INHA UNIVERSITY

ICE4027 디지털영상처리설계

실습 7주차

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 12일

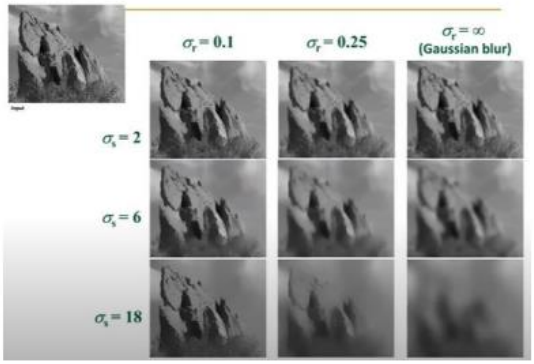
학부 정보통신공학
학년 3
성명 김동한
학번 12191727

1. 개요

Homework

실습 및 과제

- salt_pepper2.png에 대해서 3x3, 5x5의 Mean 필터를 적용해보고 결과를 분석할 것 (잘 나오지 않았다면 그 이유와 함께 결과를 개선해볼 것)
- rock.png에 대해서 Bilateral 필터를 적용해볼 것 (아래 table을 참고하여 기존 gaussian 필터와의 차이를 분석해볼 것)
- OpenCV의 Canny edge detection 함수의 파라미터를 조절해 여러 결과를 도출하고 파라미터에 따라서 처리시간이 달라지는 이유를 정확히 서술할 것



1. 구현과정과 결과 분석을 반드시 포함할 것
2. 보고서에도 코드와 실험결과 사진을 첨부할 것
3. 반드시 바로 실행가능한 코드(.cpp)를 첨부할 것
4. 별도의 언급이 없으면 OpenCV가 아닌 직접 구현 함수를 쓸 것

강의 영상 참고

2. 상세 설계 내용

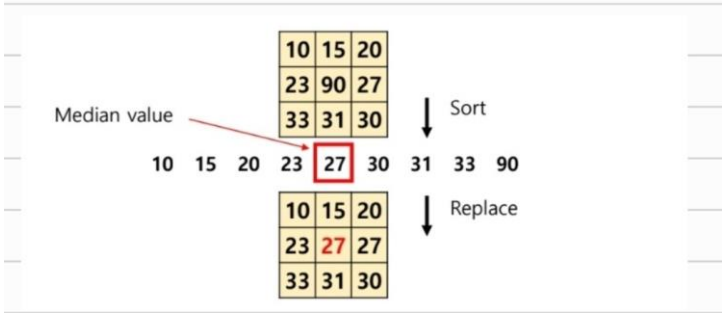
#1.

먼저 아래 사진은 salt_pepper2.png이다. salt_pepper1과 다르게 흰색점과 검은색점의 noise가 더 많은 것을 알 수 있다. 또한, 선으로 이어져있는 듯 한 noise도 보인다.

이 png에 3x3 median filter를 적용하면 아래와 같은 결과가 나온다.

모든 noise가 다 제거되지 않은 결과창을 볼 수 가 있다. 그 이후 5x5 median filter를 적용한 출력화면이다.

3x3 median filter와 다르게 white,pepper outlier가 모두 제거된 것을 확인 할 수 있다.



median filter는 마스크에 해당하는 픽셀값들을 sorting한뒤 중간값에 해당하는 값으로 대체하는 filter이

다. 여기서 인위적으로 만들지않은 영상은 인접한 화소와의 픽셀값이 급격하게 차이 날 일이 거의 없기 때문에, white,pepper outlier값들을 골라내서 filtering가능한것이다. 많이 등장하는 픽셀값들이 중간값이 될 확률이 높기 때문에 smoothing이 잘되고, 주변의 영향을 많이 받을 수 있게끔 filter의 크기가 커지면 smoothing이 더 많이 된다. 따라서, 3x3 filter에서는 사라지지않은 outlier들이 5x5filter에서는 사라진것이다. median filter는 화소값의 순서를 따지기 때문에 linear한 filter는 아니다.

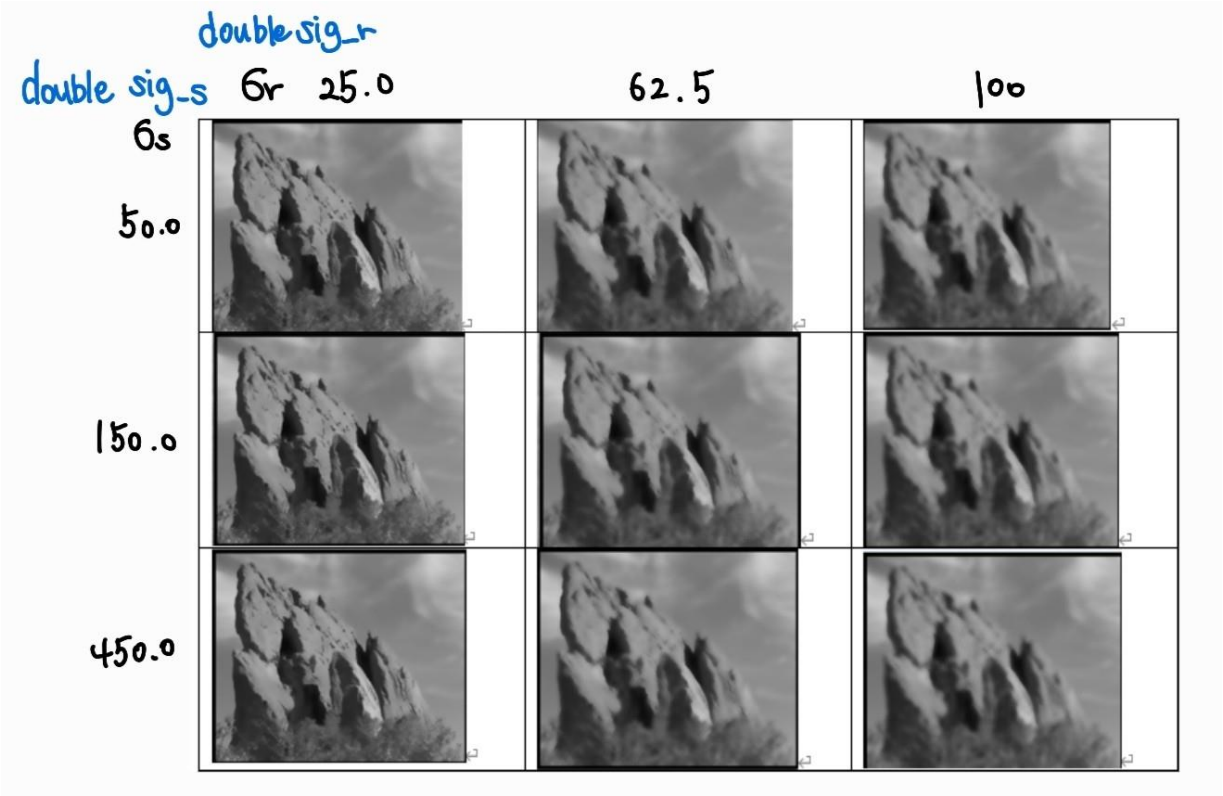
```
void doMedianEx(int kernel_size) {
    cout << "--- doMedianEx() --- ₩n" << endl;

    // 입력
    Mat src_img = imread("C:₩₩images₩₩salt_pepper2.png", 0);
    if (!src_img.data) printf("No image data ₩n");

    //Median 필터링 수행
    Mat dst_img;
#if USE_OPENCV
    medianBlur(src_img, dst_img, 3);
#else
    myMedian(src_img, dst_img, Size(kernel_size, kernel_size));
#endif
    //출력
    Mat result_img;
    hconcat(src_img, dst_img, result_img);
    imshow("doMedianEx()", result_img);
    waitKey(0);
}
```

medianfilter를 실행하는 함수에서 myMedian의 Size에 해당하는 argument를 변수로 바꿔준뒤, main함수에서 3과5를 대입하는 것으로 3x3 filter와 5x5 filter를 구현했다.

```
void myMedian(const Mat& src_img, Mat& dst_img, const Size& kn_size) {

    // 임시 저장해서 테이블을 정렬
    dst_img = Mat::zeros(src_img.size(), CV_8UC1);

    int wd = src_img.cols;
    int hg = src_img.rows;
    int kwd = kn_size.width; int khg = kn_size.height;
    int rad_w = kwd / 2; int rad_h = khg / 2;

    uchar* src_data = (uchar*)src_img.data;
    uchar* dst_data = (uchar*)dst_img.data;

    float* table = new float[kwd * khg](); //커널 테이블 동적할당
    float tmp;

    //픽셀 인덱싱(가장자리 제외)
    for (int c = rad_w + 1; c < wd - rad_w; c++) {
        for (int r = rad_h + 1; r < hg - rad_h; r++) {
            tmp = 0.f;
            for (int kc = -rad_w ; kc <=rad_w ; kc++)
            {
                for (int kr = -rad_h;kr <= rad_h;kr++) {
                    tmp = (float)src_data[(r + kr) * wd + (c + kc)];
                    table[(kr + rad_h) * kwd + (kc + rad_w)] = tmp;
                }
            }
            sort(table, table + kwd * khg);
            dst_data[r * wd + c] = (uchar)table[(kwd * khg) / 2];

        }
    }
    delete table;
}
```

위는 myMedian filter함수이다. median filter를 적용할 이미지에서 커널에 포함된 영역의 픽셀값을 tmp에 저장한다. (실수형으로 초기화) 모든 픽셀의 값을 tmp변수에 저장하고 그 값을 table에 저장한다. 그 뒤, sort를 이용해서 table에서의 중간값을 찾는다(median). kc,kr은 커널 안에서의 가로, 세로 방향 인덱스이다.

#2.




```
void doBilateralEx(double sig_r, double sig_s) {
    cout << "--- doBilateralEx() --- ₩n" << endl;

    //입력
    Mat src_img = imread("C:₩₩images₩₩rock.png", 0);
    Mat dst_img;
    if (!src_img.data) printf("No image Data₩n");

    //bilateral 필터링 수행
#if USE_OPENCV
    bilateralFilter(src_img, dst_img, 5, 25.0, 50.0);
#else
    myBilateral(src_img, dst_img, 10, sig_r, sig_s);
#endif
    //출력
    Mat result_img;
    hconcat(src_img, dst_img, result_img);
    imshow("doBilateralEx()", result_img);
    waitKey(0);
}
```

bilateral함수에서 강도 가중치 sig_r과 공간 가중치 sig_s를 변수로 받아 강의노트에 있는 표에 해당하게 끔 입력값을 다르게 받아서 출력화면을 내봤다.

```
void bilateral(const Mat& src_img, Mat& dst_img, int c, int r, int diameter, double sig_r, double sig_s) {

    int radius = diameter / 2;

    double gr, gs, wei;
    double tmp = 0.;
    double sum = 0.;

    //커널 인덱싱
    for (int kc = -radius; kc <= radius; kc++) {
        for (int kr = -radius; kr <= radius; kr++) {
            //range calc
            gr = gaussian((float)src_img.at<uchar>(c + kc, r + kr) - (float)src_img.at<uchar>(c, r), sig_r);
            //spatial calc
            gs = gaussian(distance(c, r, c + kc, r + kr), sig_s);
            wei = gr * gs;
            tmp += src_img.at<uchar>(c + kc, r + kr) * wei;
            sum += wei;
        }
    }
    dst_img.at<double>(c, r) = tmp / sum; //정규화
}
```

bilateral 함수 내의 diameter는 필터링에 사용되는 커널의 크기를 결정해준다. 이 값을 10으로 변경해주었다.(화면에 눈에띄는 차이가 없었다) 그리고, 각각의 가중치는 표에 있는 값들 sig_r 0.1 0.25 / sig_s 2 6 18의 비율에 맞게끔 값을 설정해서 실행해보았다. bilateral filter는 gaussian filter와 다르게 edge가 흐려흐려 않는 것을 알 수 출력화면을 통해서 알 수 있었다.

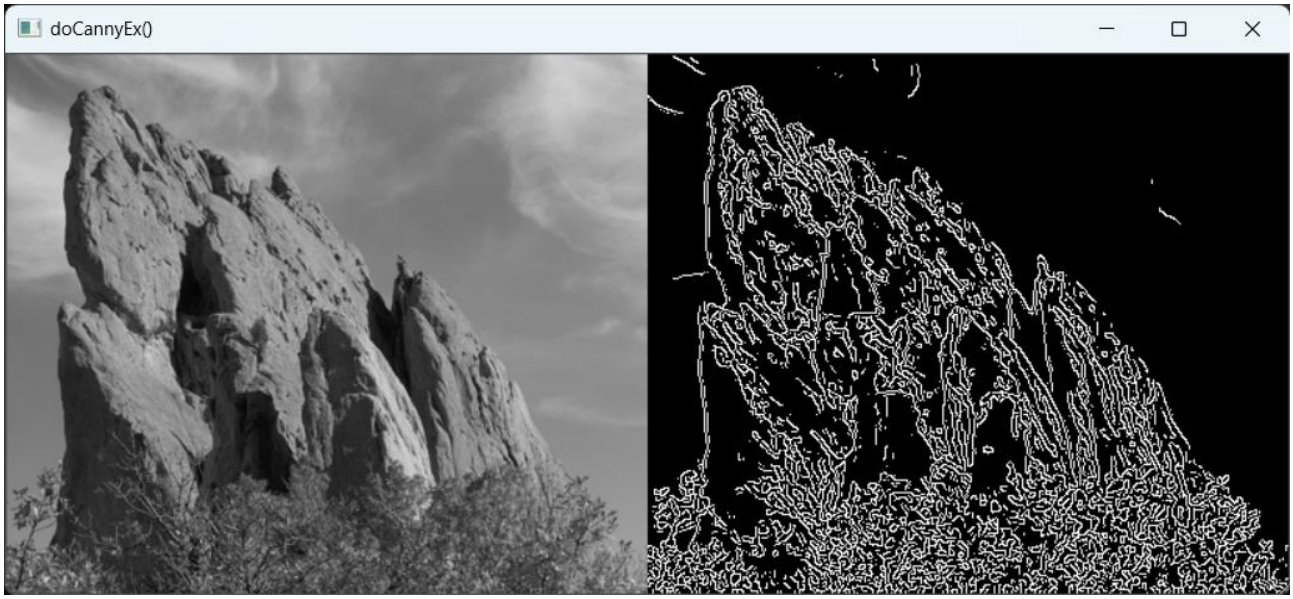
#3.

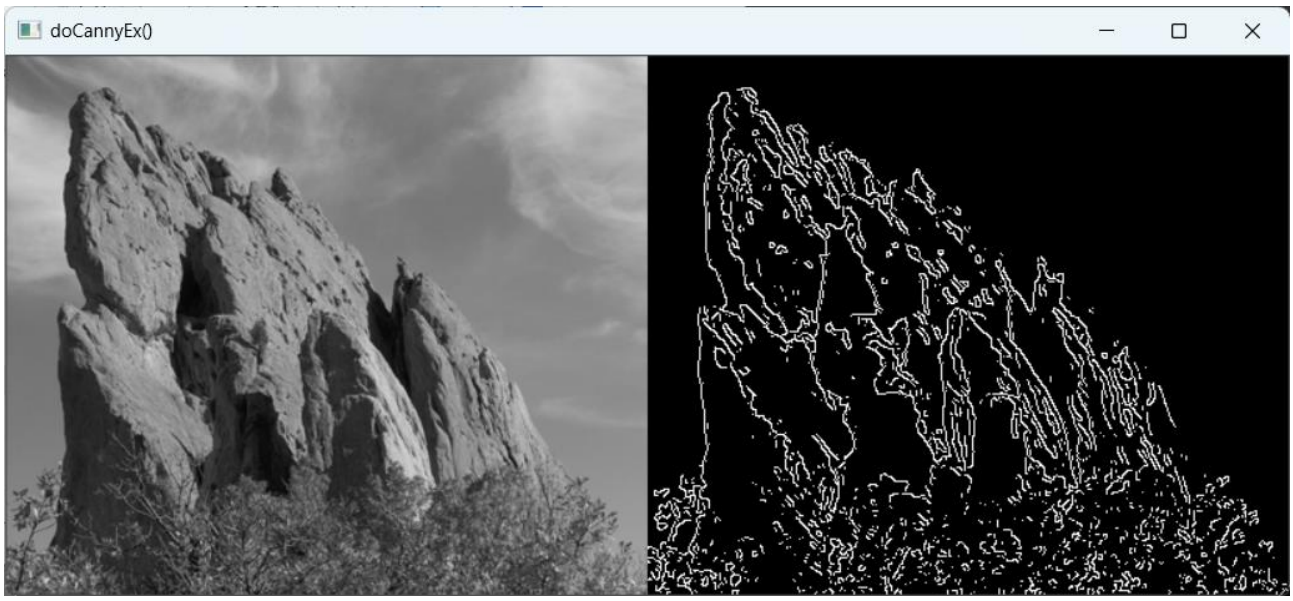
cannyedge detection 의 파라미터중 임계값에 관한 파라미터를 수정했다.

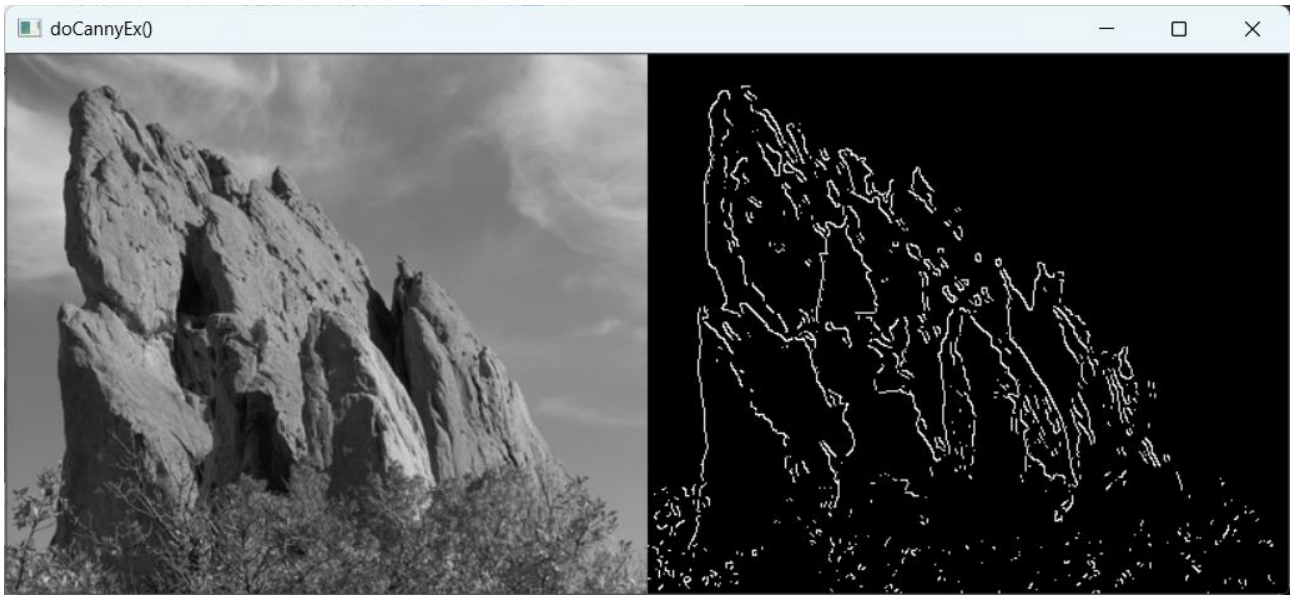
```
void doCannyEx(int threshold_A, int threshold_B) {
    cout << "--- doCannyEx() --- \n" << endl;
    //입력
    Mat src_img = imread("C:\\images\\rock.png", 0);
    if (!src_img.data) printf("No image data\n");
    Mat dst_img;
#if USE_OPENCV
    //Canny edge 탐색 수행
    Canny(src_img, dst_img, 180, 240);
#else

    clock_t start, end;
    start = clock();

    //가우시안 필터 기반 노이즈 제거
    Mat blur_img;
    GaussianBlur(src_img, blur_img, Size(3, 3), 1.5);

    //소벨 엣지 detection
    Mat magX = Mat(src_img.rows, src_img.cols, CV_32F);
    Mat magY = Mat(src_img.rows, src_img.cols, CV_32F);
    Sobel(blur_img, magX, CV_32F, 1, 0, 3);
    Sobel(blur_img, magY, CV_32F, 0, 1, 3);

    Mat sum = Mat(src_img.rows, src_img.cols, CV_64F);
    Mat prodX = Mat(src_img.rows, src_img.cols, CV_64F);
    Mat prodY = Mat(src_img.rows, src_img.cols, CV_64F);
    multiply(magX, magX, prodX);
    multiply(magY, magY, prodY);
    sum = prodX + prodY;
    sqrt(sum, sum);

    Mat magnitude = sum.clone();
```

```
    // Non-maximum suppression
    Mat slopes = Mat(src_img.rows, src_img.cols, CV_32F);
    divide(magY, magX, slopes);

    //gradient의 방향 계산
    nonMaximumSuppreession(magnitude, slopes);

    //Edge tracking by hysteresis
    edgeDetect(magnitude, threshold_A, threshold_B, dst_img);
    dst_img.convertTo(dst_img, CV_8UC1);

    end = clock();


#endif
    //출력
    Mat result_img;
    hconcat(src_img, dst_img, result_img);
    imshow("doCannyEx()", result_img);
    cout << threshold_A << ", " << threshold_B <<" "<< end - start << "ms₩n";
    waitKey(0);
}
```

먼저 GaussianBlur를 이용해 이미지에서 노이즈를 제거한다. Sobel filter를 사용해서 X,Y방향의 gradient를 구하고, 제곱하여 더한후 제곱근을 구해 magnitude에 크기를 저장한다. 마지막 edgeDetect()함수에서 강한 엣지, 약한 엣지의 threshold에 해당하는 threshold_A와 threshold_B를 변수로 함수에 전달한다. threshold값이 크면 강력한 edge만 검출하고 threshold값이 낮으면 edge의 계단이 약한것으로 grey+white값을 검출한다. canny edge detector는 high threshold에 대한 edge에서 끊어진것으로 보이는 부분에 low threshold로 인한 값을 더해서 이어주는 방식으로 이루어진다. 여기서 high threshold와 low threshold의 값차이가 커지면, edgeDetect함수에서 edge로 판단하는 개수가 변하게 되서 코드 실행시간에 영향을준다. 실제로 low threshold를 50, high threshold를 240으로 설정했을때의 실행시간이 제일 오래걸렸다.








```
--- doCannyEx() ---

50, 240 113ms
--- doCannyEx() ---

100, 240 107ms
--- doCannyEx() ---

150, 240 94ms
```