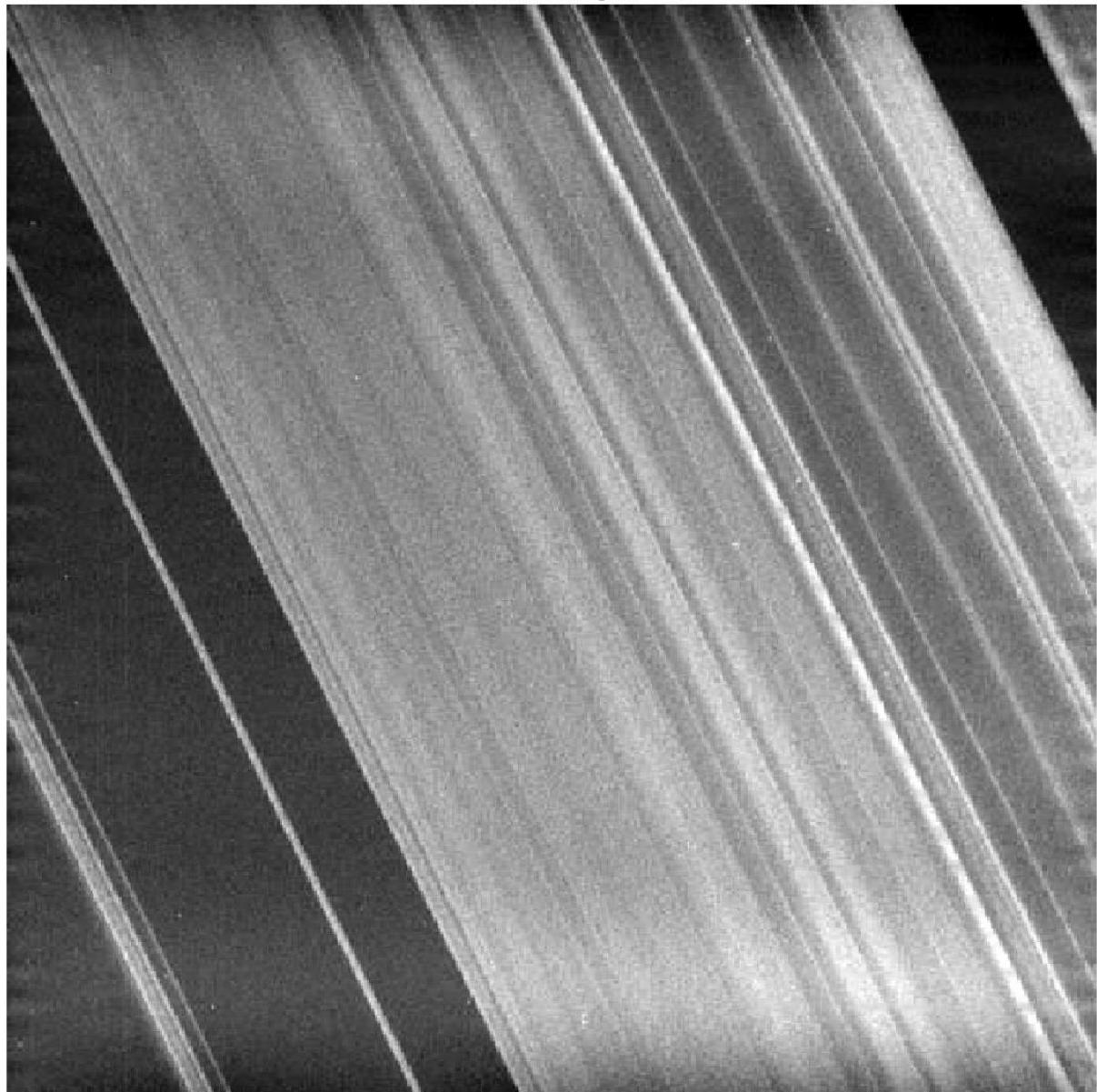




Input Image



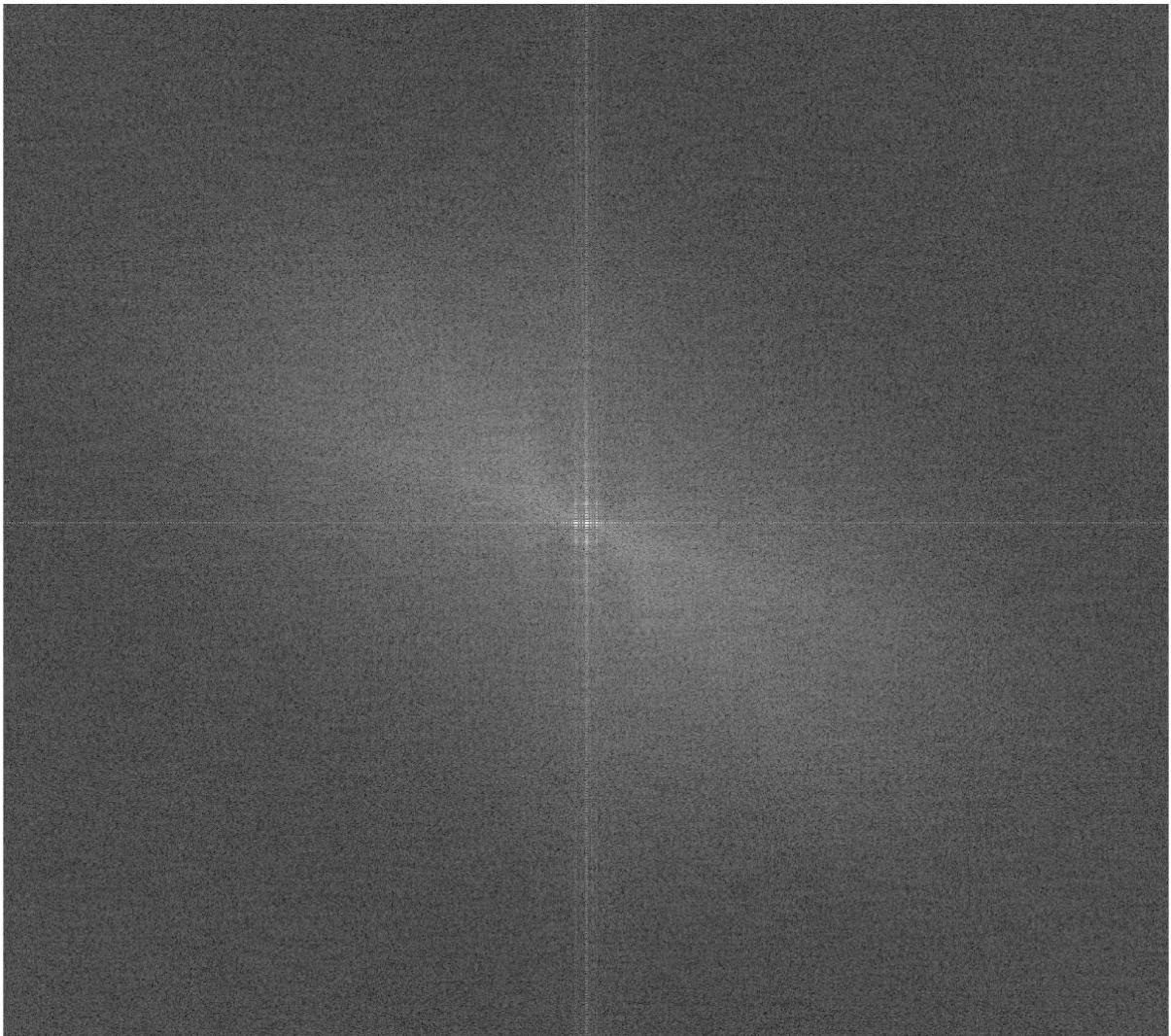
Result Image



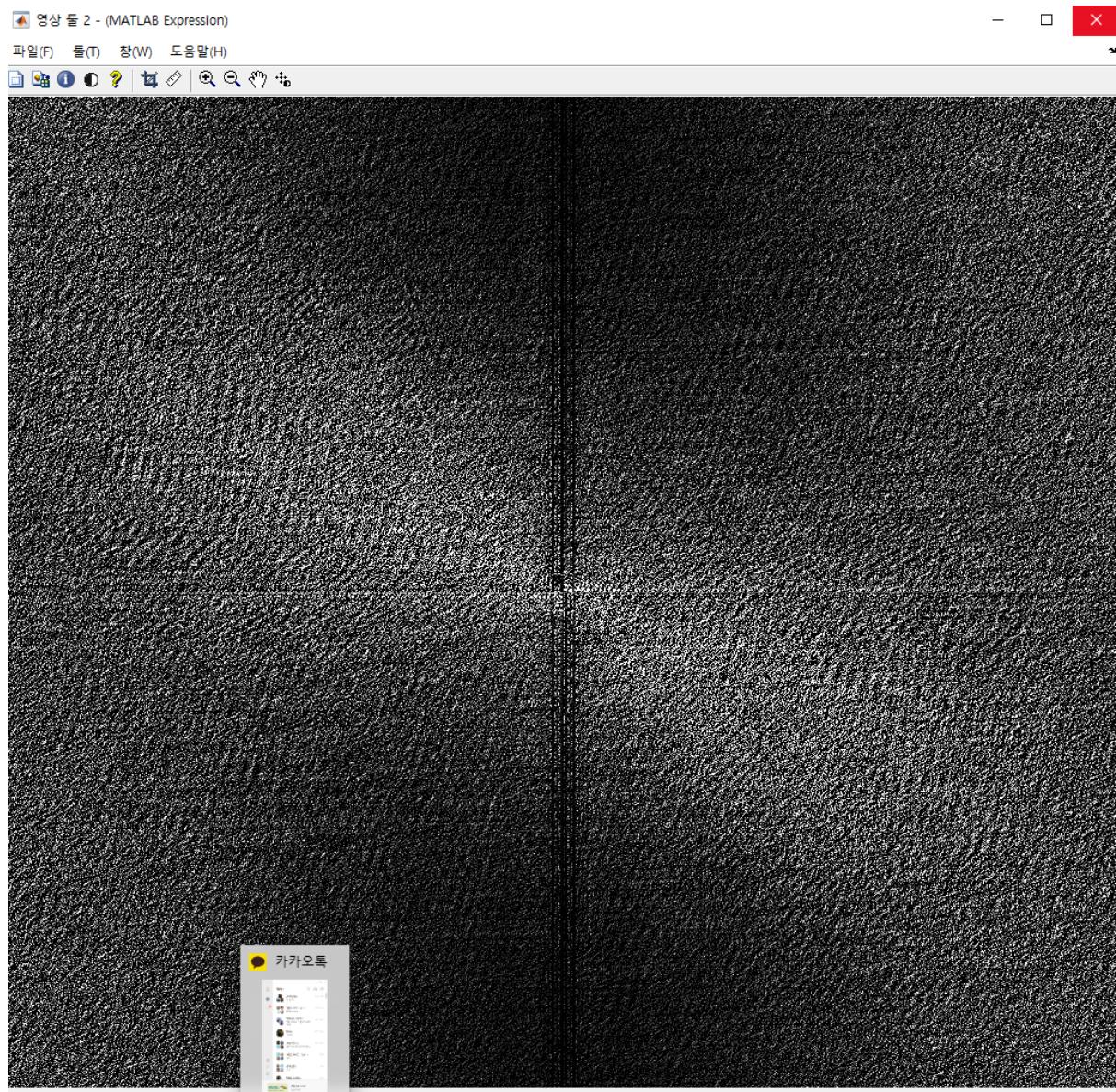
좁은 직사각형 형태의 notch를 놓음으로 반복되는 노이즈 패턴을 제거한 모습이다. 해당 노치 필터를 반전시킨다면 오히려 노이즈 패턴을 추출할 수도 있다.

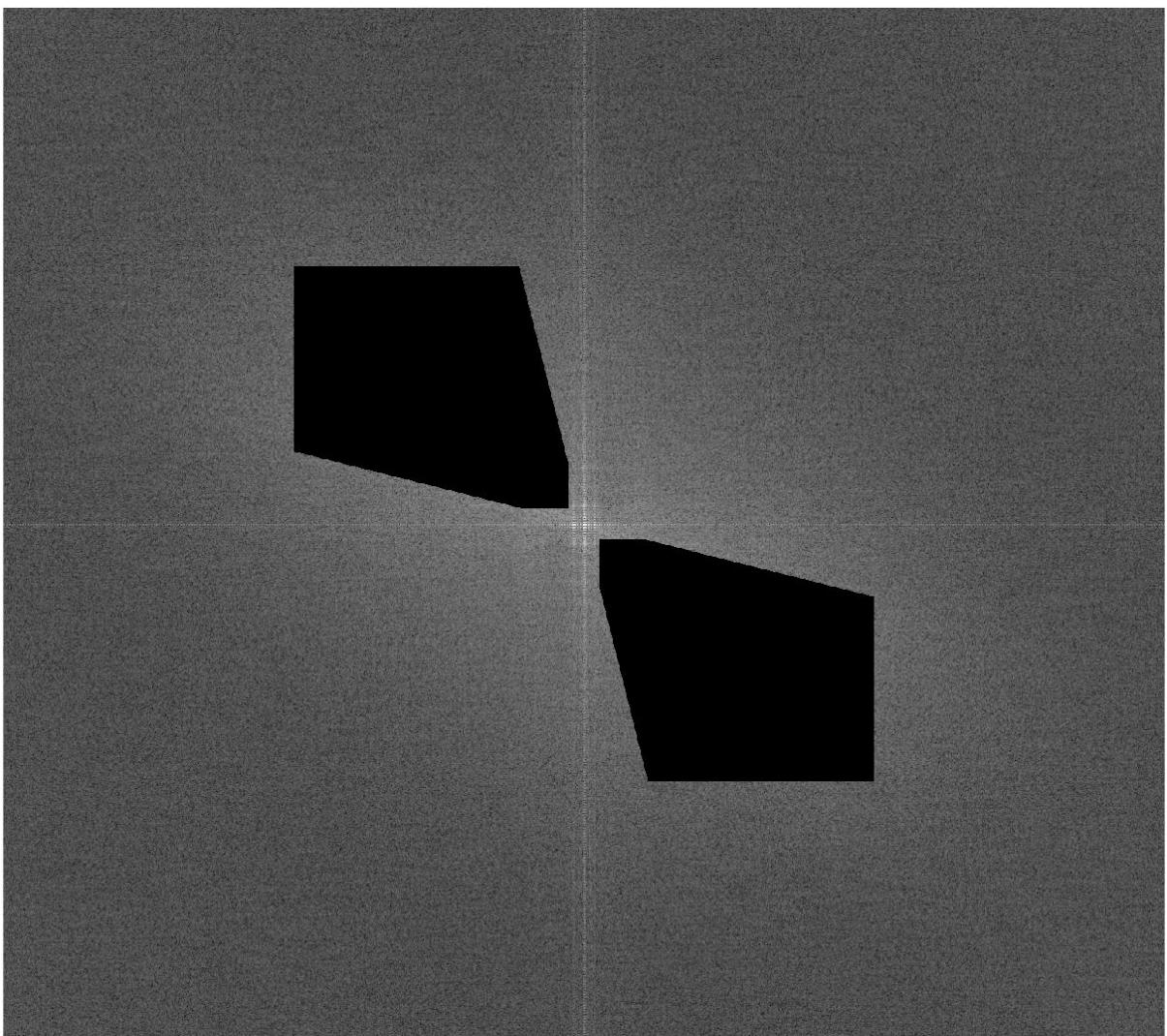
(3) text-sineshade 이미지

제시된 순서 상으론 원래 두 번째로 위치하려고 했으나, 굉장히 고민이 많았기 때문에 가장 마지막에 작성했다. 그 이유는 아래의 frequency domain 이미지를 보면 알 수 있다.

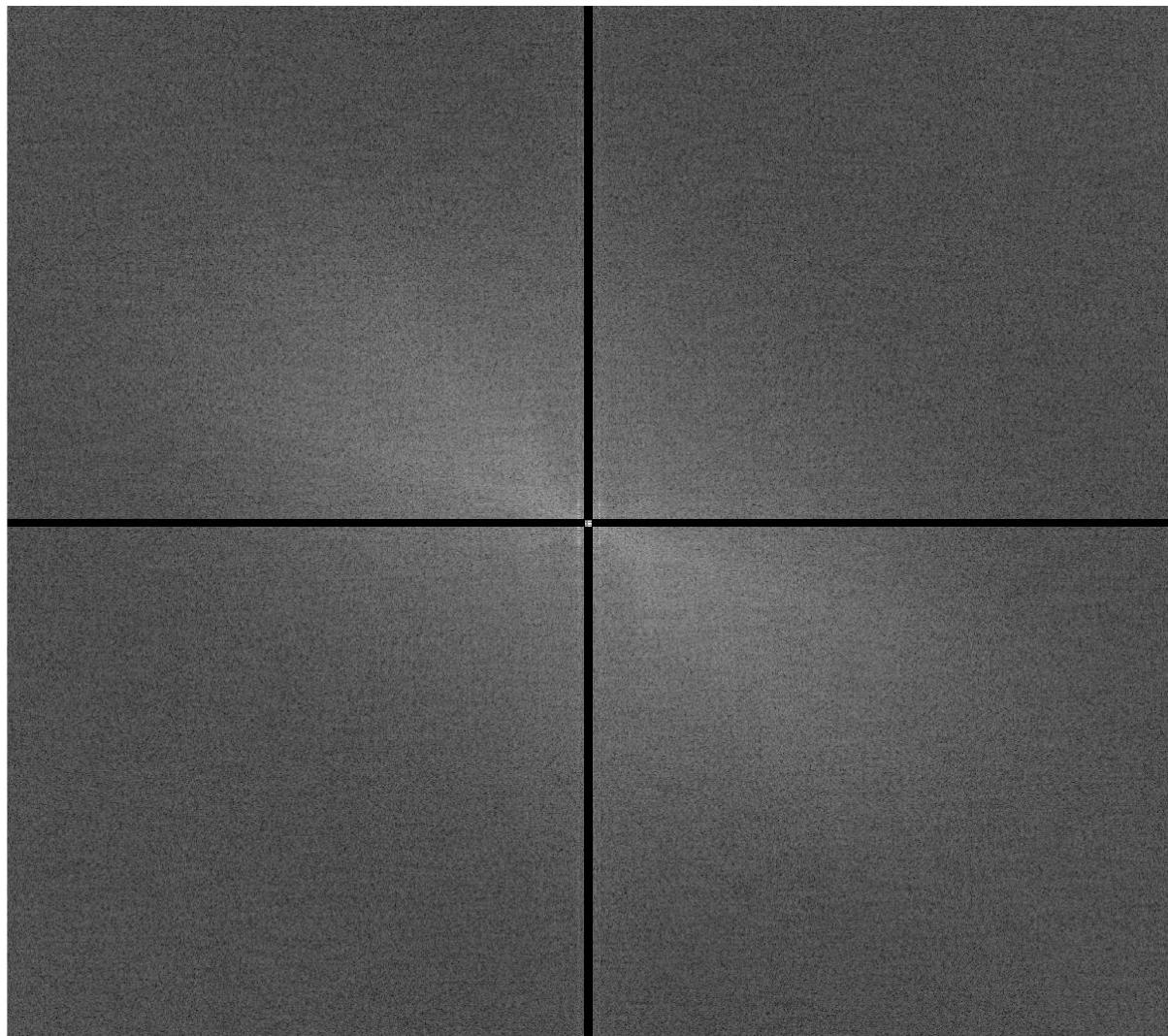


노치를 어디에 적용할지에 대한 고민이 매우 컸다. 어떻게 보면 원상단에서 우향하는 방향쪽에 스펙트럼이 밝은 부분인 것 같기도 하다. 아래 이미지는 imtool 명령어를 통해 normalize된 스펙트럼 이미지이다. Imtool(normalize(F))





그렇기에 앞선 생각을 바탕으로 다음과 같이 적용하려 했으나 효과적이지 못했다. 결국 앞선 노치와 비슷한 방식으로 스펙트럼 직사각형의 가운데 십자가 모양과 유사하게 좁은 직사각형 notch 쌍을 선택하여 다음과 같이 적용했다.



다음은 원본 이미지와 notch filtering 처리 이후의 결과 이미지이다.

Twenty Six between Stockley
of Knx. And State of Tennessee
Andrew Jackson of the County
State aforesaid of the other part
Paid Stockley Donelson for A
of the sum of two thousand
and paid thy receipt where
 hath And by these presents
 sell alien encoff And confirm
 Jackson his heirs And a
 certain traits or parcels of La
 sand acres one thousand acre
 more or less And his

Result Image

Indinty Six between Stockley
of Knx and State of Tennessee
Andrew Jackson of the County
State aforesaid of the other part
Paid Stockley Donelson for a
of the Sum of two thousand
Dollars paid the receipt where
Rath And by these presents
Bill alien enccoff And Confer
Jackson his heirs and a
Certain tracts or parcels of La
sand Acre 1 one thousand acre
more or less in the said land

다소 깨끗하진 못하지만, 그래도 원본 이미지 안에 있던 물결 같은 반복적인 noise 패턴에 대해선 완화된 모습이다. 아쉽게도 어두운 부분 안에 있던 text는 밝아지게 함에 따라 그 안에 있던 text 가 옅어지면서 가시성은 다소 떨어지는 결과를 보였다.

아래는 이에 대한 코드이다.

```
%% Case 3 :: for 'text-sineshade.tif'
%%

H1 = ones(PQ(1), PQ(2));
H1neg = ones(PQ(1), PQ(2));

for i = 1:PQ(1)/2 - 5
    for j = PQ(2)/2 - 5: PQ(2)/2 + 5
        H1(i, j) = 0;
```

```
H1neg(PQ(1)-i, PQ(2)-j) = 0;
end
end

for i = 1:PQ(2)/2 - 5
    for j = PQ(1)/2 - 5: PQ(1)/2 + 5
        H1(j, i) = 0;
        H1neg(PQ(1)-j, PQ(2)-i) = 0;
    end
end

Hnotch = H1 .* H1neg;
F = Hnotch .* F;
```