디지털 영상 처리 설계 과제 보고서 (3주차)

정보통신공학과

12171797

신원철

1. Introduction

A. 직접 gaussian filter와 sobel filter을 구현하여 영상에 적용하여 결과를 확인한다.

B. Gaussian pyramid와 Laplacian pyramid를 구축하고 복원한다.

2. Problem

A. 9x9 Gaussian filter를 구현하고 결과를 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; }//합 1로 정규화
    else { return sum; }
}
```

```
Mat myGaussianFilter(Mat srcImg) {
#if USE_OPENCV
    Mat dstImg(srcImg.size(), CV_8UC1);
    cv::GaussianBlur(srcImg, dstImg, Size(3, 3), 0);

#else
    int width = srcImg.cols;
    int height = srcImg.rows;
    int kernel[9][9] = { 0, 0, 0, 0, 1, 0, 0, 0, 0,
                         0, 0, 2, 11, 18, 11, 2, 0, 0,
                         0, 2, 29, 131, 215, 131, 29, 2, 0,
                         0, 11, 131, 585, 965, 585, 131, 11, 0,
                         1, 18, 215, 965, 1592, 965, 215, 18, 1,
                         0, 11, 131, 585, 965, 585, 131, 11, 0,
                         0, 2, 29, 131, 215, 131, 29, 2, 0,
                         0, 0, 2, 11, 18, 11, 2, 0, 0,
                         0, 0, 0, 0, 1, 0, 0, 0, 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;
#endif
}
```

Mykernelconv9x9함수는 주어진 행렬과 kernel행렬사이의 convolution연산을 수행한다. mygaussianfilter함수에서는 이 함수를 사용하여 영상의 이미지에서 9x9픽셀에서 값을 행렬화 하고 이를 9x9형태의 gaussian filter와 conv 연산을 수행하여 gaussian filter을 적용시킨다.

B. 9x9 Gaussian filter를 적용했을 때 히스토그램이 어떻게 변하는지 확인할 것

위 수행을 통해 나온 결과를 histogram으로 표현하기 위해 2주차 실습에 사용하였던 GetHistogram함수를 사용하여 해당 이미지를 히스토그램으로 이미지를 분석하였다.

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

```
void SpreadSalts(Mat img, int num) {
    for (int n = 0; n < num; n++) {
        int x = rand() % img.cols;
        int y = rand() % img.rows;
        int x_1 = rand() % img.cols;
        int y_1 = rand() % img.rows;
        int x_2 = rand() % img.cols;
        int y_2 = rand() % img.rows;

        if (img.channels() == 1) {
            img.at<uchar>(y, x) = 255;
            img.at<uchar>(y_1, x_1) = 0;
        }
        else {
            img.at<Vec3b>(y, x)[0] = 0; //b
            img.at<Vec3b>(y, x)[1] = 255; //g
            img.at<Vec3b>(y, x)[2] = 0; //r

            img.at<Vec3b>(y_1, x_1)[0] = 255; //b
            img.at<Vec3b>(y_1, x_1)[1] = 0; //g
            img.at<Vec3b>(y_1, x_1)[2] = 0; //r

            img.at<Vec3b>(y_2, x_2)[0] = 0; //b
            img.at<Vec3b>(y_2, x_2)[1] = 0; //g
            img.at<Vec3b>(y_2, x_2)[2] = 255; //r
        }
    }
}
```

위와 같이 salt and pepper noise를 주는 함수를 만들어 원본 영상에 적용한 뒤 위와 같은 방법으로 gaussian filter를 적용하였다.

D. 45도와 135도의 대각 에지를 검출하는 Sobel filter를 구현하고 결과를 확인할 것

```
int kernelX[3][3] = { 0, 1, 2,
                      -1, 0, 1,
                      -2, -1, 0 };
int kernelY[3][3] = { -2, -1, 0,
                      -1, 0, 1,
                      0, 1, 2 };
```

기존 kernel을 대각 에지를 검출하도록 위와 같이 수정하여 적용하였다.

E. 컬러영상에 대한 Gaussian pyramid를 구축하고 결과를 확인할 것

컬러 영상에 대해 gaussian pyramid를 구축하기 위하여 mySampling함수를 다음과 같이 수정하였다.

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

이를 통해 mySampling함수는 채널 별로 픽셀 값을 2픽셀 단위로 샘플링하여 해당 이미지를 리턴한다.

다음으로 mygausiancolorfilter함수를 작성하였다.

```
Mat myGaussiancolorFilter(Mat srcImg) {
    int width = srcImg.cols;
    int height = srcImg.rows;
    int kernel[9][9] = { 0, 0, 0, 0, 1, 0, 0, 0, 0,
                         0, 0, 2, 11, 18, 11, 2, 0, 0,
                         0, 2, 29, 131, 215, 131, 29, 2, 0,
                         0, 11, 131, 585, 965, 585, 131, 11, 0,
                         1, 18, 215, 965, 1592, 965, 215, 18, 1,
                         0, 11, 131, 585, 965, 585, 131, 11, 0,
                         0, 2, 29, 131, 215, 131, 29, 2, 0,
                         0, 0, 2, 11, 18, 11, 2, 0, 0,
                         0, 0, 0, 0, 1, 0, 0, 0, 0 };
    Mat dstImg(srcImg.size(), CV_8UC3);
    uchar* srcData = srcImg.data;
    uchar* dstData = dstImg.data;

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            if ((x >= 4) && (y >= 4) && (x < width - 4) && (y < height - 4)) {
                for (int c = 0; c < srcImg.channels(); c++) {
                    int sum = 0;
                    int sumKernel = 0;
                    for (int j = -4; j <= 4; j++) {
                        for (int i = -4; i <= 4; i++) {
                            sum += srcData[((y + j) * width + (x + i)) * srcImg.channels() + c] * kernel[i + 4][j + 4];
                            sumKernel += kernel[i + 4][j + 4];
                        }
                    }
                    if (sumKernel != 0) { dstData[(y * width + x) * srcImg.channels() + c] = sum / sumKernel; }//합이1로 정규화
                    else { dstData[(y * width + x) * srcImg.channels() + c] = sum; }
                }
            }
            else {
                for (int c = 0; c < srcImg.channels(); c++) {
                    dstData[(y * width + x) * srcImg.channels() + c] = srcData[(y * width + x) * srcImg.channels() + c];
                }
            }
        }
    }

    return dstImg;
}
```

위 함수는 입력 소스에 대해 각 채널별로 gaussian 필터를 적용하여 이미지를 벡터에 저장하였다.

F. 컬러영상에 대한 Laplacian pyramid를 구축하고 복원을 수행한 결과를 확인할 것

```
for (int i = 0; i < VecLap.size(); i++) {
    string fname = "ex5_lap_pyr_recover" + to_string(i) + ".png";
    dst_img = VecLap[i];
    imshow(fname, dst_img);
    if (i == 0) {
        dst_img = VecLap[i];
    }
    else {
        resize(dst_img, dst_img, VecLap[i].size());
        vector<Mat> channels_dst;
        split(dst_img, channels_dst);
        vector<Mat> channels_lap;
        split(VecLap[i], channels_lap);
        for (int c = 0; c < dst_img.channels(); c++) {
            channels_dst[c] = channels_dst[c] + channels_lap[c] - 128;
        }
        merge(channels_dst, dst_img);
    }

    //imwrite(fname, dst_img);
    //imshow(fname, dst_img);
    //waitKey(0);
    //destroyWindow("EX5");
}
```

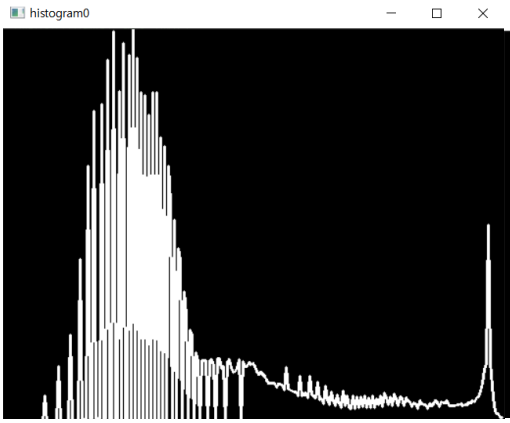
위 코드와 같이 채널 별로 확대된 gaussian pyramid와 Laplacian pyramid를 합쳐 원래 영상을 복원하였다.

3. Result

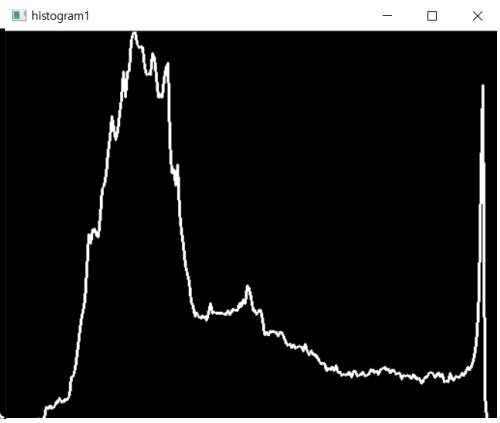
A.

위와 같이 원본 영상이 9x9 gaussian filter가 적용된 모습을 확인할 수 있었다.

B.



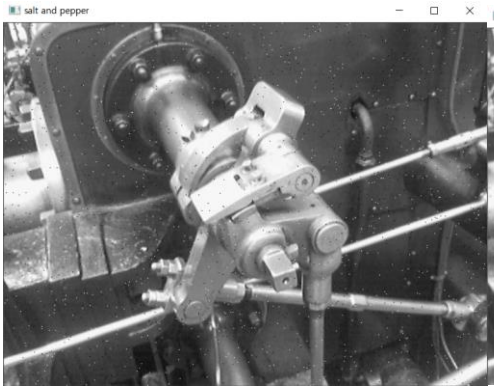
<원본 영상 히스토그램>



<filter 통과 후 히스토그램>

Gaussian filter을 통과한 후 히스토그램과 원본 히스토그램을 비교한 결과 위와 같았다. 해당 필터를 통과하니 원본 영상에 비해 노이즈가 줄어들고 영상의 특징을 보다 잘 표현하는 것을 확인할 수 있었다.

C.

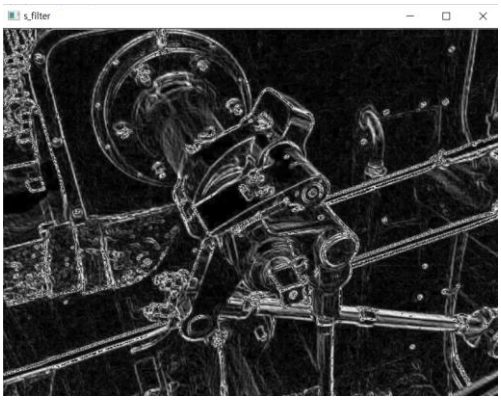


<gaussian filter 적용 전>

<gaussian filter 적용 후>

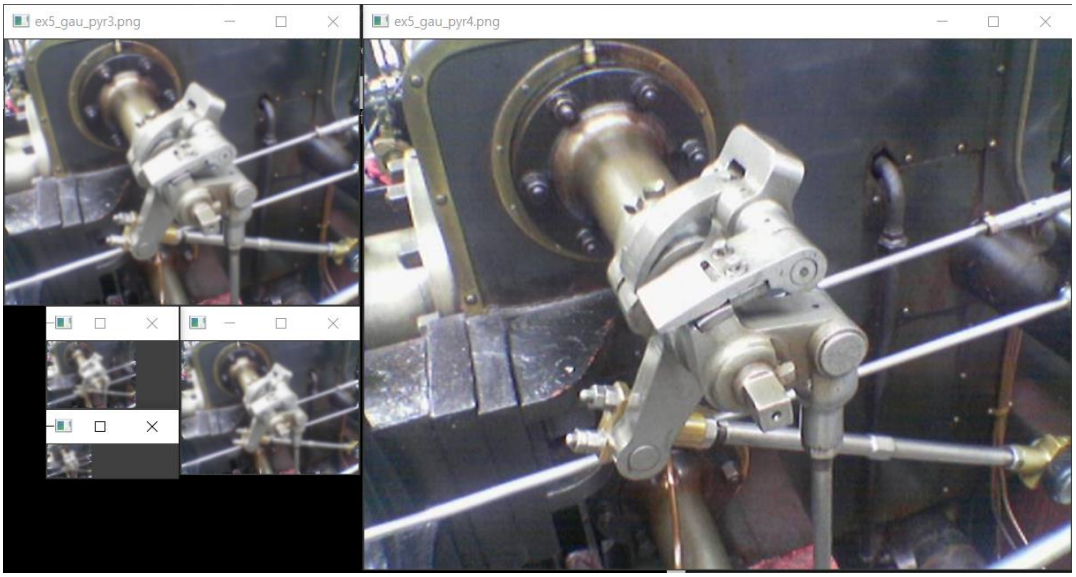
Gaussian filter적용 후 적용 전에 비해 salt and pepper 노이즈가 상당히 완화된 모습을 확인할 수 있었다.

D.



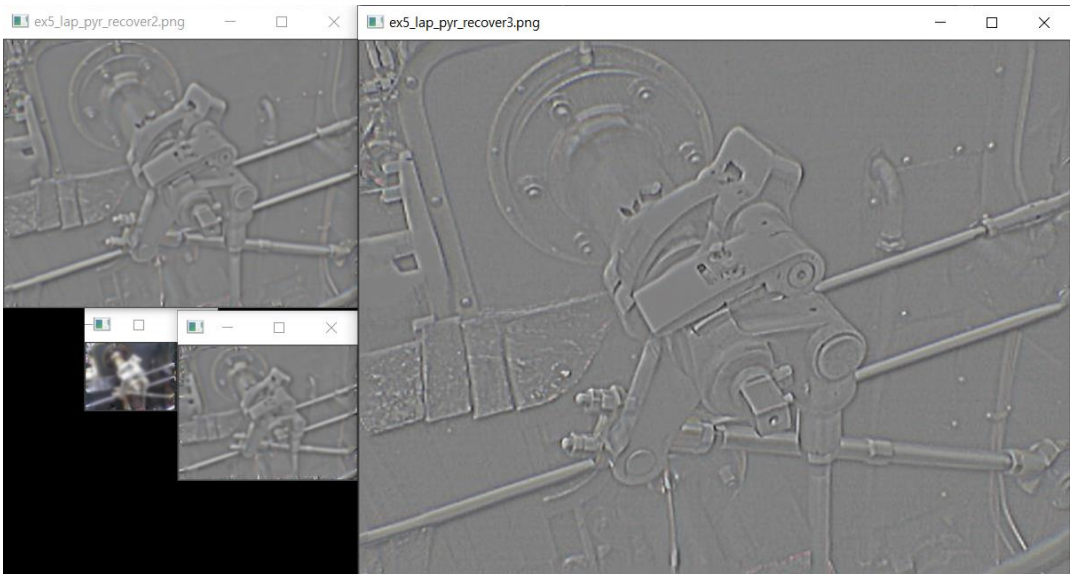
위와 같이 대각선 방향으로 sobel filter가 적용된 사실을 확인할 수 있었다.

E.

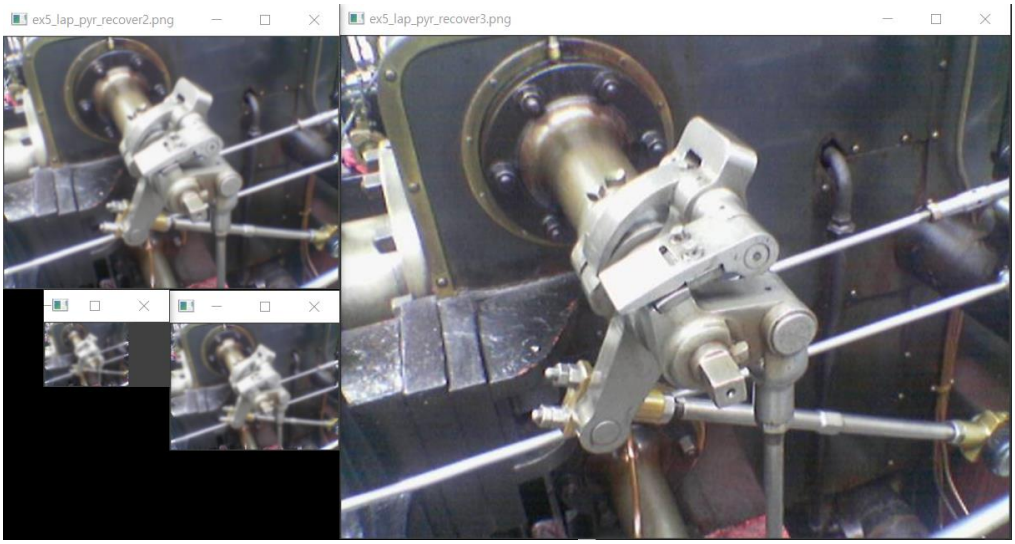


위와 같이 gaussian pyramid가 정상적으로 생성된 것을 확인할 수 있었다.

F.



위와 같이 Laplacian pyramid가 정상적으로 적용된 사실을 확인할 수 있었다.



위 Laplacian pyramid를 사용하여 원본 영상을 복원하니 다음과 같았다.

4. Conclusion