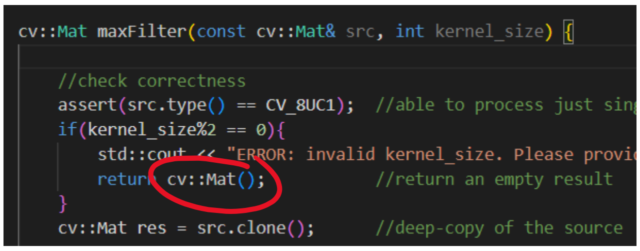
Lab 2 – Filters and histogram equalization Morselli Alberto (2102413)

-short report-

≥ **./task1 -** nothing in particular to be reported.

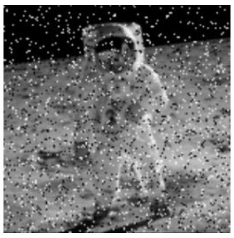
≥ ./task2 – ImageFilters.h

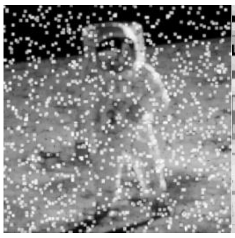
Regarding the design choices on implementation, I preferred to return a *cv::Mat* object rather than passing it as an argument because they implements filters that alters the original image sequentially, and thus they cannot process the same image passed as input, but are forced to create a new one. So, in case of error of the *kernel_size* it returns an empty *cv::Mat* that is after handled by the main.

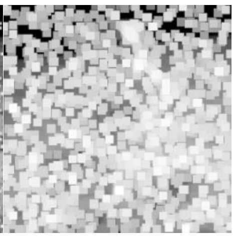


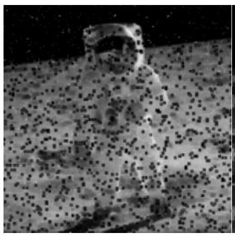
```
cv::Mat maxFilter(const cv::Mat& src, int kernel_size) {

    //check correctness
    assert(src.type() == CV_8UC1);  //able to process just sing
    if(kernel_size%2 == 0){
        std::cout << "ERROR: invalid kernel_size. Please provic
        return cv::Mat();           //return an empty result
    }
    cv::Mat res = src.clone();      //deep-copy of the source
```

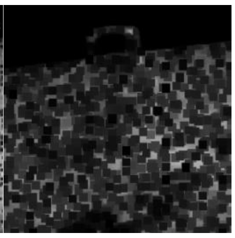
The application of the min and max filters led to pretty bad results, also because of the big amount of noise present in the image, not correctly manageable with two simple filters like those. As can be seen, as the size of the kernel increases, results of the application on *astronaut_salt_pepper.png* quickly deteriorate both for min than for max filters.


original

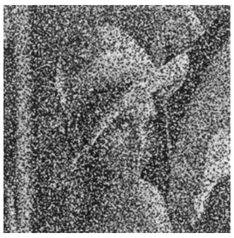

maxFilter3x3

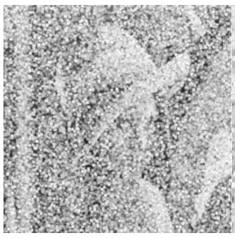

maxFilter11x11

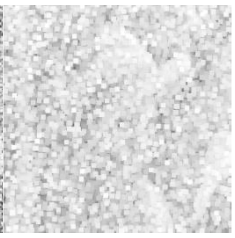

minFilter3x3

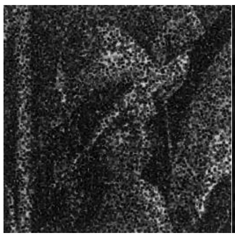

minFilter11x11

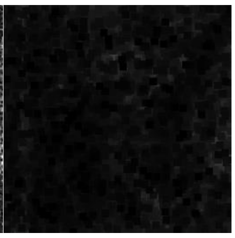
The same results hold for *lena_corrupted.png* image, as can be seen below.


original

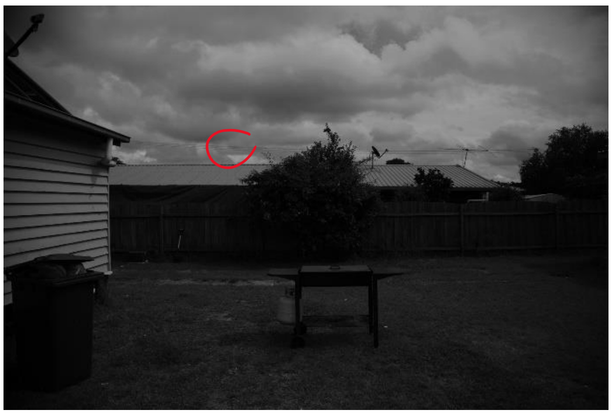

maxFilter3x3

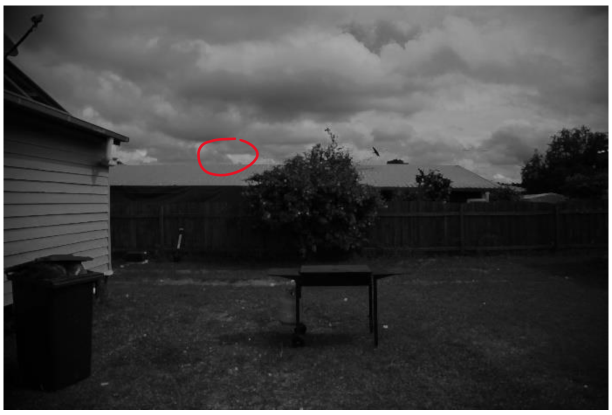

maxFilter11x11


minFilter3x3


minFilter11x11

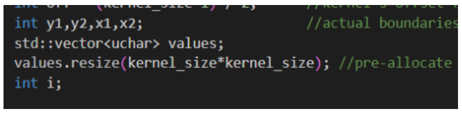
Instead for removing the electric cables in the background of *Garden_grayscale.jpg*, being them black on a lighter background (the sky) a max filter 5x5 led to really good results, without compromising the image that much.


original

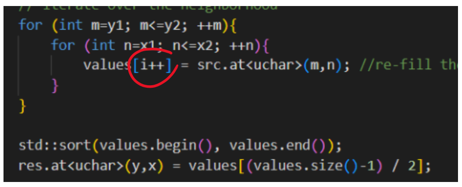

after the application of maxFilter5x5

≥ ./task3 – ImageFilters.h

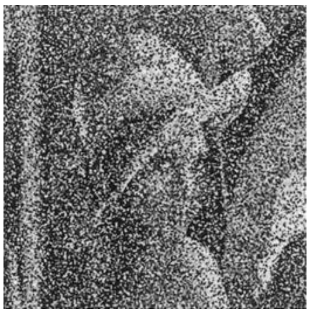
The application of the median filter showed instead a significant computational effort, appreciable on the application on the bigger image *garden_grayscale.jpg*. I then tried to optimize then function declaring outside the variables used internally in the loop cycle, trying to avoid the re-definition ad every iteration where <u>possible</u>, and resizing a priori the vector used for sorting values in the patch, trying to avoid the resizing at every iteration, but this didn't show appreciable improvements. The application is still do-able on the smaller images and on the bigger one too with small kernel sizes (<=9). The results for different kernel sizes are shown below, both for lena's and for the astronaut's image.

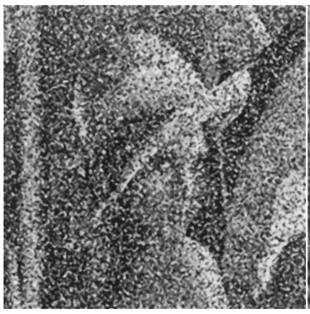


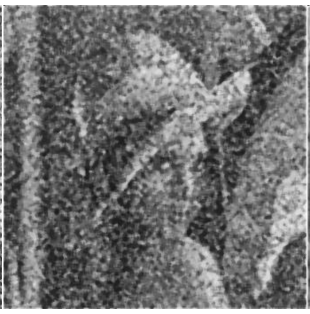
```
int y1,y2,x1,x2;                    //actual boundaries
std::vector<uchar> values;
values.resize(kernel_size*kernel_size); //pre-allocate
int i;
```

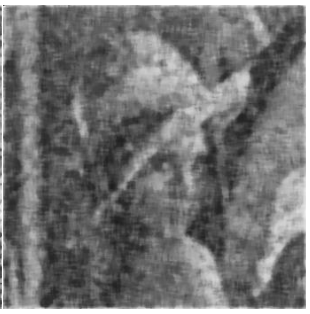


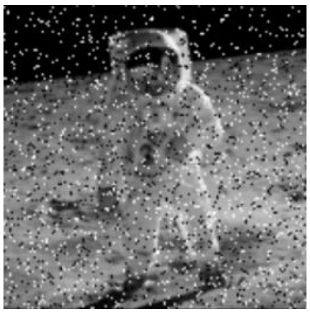
```
for (int m=y1; m<=y2; ++m){
    for (int n=x1; n<=x2; ++n){
        values[i++] = src.at<uchar>(m,n); //re-fill th
    }
}

std::sort(values.begin(), values.end());
res.at<uchar>(y,x) = values[(values.size()-1) / 2];
```

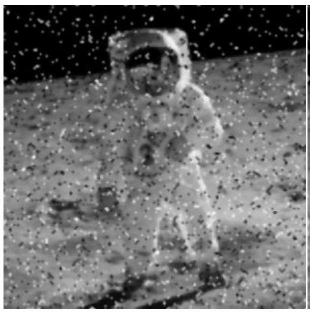

original

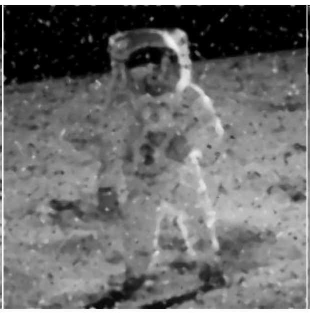

medFilter3x3

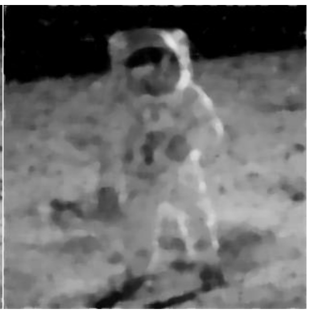

medFilter5x5


medFilter9x9


original


medFilter3x3

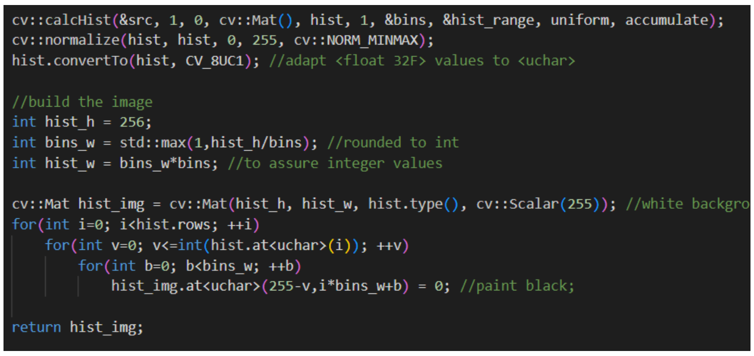

medFilter5x5


medFilter9x9

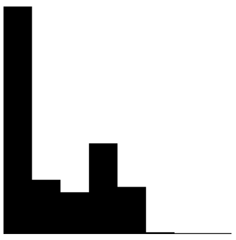
≥ ./task4 - nothing in particular to be reported.

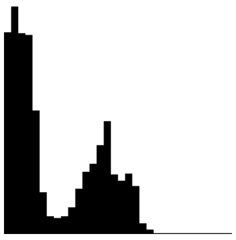
≥ ./task5

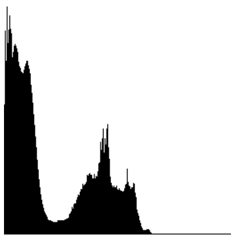
I decided to handle manually the hist obtained from *cv::calcHist* and the correspondent image generation, by first normalizing the values obtained in the interval 0-255 and after creating a 256x(≈256) white image and then writing black pixel in correspondence of the columns of the histogram. The ≈ means that, based on the amount of bins selected the width is adjusted such that every column has the same integer number of pixel per column



```
cv::calcHist(&src, 1, 0, cv::Mat(), hist, 1, &bins, &hist_range, uniform, accumulate);
cv::normalize(hist, hist, 0, 255, cv::NORM_MINMAX);
hist.convertTo(hist, CV_8UC1); //adapt <float 32F> values to <uchar>

//build the image
int hist_h = 256;
int bins_w = std::max(1,hist_h/bins); //rounded to int
int hist_w = bins_w*bins; //to assure integer values

cv::Mat hist_img = cv::Mat(hist_h, hist_w, hist.type(), cv::Scalar(255)); //white backgro
for(int i=0; i<hist.rows; ++i)
    for(int v=0; v<=int(hist.at<uchar>(i)); ++v)
        for(int b=0; b<bins_w; ++b)
            hist_img.at<uchar>(255-v,i*bins_w+b) = 0; //paint black;

return hist_img;
```



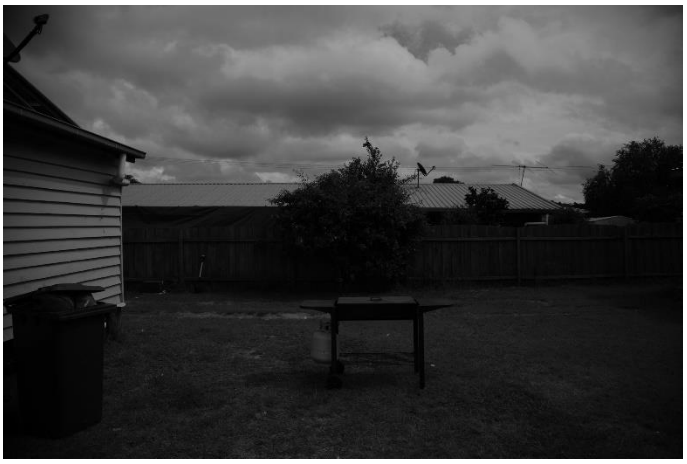


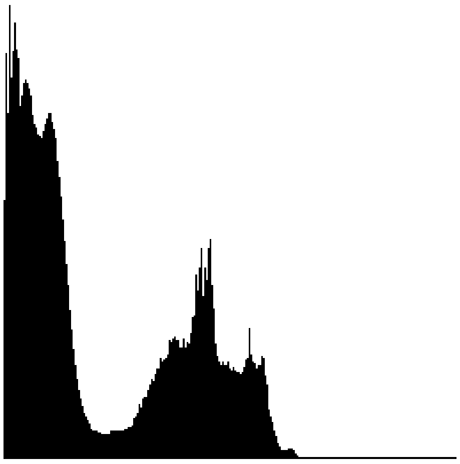


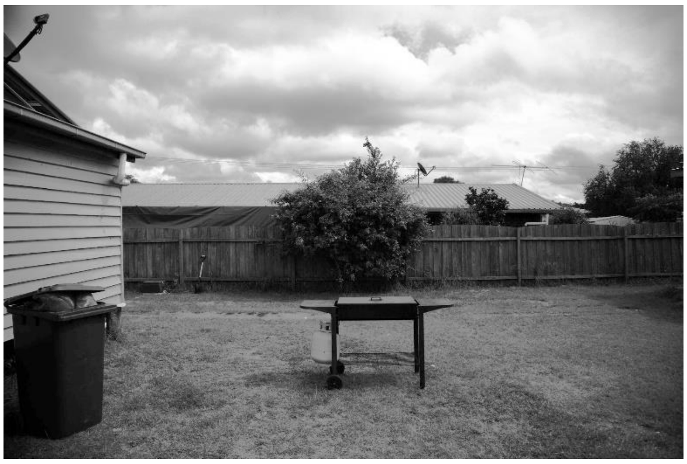
I got pretty good results, shown on the left, respectively, with 8, 32 and 255 bins. As expected, the shape is kept the same but the discretization is different.

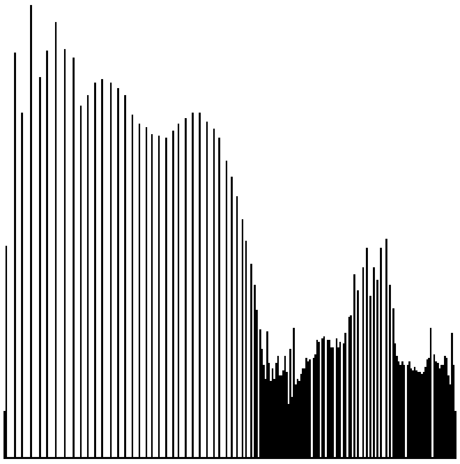
≥ ./task6

The image quality strongly increased, and the equalized histogram shows, as expected, a not perfectly flat-tization due to the discretized nature of the operation. As i noticed and as expected, the equalization process is strongly dependent from the amount of bins selected.


original


histogram


equalized image


equalized histogram

Inside the folder *Results/* can be found all the images processed and the histograms obtained.