Computer Vision hw_8

By R01922124 許彥彬

1. Intro of this homework:

   In this homework we first generate Gaussian noise image and salt-and-pepper noise image on lena.bmp. Then we use box filter, median filter, opening followed by closing, and closing followed by opening to clean the noise.

   These are the functions that are going to use in this homework.

```
void AddGaussianNoise ( const Mat&, Mat& , int );
void AddSaltAndPepperNoise ( const Mat& , Mat& , double );
void BoxFilter ( const Mat&, Mat& , int , const vector<vector<int> >& );
void MedianFilter ( const Mat &, Mat& , int  , const vector<vector<int> >& );
void Open_Close(Mat &, Mat& , Kernel *);
void Close_Open(Mat &, Mat& , Kernel *);

//previous hw
void pixel_set(Mat *, int, int ,int);
uchar pixel_get(Mat *, int ,int);
Mat * dilation(Mat *, Kernel *);
uchar di_ele(Mat *, Kernel *, int , int );
Mat * erosion(Mat *, Kernel *);
uchar ero_ele(Mat *, Kernel *, int , int );
Mat * opening(Mat *, Kernel *);
Mat * closing(Mat *, Kernel *);
```

2. Generate additive white Gaussian noise

   I. Formula:

   $$I(nim,i,j) = I(im,i,j) + amplitude * N(0,1)$$

   II. Code:

```
void AddGaussianNoise ( const Mat& src, Mat& dest, int amp ) {
        Mat temp = src.clone();
        int rows = temp.rows;
        int cols = temp.cols;
        RNG rng;
        for ( int i = 0 ; i<rows ; i++ ) {
                uchar* row_pointer = temp.ptr(i);
                for ( int j=0 ; j<cols ; j++ ) {
                        double rnd = rng.gaussian(1);
                        int v = (int)row_pointer[j]+amp*rnd;
                        row_pointer[j] = v;
                }
        }
        dest = temp;
}
```

3. Generate salt-and-pepper noise

   I. Formula:

$$I(nim,i,j) = 0 \;\; if \;\; uniform(0,1) < 0.05$$
$$I(nim,i,j) = 255 \;\; if \;\; uniform(0,1) > 1 - 0.05$$
$$I(nim,i,j) = I(im,i,j) \;\; otherwise$$
$$uniform(0,1) : random\ variable\ uniformly\ distributed\ over\ [0,1]$$
$$try\ both\ 0.05\ and\ 0.1$$

   II. Code:

```
void AddSaltAndPepperNoise ( const Mat& src, Mat& dest, double threshold ){
        Mat temp = src.clone();
        int rows = temp.rows;
        int cols = temp.cols;
        RNG rng;
        for ( int i = 0 ; i<rows ; i++ ) {
                uchar* row_pointer = temp.ptr(i);
                for ( int j=0 ; j<cols ; j++ ) {
                        double rnd = rng.uniform((double)0,(double)1);
                        if ( rnd<threshold ) {
                                row_pointer[j]=0;
                        }
                        else if ( rnd > 1-threshold ) {
                                row_pointer[j]=255;
                        }
                }
        }
        dest = temp;
}
```

4. Box filter

   I. Idea: Average the gray scale values in a given mask and set the average value to target image.

   II. Code:

```
void BoxFilter ( const Mat& src, Mat& dest , int mask_size, const vector<vector<int> >& mask ){
        Mat temp = src.clone();
        int sum = 0;
        for ( int i=0 ; i<mask_size ; i++ ) {
                for ( int j=0 ;j<mask_size; j++ ) {
                        sum+=mask[i][j];
                }
        }
        int rows = temp.rows;
        int cols = temp.cols;
        int tr,tc;
        for ( int r = 0 ; r<rows ; r++ ) {
                uchar* row_pointer = temp.ptr(r);
                for ( int c=0 ; c<cols ; c++ ) {
                        double t = 0;
                        for ( int i=0 ; i<mask_size ; i++ ) {
                                int offset_r = mask_size/2-i;
                                tr = offset_r + r;//find the neighbour in image
                                if ( tr<0 || tr>=rows ) continue;
                                uchar* mask_row_pointer =temp.ptr(tr);
                                for ( int j=0; j<mask_size ; j++ ) {
                                        int offset_c = mask_size/2-j;
                                        tc = offset_c + c;
                                        if ( tc>=0 && tc<cols ) {
                                t+=(double)mask[i][j]/(double)sum * (int)mask_row_pointer[tc];
                                        }
                                }
                        }
                        row_pointer[c] = (uchar)t;
                }
        }
        dest = temp;
}
```

5. Median Filter
   I. Idea: Set the target image using the median value of a given mask.
   II. Code:

```
void MedianFilter ( const Mat &src, Mat& dest, int mask_size , const vector<vector<int> >& mask){
        Mat temp = src.clone();
        dest = temp.clone();
        int rows = temp.rows;
        int cols = temp.cols;
        int tr,tc;
        for ( int r = 0 ; r<rows ; r++ ) {
                uchar* row_pointer = dest.ptr(r);
                for ( int c=0 ; c<cols ; c++ ) {
                        vector<int> neighbours;
                        for ( int i=0 ; i<mask_size ; i++ ) {
                                tr = mask_size/2-i+r;
                                if ( tr<0 || tr>=rows ) continue;
                                        uchar* mask_row_pointer = temp.ptr(tr);
                                for ( int j=0; j<mask_size ; j++ ) {
                                        tc = mask_size/2-j+c;
                                        if ( tc>=0 && tc<cols ) {
                                                neighbours.push_back((int)mask_row_pointer[tc]); //find neighbours
                                        }
                                }
                        }
                        sort(neighbours.begin(),neighbours.end());
                        row_pointer[c] = neighbours[neighbours.size()/2];
                }
        }
}
```

6. Opening followed by closing and closing followed by opening
   I. Idea: These two functions were discussed in homework 5. The kernel I used here is kernel octagon.
   II. Code:

```
void Open_Close(Mat &src, Mat& dest, Kernel *k){
        Mat *pic;
        pic = opening(&src, k);
        pic = closing(pic, k);
        dest = pic->clone();
}

void Close_Open(Mat &src, Mat& dest, Kernel *k){
        Mat *pic;
        pic = closing(&src, k);
        pic = opening(pic, k);
        dest = pic->clone();
}
```

7. Results:
   I. Noise Images:

Gaussian noise with amplitude 10

Gaussian noise with amplitude 30

Salt-and-pepper noise with probability 0.05

Salt-and-pepper noise with probability 0.1

II. Filter on Gaussian noise image with amplitude 10:

3by3 box filter on Gaussian noise with amplitude 10

5by5 box filter on Gaussian noise with amplitude 10

3by3 median filter on Gaussian noise with amplitude 10

5by5 median filter on Gaussian noise with amplitude 10

Opening followed by closing on Gaussian noise with amplitude 10

Closing followed by opening on Gaussian noise with amplitude 10

III. Filter on Gaussian noise image with amplitude 30:

3by3 box filter on Gaussian noise with amplitude 30

5by5 box filter on Gaussian noise with amplitude 30

3by3 median filter on Gaussian noise with amplitude 30

5by5 median filter on Gaussian noise with amplitude 30

Opening followed by closing on Gaussian noise with amplitude 30

Closing followed by opening on Gaussian noise with amplitude 30

IV. Filter on salt-and-pepper noise image with probability 0.05:

3by3 box filter on salt-pepper noise with probability 0.05

5by5 box filter on salt-pepper noise with probability 0.05

3by3 median filter on salt-pepper noise with probability 0.05

5by5 median filter on salt-pepper noise with probability 0.05

Opening followed by closing on salt-pepper noise with probability 0.05

Closing followed by opening on salt-pepper noise with probability 0.05

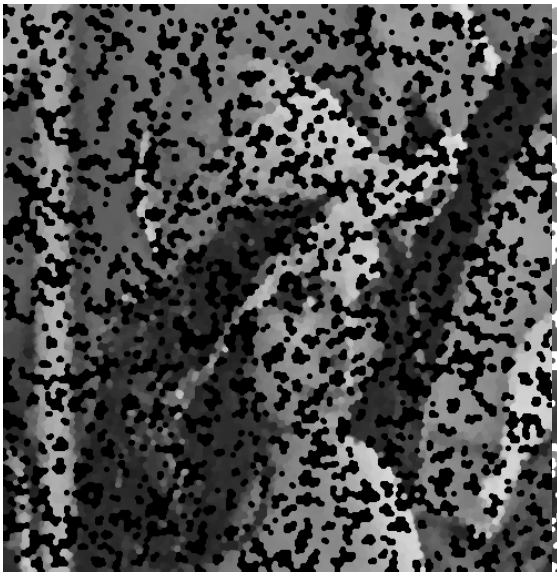
V. Filter on salt-and-pepper noise image with probability 0.1：

3by3 box filter on salt-pepper noise with probability 0.1

5by5 box filter on salt-pepper noise with probability 0.1

3by3 median filter on salt-pepper noise with probability 0.1

5by5 median filter on salt-pepper noise with probability 0.1



Opening followed by closing on salt-pepper noise with probability 0.1

Closing followed by opening on salt-pepper noise with probability 0.1

8. Appendix
    I. build_all.sh
        “sh build_all.sh” will automatically compile the code in terminal.
    II. R01922124_HW8.cpp
        source code
    III. lena.bmp
        original lena image
    IV. Many result images
    V. R01922124_HW8.pdf
        report