5nd Week Lab Assignment

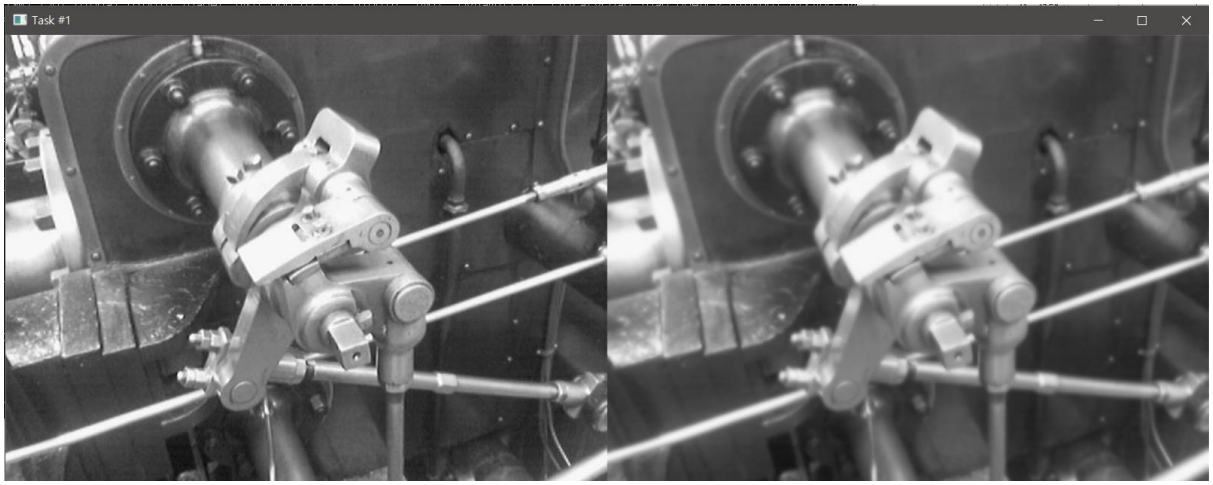
12201928 이상혁

Task 1

▪ 9x9 Gaussian filter를 구현하고 결과를 확인할 것, 히스토그램이 어떻게 변하는지 확인할 것

```
int myKernelConv9x9(uchar* arr, int kernel[][9], int x, int y, int width, int height) {
    int sum = 0;
    int sumKernel = 0;
    for (int j = -4; j <= 4; j++) {
        for (int i = -4; i <= 4; i++) {
            if ((y + j) >= 0 && (y + j) < height && (x + i) >= 0 && (x + i) < width) {
                sum += arr[(y + j) * width + (x + i)] * kernel[i + 4][j + 4];
                sumKernel += kernel[i + 4][j + 4];
            }
        }
    }
    if (sumKernel != 0) { return sum / sumKernel; }
    else { return sum; }
}
Mat myGaussianFilter(Mat srcImg) {
    int width = srcImg.cols;
    int height = srcImg.rows;
    int kernel[9][9] = { 0,1,1,2,2,2,1,1,0,
                         1,2,4,5,5,5,4,2,1,
                         1,4,5,3,0,3,5,4,1,
                         2,5,3,12,24,12,3,5,2,
                         2,5,0,24,40,24,0,5,2,
                         2,5,3,12,24,12,3,5,2,
                         1,4,5,3,0,3,5,4,1,
                         1,2,4,5,5,5,4,2,1,
                         0,1,1,2,2,2,1,1,0
    };
    Mat dstImg(srcImg.size(), CV_8UC1);
    uchar* srcData = srcImg.data;
    uchar* dstData = dstImg.data;
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            dstData[y * width + x] = myKernelConv9x9(srcData, kernel, x, y, width, height
        }
    }
    return dstImg;
}
```

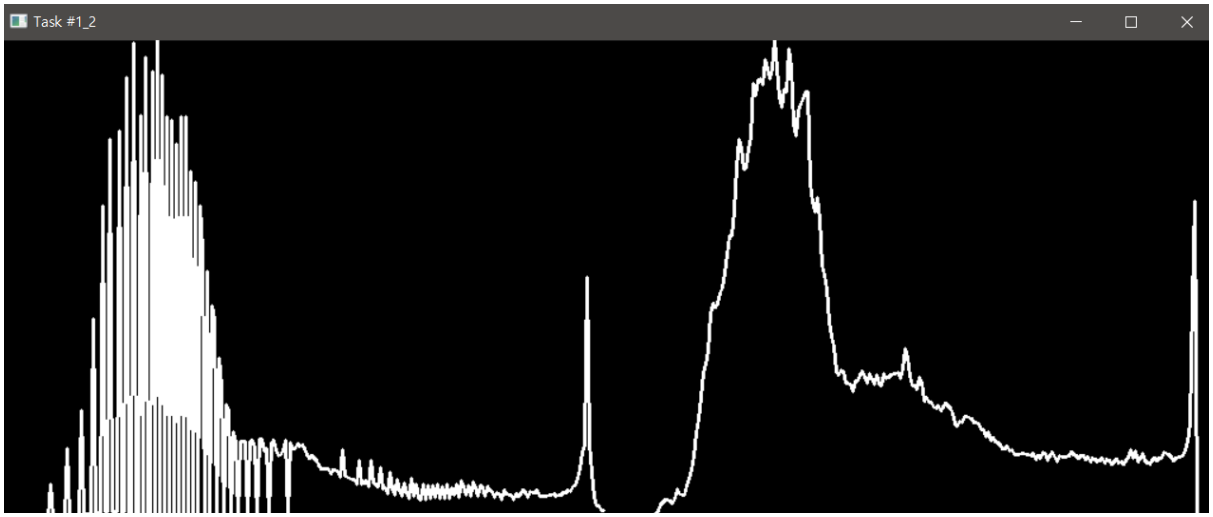
3x3에서 -1 ~ 1의 범위 였던 값들을 모두 필터사이즈인 9/2인 4로 변경해주었다. 이후 앞서 gaussian을 통하여 구한 9x9의 커널값을 근사시킨 것을 이용해 cov해 gaussian 필터를 적용하였다.



위 사진처럼 가우시안 필터를 적용한 뒤에는 smoothing 해지는 효과를 볼 수 있다.

```
Mat GetHistogram(Mat src) {
    Mat histogram;
    const int* channel_numbers = { 0 };
    float channel_range[] = { 0.0, 255.0 };
    const float* channel_ranges = channel_range;
    int number_bins = 255;

    calcHist(&src, 1, channel_numbers, Mat(), histogram, 1, &number_bins, &channel_ranges);

    int hist_w = 512;
    int hist_h = 400;
    int bin_w = cvRound((double)hist_w / number_bins);

    Mat histImage(hist_h, hist_w, CV_8UC1, Scalar(0, 0, 0));
    normalize(histogram, histogram, 0, histImage.rows, NORM_MINMAX, -1, Mat());

    for (int i = 1; i < number_bins; i++) {
        line(histImage, Point(bin_w * (i - 1), hist_h - cvRound(histogram.at<float>(i - 1))),
            Point(bin_w * (i), hist_h - cvRound(histogram.at<float>(i))),
            Scalar(255, 0, 0), 2, 8, 0);
    }

    return histImage;
}
```

사진에 나와 있는 함수로 히스토그램을 생성하였다.



히스토그램을 통해 분포를 확인해보니 원본이미지인 좌측에서는 히스토그램이 튀거나 끊기는 부분이 보인다.

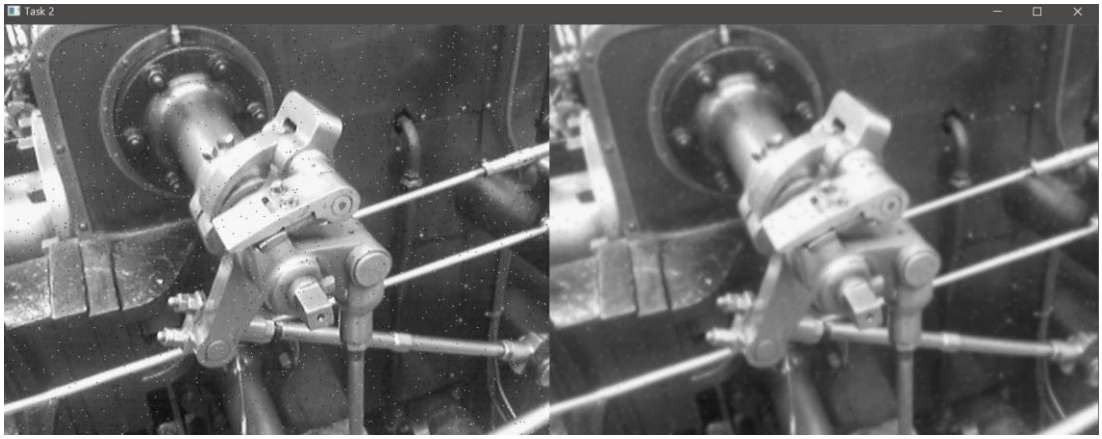
이를 가우시안 필터를 적용하여 보니 약간의 평탄화 작업뿐만아니라 주변 인접한 픽셀 값으로 정규화 되기 때문에 매끄러운 모습을 확인할 수 있다. 블러 처리가 되었기 때문에 discrete 하지 않고 continuous 된 것을 볼 수 있다.

Task 2

영상에 Salt and pepper noise를 주고, 구현한 9x9 Gaussian filter를 적용해볼 것

```
Mat SpreadSalts_pepers(Mat img, int num) {
    Mat dst_img = img;
    for (int n = 0; n < num; n++) {
        int x = rand() % dst_img.cols;
        int y = rand() % dst_img.rows;
        img.at<uchar>(y, x) = 255;
    }
    for (int n = 0; n < num; n++) {
        int x = rand() % dst_img.cols;
        int y = rand() % dst_img.rows;
        img.at<uchar>(y, x) = 0;
    }
    return dst_img;
}
```

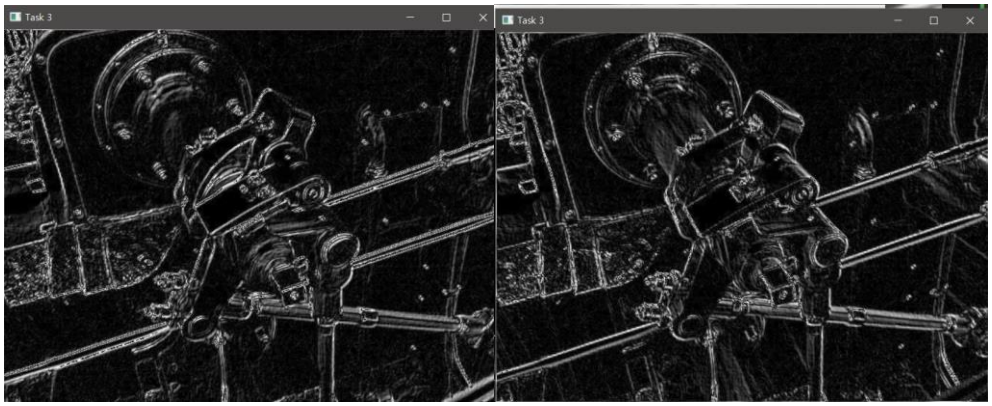
Salt와 pepper를 spread하는 함수는 다음과 같이 제작하였다. 픽셀 값 접근을 사용하여 랜덤으로 점을 검은색, 흰색으로 찍히도록 하였다. 매개변수로 1000을 전달하여 1000개의 점들이 찍히도록 설정하였다.



위 사진처럼 salt와 pepper가 제거되는 모습을 확인할 수 있다. 약간의 희미한 모습이 남아있긴 하지만 고주파인 노이즈들이 필터를 통하여 제거하였기에 이러한 결과가 나타남을 예측할 수 있다.

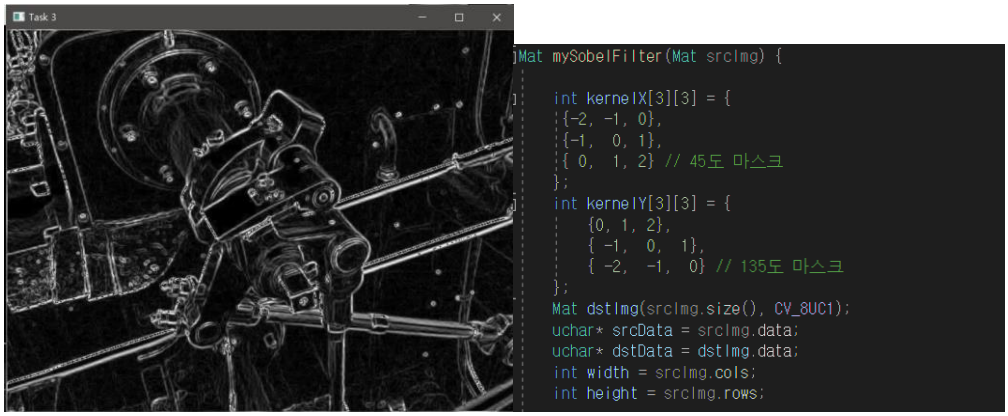
Task 3

§ 45 도와 135 도의 대각 에지를 검출하는 Sobel filter 를 구현하고 결과를 확인할 것



좌측 사진이 45 도의 대각 에지만을 검출하였을때의 결과이고, 오른쪽은 135 도의 대각 에지를 검출 하였을때의 결과이다.

둘을 합친 결과는 아래와 같다.



```
for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
        int gx = abs(myKernelConv3x3(srcData, kernelX, x, y, width, height));
        int gy = abs(myKernelConv3x3(srcData, kernelY, x, y, width, height));
        dstData[y * width + x] = (gx + gy) / 2;
    }
}
//for (int y = 0; y < height; y++) {    ////45도
//    for (int x = 0; x < width; x++) {
//        int gx = abs(myKernelConv3x3(srcData, kernelX, x, y, width, height));
//        int gy = abs(myKernelConv3x3(srcData, kernelY, x, y, width, height));
//        dstData[y * width + x] = gx;
//    }
//}
//for (int y = 0; y < height; y++) {    ////135도
//    for (int x = 0; x < width; x++) {
//        int gx = abs(myKernelConv3x3(srcData, kernelX, x, y, width, height));
//        int gy = abs(myKernelConv3x3(srcData, kernelY, x, y, width, height));
//        dstData[y * width + x] = gy;
//    }
//}
return dstImg;
```

먼저 각도에 따른 마스크를 설정해준다음 아래에 이와 같이 각각 구할 수 있는 코드와, 둘을 합치고 평균을 낸 버전으로 코드를 작성하고 결과를 도출하였다. 엣지가 각도에 맞게 잘 검출됨을 확인할 수 있다.

Task 4

§ 컬러영상에 대한 Gaussian pyramid 를 구축하고 결과를 확인할 것

```
for (int j = -4; j <= 4; j++) {
    for (int i = -1; i <= 1; i++) {
        if ((y + j) >= 0 && (y + j) < height && (x + i) >= 0 && (x + i) < width) {
            //영상 가장자리에서 영상 밖의 화소를 읽지 않도록 하는 조건문
            sum += arr[(y + j) * width * 3 + (x + i) * 3 + color] * kernel[i + 4][j +
            sumKernel += kernel[i + 4][j + 4];
        }
    }
}
if (sumKernel != 0) {
```

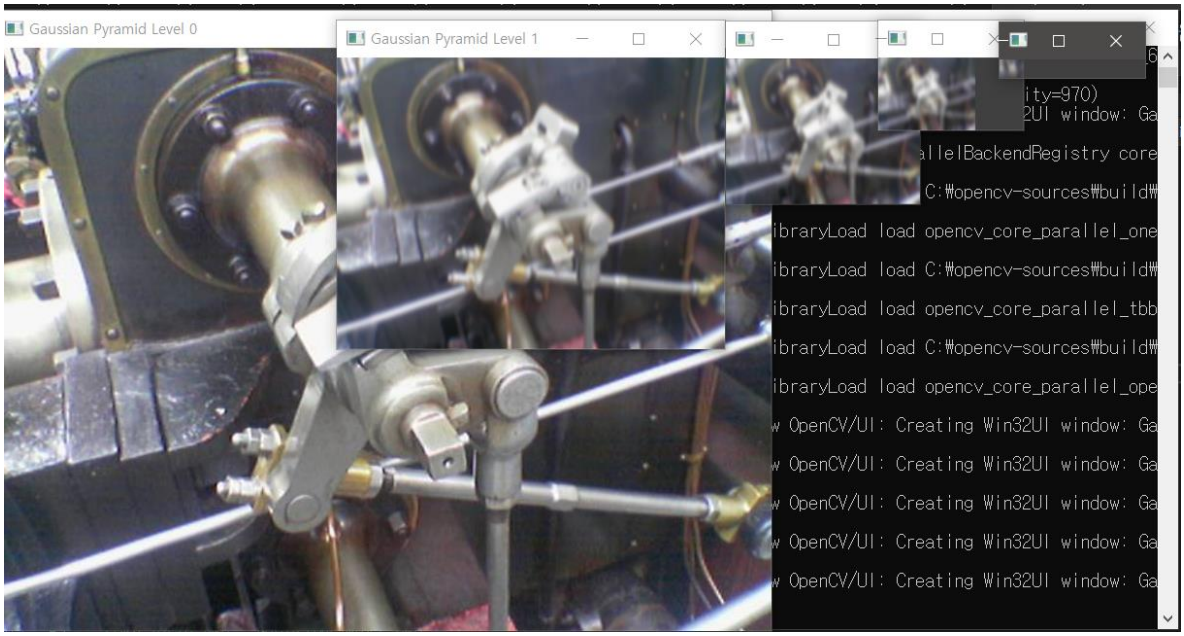
```
for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
        dstData[y * width * 3 + x * 3] = myKernelConv9x9_Color(srcData, kernel, x, y, width, height, 0);
        dstData[y * width * 3 + x * 3 + 1] = myKernelConv9x9_Color(srcData, kernel, x, y, width, height, 1);
        dstData[y * width * 3 + x * 3 + 2] = myKernelConv9x9_Color(srcData, kernel, x, y, width, height, 2);
    }
}
return dstImg;
```

```
Mat mySampling(Mat srcImg) {
    int width = srcImg.cols / 2;
    int height = srcImg.rows / 2;
    Mat dstImg(height, width, CV_8UC3);

    uchar* srcData = srcImg.data;
    uchar* dstData = dstImg.data;

    for (int y = 0; y < height; y++)
    {
        for (int x = 0; x < width; x++)
        {
            dstData[y * width * 3 + x * 3] = srcData[(y * 2) * (width * 2) * 3 + (x * 2) * 3];
            dstData[y * width * 3 + x * 3 + 1] = srcData[(y * 2) * (width * 2) * 3 + (x * 2) * 3 + 1];
            dstData[y * width * 3 + x * 3 + 2] = srcData[(y * 2) * (width * 2) * 3 + (x * 2) * 3 + 2];
        }
    }
    return dstImg;
}
```

컬러인 3 채널에 접근하기 위해서 가우시안 필터와 conv 함수를 픽셀 값, 채널값에 맞게 조정해주었다. 그후 영상을 다운샘플링하는 함수를 제작해 주었다.

```
vector<Mat> myGaussianPyramid(Mat srcImg, int levels) {
    vector<Mat> pyramid;
    pyramid.push_back(srcImg);

    for (int i = 0; i < levels; i++) {
        srcImg = mySampling(srcImg);
        srcImg = myGaussianFilter_Color(srcImg);
        pyramid.push_back(srcImg);
    }

    return pyramid;
}
void exGaussianPyramid(Mat srcImg) {
    vector<Mat> gaussianPyramid = myGaussianPyramid(srcImg, 5);

    // Display the pyramid images
    for (size_t i = 0; i < gaussianPyramid.size(); ++i) {
        imshow("Gaussian Pyramid Level " + to_string(i), gaussianPyramid[i]);
    }
}
```

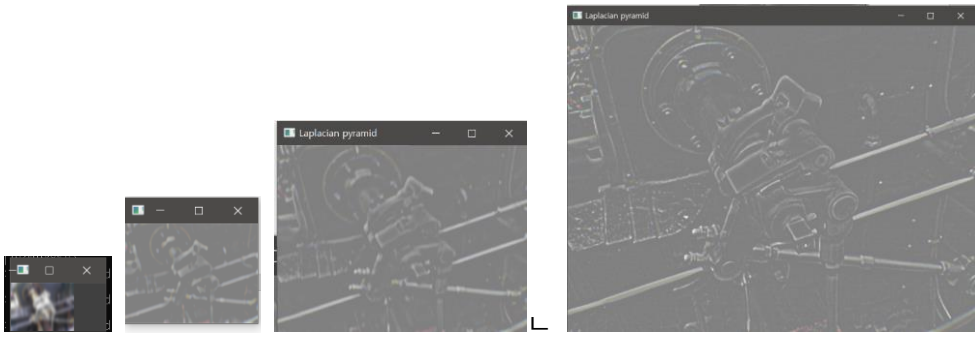
그 후 이를 한번에 보기 위해 vector를 사용했고 5레벨 까지 확인해보기로 하였다.



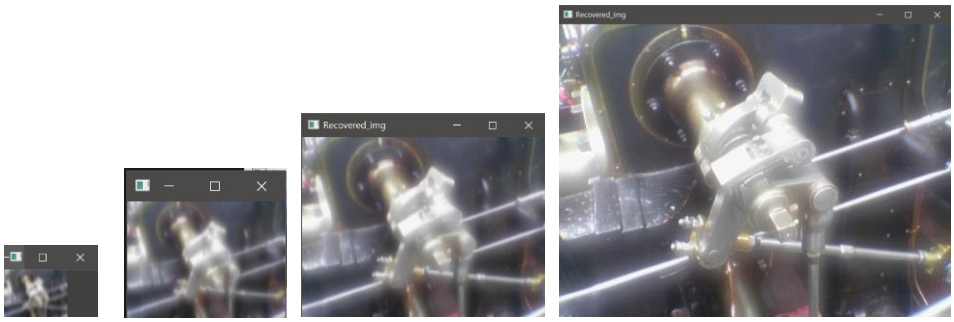
Task 5

컬러영상에 대한 Laplacian pyramid 를 구축하고 복원을 수행한 결과를 확인할 것

```
vector<Mat> myLaplacianPyramid(Mat srcImg, int levels) {
    vector<Mat> pyramid;
    pyramid.push_back(srcImg);
    Mat FirstImg = srcImg;
    for (int i = 0; i < levels; i++) {
        if (i != levels - 1) {
            srcImg = mySampling(srcImg);
            srcImg = myGaussianFilter_Color(srcImg);
            Mat lowImg = srcImg;
            resize(lowImg, lowImg, FirstImg.size());
            pyramid.push_back(FirstImg - lowImg + 128);
        }
        else
            pyramid.push_back(srcImg);
    }

    return pyramid;
}
```



라플라스 피라미드 이미지



복원한 이미지

고찰 : 피라미드를 구현하는 것이 매우 힘들었다. Opencv 내의 함수를 사용할 수 없어 직접 샘플링을 구현하고 적용하는 과정에서 많은 구글링과 시간을 들였다. 가우시안 피라미드와 라플라스 피라미드를 이해할 수 있었다.