Getting Started with Images

Prev Tutorial: Building OpenCV for Tegra with CUDA

Next Tutorial: Writing documentation for OpenCV

Original author	Ana Huamán
Compatibility	OpenCV >= 3.4.4

Warning
This tutorial can contain obsolete information.

Goal

In this tutorial you will learn how to:

- Read an image from file (using **cv::imread**)
- Display an image in an OpenCV window (using **cv::imshow**)
- Write an image to a file (using **cv::imwrite**)

Source Code

C++ Python

- **Downloadable code**: Click here
- **Code at glance:**

```
#include <opencv2/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>

#include <iostream>

using namespace cv;

int main()
{
    std::string image_path = samples::findFile("starry_night.jpg");
    Mat img = imread(image_path, IMREAD_COLOR);

    if(img.empty())
    {
        std::cout << "Could not read the image: " << image_path << std::endl;
        return 1;
    }

    imshow("Display window", img);
    int k = waitKey(0); // Wait for a keystroke in the window

    if(k == 's')
    {
        imwrite("starry_night.png", img);
    }

    return 0;
}
```

Explanation

C++ Python

In OpenCV 3 we have multiple modules. Each one takes care of a different area or approach towards image processing. You could already observe this in the structure of the user guide of these tutorials itself. Before you use any of them you first need to include the header files where the content of each individual module is declared.

You'll almost always end up using the:

ed the basic building blocks of the library
vides functions for reading and writing

- **highgui** module, as this contains the functions to show an image in a window

We also include the *iostream* to facilitate console line output and input.

By declaring `using namespace cv;` , in the following, the library functions can be accessed without explicitly stating the namespace.

```
#include <opencv2/core.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>

#include <iostream>

using namespace cv;
```

Now, let's analyze the main code. As a first step, we read the image "starry_night.jpg" from the OpenCV samples. In order to do so, a call to the **cv::imread** function loads the image using the file path specified by the first argument. The second argument is optional and specifies the format in which we want the image. This may be:

- IMREAD_COLOR loads the image in the BGR 8-bit format. This is the **default** that is used here.
- IMREAD_UNCHANGED loads the image as is (including the alpha channel if present)
- IMREAD_GRAYSCALE loads the image as an intensity one

After reading in the image data will be stored in a **cv::Mat** object.

```
    std::string image_path = samples::findFile("starry_night.jpg");
    Mat img = imread(image_path, IMREAD_COLOR);
```

Note

OpenCV offers support for the image formats Windows bitmap (bmp), portable image formats (pbm, pgm, ppm) and Sun raster (sr, ras). With help of plugins (you need to specify to use them if you build yourself the library, nevertheless in the packages we ship present by default) you may also load image formats like JPEG (jpeg, jpg, jpe), JPEG 2000 (jp2 - codenamed in the CMake as Jasper), TIFF files (tiff, tif) and portable network graphics (png). Furthermore, OpenEXR is also a possibility.

Afterwards, a check is executed, if the image was loaded correctly.

```
    if(img.empty())
    {
        std::cout << "Could not read the image: " << image_path << std::endl;
        return 1;
    }
```

Then, the image is shown using a call to the **cv::imshow** function. The first argument is the title of the window and the second argument is the **cv::Mat** object that will be shown.

Because we want our window to be displayed until the user presses a key (otherwise the program would end far too quickly), we use the **cv::waitKey** function whose only parameter is just how long should it wait for a user input (measured in milliseconds). Zero means to wait forever. The return value is the key that was pressed.

```
    imshow("Display window", img);
    int k = waitKey(0); // Wait for a keystroke in the window
```

In the end, the image is written to a file if the pressed key was the "s"-key. For this the **cv::imwrite** function is called that has the file path and the **cv::Mat** object as an argument.

```
    if(k == 's')
    {
        imwrite("starry_night.png", img);
    }
```