Lab 1

October 19, 2017

1 Lab 1: Demosicing and Color Correction

In this lab we will explore and implement some of the most common processes that are applied to images from raw to JPG or other formats. We will also use this to get familiar with some simple image processing in Python.

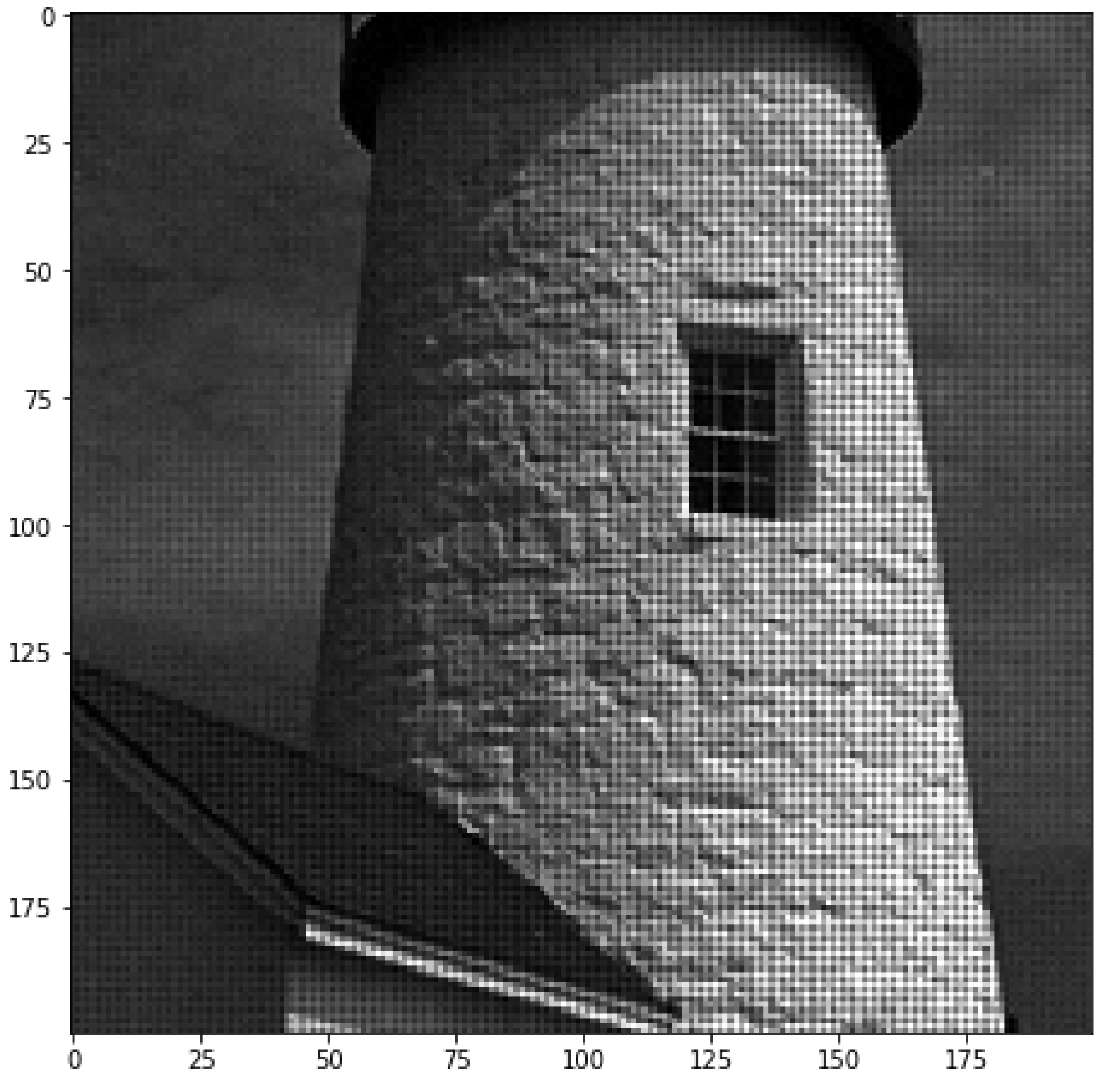
We will use scipy.ndimage to load and save images which have a simple interface and stores the data as simple numpy arrays.

You are free to change the code I give you or not to use it at all.

```
In [1]: #some magicto show the images inside the notebook
        %pylab inline
        %matplotlib inline

        import matplotlib.pyplot as plt
        from scipy import ndimage
        import numpy as np

        # A hepler function for displaying images within the notebook.
        # It displays an image, optionally applies zoom the image.
        def show_image(img, zoom=1.5):
            dpi = 77
            plt.figure(figsize=(img.shape[0]*zoom/dpi,img.shape[0]*zoom/dpi))
            if len(img.shape) == 2:
                img = np.repeat(img[:,:,np.newaxis],3,2)
            plt.imshow(img, interpolation='nearest')


        # A hepler function for displaying images within the notebook.
        # It may display multiple images side by side, optionally apply gamma transform, and zoo
        def show_images(imglist, zoom=1, needs_encoding=False):
            if type(imglist) is not list:
               imglist = [imglist]
            n = len(imglist)
            first_img = imglist[0]
            dpi = 77 # pyplot default?
            plt.figure(figsize=(first_img.shape[0]*zoom*n/dpi,first_img.shape[0]*zoom*n/dpi))
            for i in range(0,n):
```

```
            img = imglist[i]
            plt.subplot(1,n,i + 1)
            plt.tight_layout()
            plt.axis('off')
            if len(img.shape) == 2:
               img = np.repeat(img[:,:,np.newaxis],3,2)
            plt.imshow(img, interpolation='nearest')

    lighthouse = "data/lighthouse_RAW_noisy_sigma0.01.png"
    img=ndimage.imread(lighthouse)

    show_image(img[200:400,200:400],3) #takes a region of the image
```

```
Populating the interactive namespace from numpy and matplotlib
```



1.1 Task 1: DCRAW (10)

The first task is to get familiar with the tool DCRAW. www.cybercom.net/~dcoffin/dcraw/. You are given an image _MG_4257.CR2 which is the RAW format for Canon DSLR cameras. Install it in your computer and run it, you will get the options and usage. In this task you should define some functions that simply call the dcraw command with the appropriate options from the a python programm.

You only need to write functions so dcraw produces a new file.

1.1.1 Convert to Color image (PPM) from RAW

Convert to a normal jpg in sRGB color space. Test different options (for example GAMMA) and compare the results. You are given a JPG file for this image which is the one produced by the camera

1.1.2 Extract RAW data

Identify the option for this and recover the original raw data in a standard image format (pgm). This will be a black and white image with a bayer pattern.

1.1.3 Convert to Linear tiff 16 bits

The standard pipeline applies color and gamma corrections, when we use images for computation is best to have the linear input and preserve as much information as possible. Find the options to recover the linear 16 bits tiff image from the raw.

1.1.4 If you have a camera, do the same with your own images

Capture an image with your camera in RAW mode, extract the RAW image using the command line software dcraw (don't process the image, just extract the linear, RAW),

In the next tasks test your code for demosaicking and gamma correction on your own raw images, compare with the results you get from dcraw (for demosaicking and conversion to sRGB). If you can't capture, you have an example in data.

```
In [2]: ###HINT: Look for the "subprocess" module

        def raw2PPM(path_to_file):
            pass

        def extract_raw(path_to_file):
            pass

        def raw2linear_tiff(path_to_file):
            pass

        #load the converted images and display using show_images()
```

1.2 Task 2: Gamma Correction(10)

Take the 16 bits linear image and apply a gamma encoding to it. https://en.wikipedia.org/wiki/Gamma_correction

Display the linear image, the gamma encoded one, and the JPG produce by dcraw.

```
In [3]: # Verbose aliases for gamma transformation
        def gamma_decode(img, gamma=2.2):
            #Input is a gamma encoded image, output is a linear image
            #your code here
            return img #!change accordingly

        def gamma_encode(img, gamma=2.2):
            #Input is a linear image, output is a gamma encoded image
            #your code here
            return img #!change accordingly

        #load the converted images and display using show_images()
```

1.3 Task 3: White Balance (10)

White balance is the process of setting whites to white. Depending on the lighting, white pixels will look of the colour of the light, in our example whites look quite yellowish. https://en.wikipedia.org/wiki/Color_balance

Different methods exist to estimate the color of the light, you will implement two simple white balance methods.

The white point simple takes in a pixel (three values) which we know is white, for example a pixel from the fence, and use this pixel to set it to white, and the rest of the image accordingly.

The second method is the "grey world assumtion". The idea here is to compensate the color channels so their average is grey (all channels have the same value). To do this, compute the mean of each channel (RGB), normalize to the maximum value of the triplet, and use this as your compensation factor.

```
In [4]: def white_balance_white_point(img, whitePoint):
            #your code here
            #HINT: You will probably need to use np.clip
            return img #!change accordingly


        def white_balance_gray(img):
            #your code here
            return img #!change accordingly

        im = ndimage.imread("data/lighthouse.png")/255

        im_wp = white_balance_white_point(im,im[309:310,348:349,:].flatten()) #you can select a

        im_g = white_balance_gray(im)
```

```
show_images([im,im_wp,im_g],0.5)
```

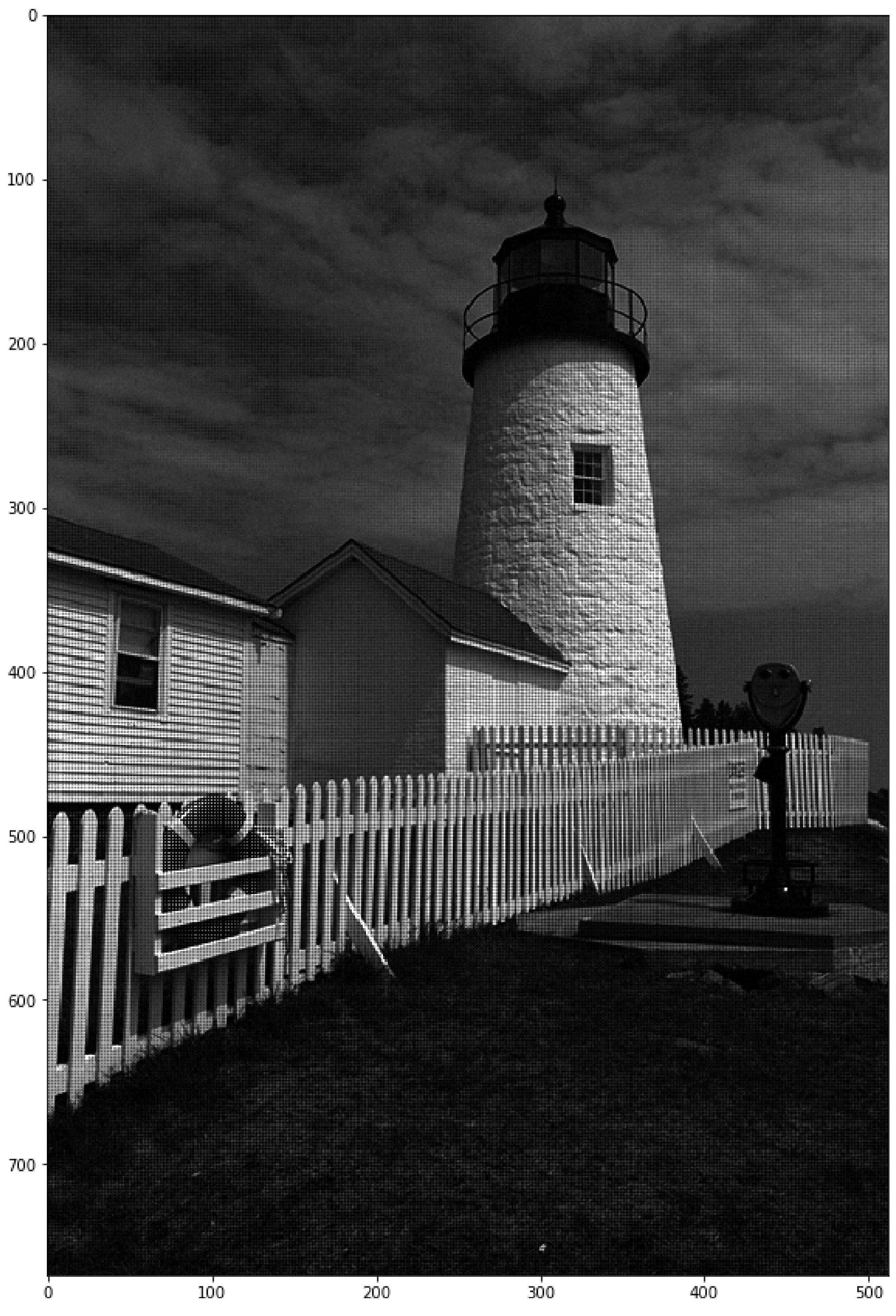
```
In [5]: #White balance other images and display using show_images()
```

1.4 Task 4: Basic Demosaicing (20)

Implement demosaicking using linear interpolation. The example image 'lighthouse_RAW_colorcoded.png' shows you in false colors which subpixel measures with color channel. Don't use that for demosaicking, use the image 'lighthouse_RAW_noisy_sigma0.01.png' Check your implementation with other images from data/raw. Lecture 2 slides here http://franchomelendez.com/Uwr/teaching/COMPHO/_LECTURES/L2/digital_photography.html

Unfortunatelly, not all the sensors have the same pattern, so you will find that the patern for lighthouse image is not valid for the other raw images. In essence in these cases the result is that the blue and red channels are exchanged. You can ignore this for the naive solution and hardcode the pattern, but you should take it into account when developing the other solutions.

```
In [6]: # A helper function for merging three single-channel images into an RGB image
        def combine_channels(ch1, ch2, ch3):
            return np.dstack((ch1, ch2, ch3))


        def demosaic_simple_naive(img):
        ### Start with a simple double loop implementation of the simple demosicing algorithm.
        ### If you are already familiar with other ways to process images in python, you can ski
        ### your code here
            pass
            return img #!change accordingly

        show_image(demosaic_simple_naive(img)) #should show a color image
```


0
100
200
300
400
500
600
700
0
100
200
300
400
500