Intensity transformations and linear filtering

RoVi1 vision exercise 2

Kim Lindberg Schwaner

University of Southern Denmark
Faculty of Engineering
The Maersk Mc-Kinney Moller Institute

September 14, 2017

Previous exercise

- Draw a black rectangle.
- Draw colored rectangle.
- Convert image to grayscale.

Black rectangle

To “paint” image pixels, we basically have to

- select the right pixels; and
- change their value.

Black rectangle “method 2”

- Select image pixels with the `Mat::at()` method.
- Set the intensity of all three channels to zero “in one go”.

```
67  void black_box_2(const cv::Mat& src)
68  {
69      cv::Mat img = src.clone();
70
71      for (int i = Y1; i < Y2; i++) { // Iterate rows
72          for (int j = X1; j < X2; j++) { // Iterate columns
73              img.at<cv::Vec3b>(i, j) = cv::Vec3b::all(0); // Set all 3 channels
                    to zero
74          }
75      }
76
77      cv::imshow("Black box 2", img);
78      cv::waitKey();
79  }
```

Listing 1: ex1.cpp

Black rectangle “method 5”

- Select a region of interest (ROI) in the image (remember `cv::Mat` copy constructor?).
- Use built-in OpenCV magic to set all pixels in the ROI at once (behind the scenes pixels will most likely still be iterated over).

```
110 void black_box_5(const cv::Mat& src)
111 {
112     cv::Mat img = src.clone();
113
114     cv::Mat roi(img, cv::Rect(X1, Y1, X2-X1, Y2-Y1));
115     roi = cv::Vec3b::all(0);
116
117     cv::imshow("Black box 5", img);
118     cv::waitKey();
119 }
```

Listing 2: ex1.cpp

Colored rectangle

- Select ROI
- Set pixel values to something other than all-zero (remember there are more than one channel).

```
122 void colored_box(const cv::Mat& src)
123 {
124     cv::Mat img = src.clone();
125
126     cv::Mat roi(img, cv::Rect(X1, Y1, X2-X1, Y2-Y1));
127     roi = cv::Vec3b(0, 255, 255); // Paint the ROI yellow
128
129     cv::imshow("Colored box", img);
130     cv::waitKey();
131 }
```

Listing 3: ex1.cpp

Grayscale

- Use `cv::cvtColor` to convert to another color space.
- Three channels are mapped to one (how?).
- Information is lost.

```
134 void grayscale(const cv::Mat& img)
135 {
136     cv::Mat gray;
137     cv::cvtColor(img, gray, cv::COLOR_BGR2GRAY);
138     // we can also load images as grayscale directly with
139     // cv::imread(filename, cv::IMREAD_GRAYSCALE);
140 
141     std::cout << "Grayscale type: " << type2str(gray.type()) << std::endl;
142 
143     for (int i = Y1; i < Y2; i++) {
144         for (int j = X1; j < X2; j++) {
145             gray.at<uchar>(i, j) = 0; // Note that elements are single-channel
146         }
147     }
148 
149     cv::imshow("Grayscale with box", gray);
150     cv::waitKey();
151 }
```

Listing 4: ex1.cpp

Exercise 2

1. Apply intensity transformation to an image.
2. Calculate and visualize histograms.
3. Perform histogram equalization.
4. Perform linear filtering.

Intensity transformations

- Very close to what we did in exercise 1.
- Pay attention to the case where an increased intensity is larger than what the image value type can hold (e.g. `uchar` can hold values in $[0, 255]$)

Intensity transformation result

Figure: Original.

Figure: No clamping.

Figure: With clamp.

Histogram calculation and visualization

- OpenCV's `calcHist` function can do it for you (and you may use it).
- Visualize by drawing "bars" in a new image, according to the calculated histogram.
  - With 256 bins, and easy approach is to use a 256 px wide image and fill columns to height h according to bin value.
  - $h(x_i) = N \frac{x_i}{x_{\max}}$, where x_i is the i'th bin value.
- For help, examine the OpenCV histogram tutorial.

Histogram equalization

- Contrast adjustment using the histogram of the image.
- “Spreads out” the most frequently occurring intensity values.

Histogram equalization

$$T(k) = \frac{1-L}{MN} \sum_{j=0}^{k} n_j \qquad k = 0, 1, \ldots, L-1$$

where n_j is the number of pixels with intensity level j for $j = 0, 1, \ldots, L-1$

Before and after histogram eq.

Figure: Original image.

$\xrightarrow{T(k)}$

Figure: Equalized image.

Before and after histogram eq.

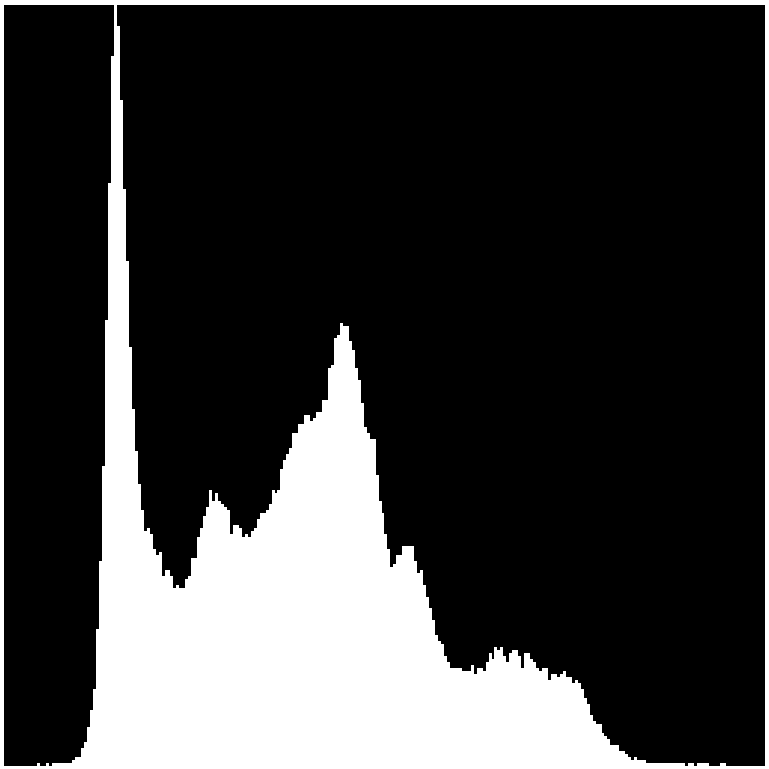

Figure: Histogram of original image.

$\xrightarrow{T(k)}$

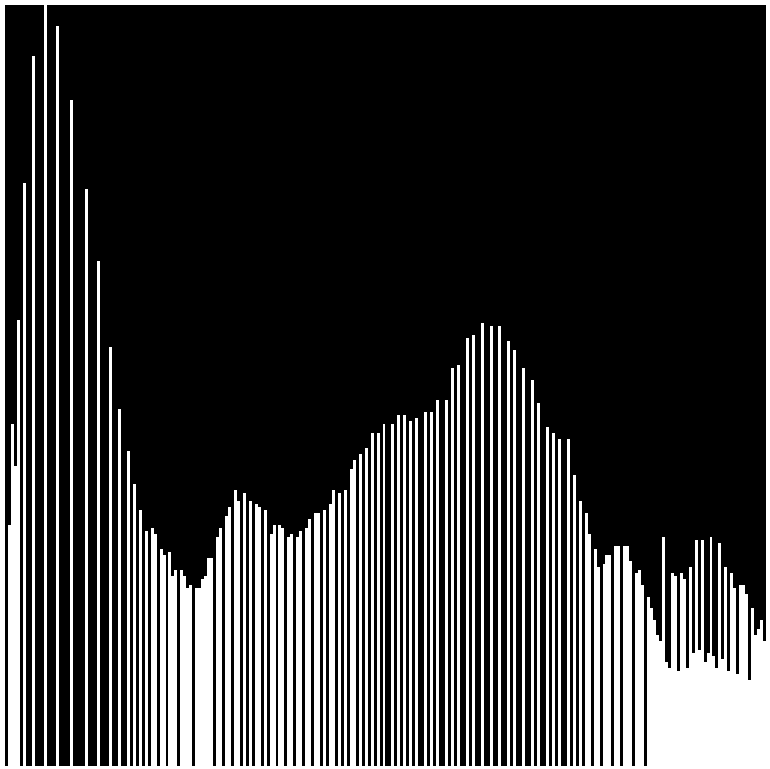

Figure: Histogram of equalized image.

Linear filtering

- First try with `cv::filter2D`
- Then make your own filtering function that implements the following steps:
  - For each pixel, sort the 9 pixel values of the 3-by-3 neighborhood (including the pixel itself).
  - Take the mean of the three middle values and store that as the filtered pixel value.
  - Since border pixels do not have 9 neighbors, you can either use padding or offset your algorithm to avoid the borders.
  - Help: `cv::copyMakeBorder` and `cv::sort`.