MIT EECS 6.815/6.865: Assignment 1:

Basic Image Processing

Due Monday February 13 at 9pm.

1 Summary

This assignment is mostly a warm up to make sure your environment is installed, but you need to implement the following requirements:

- Basic point operations: brightness and contrast
- Black frame using "`:`" indexing
- Black and white conversion
- Luminance-chrominance conversion and luminance-only brightness-contrast
- YUV conversion and saturation increase
- Spanish Castle illusion
- Histogram

We will use the provided `a1Requirements.py` script to test your code.

2 Installation

We will be using Python 2.6 or 2.7.

2.1 Numpy

Install numpy and scipy from http://numpy.scipy.org/ Familiarize yourself with the array data structure. This is what we will use to represent digital images. We recommend http://docs.scipy.org/doc/numpy/user/whatisnumpy.html and http://docs.scipy.org/doc/numpy/user/basics.html

One advantage of using numpy arrays is that it provides many built-in efficient operators that enable simple algebraic notations. For example, given two arrays `a` and `b`, you can easily write `c=a+42*b`.

`numpy` also has powerful indexing options that make it easy to operate on entire slices of the array. For example, `im[:, :, 0]=0` sets the entire red channel to zero in our image representation.

In general, it's a good idea to avoid explicit `for` loops when you can because the built-in `numpy` operations are much much faster. But we will not always be able to avoid `for`.

2.2 pypng and display

Install pypng from http://code.google.com/p/pypng/

You will need to rely on the image display capabilities of your OS. Alternatively, you are welcome to use the image display and IO capabilities of the Python Imaging Library (PIL), although we do not require it and you will not be allowed to use features of PIL that implement requirements from the assignments. Installation of PIL can be tricky (e.g. get the correct version of libjpeg, version 6, not the latest one) which is why we do not require it.

2.3 ImageIO

Install the module `ImageIO` provided on the class web page.

The module provides a wrapper for pypng. It reads and writes the popular PNG image format. Images are returned or taken as numpy arrays. The image dimensions are organized in the order Y, X, color channel. This means that you access a given value as `im[y, x, c]`

The module expects input images to be in an `Input` folder and the output ones to be in an `Output` folder. You can change this behavior by editing the variables `baseInputPath` and `baseOutputPath`.

As a first test, load the provided image `test.png` and save it under a different name using `myarray=imread(mypath)` and `imwrite(myarray, myotherpath)` in `imageIO`

Check the image size using `numpy.shape()`. If you're adventurous, see if you can rotate the image using `numpy`'s `transpose`.

3 Basic operations

3.1 Brightness and contrast

Implement a `brightness(im, factor)` function that multiplies all the values in an image by a constant `factor`.

Implement a `contrast(im, factor, midpoint=0.3)` function that implements a simple affine function that maintains a given value `midpoint` and has a slope `factor`.

Extra Credit: Deal with clipping gracefully for both brightness and contrast.

3.2 Framing

Write a `frame(im)` function that uses the `:` indexing operator to have a 1-pixel black frame around an image. That is, set the first and last row and column of pixels to black.

4 Color space manipulations

4.1 Convert to black and white

Write a function `BW(im, weights=[0.3, 0.6, 0.1])` that returns a black and white version of the input image `im`. The greyscale value at a given pixel is a weighted combination of the RGB channel as specified by weights.

Hint: `numpy` has funcion `dot(a, b)` that computes matrix multiplications in general, and dot products in particular.

4.2 Luminance-chrominance

Implement a function `lumiChromi(im)` that decomposes an image into the product of a luminance (black and white) image and a chrominance one. Your function must return two images, one for luminance, and one for chrominance. Hint: use the above `BW` function.

Implement a function `brightnessContrastLumi(im, brightF, contrastF, midpoint=0.3` that changes the brightness and then the contrast of the luminance of an image without affecting its chrominance. It should return a new image.

4.3 YUV

Implement a function that converts an image from RGB to YUV and another from YUV to RGB

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.13983 \\ 1 & -0.39465 & -0.58060 \\ 1 & 2.03211 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ U \\ V \end{bmatrix}$$

Write a function `saturate(im, k)` that modifies the saturation of an image by multiplying its U and V components by a given factor `k`.

4.4 Discussion

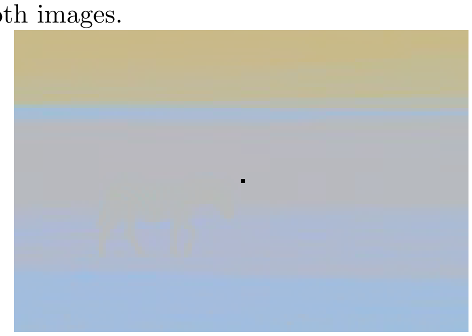
The chrominance-luminance and the YUV conversions perform similar operations: they decouple an image's "intensity" from its "color." There are, however, important differences. YUV is obtained by a purely linear transformation, whereas our chrominance-luminance decomposition requires a division. Furthermore, the latter is overcomplete (we now need 4 numbers, or even 6 the way we have encoded it), while YUV only requires 3.

YUV does a better job of organizing color along "opponents" and the notion of a negative is more perceptually meaningful. On the other hand, the separation between "intensity" and "color" is not as good as with the ratio computation used for luminance-chrominance. As a result, modifying Y without updating U and V changes not only the luminance but also the apparent saturation of the color. In contrast, because the luminance-chrominance decomposition relies on ratios, it preserves colors better when luminance is modified. This is because the human visual system tends be sensitive to ratios of color channels, and it discounts constant multiplicative factors. The "color" elicited by r, g, b, is the same as the color impression due to kr, kg, kb, only the brightness/luminance is changed. This makes sense because we want objects to appear to have the same color regardless of the amount of light that falls upon them.

5 Spanish Castle illusion

Your goal is to create a function `spanish(im)` that takes one image as input and writes two images `L.png` and `C.png` that elicit the Spanish castle illusion described at
`http://www.johnsadowski.com/big_spanish_castle.php`.
The first image has a constant luminance (Y) and inverted chrominance (U and V), and the second image is a black-and-white version of the original.

To help people focus on the same location, add a black dot in the middle of both images.




6 Histograms

A histogram can be represented as an array `h` with `N` bins where the number stored at `h`[`k`] states that the fraction of pixels that have a value between k/N and $(k+1)/N$ is `h`[`k`]. It can be computed by going through all the pixels and incrementing the appropriate bin.

6.1 Histogram computation

Write a function `histogram(im, N)` that return an array of size `N` containing the normalized histogram of luminance of image `im`. That is, the sum of the bin values should be 1.

Then write a function `printHisto(im, N, scale)` that prints histograms as below. Each row corresponds to a bin and the number of `X` is proportional to the bin value, multiplied by `N` and `scale`.

```
>>> printHisto(im, 20, 300)
XX
XX
XXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXX
```

7 Submission

Turn in your python files and make sure all your functions can be called from a module `a1.py`

Include a `README.txt` containing the answer to the following questions:

- How long did the assignment take?
- Potential issues with your solution and explanation of partial completion (for partial credit)
- Collaboration acknowledgement (but again, you must write your own code)
- What was most unclear/difficult?
- What was most exciting?