Lesson 3:
Color Spaces

Jorge Beltrán, Arturo de la Escalera

Perception Systems

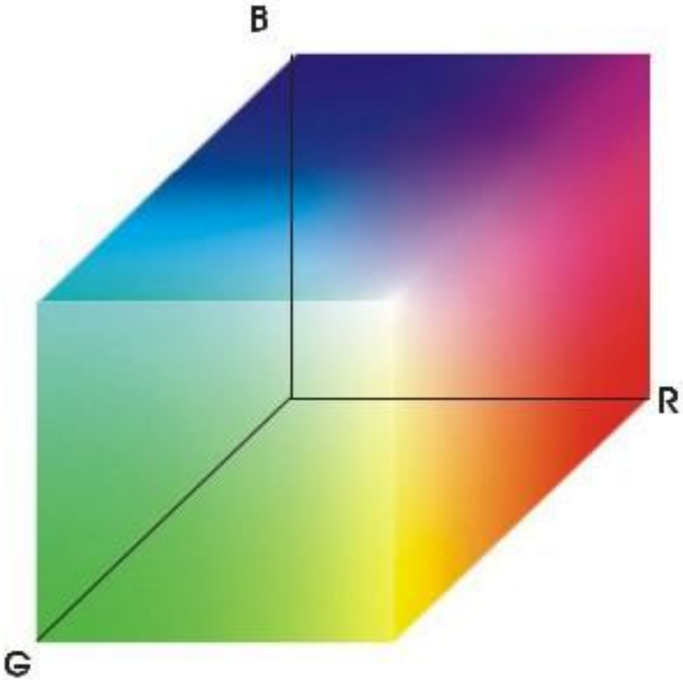
Course 2019-2020

Lesson 3: Color spaces

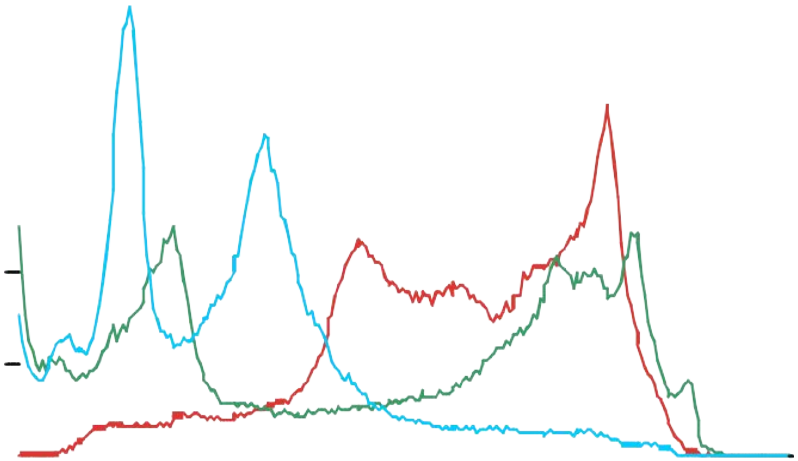
- Color spaces
  - RGB
  - HSV
- Advantages
  - One of the most important features of objects in the environment
- Disadvantages
  - High computational and memory costs

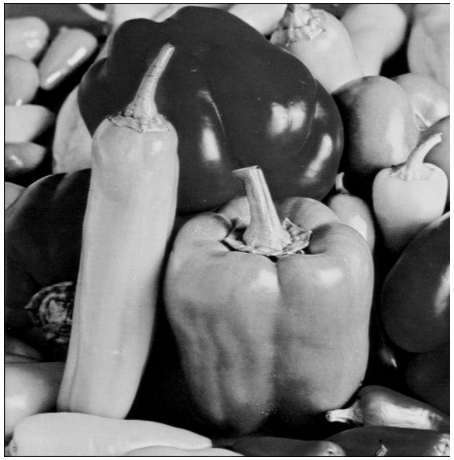
RGB color space

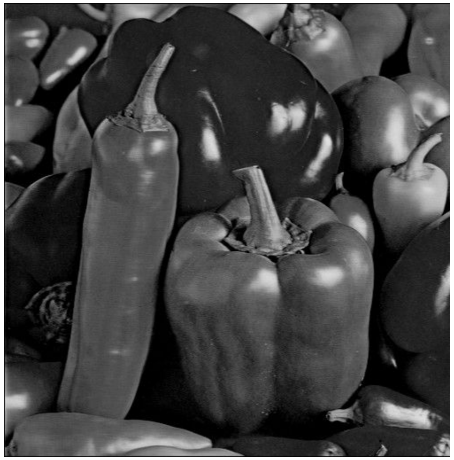
- Each pixel has three values:
  - Red
  - Blue
  - Green
- Advantage
  - Intuitive
- Disadvantage
  - Mixed information



RGB color space







Exercise 1. Display the number of channels of an image

- Load a grayscale and a color image
- Get the number of channels of each of them
- Display both images
- Print the number of channels of the two images
- Free memory

§ channels()

`int cv::Mat::channels ( ) const`

Returns the number of matrix channels.

The method returns the number of matrix channels.

Examples:

samples/cpp/pca.cpp.

Lesson 3: Color spaces

```
Original Image has: 3 channels
Grayscale Image has: 1 channels
```

Lesson 3: Color spaces

```
#include "opencv\cv.hpp"
#include <iostream>

using namespace cv;
using namespace std;

int main(int argc, char* argv[])
{
    // initialize object
    Mat original_image, grayscale_image;

    // load image from disk
    original_image = imread("lena.jpg", IMREAD_COLOR);

    // load image from disk as gray scale
    grayscale_image = imread("lena.jpg", IMREAD_GRAYSCALE);

    // check if the image is available
    if (!original_image.data || !grayscale_image.data)
    {
        cout << "Error in loading the image!" << endl;
    }
    else
    {
        // print out number of channels colors in the command window
        cout << "Original Image has: " << original_image.channels() << " channels" << endl;
        cout << "Grayscale Image has: " << grayscale_image.channels() << " channels" << endl;

        // create window canvas to show image
        namedWindow("L03_E01_Original", CV_WINDOW_AUTOSIZE);
        namedWindow("L03_E01_Grayscale", CV_WINDOW_AUTOSIZE);

        // add the image to the window
        imshow("L03_E01_Original", original_image);
        imshow("L03_E01_Grayscale", grayscale_image);

        // wait till a key is pressed
        waitKey(0);

        // free memory
        destroyAllWindows();
    }
}
```

Exercise 2. Split the channels of the color image and save each of them as a separate grayscale image.

- Load color image (default in BGR)
- Split the channels into different images
- Compute the histogram for each channel
- Display the original image, and each of the channels
- Free memory

Exercise 2. Split the channels of the color image and save each of them as a separate grayscale image.

split

Divides a multi-channel array into several single-channel arrays.

C++: void `split`(const Mat& **src**, Mat* **mvbegin**)

C++: void `split`(InputArray **m**, OutputArrayOfArrays **mv**)

Parameters:
- **src** – input multi-channel array.
- **mv** – output array or vector of arrays; in the first variant of the function the number of arrays must match `src.channels()`; the arrays themselves are reallocated, if needed.

The functions `split` split a multi-channel array into separate single-channel arrays:

The method *split* requires an output array to store the resulting single-channel images.

Exercise 2. Split the channels of the color image and save each of them as a separate grayscale image.

cvtColor

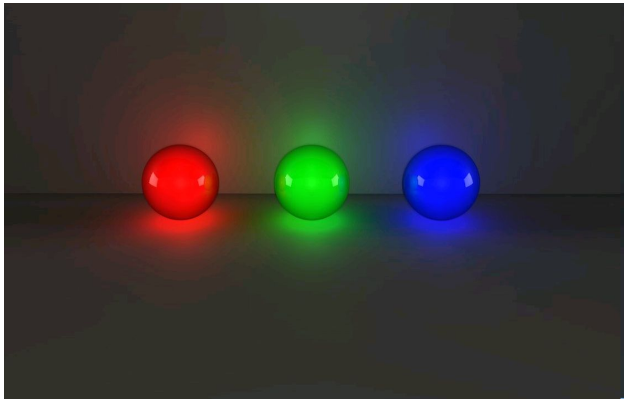
Converts an image from one color space to another.

C++: void `cvtColor`(InputArray **src**, OutputArray **dst**, int **code**, int **dstCn**=0)

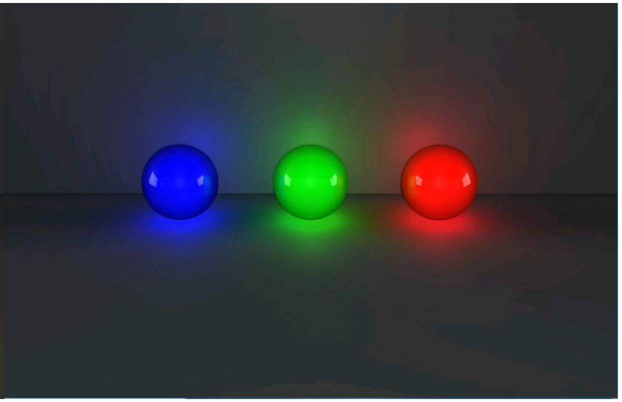
Parameters:
- **src** – input image: 8-bit unsigned, 16-bit unsigned (`CV_16UC...`), or single-precision floating-point.
- **dst** – output image of the same size and depth as `src`.
- **code** – color space conversion code (see the description below).
- **dstCn** – number of channels in the destination image; if the parameter is 0, the number of the channels is derived automatically from `src` and `code` .

The function converts an input image from one color space to another. In case of a transformation to-from RGB color space, the order of the channels should be specified explicitly (RGB or BGR). Note that the default color format in OpenCV is often referred to as RGB but it is actually BGR (the bytes are reversed). So the first byte in a standard (24-bit) color image will be an 8-bit Blue component, the second byte will be Green, and the third byte will be Red. The fourth, fifth, and sixth bytes would then be the second pixel (Blue, then Green, then Red), and so on.

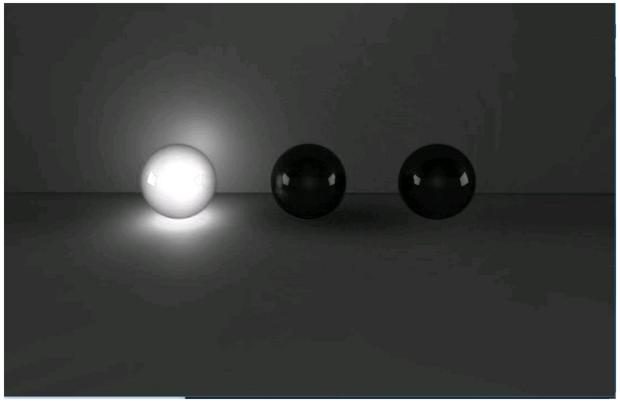
Exercise 2. Split the channels of the color image and save each of them as a separate grayscale image.



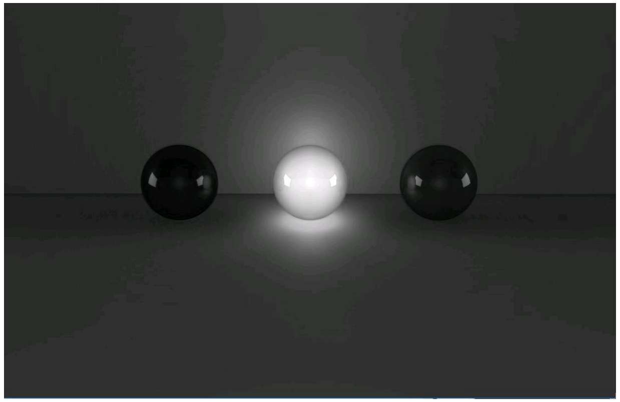
RGB



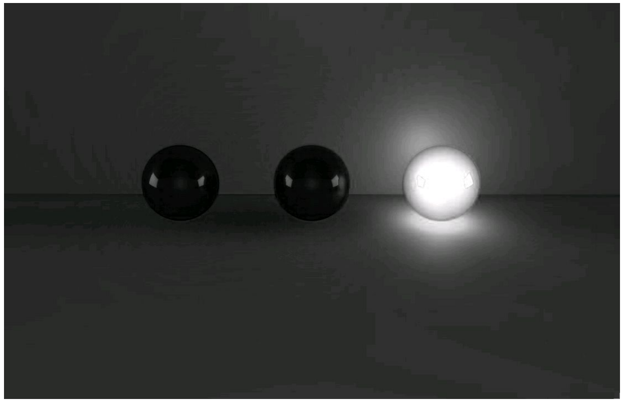
BGR



R



G



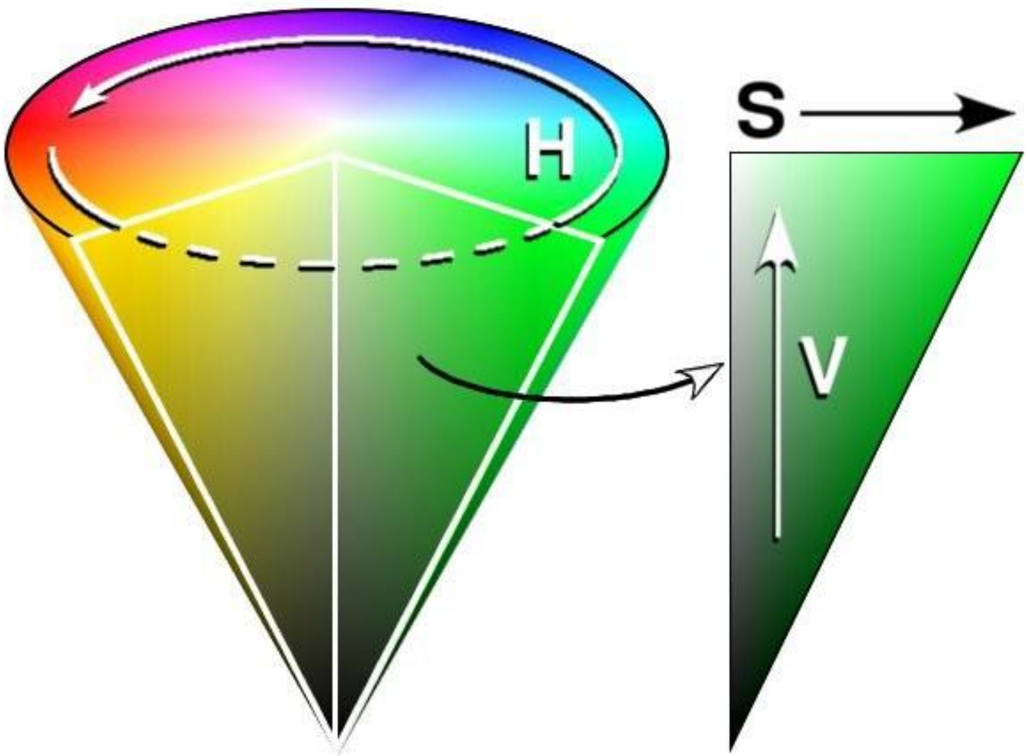
B

Exercise 2. Split the channels of the color image and save each of them as a separate grayscale image.

```cpp
#include "opencv\cv.hpp"
#include <iostream>

using namespace cv;
using namespace std;

int main(int argc, char* argv[])
{
    // initialize object
    Mat original_image, modified_image;

    // load image from disk
    original_image = imread("lightballs.jpg", IMREAD_COLOR);

    // check if the image is available
    if (!original_image.data)
    {
        cout << "Error in loading the image!" << endl;
    }
    else
    {
        // convert the image from BGR to RGB
        cvtColor(original_image, modified_image, CV_BGR2RGB);

        // create a channel for each color with image size
        Mat red_channel(modified_image.size(), CV_8UC1);
        Mat green_channel(modified_image.size(), CV_8UC1);
        Mat blue_channel(modified_image.size(), CV_8UC1);
        // create an array of all channels
        Mat channels_array[] = {red_channel, green_channel, blue_channel};
        // split the image to separate channels
        split(modified_image, channels_array);

        // create window canvases to show images
        namedWindow("L03_E02_Original", CV_WINDOW_AUTOSIZE);
        namedWindow("L03_E02_Modified", CV_WINDOW_AUTOSIZE);
        namedWindow("L03_E02_Red_Color", CV_WINDOW_AUTOSIZE);
        namedWindow("L03_E02_Green_Color", CV_WINDOW_AUTOSIZE);
        namedWindow("L03_E02_Blue_Color", CV_WINDOW_AUTOSIZE);

        // add images to windows
        imshow("L03_E02_Original", original_image);
        imshow("L03_E02_Modified", modified_image);
        imshow("L03_E02_Red_Color", red_channel);
        imshow("L03_E02_Green_Color", green_channel);
        imshow("L03_E02_Blue_Color", blue_channel);

        // wait till a key is pressed
        waitKey(0);

        // free memory
        destroyAllWindows();
    }
}
```

HSV color space

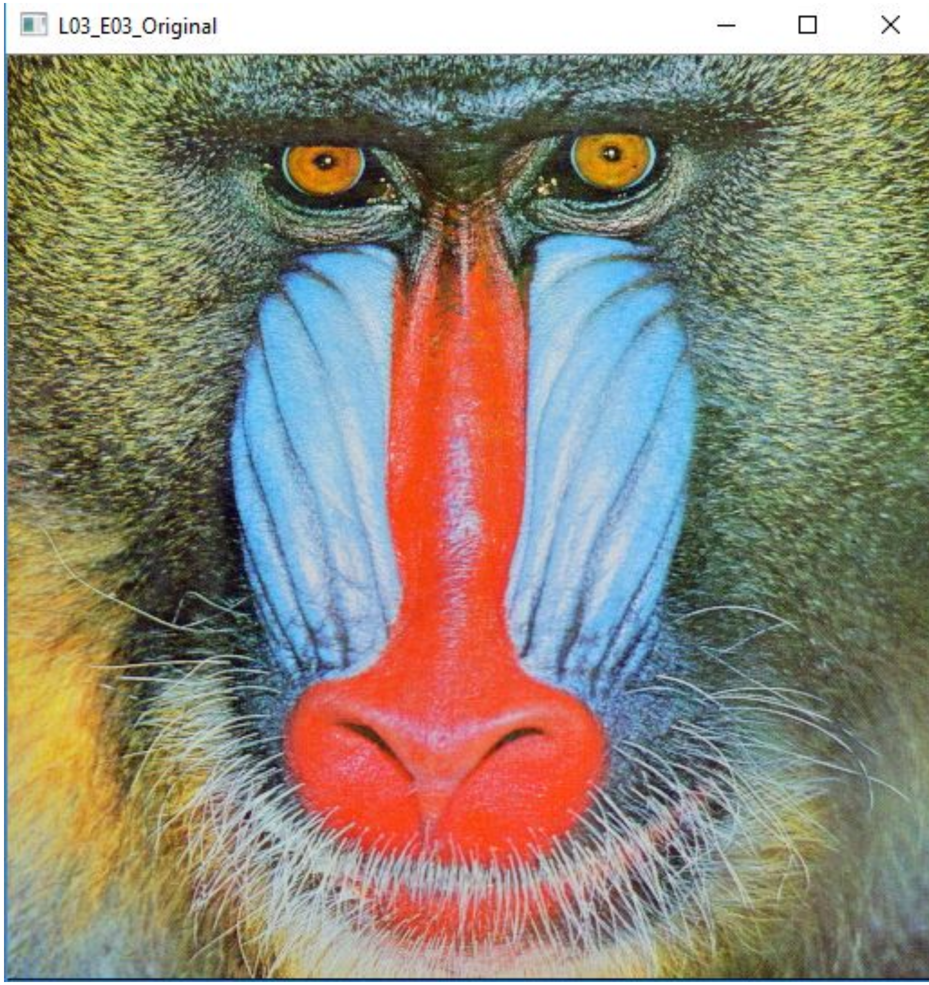
- Each pixel has three values:
  - Hue
  - Saturation
  - Value

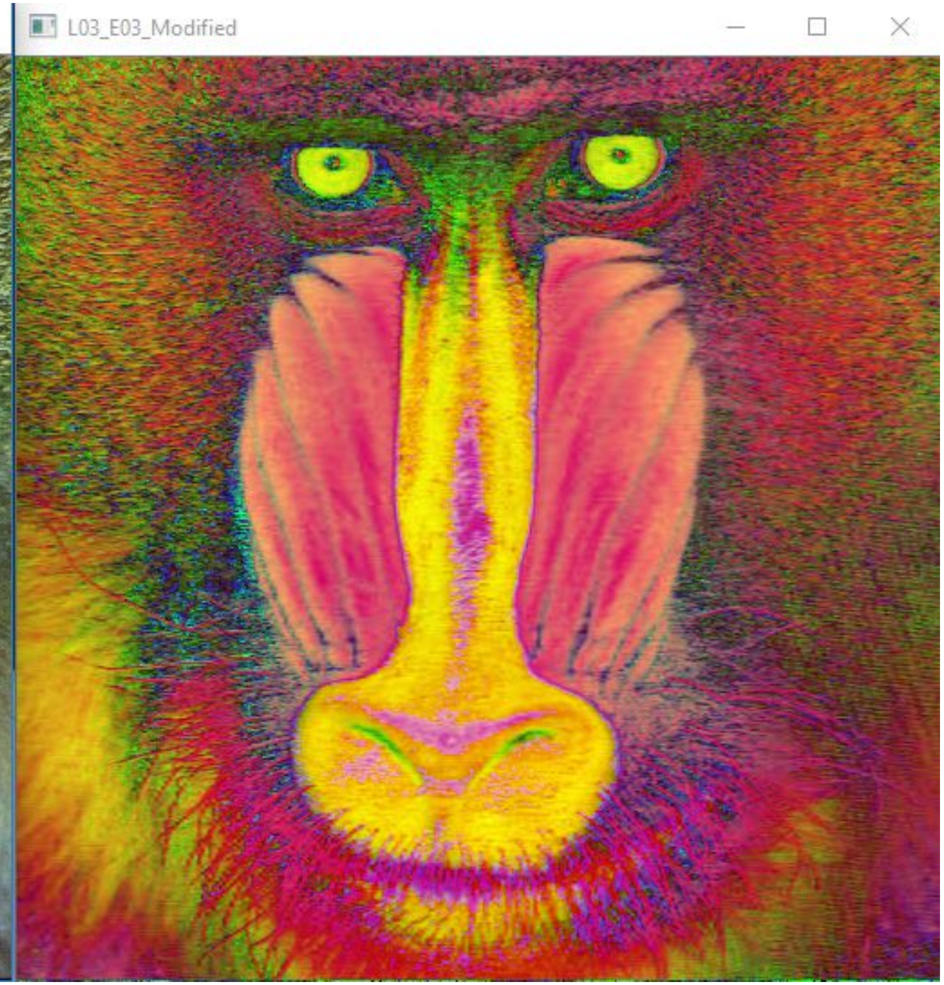


Exercise 3. Convert to HSV and plot the histogram of each channel

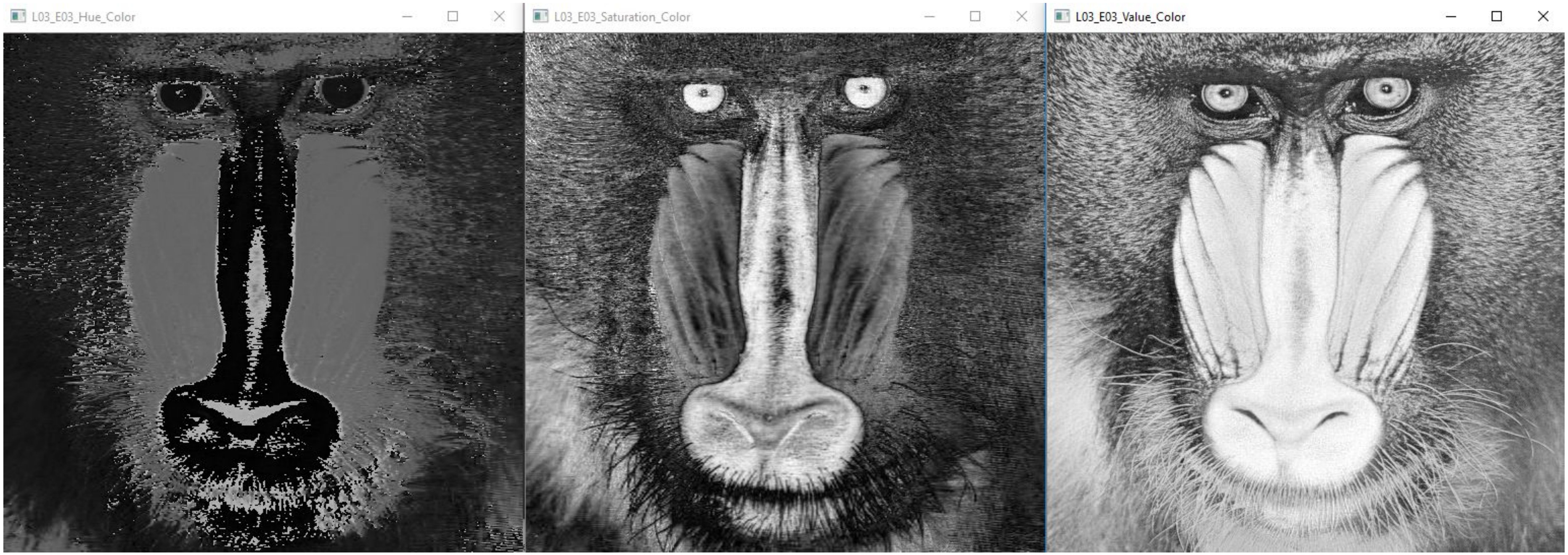
- Load color image (default in RGB)
- Convert the image to HSV color space
- Split the channels into different images
- Compute the histogram for each channel
- Draw the histograms
- Display original image, H, S, and V images and their corresponding histograms
- Free memory

Exercise 3. Convert to HSV and plot the histogram of each channel

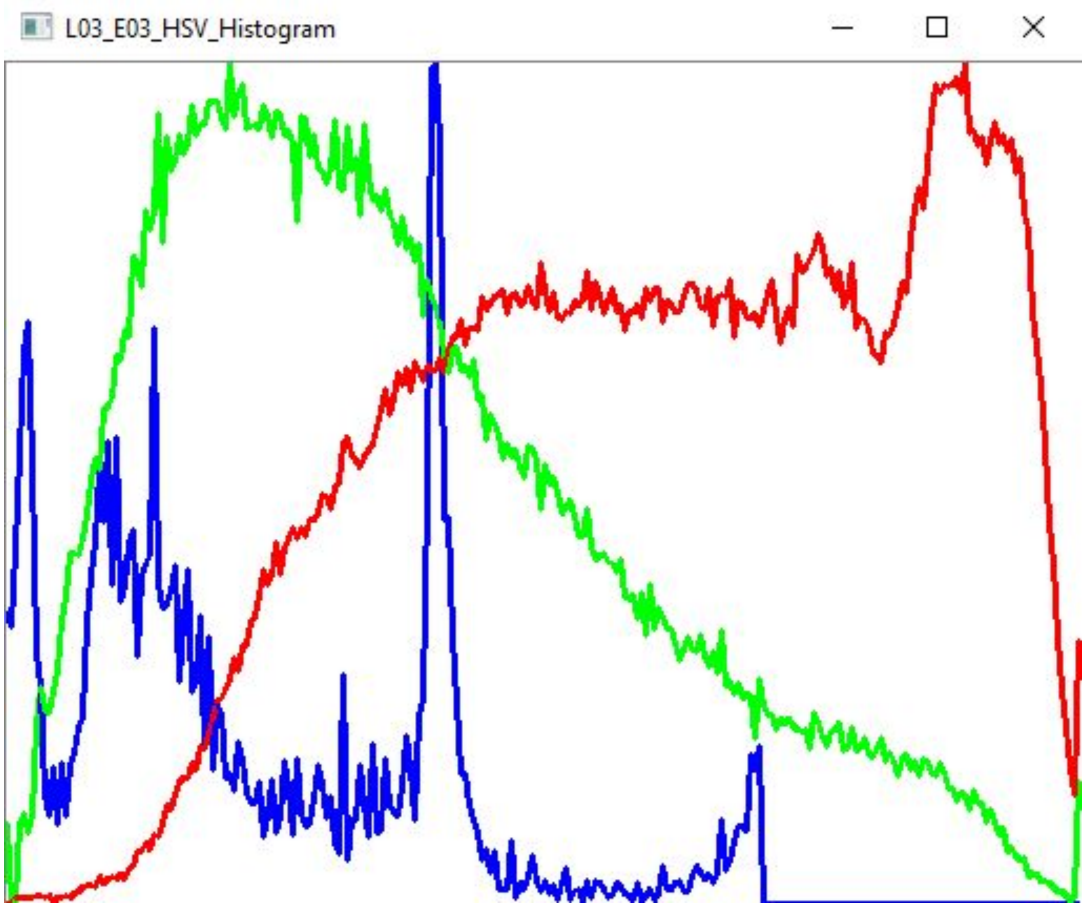




Exercise 3. Convert to HSV and plot the histogram of each channel



Exercise 3. Convert to HSV and plot the histogram of each channel



Lesson 3: Color spaces

```cpp
#include "opencv\cv.hpp"
#include <iostream>

using namespace cv;
using namespace std;

int main(int argc, char* argv[])
{
    // initialize object
    Mat original_image, modified_image;

    // load image from disk
    original_image = imread("mandril.jpg", IMREAD_COLOR);

    // check if the image is available
    if (!original_image.data)
    {
        cout << "Error in loading the image!" << endl;
    }
    else
    {
        // convert the image from BGR to RGB
        cvtColor(original_image, modified_image, CV_BGR2HSV);

        // create a channel for each color with image size
        Mat hue_channel(modified_image.size(), CV_8UC1);
        Mat saturation_channel(modified_image.size(), CV_8UC1);
        Mat value_channel(modified_image.size(), CV_8UC1);
        // create an array of all channels
        Mat channels_array[] = { hue_channel, saturation_channel, value_channel };
        // split the image to separate channels
        split(modified_image, channels_array);

        // initialize histogram calculating parameters
        int histogram_size = 256;
        float histogram_range[] = { 0, 256 };
        const float* histogram_ranges[] = { histogram_range };
        Mat hue_histogram, saturation_histogram, value_histogram;
```

Lesson 3: Color spaces

```
// calculate image histograms
calcHist(&hue_channel, 1, 0, Mat(), hue_histogram, 1, &histogram_size, histogram_ranges);
calcHist(&saturation_channel, 1, 0, Mat(), saturation_histogram, 1, &histogram_size, histogram_ranges);
calcHist(&value_channel, 1, 0, Mat(), value_histogram, 1, &histogram_size, histogram_ranges);

// initialize histogram plotting parameters
int bin_width = 2;
int histogram_width = 512;
int histogram_height = 400;
Mat normalized_hue_histogram, normalized_saturation_histogram, normalized_value_histogram;

// empty image for the histogram plot
Mat image_histogram(histogram_height, histogram_width, CV_8UC3, Scalar(255, 255, 255));
// normalize histograms to fit the window
normalize(hue_histogram, normalized_hue_histogram, 0, histogram_height, NORM_MINMAX, -1, Mat());
normalize(saturation_histogram, normalized_saturation_histogram, 0, histogram_height, NORM_MINMAX, -1, Mat());
normalize(value_histogram, normalized_value_histogram, 0, histogram_height, NORM_MINMAX, -1, Mat());

for (int i = 1; i < histogram_size; i++)
{
    Point p1(bin_width*(i - 1), histogram_height - cvRound(normalized_hue_histogram.at<float>(i - 1)));
    Point p2(bin_width*(i), histogram_height - cvRound(normalized_hue_histogram.at<float>(i)));
    line(image_histogram, p1, p2, Scalar(255, 0, 0), 2);

    Point p3(bin_width*(i - 1), histogram_height - cvRound(normalized_saturation_histogram.at<float>(i - 1)));
    Point p4(bin_width*(i), histogram_height - cvRound(normalized_saturation_histogram.at<float>(i)));
    line(image_histogram, p3, p4, Scalar(0, 255, 0), 2);

    Point p5(bin_width*(i - 1), histogram_height - cvRound(normalized_value_histogram.at<float>(i - 1)));
    Point p6(bin_width*(i), histogram_height - cvRound(normalized_value_histogram.at<float>(i)));
    line(image_histogram, p5, p6, Scalar(0, 0, 255), 2);
}

// create window canvases to show images
namedWindow("L03_E03_Original", CV_WINDOW_AUTOSIZE);
namedWindow("L03_E03_Modified", CV_WINDOW_AUTOSIZE);
namedWindow("L03_E03_Hue_Color", CV_WINDOW_AUTOSIZE);
namedWindow("L03_E03_Saturation_Color", CV_WINDOW_AUTOSIZE);
namedWindow("L03_E03_Value_Color", CV_WINDOW_AUTOSIZE);
namedWindow("L03_E03_HSV_Histogram", CV_WINDOW_AUTOSIZE);
```

Lesson 3:
Color Spaces

Jorge Beltrán, Arturo de la Escalera

Perception Systems

Course 2019-2020