Report lab2 Computer Vision - Matteo De Gobbi

Task 1

In the first task we just need to call the function cv::cvtColor(img, out, cv::COLOR_RGB2GRAY); which converts img to Grayscale and the result gets stored in the parameter out.

Task 2 and Task 3

In these tasks we are asked to implement max filter, min filter and median filter that operate on square kernel of odd size (provided as input).

```
void maxFilter(cv::Mat &img, cv::Mat &out, int kernel_size) {
  if (kernel_size % 2 == 0) {
    std::cout << "Error kernel must have odd size" << std::endl;
    return;
  }

  for (int i = kernel_size / 2; i < img.rows - kernel_size / 2; i++) {
    for (int j = kernel_size / 2; j < img.cols - kernel_size / 2; j++) {
      uchar temp_max = img.at<uchar>(i, j);
      for (int k = -kernel_size / 2; k ≤ kernel_size / 2; k++) {
        for (int h = -kernel_size / 2; h ≤ kernel_size / 2; h++) {
          uchar curr_pix = img.at<uchar>(i + k, j + h);
          temp_max = curr_pix > temp_max ? curr_pix : temp_max;
        }
      }
      out.at<uchar>(i, j) = temp_max;
    }
```

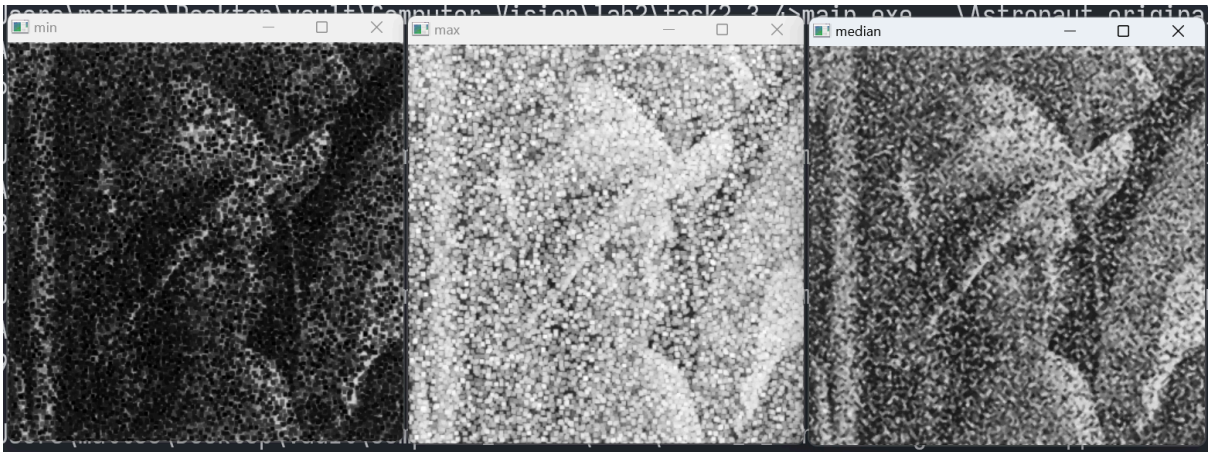
This is the code for the max filter, we can see how in the two external loops we move along the original image (indices i,j) but we ignore the first and last kernel_size/2 pixels both along rows and columns. This is one possible strategy to cope with the fact that for these pixels the kernel would go outside the bounds of the image (another possible strategy would be to pad with zeros or to reflect the pixels inside the image along the borders).
In the two internal loops (indices k,h) we look in the pixels around the current pixel (i,j) and save the current maximum found in the k,h loops, then we set the i,j pixel in the output image to the max found.
The code for the min and median filter is the same but with a different condition to save the min/median.

In the case of the median i found a neat optimization that reduced the execution of the median filter on “Garden.jpg” from 1s to around 0.1s (with -O3):

```
std::vector<uchar> v;
v.reserve(kernel_size * kernel_size);
for (int i = kernel_size / 2; i < img.rows - kernel_size / 2; i++) {
  for (int j = kernel_size / 2; j < img.cols - kernel_size / 2; j++) {
    v.clear();
    for (int k = -kernel_size / 2; k ≤ kernel_size / 2; k++) {
      for (int h = -kernel_size / 2; h ≤ kernel_size / 2; h++) {
        uchar curr_pix = img.at<uchar>(i + k, j + h);
        v.push_back(curr_pix);
      }
    }
    std::nth_element(v.begin(), v.begin() + v.size() / 2, v.end());
    out.at<uchar>(i, j) = v[v.size() / 2];
  }
```

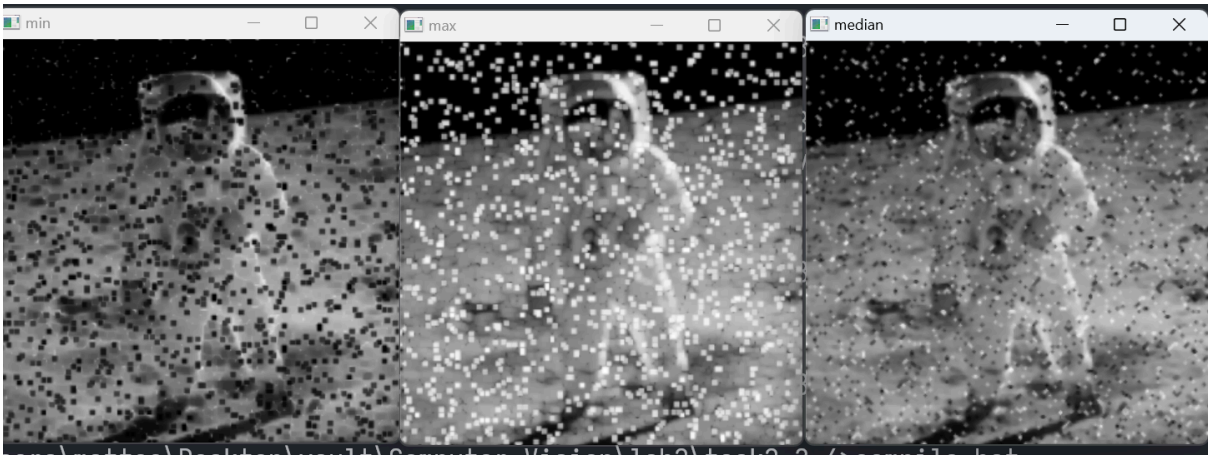
We reserve the memory for the vector just once outside the loops and only empty it with the clear function once we have found the median. In this way we only need to allocate the vector once (no reallocations due to the size of v increasing) making the function faster.

To evaluate the effect of the filter we can use them on the “Lena_corrupted.png”. The best kernel size I found was 3x3, any bigger than that and the image becomes very blurry and still doesn’t remove the noise well since the original image is very noisy:

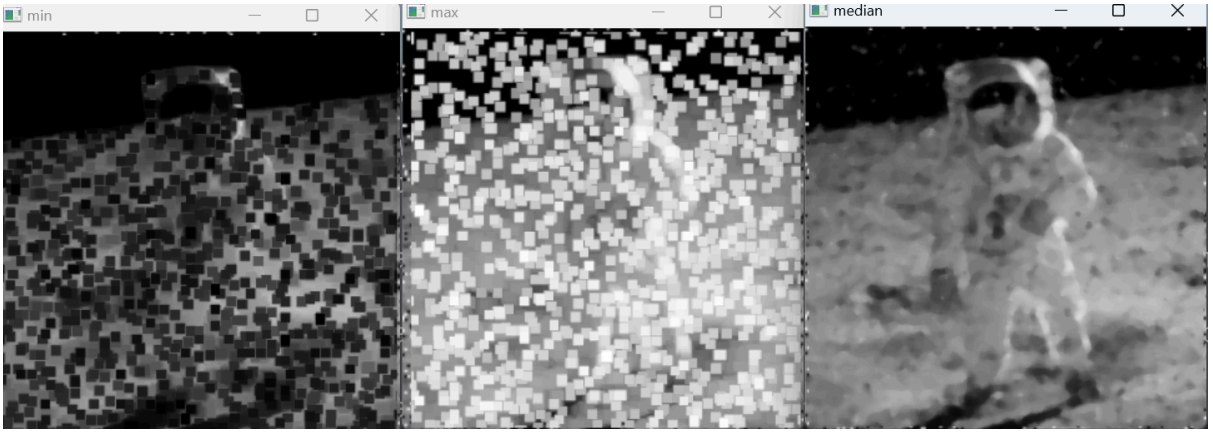


In this image we can see the effects of min, max and median filtering, as seen in the lecture notes the min filter removes salt noise but increases pepper noise, the max filter does the opposite, the median filter performs the best (Lena’s face is more visible) but still the image is corrupted by noise.

The same considerations can be made for the astronaut image, using kernel 3x3:



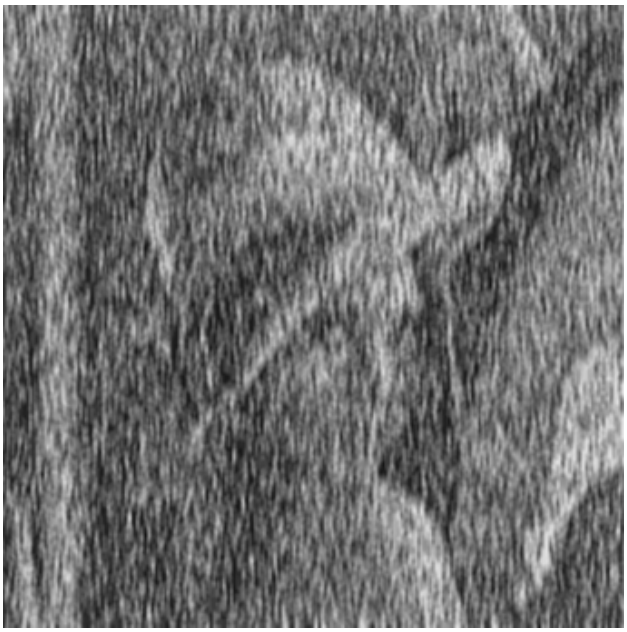
If instead we use the 7x7 kernel with the astronaut we are able to remove most of the salt and pepper noise with the median filter but the image becomes blurry (min and max are even worse in the 7x7 version than in the 3x3 version).



Lastly for the garden image we can use the max filter to remove the electrical cables since they are dark and are surrounded by lighter clouds, the smallest kernel size we can use in order to completely remove the cables is 5x5, if we use a smaller kernel the cables are still visible, if we use a bigger kernel the image has some visual artifacts. Even with the 5x5 kernel we can see how the rocks in the grass become white squares and are more visible when compared with the original image.

Task 4

For applying Gaussian Blur we can use the opencv function, cv::GaussianBlur(gray, out_blur, cv::Size(1, 9), 7); using this function call we get these results:



In the case of the Garden image we can see how the blur removes almost completely the electric cables in the background but also the rest of the image is more blurry, in this aspect the max filter is still better:

Task 5 and Task 6

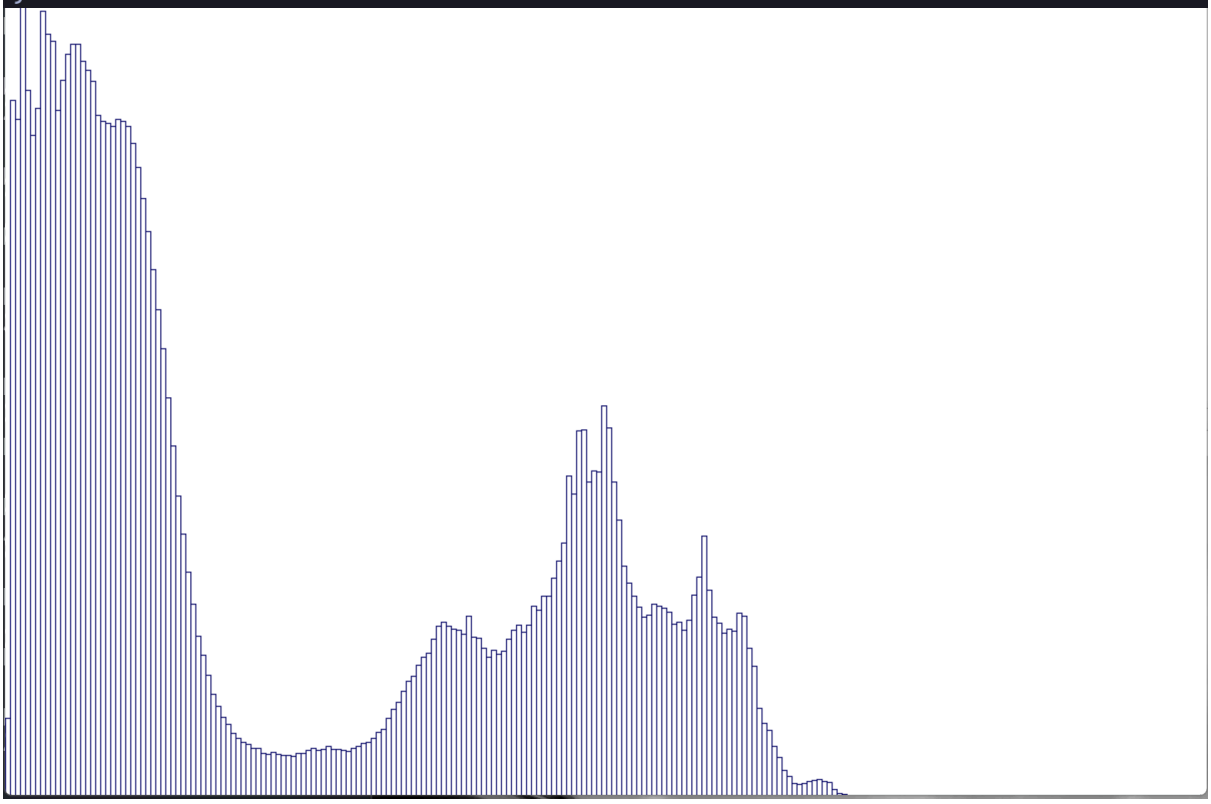
To evaluate the histogram we can use the calcHist function provided by opencv:

```
constexpr float range[2] = {0, 256};
const float *histRange[2] = {range};
constexpr int bins = 256;
constexpr bool uniform = true, accumulate = false;
calcHist(&originalGs, 1, 0, cv::Mat(), hist, 1, &bins, histRange, uniform,
         accumulate);
```

The histogram will be stored in the originalGs Mat passed by address.

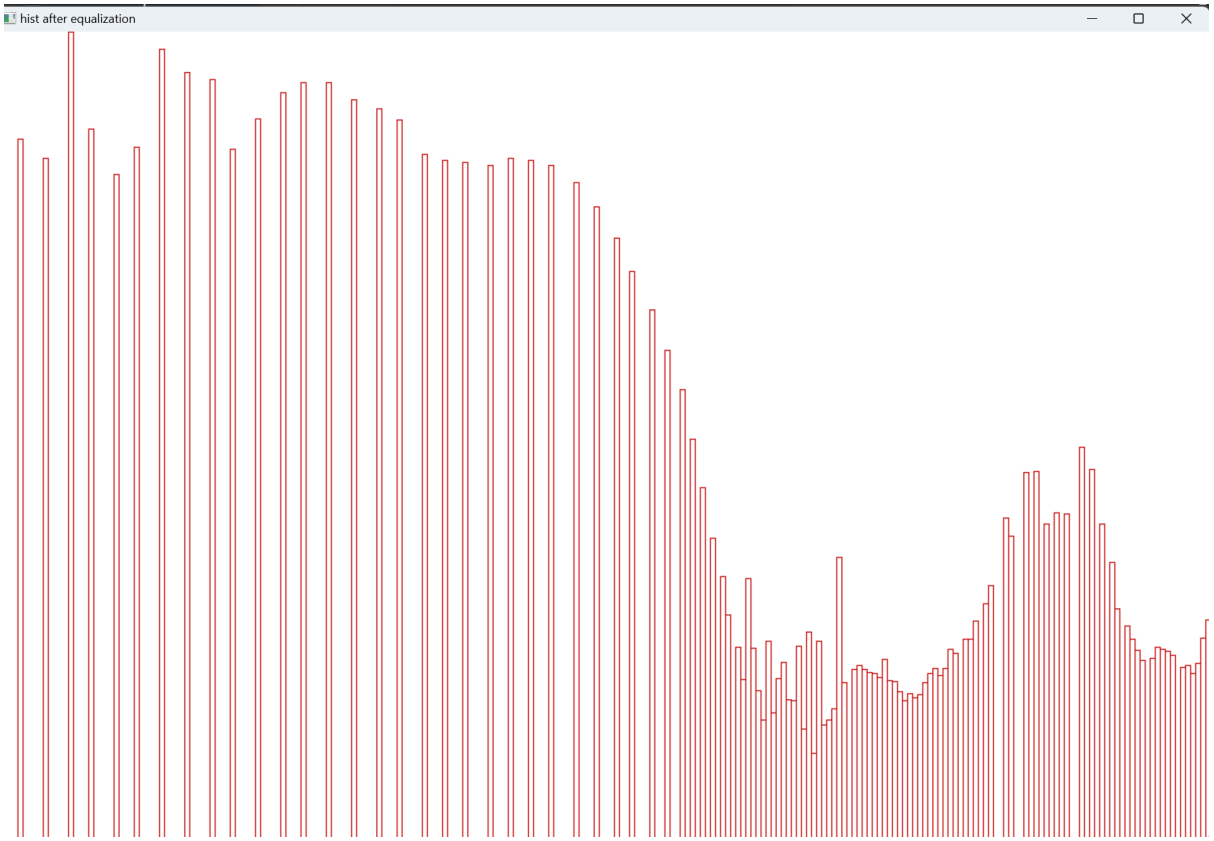
To draw the histogram I wrote an helper function:

```
void draw_hist(const cv::Mat &hist, cv::Mat &histImage, int histSize, int width,
               int height, cv::Scalar color) {
  int bin_width = cvRound((double)width / histSize);
  cv::normalize(hist, hist, 0, histImage.rows, cv::NORM_MINMAX, -1, cv::Mat());
  for (int i = 1; i < histSize; i++) {
    using namespace cv;
    rectangle(
        histImage,
        Point(bin_width * (i - 1), height - cvRound(hist.at<float>(i - 1))),
        Point(bin_width * (i), height), color, 1);
  }
}
```



We can see that it's very unbalanced: the last bins (corresponding to lighter colors) are completely empty.

To make the histogram more balanced (and enhance the contrast) we can apply histogram normalization with the function cv::equalizeHist(originalGs, equalizedGs);
After equalization the histogram is more balanced, spanning all the range of the 256 bins:



The effect on the image is that the image becomes lighter and the contrast increases, we can see this in the original vs equalized histogram images: