INHA UNIVERSITY

ICE4027 디지털영상처리설계

실습 3주차

보고서 작성 서약서

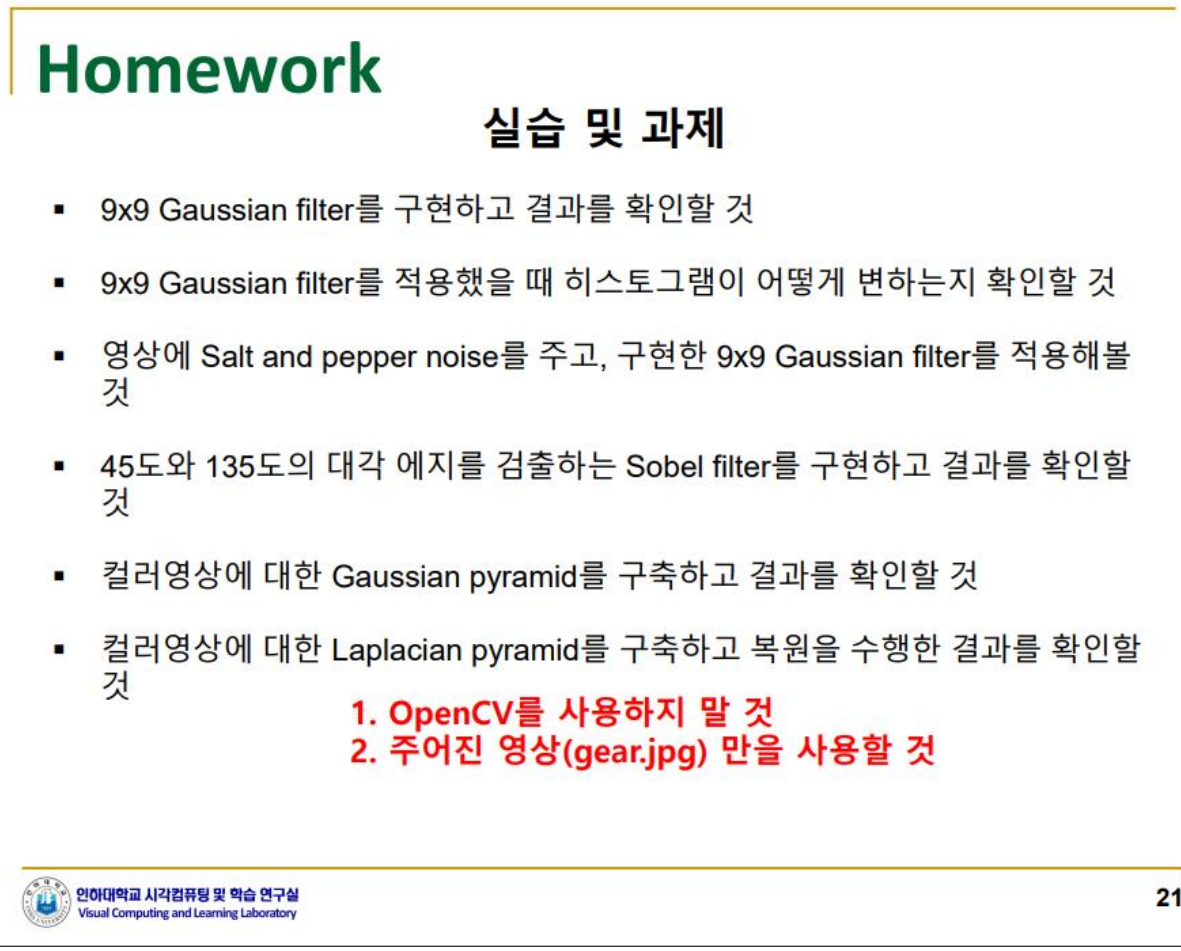
1. 나는 타학생의 보고서를 베끼거나 여러 보고서의 내용을 짜집기하지 않겠습니다.
2. 나는 보고서의 주요 내용을 인터넷사이트 등을 통해 얻지 않겠습니다.
3. 나는 보고서의 내용을 조작하지 않겠습니다.
4. 나는 보고서 작성에 참고한 문헌의 출처를 밝히겠습니다.
5. 나는 나의 보고서를 제출 전에 타학생에게 보여주지 않겠습니다.

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

2023년 03월 22일

학부 정보통신공학
학년 3
성명 김동한
학번 12191727

1. 개요



과제 1,2번은 실습시간에 3x3 Gaussian filter에 대해 해보았기 때문에, 이를 응용해서 9x9 Gaussian filter를 구현하였다.

과제 3번은 이전주차 실습에서 사용한 SpreadSalt함수를 이용해서 흰색점들을 source image에 뿌리고, 1번에서 구현한 Gaussian filter를 적용했다.

과제 4번은 실습시간에 수평,수직의 에지를 검출하는 sobel filter를 구현했었는데 이 코드에서 kernel부분을 수정하여 45도, 135도(90+45도) 의 대각 에지를 검출하는 sobel filter를 구현하였다.

과제 5번과 6번은 기존 실습시간에서는 1개의 채널만 사용할경우 즉 gray scale의 image인 경우에서의 Gaussian pyramid와 Laplacian pyramid였는데, 컬러영상을 위해서 3개의 채널을 사용하는것으로 코드를 수정해 구현했다.

2. 상세 설계 내용

Histogram을 나타낼 Mat타입 histImage를 반환하는 Gethistogram 함수

```
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;
}
```

먼저 이전주차에서도 사용했던 Gethistogram 함수이다. gray scale의 이미지에서 화소의 분포를 눈으로 볼 수 있게끔 해주는 함수이다.

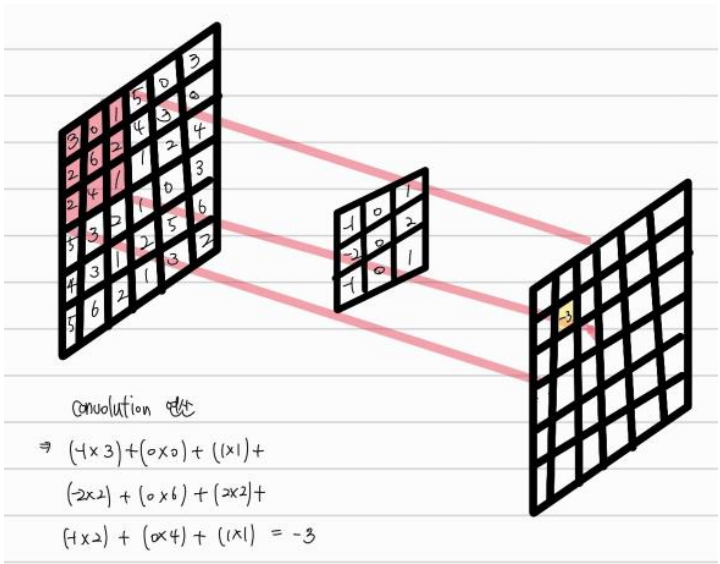
SpreadSalts함수

```
void SpreadSalts(Mat img, int num) {

    for (int n = 0; n < num; n++)
    {
        int x = rand() % img.cols;
        int y = rand() % img.rows;

        if (img.channels() == 1) {
            img.at<uchar>(y, x) = 255;
        }
        else {
            img.at<Vec3b>(y, x)[0] = 255;
            img.at<Vec3b>(y, x)[1] = 255;
            img.at<Vec3b>(y, x)[2] = 255;
        }
    }
}
```

이번 과제에서는 단일채널인 image에 흰색 점을 뿌리기 때문에, if문을 만족하여 img.at<uchar>(y,x)=255; 가 실행되어 source image에 흰색점을 무작위 위치에 흩뿌리게 된다.

myKernelConv3x3함수

```
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) {
                //영상 가장자리에서 영상 밖의 화소를 잃지 않도록 하는 조건문
                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 정규화되도록 해서 영상의 밝기 변화를 방지한다.
    }
    else
    {
        return sum;
    }
}
```



위 함수는 이와 같이 convolution 연산을 수행하는 함수로 2중for문을 이용해서 sum값에 convolution을 수행한 결과값을 저장한다. 여기서 sumKernel이 0이 아니라면 sum을 sumKernel로 나눠줘서 영상자체의 밝기가 전체적으로 변하는 것을 방지한다.

myKernelConv9x9함수

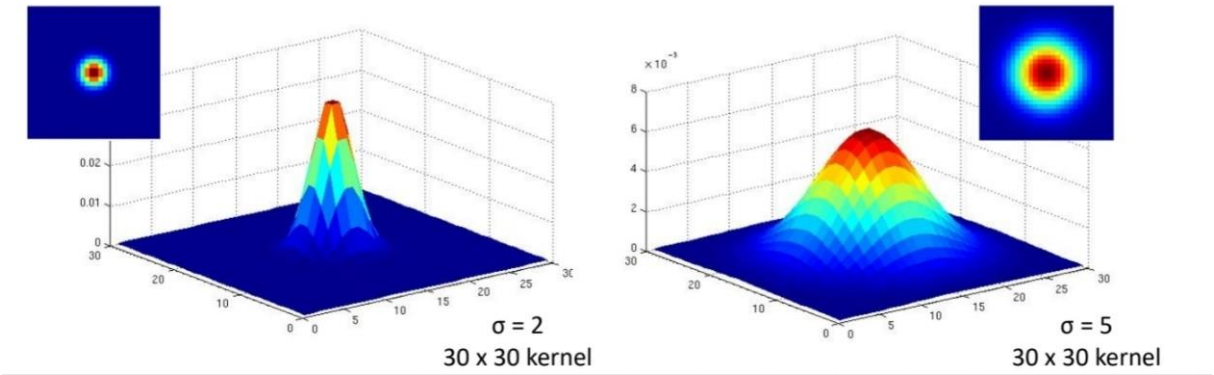
```
int myKernelConv9x9(uchar* arr, int kernel[][9], int x, int y, int width, int height, int ch, int p = 0) {
    int sum = 0;
    int sumKernel = 0;

    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) {
                //영상 가장자리에서 영상 밖의 화소를 잃지 않도록 하는 조건문
                if (ch == 1)
                    //1채널일때
                    sum += arr[(y + j) * width + (x + j)] * kernel[i + 4][j + 4];
                else if (ch == 3)
                    //3채널일때
                    sum += arr[(y + j) * width * 3 + (x + i) * 3 + p] * kernel[i + 4][j + 4];
                sumKernel += kernel[i + 4][j + 4];
            }
        }
    }
    if (sumKernel != 0) {
        return sum / sumKernel; //합이 1로 정규화되도록 해서 영상의 밝기 변화를 방지한다.
    }
    else
    {
        return sum;
    }
}
```

myKernelConv9x9함수는 3X3을 수정해서 작성했다. 먼저, -4부터 4까지의 마스크 이기 때문에, for문의 조건식을 변경했다. 또한, 컬러 영상일 때는 B,G,R을 접근해야하기 때문에 변수 p를 활용해서 p=0일때는 B,1일때는 G,2일때는 R에 접근하게 하였다.

myGaussianFilter함수

```
Mat myGaussianFilter(Mat srcImg) {
    int width = srcImg.cols;
    int height = srcImg.rows;
    int kernel[9][9] = { 1,3,5,7,10,7,5,3,1,
                         3,13,16,19,22,19,16,13,3,
                         5,16,25,28,31,28,25,16,5,
                         7,19,28,34,37,34,28,19,7,
                         10,22,31,37,40,37,31,22,10,
                         7,19,28,34,37,34,28,19,7,
                         5,16,25,28,31,28,25,16,5,
                         3,13,16,19,22,19,16,13,3,
                         1,3,5,7,10,7,5,3,1 };

    Mat dstImg(srcImg.size(), CV_8UC1); //1채널용

    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, 1);
            }
        }

        return dstImg;

}
```

9x9인 필터를 구현해야 하기 때문에, kernel을 9x9행렬로 선언했다. 필터의 폭의 절반은 표준편차의 3을 곱한 값으로 설정해야한다. 그리고, edge쪽의 값은 0과 근처의 값으로 설정해야한다. 아래의 그림과 같이 분산의 값이 더 커질수록 가운데에 값이 몰려있지않고, 퍼져있는 것을 알 수 있다.



예를 들어서 표준편차의 값이 3이라면

$3\sigma = 9$ 이기 때문에, filter의 크기는 17x17가 적당하다.

이번 과제에서는 역으로 filter의 크기를 9x9로 정해줬기 때문에, $3\sigma = 5$ 인 값을 만족하는 $\sigma = 1.6666$ 으로 표준편차를 이로 설정하여 gaussfilter를 만들었다. x,y 좌표의 값은 중요하지 않고, 원점으로부터의 거리가 중요하기 때문에 중심을 기준으로 같은 거리의 원이 있다 생각하고 알맞게 gaussian filter의 kernel을 설정했다.

myGaussianColorFilter함수

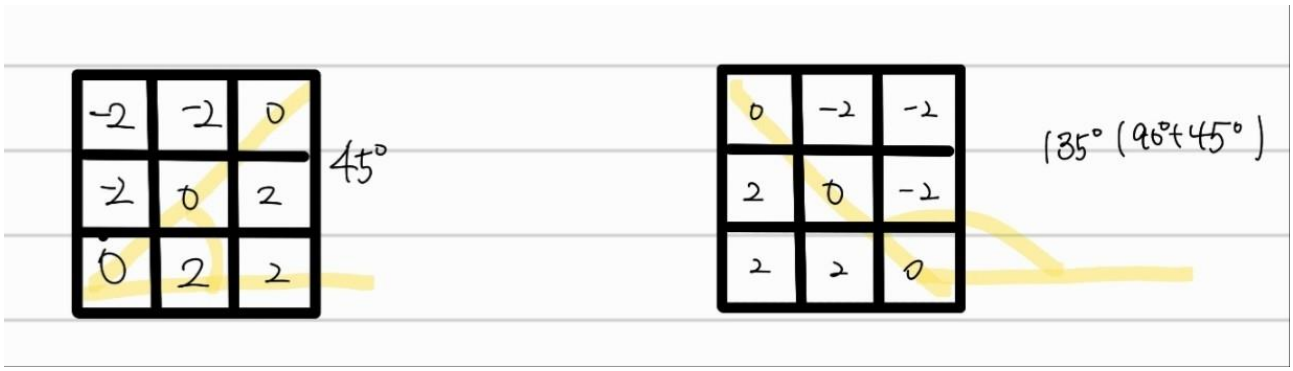
```
Mat myGaussianColorFilter(Mat srcImg) {
    int width = srcImg.cols;
    int height = srcImg.rows;
    int kernel[9][9] = { 1,3,5,7,10,7,5,3,1,
                         3,13,16,19,22,19,16,13,3,
                         5,16,25,28,31,28,25,16,5,
                         7,19,28,34,37,34,28,19,7,
                         10,22,31,37,40,37,31,22,10,
                         7,19,28,34,37,34,28,19,7,
                         5,16,25,28,31,28,25,16,5,
                         3,13,16,19,22,19,16,13,3,
                         1,3,5,7,10,7,5,3,1 };

    Mat dstImg(srcImg.size(), CV_8UC3); //3채널용

    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] = myKernelConv9x9(srcData, kernel, x, y, width, height, 3, 0);
            dstData[y * width * 3 + x * 3 + 1] = myKernelConv9x9(srcData, kernel, x, y, width, height, 3, 1);
            dstData[y * width * 3 + x * 3 + 2] = myKernelConv9x9(srcData, kernel, x, y, width, height, 3, 2);
        }
    }

    return dstImg;
}
```

myGaussian filter와의 차이는 컬러 이미지이기 때문에 픽셀 구성이 [B,G,R] [B,G,R] [B,G,R] 이기 때문에, 3을 곱하고 각각myKernelconv9x9 함수에 0 1 2 를 전달해주어 B G R에 접근하도록 구현했다.

mySobelFilter

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

    int kernelY[3][3] = { 0,-2,-2,
                         2,0,-2,
                         2,2,0 };



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

    //45도 커널
    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));
            }
        }
    }
```

```
    //135도 커널
    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));
            }
        }
    }
    //45도 135도 커널 적용한것을 합친다.
    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;

}
```



각가 45도와 135도의 edge를 검출하기 위한 커널을 위와같이 설정했다.

sel 변수를 통해서 sel=0 일 경우 dstImg가 45도 sobelfilter sel=1 일 경우 dstImg가 135도인 이미지 마지막으로 sel=2 일 경우 dstImg가 45도 와 135도 sobelfilter를 합친 결과를 반환하도록 구현했다. 두 에지의 결과의 절대값의 합의 형태로 결과를 도출했다. 그리고, sobel filter의 합은 0이 되기 때문에 1로 정

규화하는 과정은 필요가없다.

mySampling_color함수

```
Mat mySampling_color(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 * 6) + (x * 6)];
            dstData[y * width * 3 + x * 3 + 1] = srcData[(y * 2) * (width * 6) + (x * 6) + 1];
            dstData[y * width * 3 + x * 3 + 2] = srcData[(y * 2) * (width * 6) + (x * 6) + 2];
        }
    }
    return dstImg;

}
```

color이미지에 대한 GaussianPyramid와 LaplacianPyramid를 구현하는 것이기 때문에, down sampling을 할 때, 마찬가지로 B G R 각각에 대해 접근해서 down sampling했다. 여기서 3개의 채널에 대한 접근이기 때문에 CV_8UC3로 설정했다.

myGaussianPyramid_color함수

```
vector<Mat> myGaussianPyramid_color(Mat srcImg) {
    vector<Mat> Vec;

    Vec.push_back(srcImg);
    for (int i = 0; i < 4; i++)
    {   srcImg = mySampling_color(srcImg);
        srcImg = myGaussianColorFilter(srcImg);

        Vec.push_back(srcImg);
    }
    return Vec;

}
```

4회 GaussianColorFilter함수를 거쳐서 GaussianPyramid를 실행하는 함수이다.

myLaplacianPyramid_color함수

```
vector<Mat> myLaplacianPyramid_color(Mat srcImg) {
    vector<Mat> Vec;

    for (int i = 0; i < 4; i++)
    {
        if (i != 3) {
            Mat highImg = srcImg;

            srcImg = mySampling_color(srcImg);
            srcImg = myGaussianColorFilter(srcImg);
            Mat lowImg = srcImg;

            resize(lowImg, lowImg, highImg.size());
            //작아진 영상을 백업한 영상의 크기로 확대
            Vec.push_back(highImg - lowImg + 128);
            //뺀 영상을 벡터에 삽입
            //뺀 영상의 오버플로우를 방지하기 위해서 128 더한다.
        }
        else
        {
            Vec.push_back(srcImg);
        }
    }
    return Vec;
}
```

```
int main() {
    Mat src_img, dst_img;

    src_img = imread("C:₩₩images₩₩gear.jpg", 0);
    Mat src_his = Gethistogram(src_img);

    dst_img = myGaussianFilter(src_img);
    Mat dst_his = Gethistogram(dst_img);

    //1) 9x9 가우스 필터
    imshow("src_img", src_img);
    imshow("dst_img", dst_img);
    waitKey(0);

    //2) 9x9 가우스 필터 적용시 히스토그램
    imshow("src_hist", src_his);
    imshow("dst_hist", dst_his);
    waitKey(0);

    destroyWindow("src_img");
    destroyWindow("dst_img");
    destroyWindow("src_hist");
    destroyWindow("dst_hist");
    waitKey(0);
```

```
    //3) Salt and pepper noise 주고 9x9 가우스 필터 적용
    SpreadSalts(src_img, 1000);
    imshow("src_img_salts", src_img);
    waitKey(0);
    dst_img = myGaussianFilter(src_img);
    imshow("dst_img_salts", dst_img);
    waitKey(0);

    destroyWindow("src_img_salts");
    destroyWindow("dst_img_salts");
    waitKey(0);
```

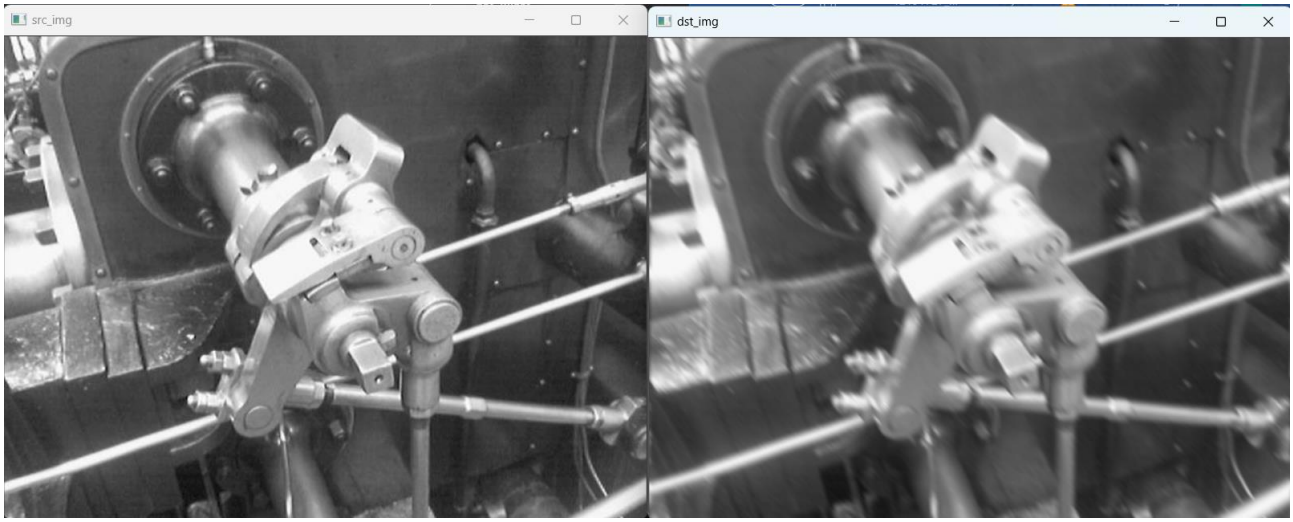
```
//4) 45도와 135도의 대각 에지를 검출하는 Sobel filter 구현
src_img = imread("C:₩₩images₩₩gear.jpg", 0);
dst_img = mySobelFilter(src_img,0);
Mat dst_img2 = mySobelFilter(src_img,1);
Mat dst_img3 = mySobelFilter(src_img,2);

imshow("45", dst_img);
imshow("135", dst_img2);
imshow("45+135", dst_img3);
waitKey(0);

destroyWindow("45");
destroyWindow("135");
destroyWindow("45+135");
waitKey(0);
```

```
//5) 컬러영상에 대한 가우스 피라미드 구축
src_img = imread("C:₩₩images₩₩gear.jpg", 1);
vector<Mat>Vec_Gau = myGaussianPyramid_color(src_img);
for (int i = 0; i < Vec_Gau.size(); i++)
{
    imshow("Gaussian pyramid", Vec_Gau[i]);
    waitKey(0);
}

destroyWindow("Gaussian pyramid");
```

```
//6) 컬러영상에 대한 라플라시안 피라미드 구축, 복원 수행 결과
src_img = imread("C:₩₩images₩₩gear.jpg", 1);
vector<Mat>Vec_Lap = myLaplacianPyramid_color(src_img);

reverse(Vec_Lap.begin(), Vec_Lap.end());
//작은 영상부터 처리하기 위해서 vector 의 순서를 반대로
for (int i = 0; i < Vec_Lap.size(); i++)
{
    imshow("Laplacian pyramid", Vec_Lap[i]);
    waitKey(0);
}
destroyWindow("Laplacian pyramid");

for (int i = 0; i < Vec_Lap.size(); i++)
{
    if (i==0)
    {
        dst_img = Vec_Lap[i];
        //가장 작은 영상은 밴 영상이 아니기 때문에 바로 불러온다.
    }
    else
    {
        resize(dst_img, dst_img, Vec_Lap[i].size());
        dst_img = dst_img + Vec_Lap[i] - 128;
        // 밴 영상을 다시 더해서 큰 영상을 복원한다.
        // 오버플로우 방지용으로 더해준 128을 다시 뺀다.
    }
    string fname = "lap_pyr" + to_string(i) + ".png";
    imwrite(fname, dst_img);
    imshow("Laplacian recovery", dst_img);
    waitKey(0);
    destroyWindow("Laplacian recovery");
}

}
```

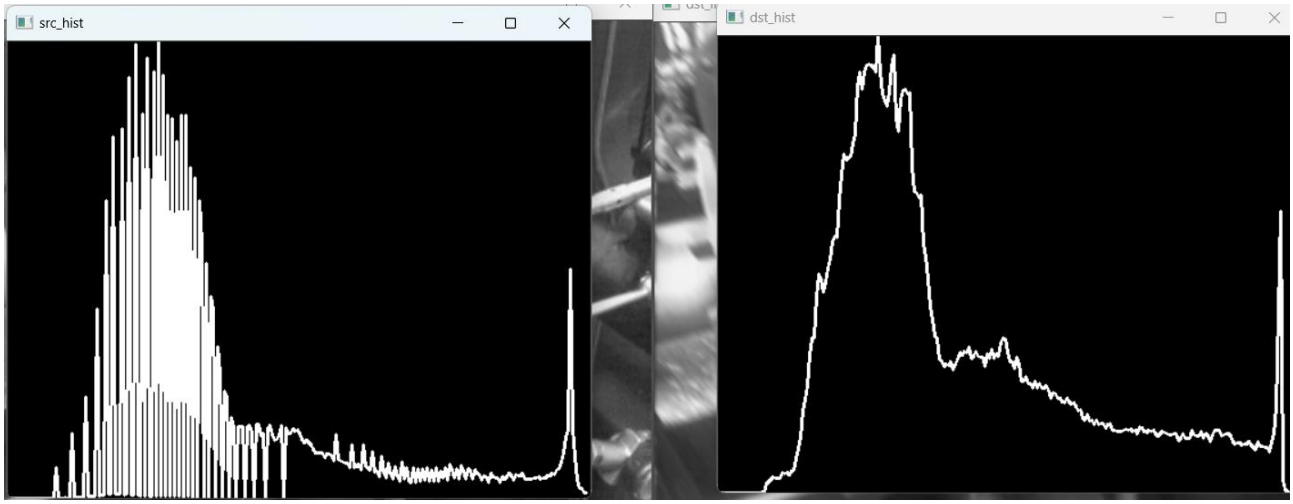
3. 실행 화면

#1



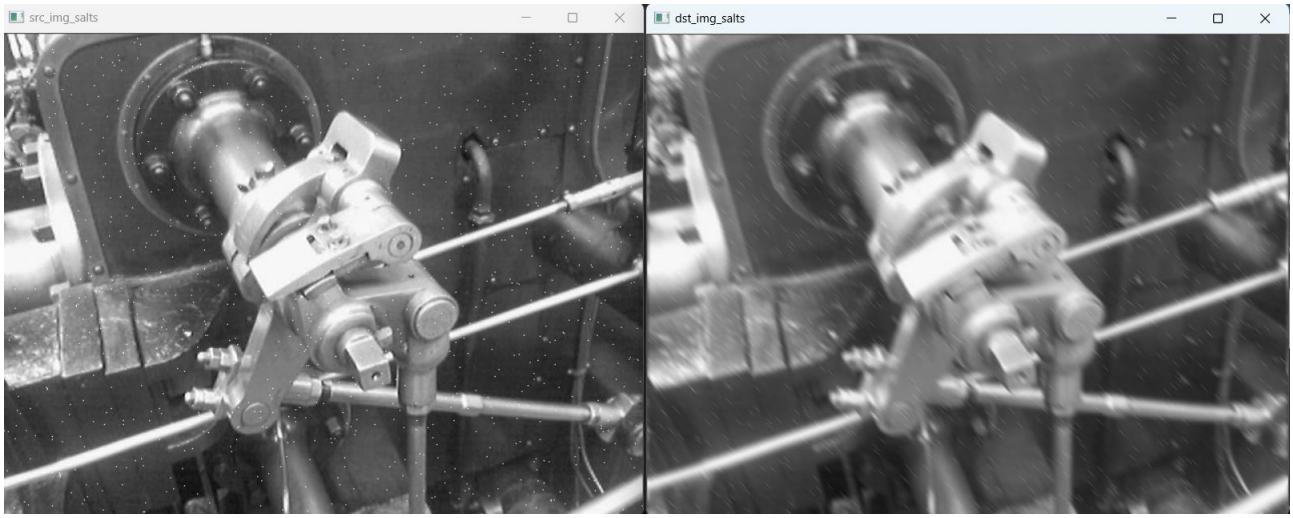
9x9 가우스필터를 적용해서 이미지가 흐려진 것을 확인할 수 있었다.

#2



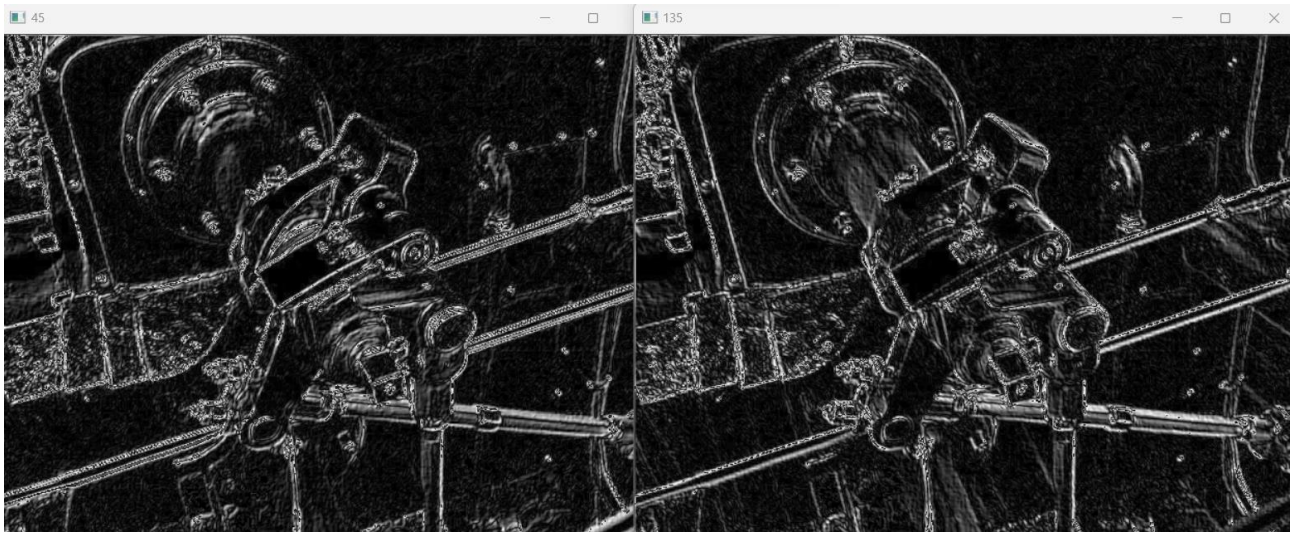
히스토그램을 비교했을 때 Gaussian filtering을통해서 smoothing이 적절히 이루어진 것을 확인할 수 있었다.

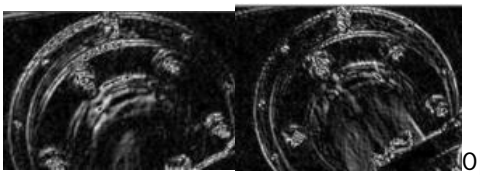
#3

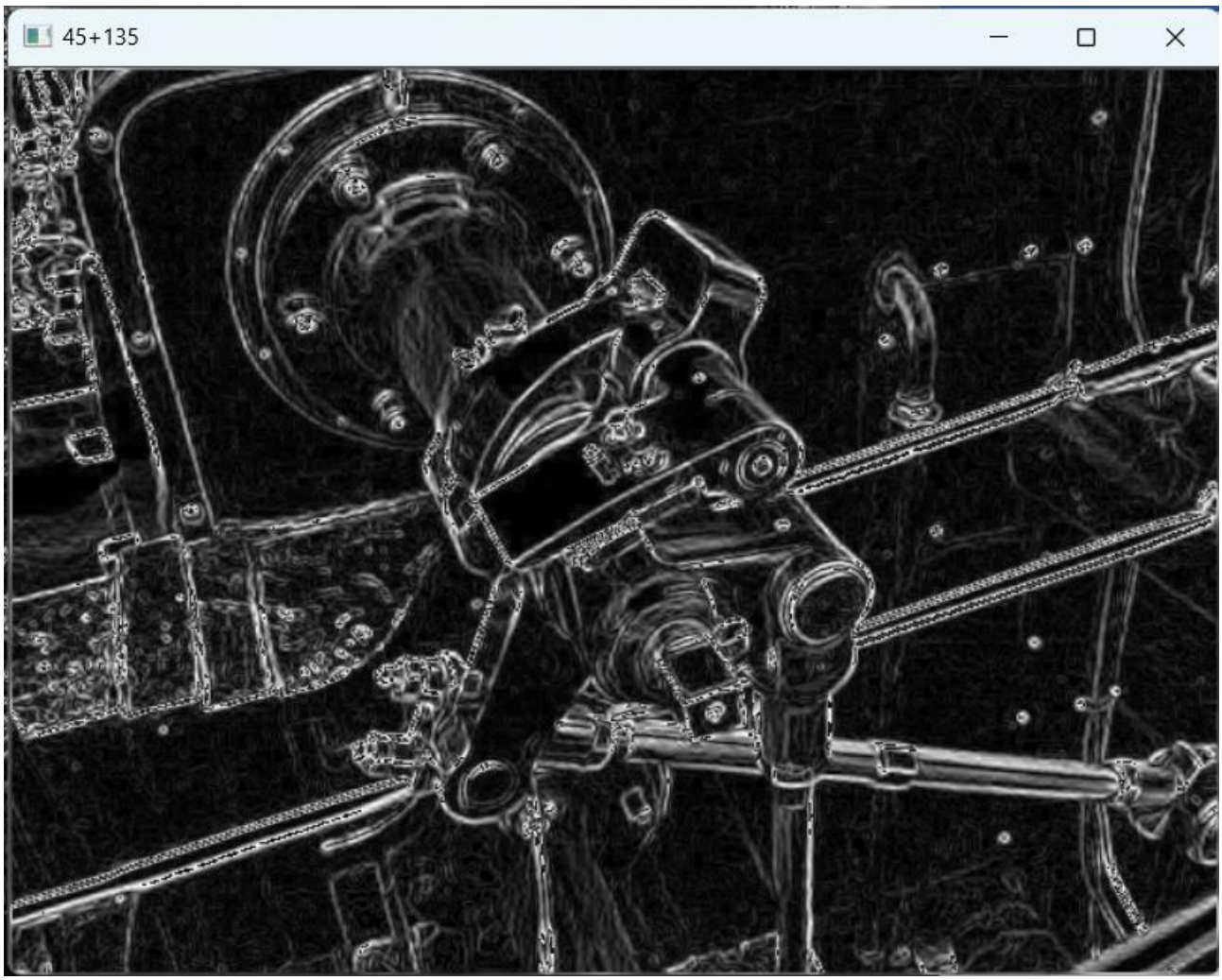


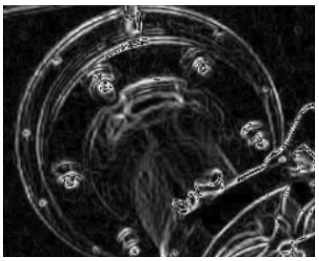
흰색점을 무작위로1000개 뿌렸고, Gaussian filter적용시 이 흰점도 흐려지는 것을 확인할 수 있었다.

#4

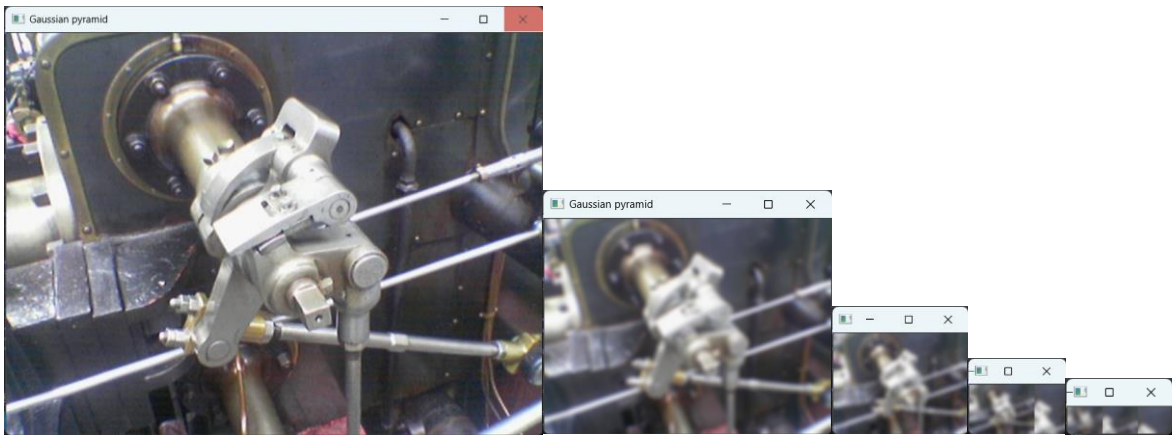


각각 45도 edge와 135도 edge에 대해 추출한것으로  이 부분을 비교해 보았을 때 잘 실행된 것을 확인할 수 있었다.



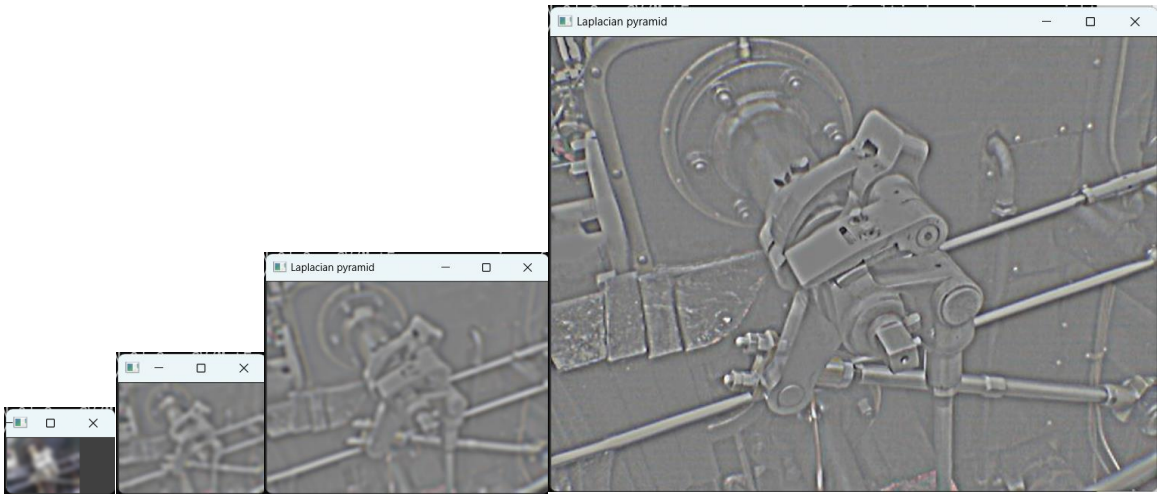
45도 와 135도 edge 검출을 합친것으로 마찬가지로  이 부분으로 확인 할 수 있었다.

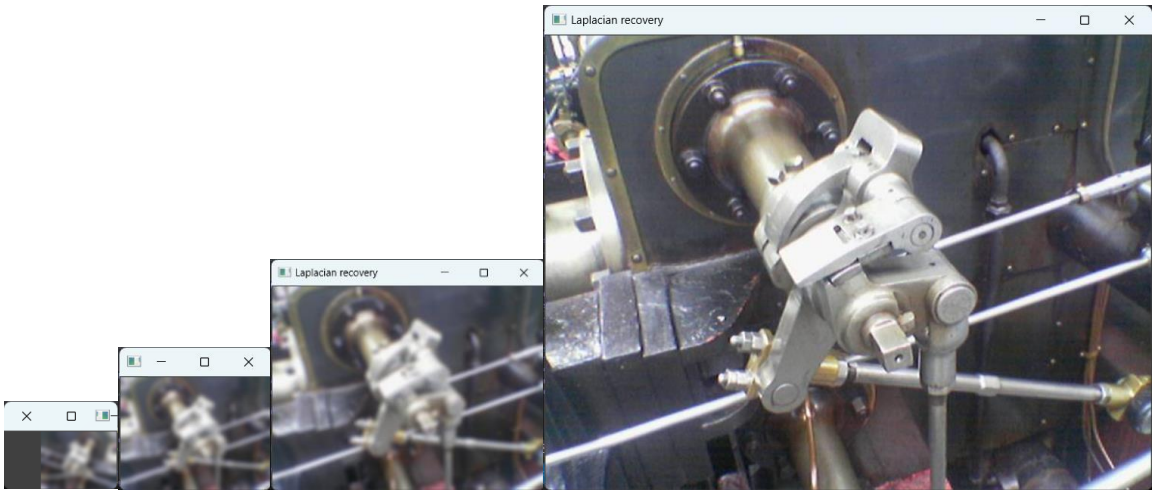
#5



컬러 이미지로 점점 흐려지는 Gaussianpyramid의 결과가 제대로 출력된 것을 확인 할 수 있었다.

#6





최종적으로 원본 사진과 같은 이미지로 복원된 것을 확인할 수 있었다.