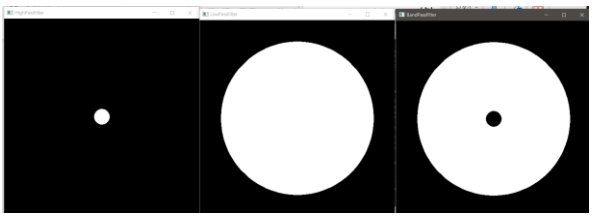
6nd Week Lab Assignment

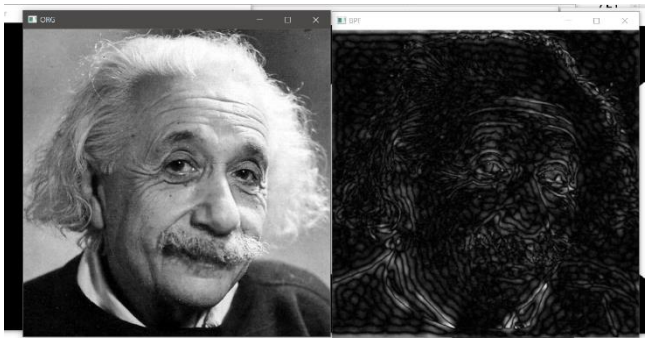
12201928 이상혁

Task 1

- img1.jpg 에 band pass filter 를 적용할 것

```
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);

    //<LPF>
    double minVal, maxVal;
    Point minLoc, maxLoc;
    minMaxLoc(magImg, &minVal, &maxVal, &minLoc, &maxLoc); //min max 범위를 통일하고
    normalize(magImg, magImg, 0, 1, NORM_MINMAX); //이 min max 에 대해서 normalize
    Mat maskImg = Mat::zeros(magImg.size(), CV_32F);
    Mat maskImg2 = Mat::zeros(magImg.size(), CV_32F);
    circle(maskImg, Point(maskImg.cols / 2, maskImg.rows / 2), 200, Scalar::all(1), -1, -1, 0);
    circle(maskImg2, Point(maskImg.cols / 2, maskImg.rows / 2), 20, Scalar::all(1), -1, -1, 0);

    Mat resultImg = maskImg - maskImg2;

    imshow("LowPassFilter", maskImg);
    imshow("HighPassFilter", maskImg2);
    imshow("BandPassFilter", resultImg);

    Mat magImg2;
    multiply(magImg, resultImg, magImg2);

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

    return myNormalize(dstImg);
}
```





이 코드는 주어진 입력 이미지에 대해 주파수 도메인에서의 (Low Pass Filtering),(High Pass Filtering) 으로 Band Pass Filter 를 수행하는 함수다.

먼저 입력 이미지를 주파수 도메인으로 변환하기 위해 이산 푸리에 변환(DFT)을 수행한다. 그 다음 변환된 이미지를 중앙에 배치하고, 이를 통해 주파수 도메인에서의 크기와 위상을 얻습니다.

먼저 크기 이미지를 정규화하고, 최솟값과 최댓값을 이용하여 이미지를 정규화한다. 그 후, 로우 패스 필터의 모양을 결정하기 위해 원 형태의 마스크를 생성한다. 이 때, 중심에서 원 바깥쪽으로 특정 반경까지 1 로 값을 채운다. 하이패스 필터링은 로우패스 필터와 유사하게 작동하지만, 원 내부에 1 을 채우고 원 바깥쪽에 0 을 채워서 반대로 작용하게 된다. 대역통과 필터링은 로우패스 필터와 하이패스 필터의 차이를 구하여 밴드패스 대역을 선택하게 됩니다.

다음으로 주파수 도메인에서의 크기 이미지와 마스크 이미지를 곱하여 필터링된 이미지를 가진다.

마지막으로 (IDFT)를 통해 주파수 도메인에서의 필터링된 이미지를 다시 공간 도메인으로 변환한다.

이러한 처리를 거쳐 만들게 되었다.

Task 2

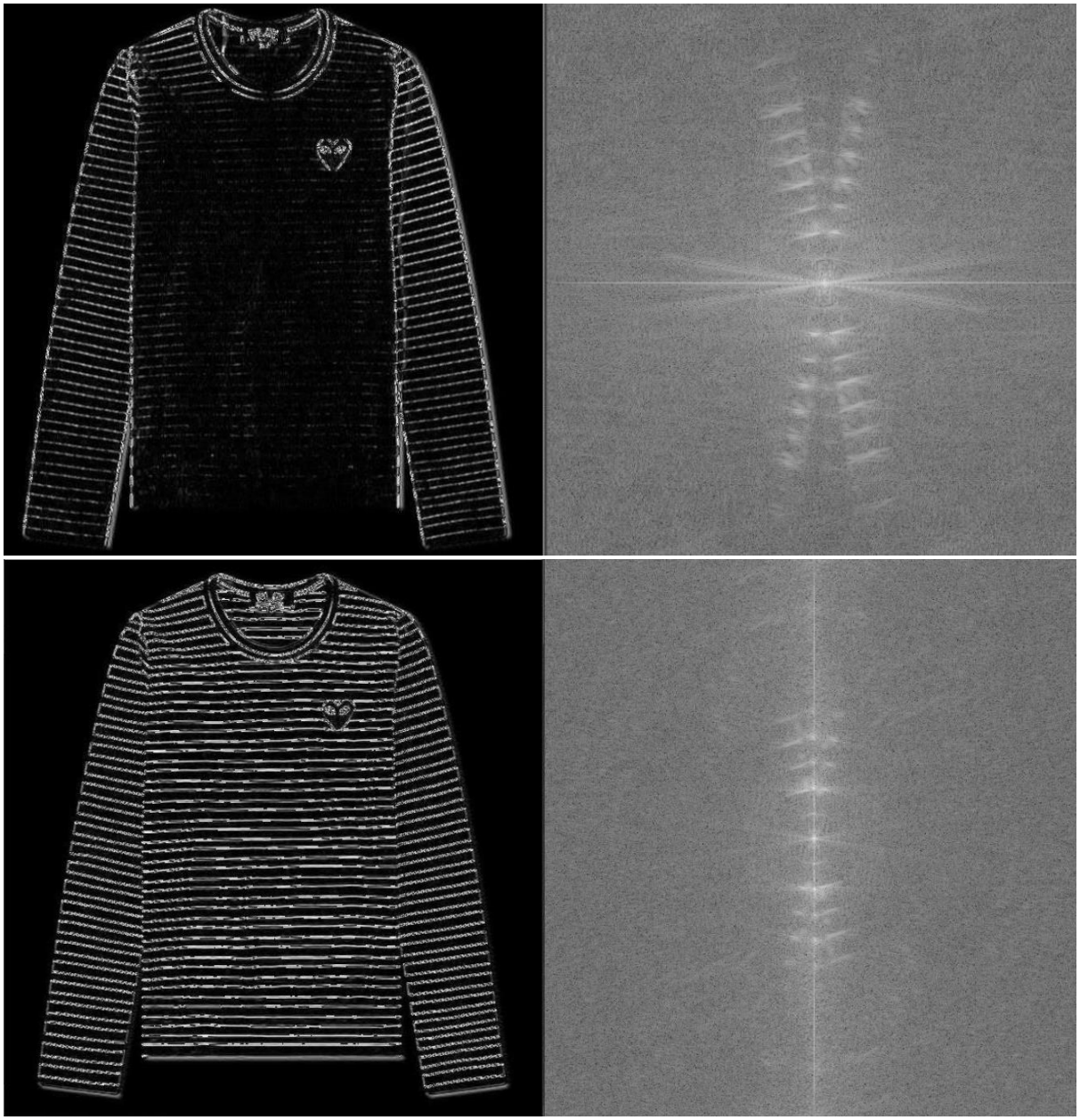
- Spatial domain, frequency domain 각각에서 sobel filter 를 구현하고 img2.jpg 에 대해 비교할 것

```
}
Mat mySobelFilter(Mat srcImg) {
    int kernelX[3][3] = { -1,0,-1,-2,0,2,1,0,1 };
    int kernelY[3][3] = { -1,-2,-1,0,0,0,1,2,1 };

    Mat dstImg(srcImg.size(), CV_8UC1);
    uchar* srcData = srcImg.data;
    uchar* dstData = dstImg.data;
    int width = srcImg.cols;
    int height = srcImg.rows;

    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(srcDa
        }
    }
    return dstImg;
}

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

    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; }
    else return sum;
```

```
Mat freqeunSober(Mat srcImg) {
    Mat padImg = padding(srcImg);
    Mat complexImg = doDft(padImg);
    Mat centerComplexImg = centralize(complexImg);
    Mat magImg = getMagnitude(centerComplexImg);
    Mat phaImg = getPhase(centerComplexImg);

    double minVal, maxVal;
    Point minLoc, maxLoc;
    minMaxLoc(magImg, &minVal, &maxVal, &minLoc, &maxLoc); //min max 범위를 통일하고
    normalize(magImg, magImg, 0, 1, NORM_MINMAX); //이 min max 에 대해서 normalizedd

    Mat maskImg = Mat::zeros(magImg.size(), CV_32F);
    int lineHeight = magImg.rows;
    int lineWidth = 400;

    maskImg(Rect(magImg.cols / 2 - lineWidth / 2, magImg.rows / 2 - lineHeight / 2, lineWidth, lineHeight)) = 1;

    int lineThickness = 400;
    int lineWidth1 = magImg.cols;


    maskImg(Rect(magImg.cols / 2 - lineWidth1 / 2, magImg.rows / 2 - lineThickness / 2, lineWidth1, lineThickness)) = 1;
    circle(maskImg, Point(maskImg.cols / 2, maskImg.rows / 2), 50, Scalar::all(0), -1, -1, 0);
    imshow("Mask222", maskImg);



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

    // 결과 이미지 출력
    imshow("Result Image", magImg2);
    //<IDFT>
    normalize(magImg2, magImg2, (float)minVal, (float)maxVal, NORM_MINMAX);
    Mat complexImg2 = setComplex(magImg2, phaImg);
    Mat dstImg = doIdft(complexImg2);

    return myNormalize(dstImg);
}
```

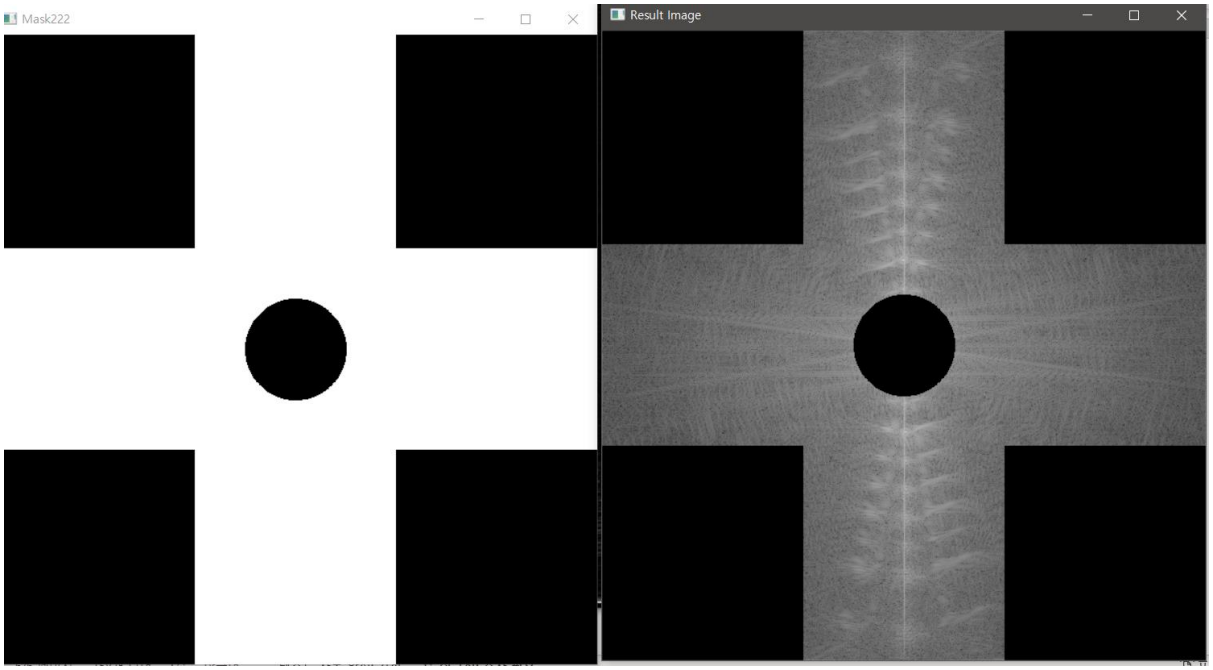
Spatial domain 에서의 코드는 저번 실습시간에 사용하였던 코드를 사용하였다.

주파수 도메인에서의 소벨필터를 만들기 위해 최대한 같은 부분을 만들도록 노력하였다.



Spatial domain 에서 x 축,y 축 각각만 소벨필터를 적용한 모습은 왼쪽과 같다. 이를 주파수 공간모양으로 변형시켜 보았더니 오른쪽 사진과 같은 모양새를 보였다.

따라서 원 사진에서 주파수 공간모양부분에서 오른쪽 사진으로 최대한 비슷하게 변형할 수 있도록 마스크를 제작하도록 하였다.



이렇게 마스크를 제작한 이후에 이걸 사진에 적용시켰다.

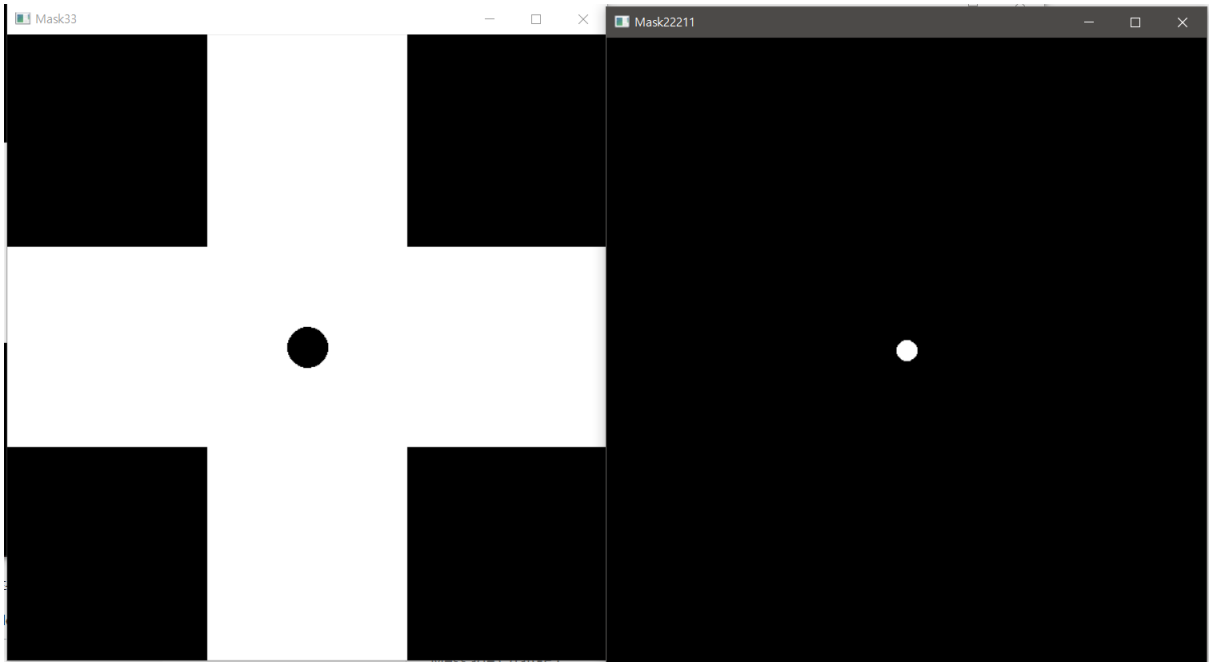
왼쪽이 주파수 공간에서 진행한 결과이고 오른쪽이 소벨필터를 적용한 것이다.

여러가지 테스트를 많이 진행한 결과이다. 실제 소벨필터를 적용한것보다 많이 흐린 결과가 표현되었다. 아무래도 선이 가진 저주파만을 찾는점이 매우 힘들었기 때문이다.

실제 소벨필터는 xy 축으로 선을 가지고 있는것만은 확실하게 디텍하기 때문인것같다.

주파수 도메인에서 이를 찾고 하나씩 마스킹해주며 해야했기에 이러한 오차가 발생했다고 생각한다.

주파수 공간에서 마스크를 만든 방법은 아래와 같다.



이 두 마스크를 뺀것이다.

여러가지 실험을 통해 이 결과가 제일 만족스러움을 확인하였다.

결과적으로 이러한 주파수 공간대를 가진 이미지가 탄생하였다!

전체적으로 코드를 어떻게 작성하였는지 적어보겠다.

앞선 DFT 와 정규화, 크기, 위상을 보내는 방법은 동일하다.

다른 점은 마스크 부분이다.

주파수 도메인에서의 소벨 필터링을 위해 마스크 이미지를 하였는데 여기서는 수평 및 수직 방향으로 강도가 감소하는 직선 모양의 마스크를 사용한다. 이러한 마스크를 생성하고 저주파 부분의 DC 부분을 약화시킬 마스크를 제작해 뺄셈을 진행해주었다.

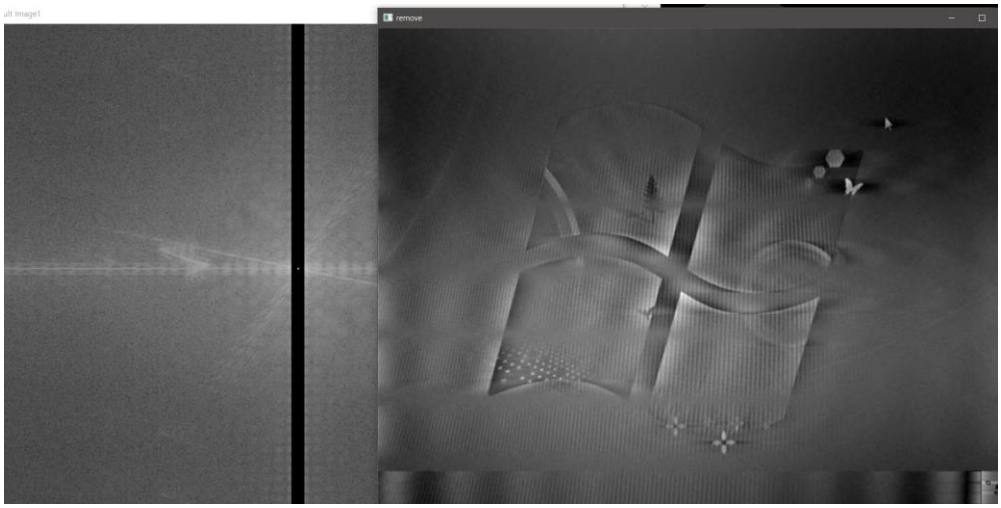
주파수 도메인에서의 크기 이미지와 소벨 필터 마스크 이미지를 곱하여 필터링된 이미지를 얻는다.

필터링된 이미지를 시각화하여 결과를 확인하는 과정을 여러 번 거쳐 최적의 마스크를 찾아 작성하게 되었다.

Task 3

img3.jpg 에서 나타나는 flickering 현상을 frequency domain filtering 을통해제거할 것

```
}
Mat removeHorizontalLines(Mat srcImg) {
    Mat padImg = padding(srcImg);
    Mat complexImg = doDft(padImg);
    Mat centerComplexImg = centralize(complexImg);
    Mat magImg = getMagnitude(centerComplexImg);
    Mat phaImg = getPhase(centerComplexImg);

    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);

    int lineHeight = magImg.rows;
    int lineWidth = 20;


    maskImg(Rect(magImg.cols / 2 - lineWidth / 2, magImg.rows / 2 - lineHeight / 2, lineWidth, lineHeight)) = 0;

    circle(maskImg, Point(maskImg.cols / 2, maskImg.rows / 2), 1, Scalar::all(1), -1, -1, 0);
    imshow("Mask222", maskImg);

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

    // 결과 이미지 출력
    imshow("Result Image", magImg2);
    //<IDFT>
    normalize(magImg2, magImg2, (float)minVal, (float)maxVal, NORM_MINMAX);
    Mat complexImg2 = setComplex(magImg2, phaImg);
    Mat dstImg = doIdft(complexImg2);

    return myNormalize(dstImg);
}
```



우선 수평으로 필커링 현상이 반복됨을 확인하였다.

수평선을 제거하기 위한 마스크 이미지를 생성하였다. 여기서는 수평선의 위치를 결정하기 위해 직사각형 형태의 마스크를 생성하고, 해당 영역을 0 으로 설정하여 제거하려 하였다. 마스크의 중심에는 원을 그려 원 내부를 1 로 설정하여 약간의 저주파 부분을 살려 조금 더 확실한 효과를 얻고자 하였다.

주파수 도메인에서의 크기 이미지와 마스크 이미지를 곱하여 필터링된 이미지를 얻은 후. 필터링된 이미지를 시각화하여 결과를 확인하였다.

이렇게 하니 가로줄 효과가 사라지는 모습을 보였다.

고찰 : 주파수 도메인에서의 영상들을 다루어 보면서 조금 더 한발자국 다가갈 수 있었다. 실제 도메인과 주파수 도메인에서의 차이점에 대해 인식하고 영상의 패턴에 따라 주파수 도메인에서의 차이점을 명확하게 인식할 수 있었다.