INHA UNIVERSITY

ICE4027 디지털영상처리설계

실습 5주차

보고서 작성 서약서

1. 나는 타학생의 보고서를 베끼거나 여러 보고서의 내용을 짜집기하지 않겠습니다.
2. 나는 보고서의 주요 내용을 인터넷사이트 등을 통해 얻지 않겠습니다.
3. 나는 보고서의 내용을 조작하지 않겠습니다.
4. 나는 보고서 작성에 참고한 문헌의 출처를 밝히겠습니다.
5. 나는 나의 보고서를 제출 전에 타학생에게 보여주지 않겠습니다.

나는 보고서 작성시 윤리에 어긋난 행동을 하지 않고 정보통신공학인으로서 나의 명예를 지킬 것을 맹세합니다.

2023년 04월 06일

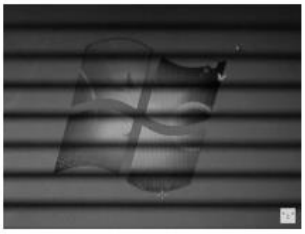
학부 정보통신공학
학년 3
성명 김동한
학번 12191727

1. 개요

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

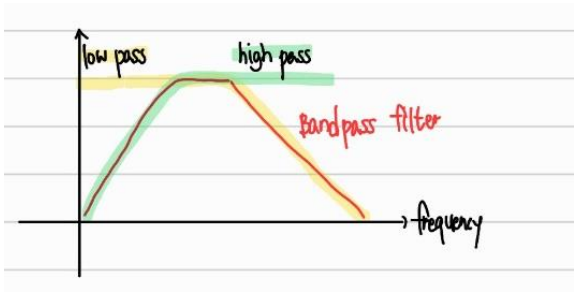


1. 구현과정과 결과 분석을 반드시 포함할 것
2. 보고서에도 코드와 실험결과 사진을 첨부할 것
3. 반드시 바로 실행가능한 코드(.cpp)를 첨부할 것

2. 상세 설계 내용

#1

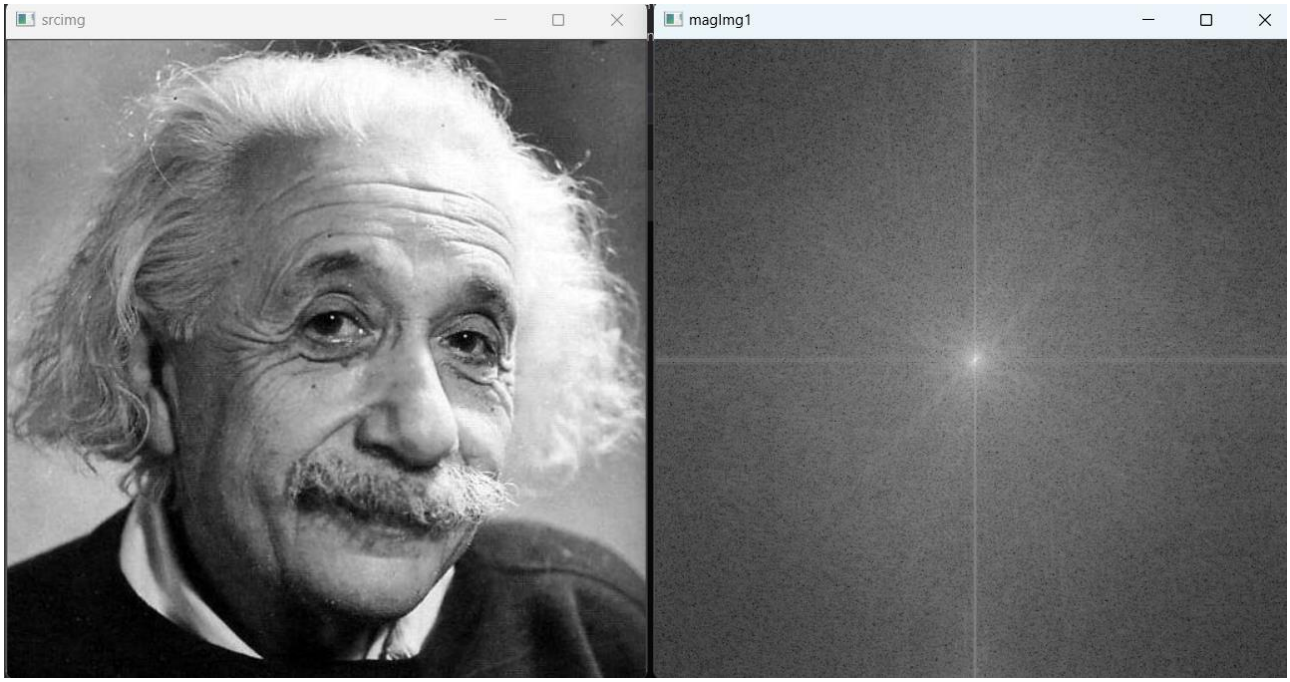
```
Mat doBPF(Mat srcImg) {
    //<DFT>
    Mat padImg = padding(srcImg);
    Mat complexImg = doDft(padImg);
    Mat centerComplexImg = centralize(complexImg);
    Mat magImg = getMagnitude(centerComplexImg);
    Mat phaImg = getPhase(centerComplexImg);
    Mat normImg = myNormalize(magImg);


    //<BPF>
    double minVal, maxVal;
    Point minLoc, maxLoc;
    minMaxLoc(magImg, &minVal, &maxVal, &minLoc, &maxLoc);
    normalize(magImg, magImg, 0, 1, NORM_MINMAX);

    Mat maskImg_band = Mat::zeros(magImg.size(), CV_32F);
    circle(maskImg_band, Point(maskImg_band.cols / 2, maskImg_band.rows / 2), 100, Scalar::all(1), -1, -1, 0); //LPF
    circle(maskImg_band, Point(maskImg_band.cols / 2, maskImg_band.rows / 2), 20, Scalar::all(0), -1, -1, 0); //HPF
    Mat magImg2;

    multiply(maskImg_band, magImg, magImg2);
    imshow("magImg1", magImg);
    imshow("magImg2", magImg2);

    //<IDFT>
    normalize(magImg2, magImg2, (float)minVal, (float)maxVal, NORM_MINMAX);
    Mat complexImg2 = setComplex(magImg2, phaImg);
    Mat dstImg = doIdft(complexImg2);

    return myNormalize(dstImg);

}
```



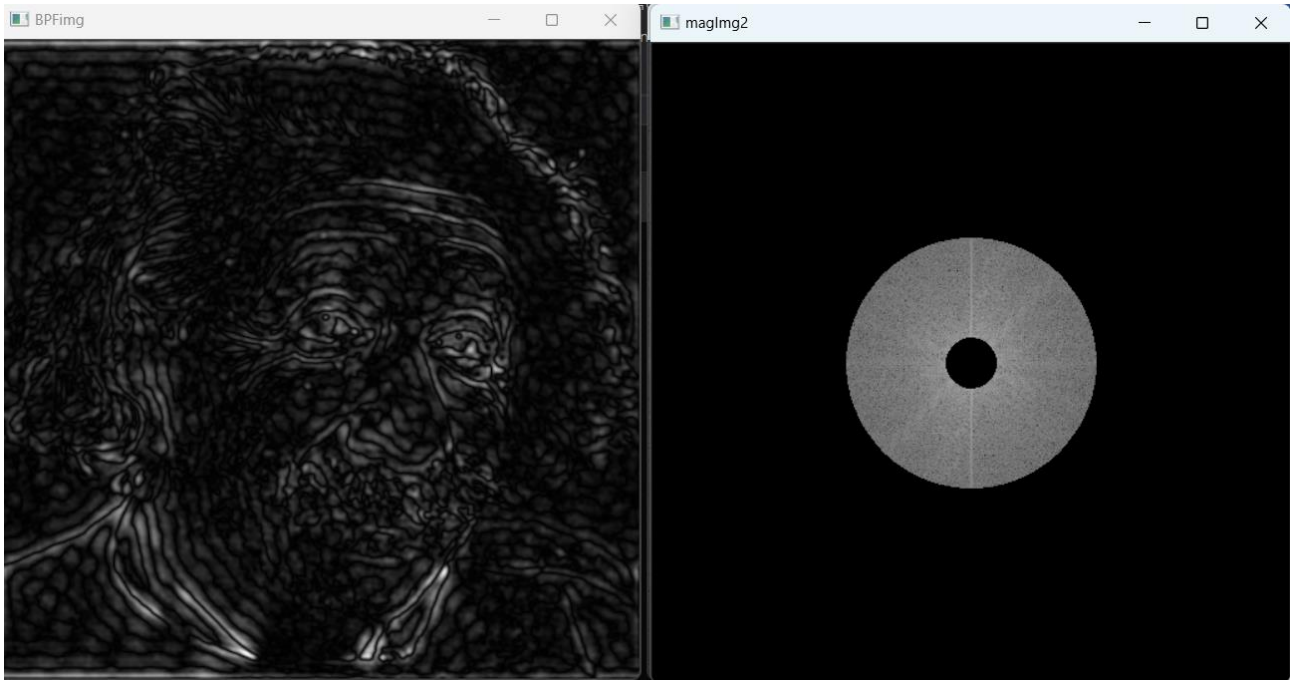
bandpass filter는 저주파수와 고주파수를 차단해야하기 때문에, high passfilter 와 low passfilter를 응용하였다.

```
Mat maskImg_band = Mat::zeros(magImg.size(), CV_32F);
circle(maskImg_band, Point(maskImg_band.cols / 2, maskImg_band.rows / 2), 100, Scalar::all(1), -1, -1, 0); //LPF
circle(maskImg_band, Point(maskImg_band.cols / 2, maskImg_band.rows / 2), 20, Scalar::all(0), -1, -1, 0); //HPF
Mat magImg2;
```

첫번째 circle함수에서는 (maskImg_band.cols/2 , maskImg_band.rows/2) 를 중심으로하는 반지름 100의 원을 그린다. 원안의 모든값을 1로 설정하고 마스크 값을 채우기 위해 '-1'값을 사용한다. 그리고 두번째 circle함수에서 반지름 20인 원으로 고주파를 차단하는 마스크를 만든다. 원안의 모든 값을 0으로 하기 때문에, 반지름 20인 원안은 0이 되고, 반지름 20인 원과 100인 원 사이는 1이 된다. 이로써 밴드패스 이미지를 생성할 수 있었다.



기존 이미지의 magnitude 창이다. 여기서 bandpassfilter를 적용하게되면,



이와 같은 결과를 얻게된다. 밴드패스 필터링으로 고주파수와 저주파수의 정보를 제어했기 때문에, 특정 대역의 정보만을 추출한 것을 알 수 있다. 중간주파수의 대역을 강조해서 영상의 어느정도 엣지가 강조되는 효과를 얻었다.

#2

```
int myKernelConv3x3(uchar* arr, int kernel[][3], int x, int y, int width, int height) {
    int sum = 0;
    int sumKernel = 0;

    // 특정 화소의 모든 이웃화소에 대해 계산하도록 반복문 구성
    for (int j = -1; j <= 1; j++) {
        for (int i = -1; i <= 1; i++) {
            if ((y + j) >= 0 && (y + j) < height && (x + i) >= 0 && (x + i) < width) {
                sum += arr[(y + j) * width + (x + i)] * kernel[i + 1][j + 1];
                sumKernel += kernel[i + 1][j + 1];
            }
        }
    }
    if (sumKernel != 0) { return sum / sumKernel; }// 합이 1로 정규화 되도록 해 영상의 밝기변화 방지
    else { return sum; }
    //color channel indexing 계산식 다르고 채널별 각 수행
}
```

```
//spatial domain에서 sobelfilter구현
Mat mySobelFilter(Mat srcImg, int sel) {
    //horizontal filter
    int kernelX[3][3] = { -1, 0, 1,
                          -2, 0, 2,
                          -1, 0, 1 };
    //vertical filter
    int kernelY[3][3] = { 1, 2, 1,
                          0, 0, 0,
                         -1, -2, -1 };

    // 마스크 합이 000 되므로 1로 정규화하는 과정은 필요 없음
    Mat dstImg(srcImg.size(), CV_8UC1);
    uchar* srcData = srcImg.data;
    uchar* dstData = dstImg.data;
    int width = srcImg.cols;
    int height = srcImg.rows;

    //horizontal filter
    if (sel == 0) {
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                dstData[y * width + x] = abs(myKernelConv3x3(srcData, kernelX, x, y, width, height));
            }
        }
    }
    //vertical filter
    if (sel == 1) {
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                dstData[y * width + x] = abs(myKernelConv3x3(srcData, kernelY, x, y, width, height));
            }
        }
    }
```

```
    if (sel == 2) {
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                dstData[y * width + x] = (abs(myKernelConv3x3(srcData, kernelX, x, y, width, height)) +
                    abs(myKernelConv3x3(srcData, kernelY, x, y, width, height))) / 2;
            }
        }
    }
    return dstImg;
}
```

먼저 spatial domain에서의 sobel filter는 기존에 작성했던 코드를 그대로 사용하면 되기 때문에 따로 구현은 하지않았다. 영상의 수직적인 edge를 살리기 위한 kernelX와 수평적인 edge를 살리기 위한 kernelY로 구성되어있다. 3x3 마스크에 가중치를 부여함으로써 구현했다.

각각 수평성분의 edge, 수직성분의 edge를 살렸다.

그리고 두 결과를 합쳐 sobel filter를 구현했었다.

```
Mat mySobelFilter_freq(Mat srcImg, int sel) {
    //<DFT>
    Mat padImg = padding(srcImg);
    Mat complexImg = doDft(padImg);
    Mat centerComplexImg = centralize(complexImg);
    Mat magImg = getMagnitude(centerComplexImg);
    Mat phaImg = getPhase(centerComplexImg);

    //horizontal filter
    if (sel == 0) {
        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);
        line(maskImg, Point(maskImg.cols / 2,0), Point(maskImg.cols / 2, (maskImg.rows/2) -50), Scalar::all(1),50);
        line(maskImg, Point(maskImg.cols / 2, (maskImg.rows / 2)+50), Point(maskImg.cols / 2, maskImg.rows ), Scalar::all(1),50);

        Mat magImg2;
        multiply(magImg, maskImg, magImg2);
        imshow("magImg2_horizontal", magImg2);

        //<IDFT>
        normalize(magImg2, magImg2, (float)minVal, (float)maxVal, NORM_MINMAX);
        Mat complexImg2 = setComplex(magImg2, phaImg);
        Mat dstImg = doIdft(complexImg2);

        return myNormalize(dstImg);
    }
```

```
    //vertical filter
    if (sel == 1) {
        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);
        line(maskImg, Point(0, maskImg.rows / 2), Point((maskImg.cols / 2) - 70, maskImg.rows / 2), Scalar::all(1), 50);
        line(maskImg, Point((maskImg.cols / 2) + 70, maskImg.rows / 2), Point(maskImg.cols, maskImg.rows / 2), Scalar::all(1), 50);

        Mat magImg2;
        multiply(magImg, maskImg, magImg2);
        imshow("magImg2_vertical", magImg2);

        //<IDFT>
        normalize(magImg2, magImg2, (float)minVal, (float)maxVal, NORM_MINMAX);
        Mat complexImg2 = setComplex(magImg2, phaImg);
        Mat dstImg = doIdft(complexImg2);

        return myNormalize(dstImg);
    }
```

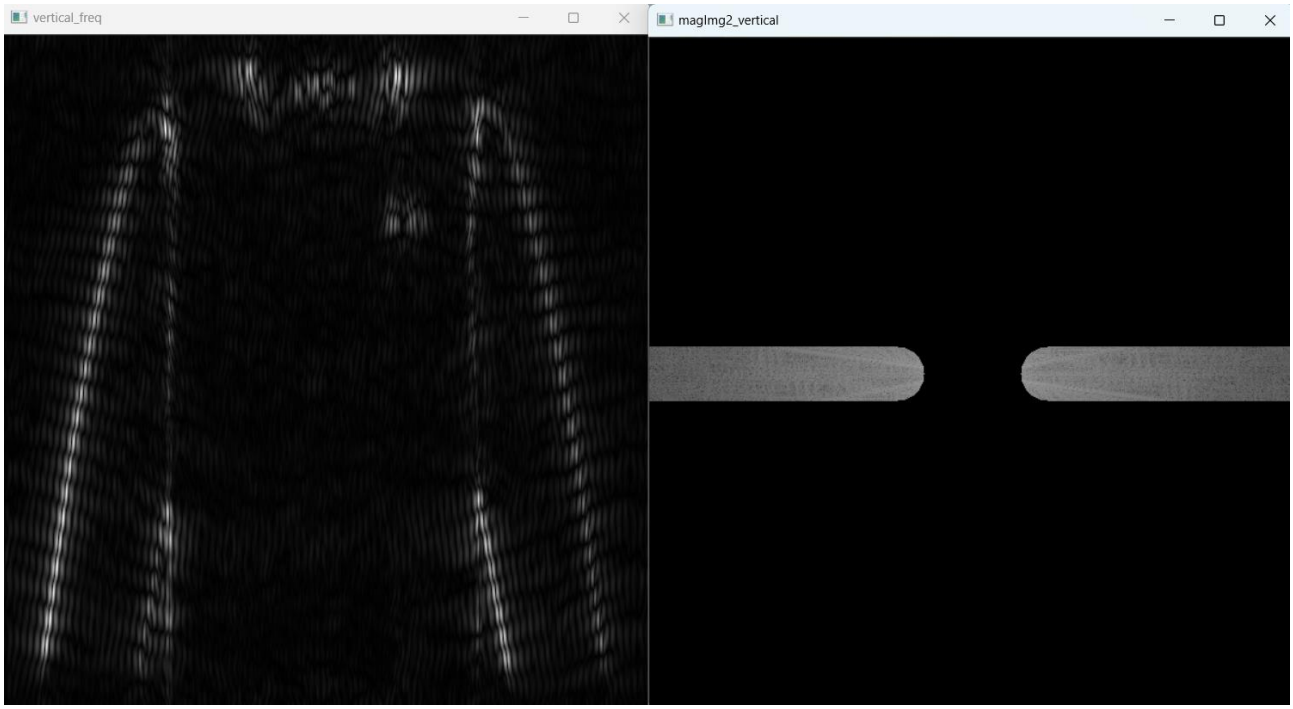
```
    if (sel == 2) {
        double minVal, maxVal;
        Point minLoc, maxLoc;
        minMaxLoc(magImg, &minVal, &maxVal, &minLoc, &maxLoc);
        normalize(magImg, magImg, 0, 1, NORM_MINMAX);

        Mat maskImg = Mat::zeros(magImg.size(), CV_32F);
        line(maskImg, Point(maskImg.cols / 2, 0), Point(maskImg.cols / 2, (maskImg.rows / 2) - 50), Scalar::all(1), 50);
        line(maskImg, Point(maskImg.cols / 2, (maskImg.rows / 2) + 50), Point(maskImg.cols / 2, maskImg.rows), Scalar::all(1), 50);
        line(maskImg, Point(0, maskImg.rows / 2), Point((maskImg.cols / 2) - 70, maskImg.rows / 2), Scalar::all(1), 50);
        line(maskImg, Point((maskImg.cols / 2) + 70, maskImg.rows / 2), Point(maskImg.cols, maskImg.rows / 2), Scalar::all(1), 50);

        Mat magImg2;
        multiply(magImg, maskImg, magImg2);
        imshow("magImg2_sobel", magImg2);

        //<IDFT>
        normalize(magImg2, magImg2, (float)minVal, (float)maxVal, NORM_MINMAX);
        Mat complexImg2 = setComplex(magImg2, phaImg);
        Mat dstImg = doIdft(complexImg2);

        return myNormalize(dstImg);
    }

}
```

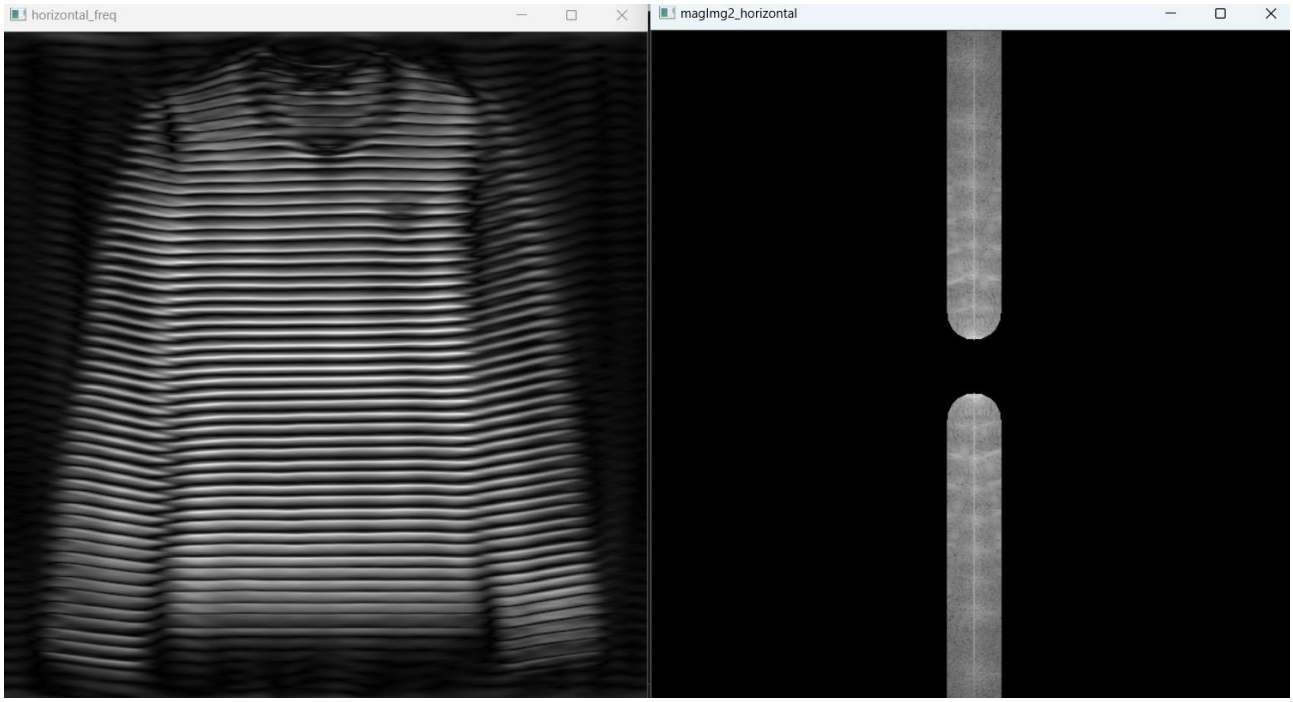
sel을 인자로 받아 0 1 2 를 받을 때 각각 horizontal filter vertical filter sobel filter로 구현했다.horizontal filter의 경우, 중심을 기준으로 위 영역과 밑 영역만 픽셀값을 1로 설정하고 나머지를 모두 0으로 설정

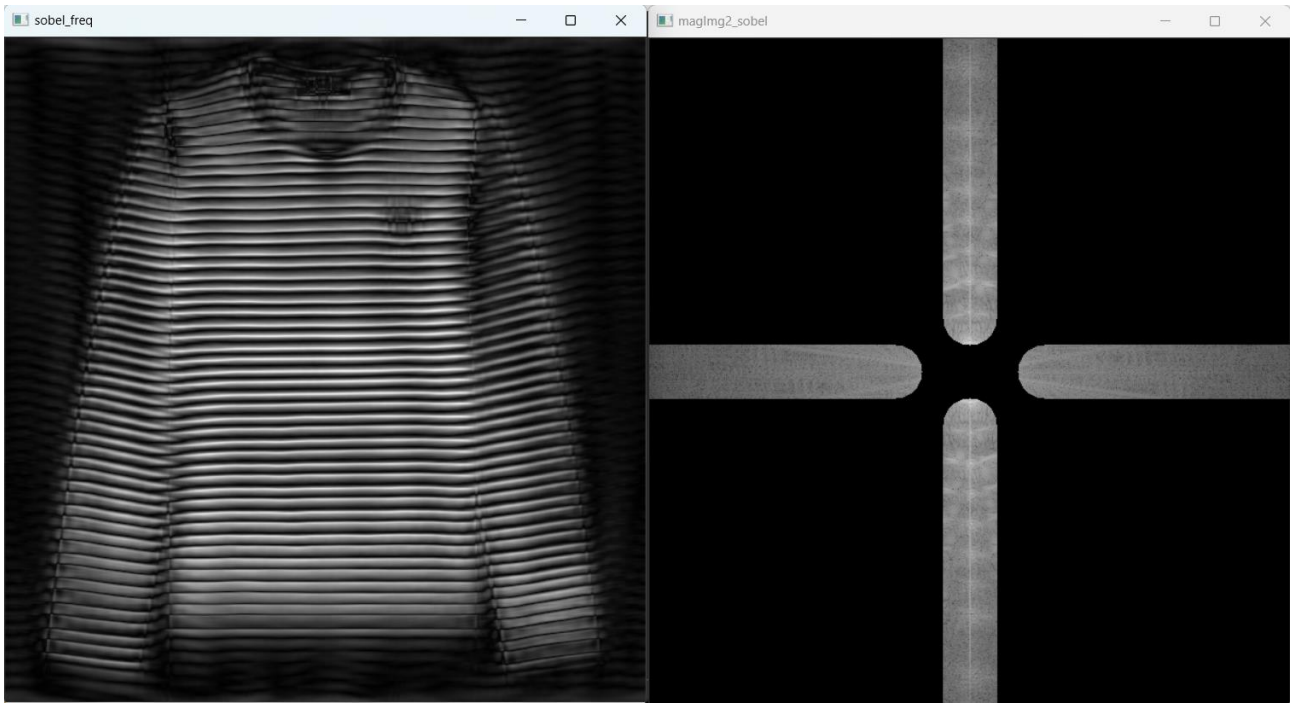
한다. 이 mask를 적용하면 magnitude이미지에서 위 아래의 부분을 제외하고 모두 0으로 처리된다. 결과적으로 수평적인 edge만 살아남게 되고 나머지는 모두 0처리되는것이다. 같은 원리로 vertical은 오른쪽 왼쪽 부분만 1로 나머지는 0으로 처리하며 수직성분의 edge만을 남겨둘 수 있는 것이다. line함수를 이용해서 구현했고,

```
Mat maskImg = Mat::zeros(magImg.size(), CV_32F);
line(maskImg, Point(0, maskImg.rows / 2), Point((maskImg.cols / 2) - 70, maskImg.rows / 2), Scalar::all(1), 50);
line(maskImg, Point((maskImg.cols / 2) + 70, maskImg.rows / 2), Point(maskImg.cols, maskImg.rows / 2), Scalar::all(1), 50);
```

첫 line함수는 중앙에서 좌측으로 이동하는 선을 그린다. (0,maskImg.rows/2)는 line의 시작점을 의미하고, 왼쪽끝이면서 중앙높이임을 의미한다. ((maskImg.cols/2)-70,maskImg.rows/2)는 끝점을 의미하고 중앙에서 좌측으로 70 픽셀 이동한 위치를 의미한다. 따라서 왼쪽 선이 되는것이다. 두께는 50으로 설정했다. 오른쪽 선은 시작부분을 중앙에서 70픽셀 오른쪽으로 이동한 점에서 시작해 오른쪽 중앙끝을 끝점으로 잡아 line함수를 구현했다.







마지막으로 수평성분과 수직성분을 합친것으로 sobel filter를 구현했다.

#3

```
//#3
Mat dstImg2;
srcImg = imread("C:₩₩images₩₩img3week5.jpg", 0);
dstImg = myGaussianFilter(srcImg, 1);
dstImg2 = doFDF(dstImg);
imshow("flickering", dstImg2);
waitKey(0);
```

```
Mat doFDF(Mat srcImg) {
    //<DFT>
    Mat padImg = padding(srcImg);
    Mat complexImg = doDft(padImg);
    Mat centerComplexImg = centralize(complexImg);
    Mat magImg = getMagnitude(centerComplexImg);
    Mat phaImg = getPhase(centerComplexImg);

    //doFDF
    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);
    line(maskImg, Point(maskImg.cols / 2, 0), Point(maskImg.cols / 2, (maskImg.rows / 2) - 15), Scalar::all(0), 25);
    line(maskImg, Point(maskImg.cols / 2, (maskImg.rows / 2) + 15), Point(maskImg.cols / 2, maskImg.rows), Scalar::all(0), 25);


    Mat magImg2;
    multiply(magImg, maskImg, magImg2);
    imshow("magImg2", magImg2);

    //<IDFT>
    normalize(magImg2, magImg2, (float)minVal, (float)maxVal, NORM_MINMAX);
    Mat complexImg2 = setComplex(magImg2, phaImg);
    Mat dstImg = doIdft(complexImg2);

    return myNormalize(dstImg);

}
```

수평방향의 flicker를 지우기 위한 frequency domain filtering이기 때문에, 고주파수를 일단 제거하기 위해서 gaussian filtering과정을 거쳤다. 기존에 구현했던 gaussian filter는 LPF의 역할을 하기 때문에 고주파수를 filtering하는 것이 가능해진다. 그 이후 수평적인 성분을 없애기 위해서 sobel filter에서 한것과 같은 방법으로 fourier transform 한 magnitude영상에서 수직 성분을 제거했다. 수평방향에 흑백선들을 성공적으로 제거할 수 있었다.