Image processing: histograms and color coding

1 Introduction

The objective of this session is to practice with colour coding and histogram based filtering, following the lecture 1 of Computer Vision course. You can get more information about lectures and associated practical sessions on my website: `http://chateaut.fr`. Essential needed knowledges includes a beginning level of computer science (Linux, Python programming with `Opencv`, `Numpy`, `Matplotlib`, and `Pytorch` (for sessions on Deep Learning)) and applied mathematics.

The listing 1 shows an "Hello Word" python script that loads and convert an image using `opencv`; and displays it with `matplotlib` and illustrates relevant functions.

2 Histogram based filtering

The histogram of an image `I` is defined by:

$$H(x) = Card\{\mathbf{p} : \mathtt{I}(\mathbf{p}) = x\} \tag{1}$$

with $x \in [0; 255]$ for an one-channel image coded on 256 levels.

Study to realize

You should work from the script `testhisto.py`.

1. **Building your own histogram function** : You have to complete the function `h=histo(I,nbclasses)`. It takes one grayscale image *im* and the number of bins *nb* as args and returns a numpy vector of size *nb* coding the histogram of *im*.

   Tips: you may have a two-folds answer. In a first time, write the histogram function that computes a 256 output vector for the 256 grayscale image. Then, modify the function to gather closer grayscale values within *nb* bins.

   The output returned histogram must be normalized:

   $$H(x) = Card(\mathtt{I})^{-1}.Card\{\mathbf{p} : \mathtt{I}(\mathbf{p}) = x\}$$

2. **Comparing your function with `opencv` histogram function**

   The aim of the previous question is to help you understanding what is really an histogram. Histograms functions already exits both in `opencv` and `numpy` libraries. We give only details for `opencv` function here:

   `hist = cv2.calcHist(images, channels, mask, histSize, ranges)`

   with:

images : it is the source image of type uint8 or float32. it should be given in square brackets, ie, "[img]".

channels : it is also given in square brackets. It is the index of channel for which we calculate histogram. For example, if input is grayscale image, its value is [0]. For color image, you can pass [0], [1] or [2] to calculate histogram of blue, green or red channel respectively.

mask : mask image. To find histogram of full image, it is given as "None". But if you want to find histogram of particular region of image, you have to create a mask image for that and give it as mask. (I will show an example later.)

histSize : this represents the BIN number. Need to be given in square brackets. For full scale, we pass [256].

ranges : this is the RANGE. Normally, it is [0,256]

This function returns *hist*: the histogram values vector.

You should modify `testhisto` script to show histograms obtained by both your handcrafted `histo` function and the opencv one. Verify that you find the same outputs for the two functions.

3. **Comparing histograms of several images** Compare and analyse the histogram computed for the images named: , `"lena1.jpg"`, `"lena2.jpg"` and `"lena3.jpg"`

   Tips: you can modify the `plt.subplot` command to display the number of subfigures you need.

4. **Histogram stretching** : The histogram of the image `"lena2.jpg"` shows that the range of the image is small, resulting to a poor-contrasted image. Increasing the contrast can be done using two main ways: 1) histogram stretching and 2) histogram equalization.

   Histogram stretching is a linear transformation that moves the image into a min-max range (generally $[0, 256]$) image:

$$\mathtt{I}_s = a * \mathtt{I} + b$$

   Histogram equalization is a non-linear transformation that moves the image into an output one such as the associated histogram is uniform (each bins have the same value).

   You have to modify `testhisto` script to add a `eqhist` function that computes a stretched image from an original one (given in args.). Use the resulting function to show the result of histogram stretching. You should produce a picture close to one of fig. 1.

   The main drawback with histogram stretching is that natural images may have noise with bright and dark pixels, resulting poor effect of this operation (since min is close to 0 and max is close to 255). Moreover, some dark/bright images may have a small bright/dark region and the resulting stretched image is not acceptable for visualisation. In this case, the equalization operation can be used. `opencv` library already proposes an histogram equalization function:

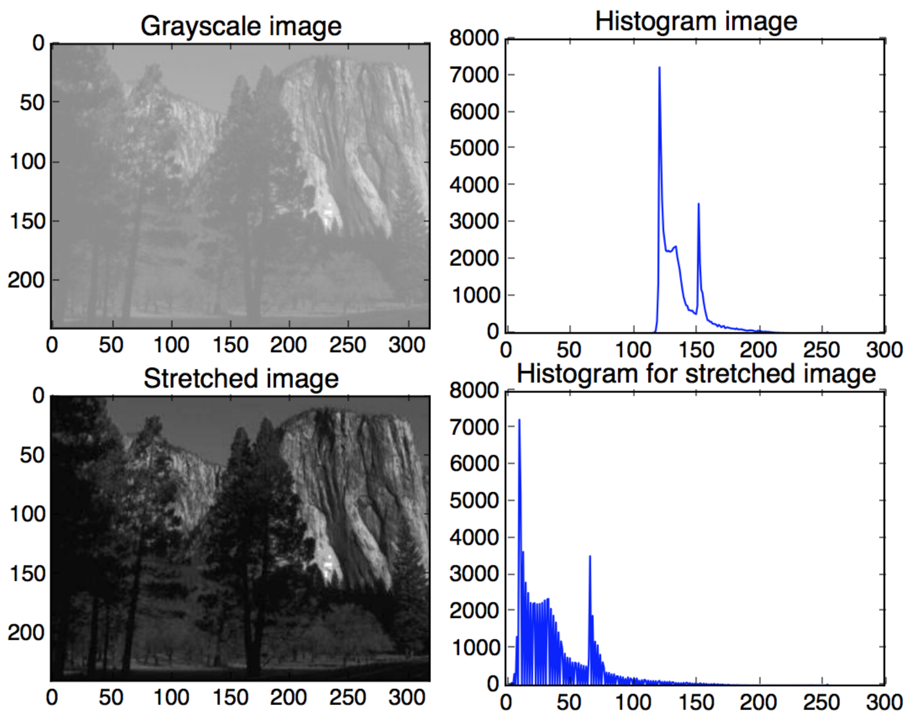

Figure 1: histogram stretching

```
imeq = cv2.equalizeHist(img)
```

You should compare histogram stretching and histogram equalization on image `sample1`

3 Colour representation and application to automatic Hue modification

Here we use histogram to extract the principal hue value into a colour image and then to modify it. Images are usually stored into a RGB (red, green, blue) coding format. The main drawback with this colour space is that colour and brightness are mixed into the three channels, resulting to variation of these three channels when the luminosity lighting changes. The HSV (Hue, Saturation, Value) is a coding space that dissociates lighting (value) from the wave length (hue) and the saturation of this wave length. `Opencv` library proposes the function `cvtColor(..)` to convert RGB to/from HSV image.

The applicative goal of this section is to automatically change the colour of a green pool rug (to obtain a blue one). The following image processing chain will be implemented:

1. Load the image and change the colour coding to HSV
2. Compute and display the hue histogram
3. Compute the maximum value of Hue.

4. Build a mask image for pixels close to the maximum value of Hue
5. Modify the Hue of selected pixels to change it from green to blue (on a $[0, 1]$ range, blue is about 0.6)
6. Convert the modified HSV image into a RGB one.
7. Display

Study to realize

You should work from the script `testhsv.py` and produce the image processing chain described bellow. You should produce a figure close to fig. 2.

Tips

- Extraction the first channel of a three channel image:
  `x1 = x[:,:,0]`
- Compute the index corresponding to the max value of a vector **x**:
  `indxmax = np.nonzero(x==max(x))[0]`
- Building a mask image from a logical condition on image **im** (example):
  `mask = im > th`

Additional questions

1. What happen if the main hue is red? (Tips: remember that hue is an angle)
2. Modify the script to remove the automatic detection of dominant hue and replace it by a manual selection of the hue to be change (by using `plt.ginput()`) to select with the mouse.

Listing 1: HelloWord Python script: load an image and convert it using opencv then display using matplotlib

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Mon Aug  6 08:46:29 2018
hello_wold simply loads an image with opencv, converts it to RGB and GRAYSCALE
and displays it using matplotlib
@author: thierrychateau
"""
import cv2
from matplotlib import pyplot as plt
import numpy as np
# This is an example of function with a string argument that returns another
# string
def my_func(my_arg):
    string2 = my_arg + ' toto ';
```

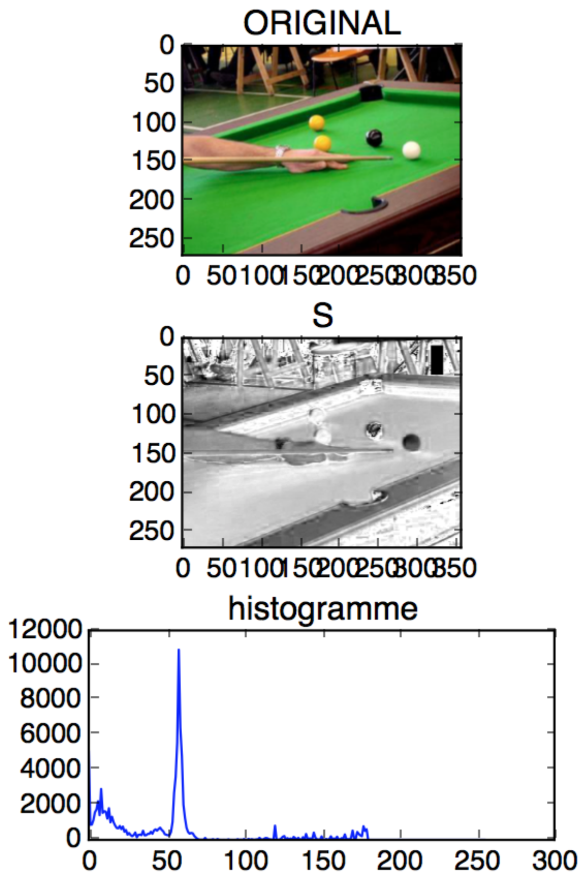

Figure 2: Automatic principal hue transformation

```
    return string2
# This is the main function that is launched when this file is run as a script
def main():
    # call function my_func
    st = my_func("my name is")
    print(st)
    # Read an image file (im is a numpy matrix
    # (nblines,nbcolums, BGR 3 channels))
    filename = "billard_large.jpg"
    img = cv2.imread(filename)
    # Convert to RGB
    imgrgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    # Convert to grayscale image
    imggray=cv2.cvtColor( imgrgb, cv2.COLOR_RGB2GRAY )
    ## Display the RGB and gray images
    # divide plot screen into 1 line, 2 columns and select first subscreen as
    # current screen
    plt.subplot(121)
    plt.imshow(imgrgb)
    plt.title('Color RGB image')
    plt.subplot(122)
    plt.imshow(imggray, 'gray',vmin=0, vmax=255)
    plt.title('Grayscale image')
#
if __name__ == "__main__":
    # execute only if run as a script
    main()
```