3

Introduction to OpenCV

Overview

In this first chapter of the book, you will start writing your first scripts in order to get to grips with the OpenCV library. Images are a key component in a computer vision project because they provide, in many cases, the input to work with. Therefore, understanding main image concepts is the basic knowledge you need to start coding your computer vision projects. Also, some of the OpenCV library peculiarities, such as the coordinate system or the BGR order (rather than RGB), will be introduced. By the end of this first chapter, you will be introduced to the world of computer vision using OpenCV and Python and acquire both theoretical and practical knowledge to start programming computer vision applications. Finally, you have also experienced with your first code files and unit testing, which conform two key points for the rest of this book.

Section 1: Concepts of pixels, colors, channels, images and color spaces

Images are the key pieces in the world of computer vision, as they contain the information to work with. In this sense, geting aquitanced with images is very important. Therefore, you will learn how this information is stored, conforming digital images. To accomplish this, some key concepts like, pixels, colors, channels and color spaces should be introduced.

A digital image can be expressed as a numeric representation of a two-dimensional (2D) view of a 3D world. For example, an Istagram photo taken with your smartphone is a digital image and it can be used as the input for your computer vision application (e.g. detect some text in the image and in such a case, output the text contained in the image performing text to speech for blind people).

As OpenCV offers a grate diversity of functionallity to work with images, it is important to differenciate between color and grayscale images. A grayscale image is simply one digital image where the only colors that appear in it are shades of gray. Usually, there are 256 distinct shades of gray in a grayscale image.

More specifically, a grayscale image can be seen or described as a 2D function, f(x,y), where (x,y) are the spatial coordinates and the value of the function f at each point (x,y) is the brightness or grayscale level of the image. In this way, f(x,y) takes the following values:

* x ∈ [0, h-1], where h is the height of the image
* y ∈ [0, w-1], where w is the width of the image
* f(x,y) ∈ [0, L-1], where L = 256 (for an 8-bit image)

For example, if f(x,y) == 0 means that the grayscale level is equals to 0 (black) and if f(x,y) == 255 means that the grayscale level is equals to 255 (white). All the intermediate values in the range [0,255] define all the shades of gray that can appear in the (grayscale) image.

In the next Figure, you can see a grayscale image. The original (color) image has been taken from the OpenCV repository (<https://github.com/opencv/opencv/blob/master/samples/data/lena.jpg>).

Note

In the next Chapters, you will see how to convert from color to grayscale images and viceversa.



Figure 1.01: Grayscale image

Note that a black and white image can be seen as a special case of a grayscale image because only two values (0 - black, and 255 - white) are allowed. Therefore, f(x,y) takes the two afforementioned values. In this sense, in the next Figure, you can see a black and white (also known as a binary) image using a threshold value of 128.



Figure 1.02: Black and white (binary) image using a threshold value of 128

Note

In the next Chapters, you will see how to convert from grayscale images to black and white (binary) images using the OpenCV function cv2.threshold().

For the sake of clarification, when converting a grayscale image to a black and white (binary) image, the (OpenCV) function cv2.threshold() is usually used and a threshold value is needed.

In a nutsell, a thresholding method replaces each pixel in the source image with a white pixel if the pixel intensity is greater than some predefined constant (threshold value) or a black pixel if the pixel intensity is less than the threshold value.

As commented above, the previous Figure was obtained applying the function cv2.threshold() to the grayscale image with a threshold value of 128. Pixels with intensity less than 128 will be black, and pixels with intensity bigger than 128 will be white. Therefore, the threshold value is a key parameter when thresholding images.

For example, and for the sake of completeness, you will see in the next Figure a binary image but using a threshold value of 160. In this case, as you will suppose, as fewer pixels in the grayscale image have a pixel intensity bigger than 160, the resulting (binary) image will have more black pixels.



Figure 1.03: Black and white (binary) image using a threshold value of 160

A color image can be expressed in the same way as grayscale images, but as they are conformed using three channels (red, green and blue), we need three functions, one for each channel (fR(x,y),fG(x,y),fB(x,y)).

In the next Figure, you can see the corresponding color image of the previous image we have been using throughout the Chapter.

It is important to remember that digital images are an approximation of the real scene because f(x,y) can only have finite discrete values. The more bits you use to represent the image, the more closer this approximation will be to the real scene.



Figure 1.04: Color image

To further understand key concepts in connection with digital images, a color model should be used. There are several color models, but the most common one is the RGB (Red, Green, Blue) model. In this color model, the primary colors (R,G,B) are mixed to create the full range of colors. Each primary color can be usually expressed as a channel. Therefore, a RGB image has three channels. Using 8 bits to represent each channel, a total of 256 discrete values (28 = 256) can be produced. As we are using three channels, a RGB image is called 24-bit color depth image. RGB color model is an additive color model. To better understand this concept, the next Figure will be used.

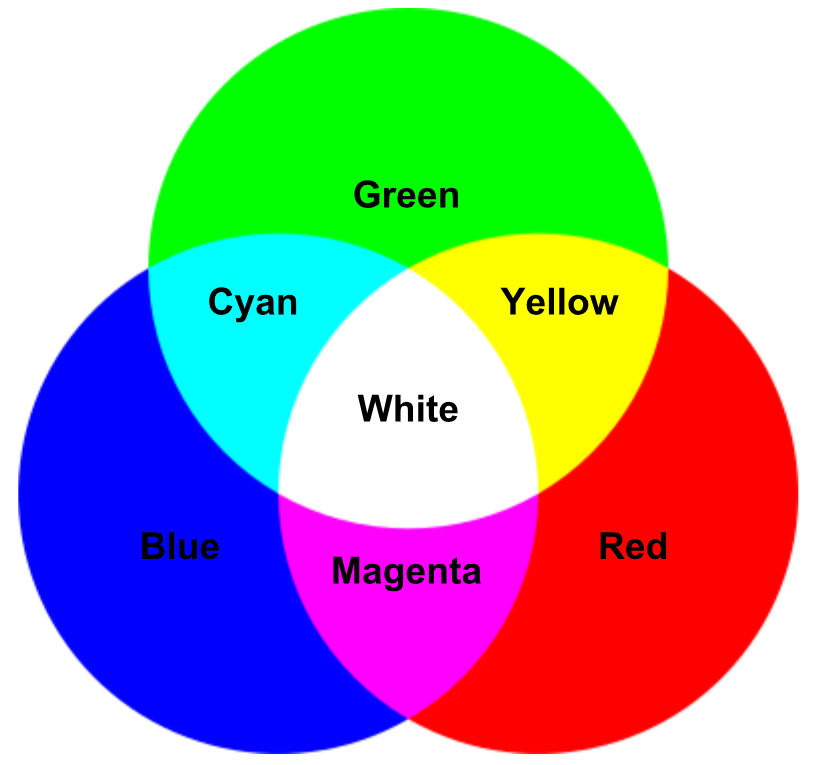


Figure 1.05: A representation of additive color mixing of the RGB color model

In the previous Figure, the additive color property can be seen. In this sense, mixing two primary colors gives the three different secondary colors:

* Mixing red and green gives yellow
* Mixing red and blue produces magenta
* Mixing green and blue generates cyan
* Mixing all three primary colors together produces white

In this previous Figure, only a limited combination of RGB values (r,g,b) are shown. If you want to experiment and play with all the possible combinations, you can access the following URL: <https://www.rapidtables.com/web/color/RGB_Color.html>. Therefore, you can see how a specific color is represented by red, green, and blue values.

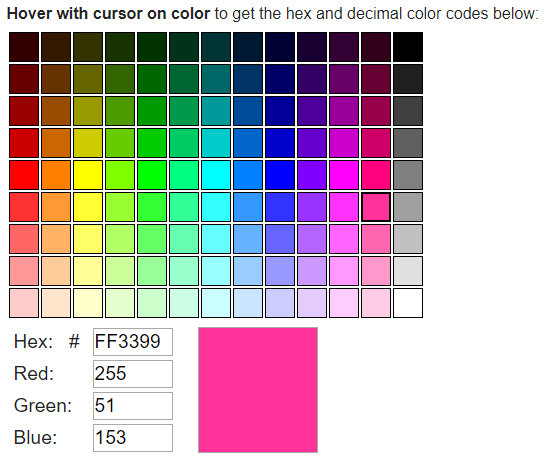


Figure 1.06: Additive property of the RGB color model: how to obtain a pinkish color

In the previous Figure, you can see how to obtain a pinkish color using a specific combination of (r,g,b) values. In this case, the values are: R=255, G=51 and B=153.

Another important concept in connection with the digital images is the image resolution. For example, an image with a resolution of 200x400 can be seen as a grid of 200 columns and 400 rows. This image will contain 80000 pixels. It should be noted that knowing how many pixels conform the image does not imply any phisical dimensions of the image. To know the phisical dimensions of the image, we should know the pixels per inch (PPI) value of the image. For example, if an image is a 200 PPI image, this image will have a phisical dimensions of 1x2 inches. A good resolution image will be in the range of [200-400] PPI.

To conclude this section, the concept of image file format is also introduced. To create, store or transmitte images, different file formats can be used. In this sense, OpenCV supports common file formats:

* Windows bitmaps: \*.bmp and \*.dib
* JPEG files: \*.jpeg, \*.jpg, and \*.jpe
* JPEG 2000 files: \*.jp2
* Portable Network Graphics: \*.png
* Portable image format: \*.pbm, \*.pgm, and \*.ppm
* TIFF files: \*.tiff and \*.tif

Exercise 1.01: Program to show the dimensions of the loaded image

A raster graphics editor needs to show the dimensions of the loaded image by the user.

As this is the first script of the first Chapter of this book, don't worry if you don't quite understand every detail of the script, because we’ll cover all the introduced concepts in detail in this but also in next Chapters.

1. Create a new folder and add a Exercise01\_01.ipynb file.
2. Import the required packages (in this case, only the OpenCV package should be included):

import cv2

1. Load the image (from disk) using the OpenCV function cv2.imread(). To load the image properly, The image should be in the working directory. Otherwise, a full path of image should be provided to the function cv2.imread(). In this case, the variable path\_img contains the path to the image to be loaded, and img is the loaded image:

img = cv2.imread(path\_img)

1. At this point we can get the dimensions of the image using img.shape, which returns a tuple containing the number of rows, columns and channels (if the image is a color one). If the loaded image is a grayscale one, img.shape returns a tuple of number of rows and columns. As in this case the loaded image is a color one, we can do the following:

(height, width, channels) = img.shape

1. Finally, we print the obtained values:

print("Height: '{}', width: '{}', channels: '{}'".format(height, width, channels))

You should see the height, width and channels of the loaded image:

Height: '99', width: '82', channels: '3'

Section 2: BGR order in OpenCV

It is important to note that OpenCV uses BGR color format instead of the RGB one. This means that the first channel in OpenCV is the blue one, instead of the red one. Additionally, the third channel in OpenCV is the red one, instead of the blue one. In other words, red and blue channels are swapped between BGR and RGB color models. The green channel does not change between both color models. BGR order in OpenCV is an important point to remember.This concept can be seen in the next Figure.

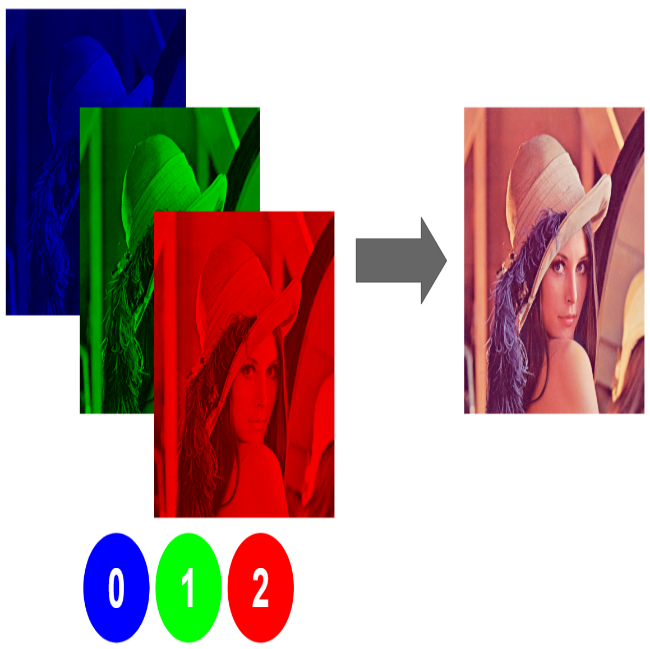


Figure 1.07: BGR order in OpenCV

It should be taken into account that other Python packages use the RGB color model rather than the BGR color model that OpenCV uses. For example, Matplotlib uses the RGB color model. Matplotlib (<https://matplotlib.org/>) is one of the most popular Python plotting libraries and offers a great diversity of plotting methods. Therefore, in order to integrate the plotting capabilities that this library offers, we have to know how to convert from/to BGR/RGB color models.

Section 3: The coordinate system in OpenCV

When accessing the pixel values of the images in OpenCV, one important point is where the coordinate of the origin is stablished.

The coordinate of the origin in OpenCV is the pixel (0,0) and it is located in the top-left corner of the image. Yes, you are right, the pixels in OpenCV are zero-indexed, meaning that the upper left corner is located at (0,0), not (1,1). The following image, which the resolution is a 20x20 pixels, is used to show how to index individual pixels in OpenCV.



Figure 1.08: Image of resolution 20x20 pixels

An individual pixel can be accessed by its row and column coordinates:

* f(x=0,y=0) == magenta (R=255,G=0,B=255)
* f(x=9,y=0) == red (R=255)
* f(x=18,y=1) == cyan (R=0, G=255, B=255)
* f(x=3,y=8) == yellow (R=255,G=255,B=0)
* f(x=19,y=10) == green (R=0, G=255, B=0)
* f(x=13,y=12) == white (R=255, G=255, B=255)
* f(x=0,y=19) == blue (R=0, G=0, B=255)

As you can see, x coordinates get larger as they go right, and y coordinates get larger as they go down indexing the image.

Mostrar como hacer el get de un pixel

Como ejercicio, hacer el get de cada uno de los píxles de la imagen

Section 4: Manipulate pixels in OpenCV

Los dos tipos que hay: set individual pixles, y el de trozo

Como ejercicio, el de hacer set a la cara