Lab 2 report

> Name: Xu LingXiao
> No. 2008500

How to run

```
mkdir build
cd build
cmake ../
make
./lab2
```

Task 1

The first task is load the image and show it.

```
Mat img = imread("../image.jpg");
imshow("task1 imag.jpg", img);
```

Then convert the imge to gray scale.

```
cvtColor(img, img_gray, COLOR_RGB2GRAY);
```

Finally save(write) it.

```
imwrite("../image_grayscale.jpg", img_gray);
```

Task 2

First create an `includes` folder and a `filter.h` file in it. As we used CMake build system, we need to add this line in our `CMakeLists.txt`：

```
include_directories(includes)
```

Now, we can start coding.

The task requires max and min filter, these two filters are almost the same, what we need to do is just find the max/min value in the neighbourhood of the pixel and replace it.

```
oid max_filter(Mat &src, Mat &dst, int size) {
    if(size % 2 == 0) {
        cout << "The size is even." << endl;
        return;
    }

    int p = (size - 1) / 2;

    int row = src.rows;
    int col = src.cols;
    cout << row << "," << col << endl;
    for(int r = 0; r < row; r++) {
        for(int c = 0; c < col; c++) {
            int maxt = 0;
            for(int i = r - p; i <= r + p; i++) {
                for(int j = c - p; j <= c + p; j++) {
                    if(i >= 0 && i < row && j >= 0 && j < col) {
                        if(maxt < src.at<uchar>(i, j))
                            maxt = src.at<uchar>(i, j);
                    }
                }
            }
            dst.at<uchar>(r, c) = maxt;
        }
    }
}
```

For the min filter, the same as before:

```
if(mint > src.at<uchar>(i, j))
    mint = src.at<uchar>(i, j);
```

It's really easy to know, if we use the max filter, we can remove the cables in the image(I tried the size 3 * 3 and 5 * 5, the bigger the kernel size is, the better the reslut is, the image also would be more blur).

Task 3

Task 3 is easy, just use the apis that OpenCV provided:

Gassian filter:

```
GaussianBlur(img, dst_gaussian, Size(KERNEL_SIZE, KERNEL_SIZE), 0, 0);
```

median filter

```
medianBlur(img, dst_median, KERNEL_SIZE);
```

What we need to set is the size of the kernel.

Task 4

Set the bins and the ranges, then call `calcHist`, normalize the result and show the histogram.

refer to this.

Task 5

For histogram equalization, just one line:

```
equalizeHist(img_gray, dst_eq);
```

Conclusion

The most difficlut part of the lab2 is the task3, which needs to access the pixel by hand. This task tells us how we can implement a filter by ourselves.