

# ICE4027 Digital Image Processing

## Lab.4 – Filtering in Frequency Domain

Prof. In Kyu Park



**인하대학교 시각컴퓨팅 및 학습 연구실**  
Visual Computing and Learning Laboratory

# Contents

Lab.1 – Get Familiar with Image Processing

Lab.2 – Point Processing and Histogram

Lab.3 – Linear Filtering

## **Lab.4 – Filtering in Frequency Domain**

Lab.5 – Median and Edge Filtering

Lab.6 – Color Processing and Clustering

Lab.7 – Clustering and Segmentation

Lab.8 – Local Features and SIFT

Lab.9 – Image Transformation

Lab.10 – Panorama Stitching

Lab.11 – Motion Estimation (KLT)

Lab.12 – High Dynamic Range (HDR)



# Frequency Domain Analysis

## ■ 주파수 영역 분석

- 영상 신호를 주파수 영역에서 분석하는 것  
→ 공간 영역에서 잘 보이지 않던 정보를 쉽게 발견할 수 있게 됨
- Fourier transform을 통해 변환할 수 있음
- Magnitude 성분과 phase 성분으로 나누어 시각화 가능
- 일반적으로는 magnitude spectrum에 대한 분석만으로 충분



$$F(\omega) = R(\omega) + iI(\omega)$$

$$A = \pm \sqrt{R(\omega)^2 + I(\omega)^2} \quad \phi = \tan^{-1} \frac{I(\omega)}{R(\omega)}$$

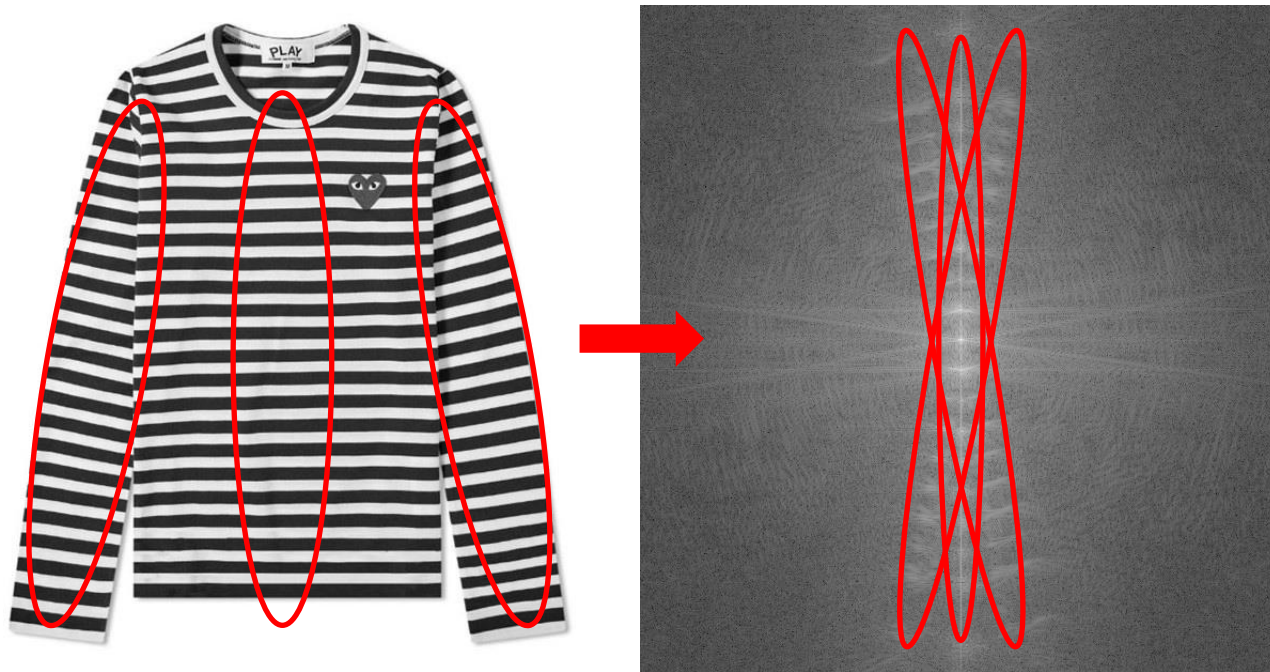
Magnitude

Phase

# Frequency Domain Analysis

## ■ 주파수 영역에서 발견할 수 있는 정보

- 영상 속에서 주기적으로 반복되는 성분을 쉽게 파악할 수 있음
- 주기적으로 반복되는 성분의 주기와 방향이 스펙트럼으로 나타나기 때문



# Discrete Fourier Transform


## ■ 2D DFT

```
void cv::dft ( InputArray  src,
               OutputArray dst,
               int         flags = 0 ,
               int         nonzeroRows = 0
             )
```

```
Mat doDft(Mat srcImg) {
    Mat floatImg;
    srcImg.convertTo(floatImg, CV_32F);

    Mat complexImg;
    dft(floatImg, complexImg, DFT_COMPLEX_OUTPUT);

    return complexImg;
}
```



$$\begin{bmatrix} ReY_{0,0} & ReY_{0,1} & ImY_{0,1} & ReY_{0,2} & ImY_{0,2} & \dots & ReY_{0,N/2-1} & ImY_{0,N/2-1} & ReY_{0,N/2} \\ ReY_{1,0} & ReY_{1,1} & ImY_{1,1} & ReY_{1,2} & ImY_{1,2} & \dots & ReY_{1,N/2-1} & ImY_{1,N/2-1} & ReY_{1,N/2} \\ ImY_{1,0} & ReY_{2,1} & ImY_{2,1} & ReY_{2,2} & ImY_{2,2} & \dots & ReY_{2,N/2-1} & ImY_{2,N/2-1} & ImY_{1,N/2} \\ \dots & & & & & & & & \\ ReY_{M/2-1,0} & ReY_{M-3,1} & ImY_{M-3,1} & \dots & ReY_{M-3,N/2-1} & ImY_{M-3,N/2-1} & ReY_{M/2-1,N/2} & & \\ ImY_{M/2-1,0} & ReY_{M-2,1} & ImY_{M-2,1} & \dots & ReY_{M-2,N/2-1} & ImY_{M-2,N/2-1} & ImY_{M/2-1,N/2} & & \\ ReY_{M/2,0} & ReY_{M-1,1} & ImY_{M-1,1} & \dots & ReY_{M-1,N/2-1} & ImY_{M-1,N/2-1} & ReY_{M/2,N/2} & & \end{bmatrix}$$

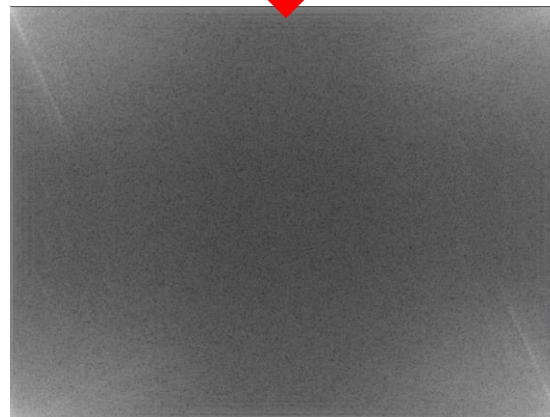
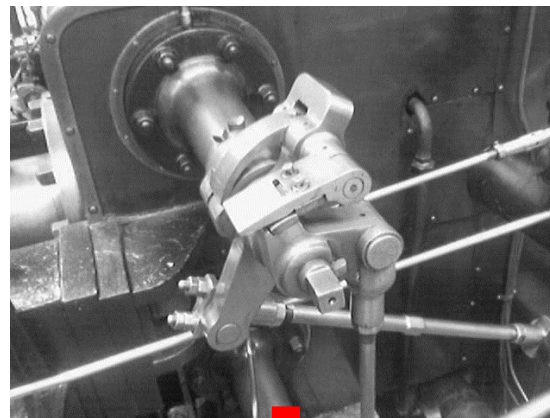
**CCS (complex-conjugate-symmetrical) format**

# Discrete Fourier Transform

## ■ Magnitude 영상 취득

```
void cv::magnitude ( InputArray  x,  
                    InputArray  y,  
                    OutputArray magnitude  
                    )
```

```
Mat getMagnitude(Mat complexImg) {  
    Mat planes[2];  
    split(complexImg, planes);  
    // 실수부, 허수부 분리  
  
    Mat magImg;  
    magnitude(planes[0], planes[1], magImg);  
    magImg += Scalar::all(1);  
    log(magImg, magImg);  
    // magnitude 취득  
    // log(1 + sqrt(Re(DFT(I))^2 + Im(DFT(I))^2))  
  
    return magImg;  
}  
  
Mat myNormalize(Mat src) {  
    Mat dst;  
    src.copyTo(dst);  
    normalize(dst, dst, 0, 255, NORM_MINMAX);  
    dst.convertTo(dst, CV_8UC1);  
  
    return dst;  
}
```



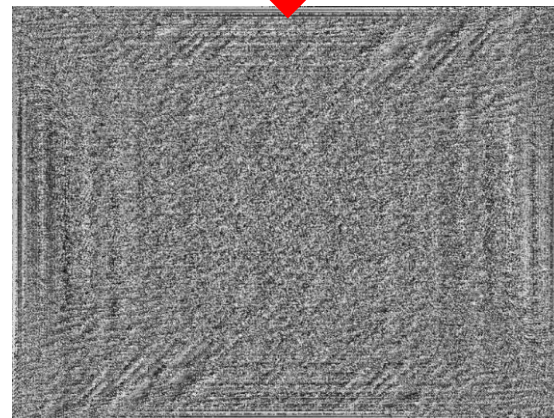
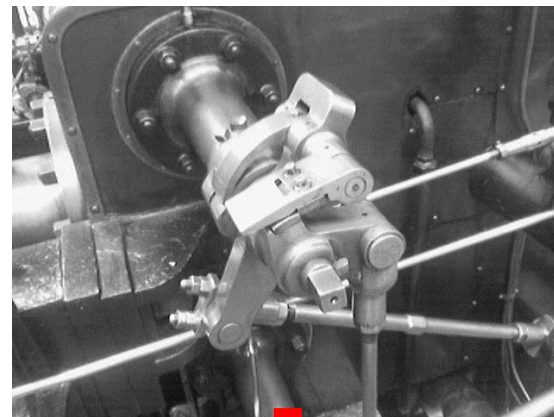
Magnitude 결과

# Discrete Fourier Transform

## ■ Phase 영상 취득

```
void cv::phase ( InputArray  x,  
                 InputArray  y,  
                 OutputArray angle,  
                 bool        angleInDegrees = false  
               )
```

```
Mat getPhase(Mat complexImg) {  
    Mat planes[2];  
    split(complexImg, planes);  
    // 실수부, 허수부 분리  
  
    Mat phaImg;  
    phase(planes[0], planes[1], phaImg);  
    // phase 취득  
  
    return phaImg;  
}  
  
Mat myNormalize(Mat src) {  
    Mat dst;  
    src.copyTo(dst);  
    normalize(dst, dst, 0, 255, NORM_MINMAX);  
    dst.convertTo(dst, CV_8UC1);  
  
    return dst;  
}
```



Phase 결과



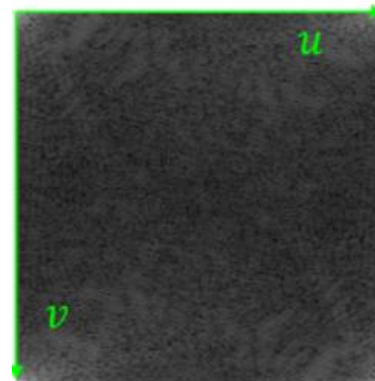
# Discrete Fourier Transform

## 좌표계 중앙 이동(centralize)

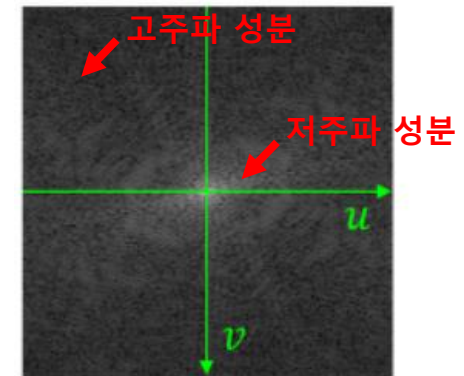
```
Mat centralize(Mat complex) {  
    Mat planes[2];  
    split(complex, planes);  
    int cx = planes[0].cols / 2;  
    int cy = planes[1].rows / 2;  
  
    Mat q0Re(planes[0], Rect(0, 0, cx, cy));  
    Mat q1Re(planes[0], Rect(cx, 0, cx, cy));  
    Mat q2Re(planes[0], Rect(0, cy, cx, cy));  
    Mat q3Re(planes[0], Rect(cx, cy, cx, cy));  
  
    Mat tmp;  
    q0Re.copyTo(tmp);  
    q3Re.copyTo(q0Re);  
    tmp.copyTo(q3Re);  
    q1Re.copyTo(tmp);  
    q2Re.copyTo(q1Re);  
    tmp.copyTo(q2Re);  
  
    Mat q0Im(planes[1], Rect(0, 0, cx, cy));  
    Mat q1Im(planes[1], Rect(cx, 0, cx, cy));  
    Mat q2Im(planes[1], Rect(0, cy, cx, cy));  
    Mat q3Im(planes[1], Rect(cx, cy, cx, cy));  
  
    q0Im.copyTo(tmp);  
    q3Im.copyTo(q0Im);  
    tmp.copyTo(q3Im);  
    q1Im.copyTo(tmp);  
    q2Im.copyTo(q1Im);  
    tmp.copyTo(q2Im);  
  
    Mat centerComplex;  
    merge(planes, 2, centerComplex);  
  
    return centerComplex;  
}
```



(a)  $f(x,y)$



(b)  $\log(|F(u,v)|)$



(c)  $\log(|F(u-W/2,v-H/2)|)$



# Discrete Fourier Transform

## ■ 2D IDFT

```
void cv::idft ( InputArray  src,
                OutputArray dst,
                int         flags = 0 ,
                int         nonzeroRows = 0
              )
```

```
Mat setComplex(Mat magImg, Mat phaImg) {
    exp(magImg, magImg);
    magImg -= Scalar::all(1);
    // magnitude 계산을 반대로 수행

    Mat planes[2];
    polarToCart(magImg, phaImg, planes[0], planes[1]);
    // 극 좌표계 -> 직교 좌표계 (각도와 크기로부터 2차원 좌표)

    Mat complexImg;
    merge(planes, 2, complexImg);
    // 실수부, 허수부 합체

    return complexImg;
}
```

```
Mat doIdft(Mat complexImg) {
    Mat idftcvt;
    idft(complexImg, idftcvt);
    // IDFT를 이용한 원본 영상 취득

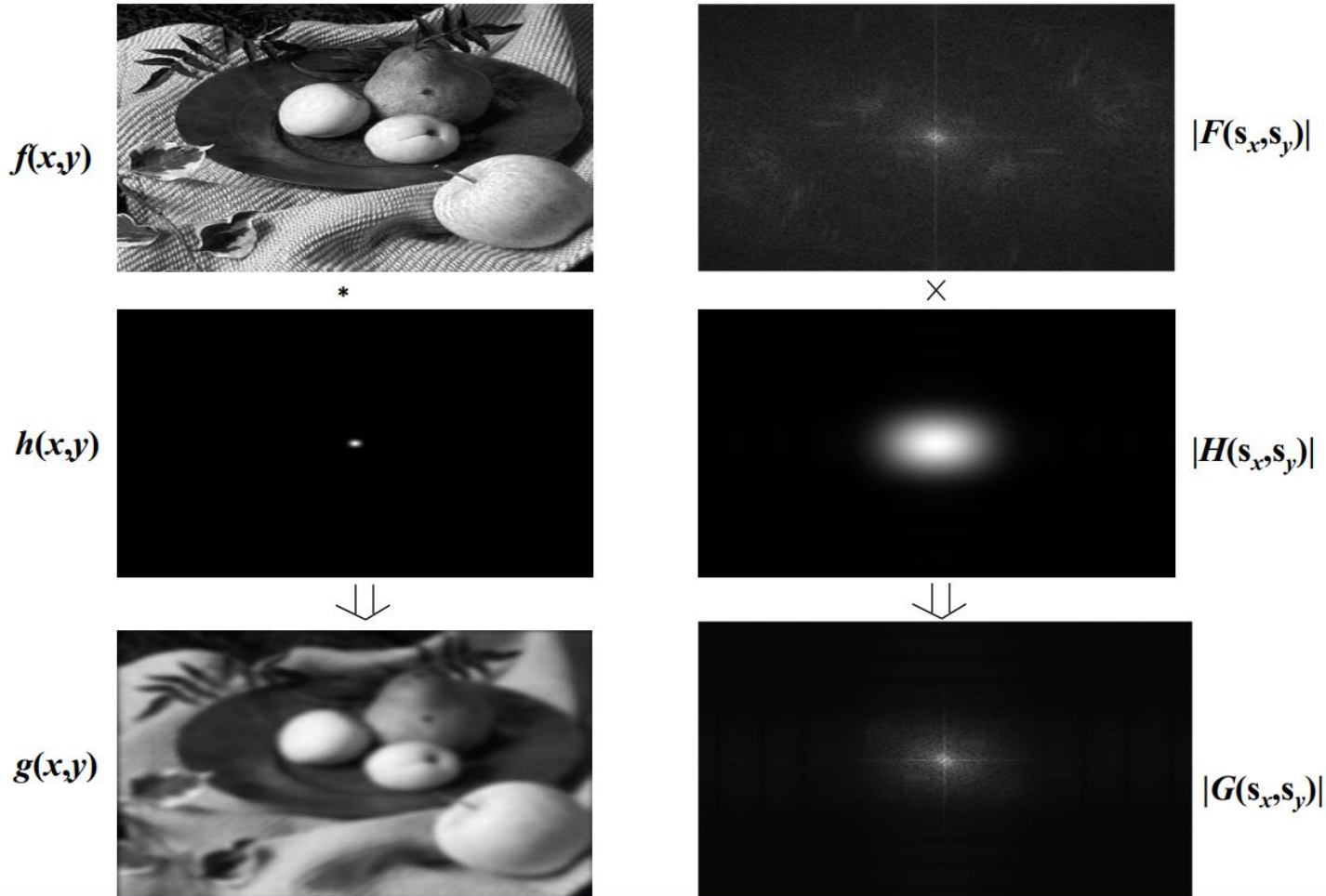
    Mat planes[2];
    split(idftcvt, planes);

    Mat dstImg;
    magnitude(planes[0], planes[1], dstImg);
    normalize(dstImg, dstImg, 255, 0, NORM_MINMAX);
    dstImg.convertTo(dstImg, CV_8UC1);
    // 일반 영상의 type과 표현범위로 변환

    return dstImg;
}
```

# Filtering

## ■ Spatial domain filtering과의 관계

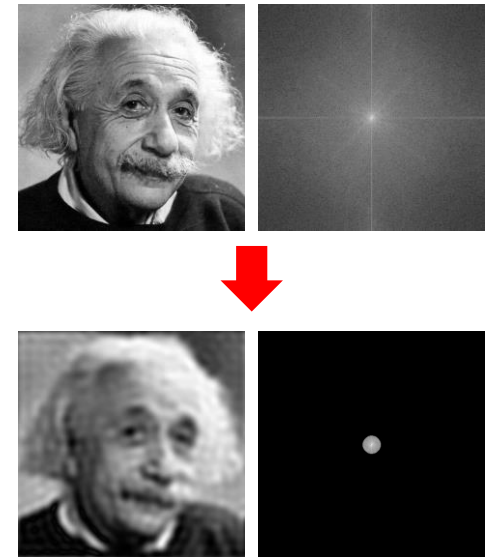


# Filtering

## ■ Low pass filtering (LPF)

- ❑ 주파수 성분에서 고주파 성분을 배제하는 필터링
- ❑ 값이 크게 바뀌는 영역에서 나타나는 고주파를 차단해 영상을 스무딩하는 효과가 있음

```
Mat doLPF(Mat srcImg) {  
    // < DFT >  
    Mat padImg = padding(srcImg);  
    Mat complexImg = doDft(padImg);  
    Mat centerComplexImg = centralize(complexImg);  
    Mat magImg = getMagnitude(centerComplexImg);  
    Mat phaImg = getPhase(centerComplexImg);  
  
    // < LPF >  
    double minVal, maxVal;  
    Point minLoc, maxLoc;  
    minMaxLoc(magImg, &minVal, &maxVal, &minLoc, &maxLoc);  
    normalize(magImg, magImg, 0, 1, NORM_MINMAX);  
  
    Mat maskImg = Mat::zeros(magImg.size(), CV_32F);  
    circle(maskImg, Point(maskImg.cols / 2, maskImg.rows / 2), 20, Scalar::all(1), -1, -1, 0);  
  
    Mat magImg2;  
    multiply(magImg, maskImg, magImg2);  
  
    // < IDFT >  
    normalize(magImg2, magImg2, (float)minVal, (float)maxVal, NORM_MINMAX);  
    Mat complexImg2 = setComplex(magImg2, phaImg);  
    Mat dstImg = doIdft(complexImg2);  
  
    return myNormalize(dstImg);  
}
```

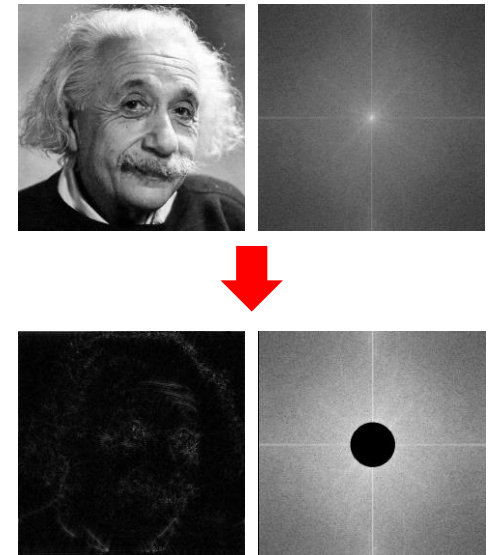


# Filtering

## ■ High pass filtering (HPF)

- 주파수 성분에서 저주파 영역을 배제하는 필터링
- 값이 크게 바뀌는 고주파 영역만을 남기게 되므로 에지를 추출하는데 사용할 수 있음

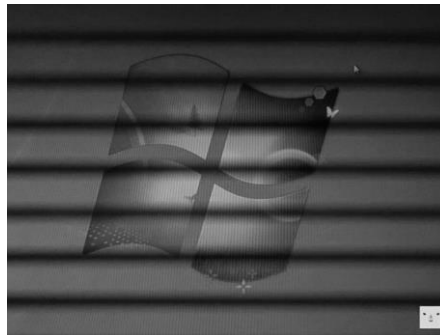
```
Mat doHPF(Mat srcImg) {  
    // < DFT >  
    Mat padImg = padding(srcImg);  
    Mat complexImg = doDft(padImg);  
    Mat centerComplexImg = centralize(complexImg);  
    Mat magImg = getMagnitude(centerComplexImg);  
    Mat phaImg = getPhase(centerComplexImg);  
  
    // < LPF >  
    double minVal, maxVal;  
    Point minLoc, maxLoc;  
    minMaxLoc(magImg, &minVal, &maxVal, &minLoc, &maxLoc);  
    normalize(magImg, magImg, 0, 1, NORM_MINMAX);  
  
    Mat maskImg = Mat::ones(magImg.size(), CV_32F);  
    circle(maskImg, Point(maskImg.cols / 2, maskImg.rows / 2), 50, Scalar::all(0), -1, -1, 0);  
  
    Mat magImg2;  
    multiply(magImg, maskImg, magImg2);  
  
    // < IDFT >  
    normalize(magImg2, magImg2, (float)minVal, (float)maxVal, NORM_MINMAX);  
    Mat complexImg2 = setComplex(magImg2, phaImg);  
    Mat dstImg = doIdft(complexImg2);  
  
    return myNormalize(dstImg);  
}
```



# Homework

## 실습 및 과제

- img1.jpg에 band pass filter를 적용할 것
- Spatial domain, frequency domain 각각에서 sobel filter를 구현하고 img2.jpg에 대해 비교할 것
- img3.jpg에서 나타나는 flickering 현상을 frequency domain filtering을 통해 제거할 것



1. 구현과정과 결과 분석을 반드시 포함할 것
2. 보고서에도 코드와 실험결과 사진을 첨부할 것
3. 반드시 바로 실행가능한 코드(.cpp)를 첨부할 것