UNIVERSITÀ DEGLI STUDI DI PADOVA

Laboratory 3 – Image Equalization, Histograms, Filters

Alberto Pretto

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

INTELLIGENT AUTONOMOUS SYSTEMS LAB

Lab Recap

- Learn how to compute histrograms with OpenCV
- Try to manipulate the histograms of RGB and HSV images
- Experiment with various types of filters

Histograms in OpenCV

```
void cv::calcHist( const Mat *images,
                   int nimages,
                   const int *channels,
                   InputArray mask,
                   OutputArray hist,
                   int dims,
                   const int *histSize,
                   const float **ranges,
                   bool uniform = true,
                   bool accumulate = false )
```

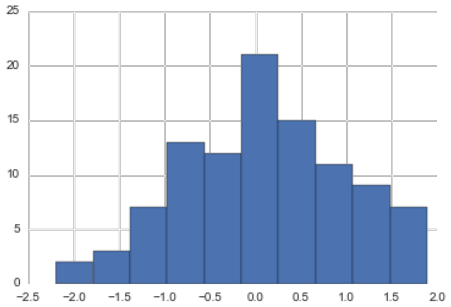
- We assume here `uniform == true`
- https://docs.opencv.org/3.4/d6/dc7/group__imgproc__hist.html#ga4b2b5fd75503ff9e6844cc4dcdaed35d

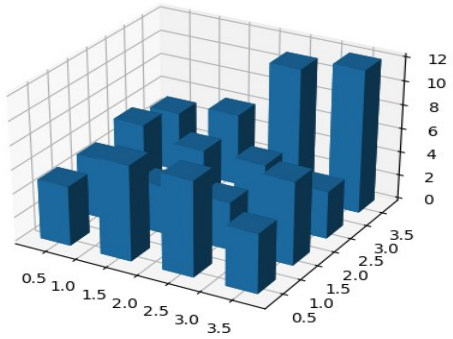
Histograms in OpenCV

- `Mat *images`: vector of source images, same size and depth, can have an arbitrary number of channels
- `int nimages`: number of source images

Histograms in OpenCV

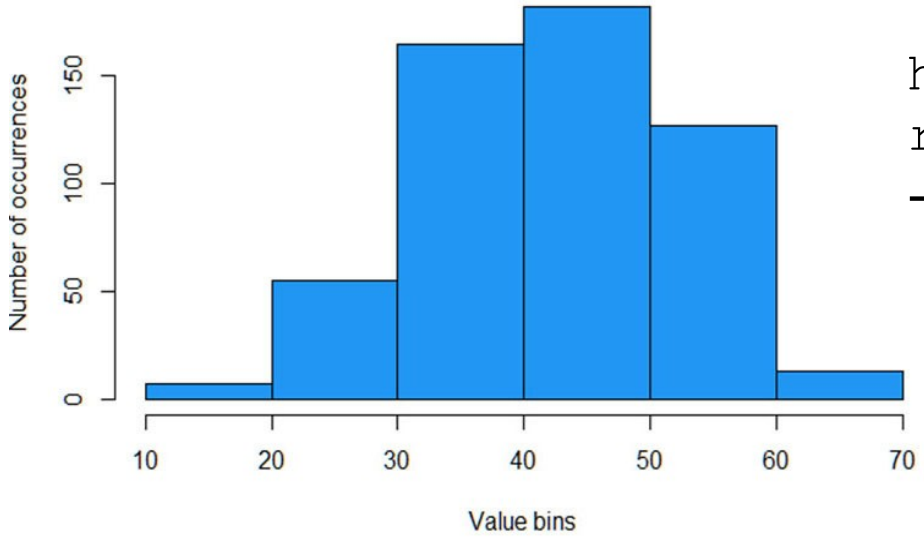
- `const int *channels`: vector of dimensions used to compute the histogram (be careful with the numbering, see docs!)
- `InputArray mask`: optional mask (just use cv::Mat() if no mask is required)
- `OutputArray hist`: Output histogram, usually just a cv::Mat with deduced size and number of channels (e.g., see dims).
- `int dims`: number of histogram dimensions (e.g., 1 for our histograms)





Histograms in OpenCV

- `const int *histSize`: vector that for each dimension reports the **histogram size**
- `const float **ranges`: if `uniform == true` (default behavior), vector of couples representing the limits for the values to be measured, one for each histogram dimension → the **bin size** is deduced!




```
histSize = {6}
ranges = {{10,70}}
```

→ bin size = 10

Example

- Compute a 2D Hue-Saturation histogram

```
cv::cvtColor(src, hsv, COLOR_BGR2HSV);
// Hue is the first channel (idx 0),
// saturation is the second one (index 1)
int channels[] = {0, 1};
// Quantize the hue to 30 levels and
// the saturation to 32 levels
int hue_bins = 30, sat_bins = 32;
int histSize[] = {hue_bins, sat_bins};
// hue varies from 0 to 179
float hue_range[] = { 0, 180 };
// saturation varies from 0 (black-gray-white) to 255
float sat_range[] = { 0, 256 };
const float* ranges[] = { hue_range, sat_range };
cv::Mat hist;

cv::calcHist( &hsv, 1, channels, Mat(), //do not use mask
              hist, 2, histSize, ranges,
              true, // the histogram is uniform
              false );
```

Histogram Equalization 1/2

1) Load a color image. Color images in OpenCv are saved as a matrix of triplets b, g, r (in this order), representing the three-dimensional RGB coordinates of the pixel color.
2) To compute an histogram for each color channel, you need to split the 3-channels color images into three one-channels images (function `cv::split()`), each one holding the intensities for a specific color channel.

Histogram Equalization 1/2

3) Use `cv::calcHist()` to compute an histogram *for each channel*.
4) Use `cv::equalizeHist()` to equalizes the R,G and B images.
5) Using the equalized channels, reassemble an BGR image by using the function `cv::merge()`
6) Visualize the input and the equalized image and the histograms of its channels by exploiting the provided helper function (`showHistogram(std::vector<cv::Mat>& hists)`).

Histogram Equalization 2/2

1) Convert the input input image into the HSV color space (`cv::cvtColor()` function, with `cv::COLOR_BGR2HSV` flag), split the resulting image into 3 single-channel images (H,S,V channel)
2) Equalize only one channel between H,S,V , and re-assemble the HSV image with one equalized channel.
3) Switch back to the RGB color space (`cv::COLOR_HSV2BGR`) and visualize the resulting image
4) Visualize the input and the equalized image in two different windows.
5) Repeat step from 1 for each H,S,V channel.

Image Filtering

- Load the input image and prepare a copy of such image for each type of filter you are going to apply. These copies will be used to store the filtered image.
- Initialize a different named window for each image (function `cv::namedWindow()`) and associate to each window a number of trackbars (one trackbar for each parameter to be tuned, function `cv::createTrackbar()`).

Image Filtering

The `cv::createTrackbar()` functions take as parameters, among others:

- `value`: Pointer to an integer variable whose value reflects the position of the slider. This variable should hold the value of the parameter to be changed.
- `count`: Maximal position of the slider: put here some reasonable value. **The minimal position is always 0**.
- `onChange`: Pointer to the **function** to be called every time the slider changes position. This function should be defined as:

```
void doSomethingOnChange(int pos, void *userdata);
```

where the first parameter is the trackbar position and the second parameter is the user data

- `userdata`: User data that is passed **as is** to the callback
- https://docs.opencv.org/4.5.2/d7/dfc/group__highgui.html#gaf78d2155d30b728fc413803745b67a9b

Image Filtering

- As user data, you may create a set of classes, e.g. `SomeFilter`, derived from the provided class `Filter`, one for each filter type, that includes both filter parameters, images and method to apply the related filter, and pass these classes to the callback through the param `void *userdata`

  → Remember to cast back from `void*` to `SomeFilter*` .
- Visualize each resulting filtered image (one for each type of filter) in the related named window (function `cv::imshow()`).

UNIVERSITÀ DEGLI STUDI DI PADOVA

Laboratory 3 – Image Equalization, Histograms, Filters

Alberto Pretto

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

INTELLIGENT AUTONOMOUS SYSTEMS LAB