Raw Image Decoder
INF01213 Computational Photography – 2020/1

Lucas N. Alegre

March, 2020

1 Introduction

The goal of this assignment is to better understand and to implement the pipeline of a raw image decoder. A raw image data captured from a Canon Rebel XS camera is first converted to Adobe DNG file format, then it is applied demosaicking, white balance and gamma encoding methods in order to produce a full-color image.

I used *Python3* programming language to implement the necessary methods. The library *rawpy* was used to read the DNG file and the library *numpy* to operate over matrices, similarly as in *MATLAB*.

2 Raw Image Decoder

2.1 Step 1 - Converting from .CR2 to .DNG

The file scene_raw.CR2 was first converted to Digital Negative (DNG) format using Adobe's Digital Negative (DNG) Converter software. The resulting DNG file is shown in Figure 1.

Figura 1: Raw image.

2.2 Step 2 - Demosaicking

The Bayer Color Filter Array (CFA) is read from the DNG file and stored in a matrix after scaling its values to the 16-bit representation. The goal of this process is to transform the CFA into a 3-dimensional matrix representing the RGB channels of the image.

Each RGB channel is obtained by first multiplying the CFA to a corresponding matrix that repeatedly presents the following patterns for each channel:

$$R_{pattern} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, G_{pattern} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, B_{pattern} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{1}$$

The next step, in order to retrieve color information of the other two missing intensities for each pixel, is to apply bilinear interpolation on each RGB channel. The bilinear inteporlation was implemented using the following convolutional filters:

$$Diamond = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, Corners = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} \tag{2}$$

For the green channel, applying the *Diamond* filter is sufficient. However, for the red and blue channels, both *Diamond* and *Corners* filters are applied to retrieve both missing channels.

The resultant image after demosaicking is shown in Figure 2.

Figura 2: Image after demosaicking.

2.3 Step 3 - White balance

After demosaicking, the result is a very greenish image. In order to adjust the color intensity of a pixel and to simulate the way the human visual system perceives it in relation to the other colors in view, we apply white balanced methods.

In Figures 3, 4 and 5, a white pixel of the image was chosen to be the *true-white* pixel of reference. Then, the RGB values of all other pixels are divided by the RGB values of the chosen white pixel.

We can observe that the resulting images are all very different. The image produced with the icon pixel (Figure 3), for instance, is darker than the image produced with the paper pixel (Figure 5). In my opinion, the image in Figure 4 (produced with a pixel from the adapter over the table) is the best one in terms of final color, appearing more similar to an image I would expect the camera to generate.

Figura 3: White balance using icon pixel (640x2132).

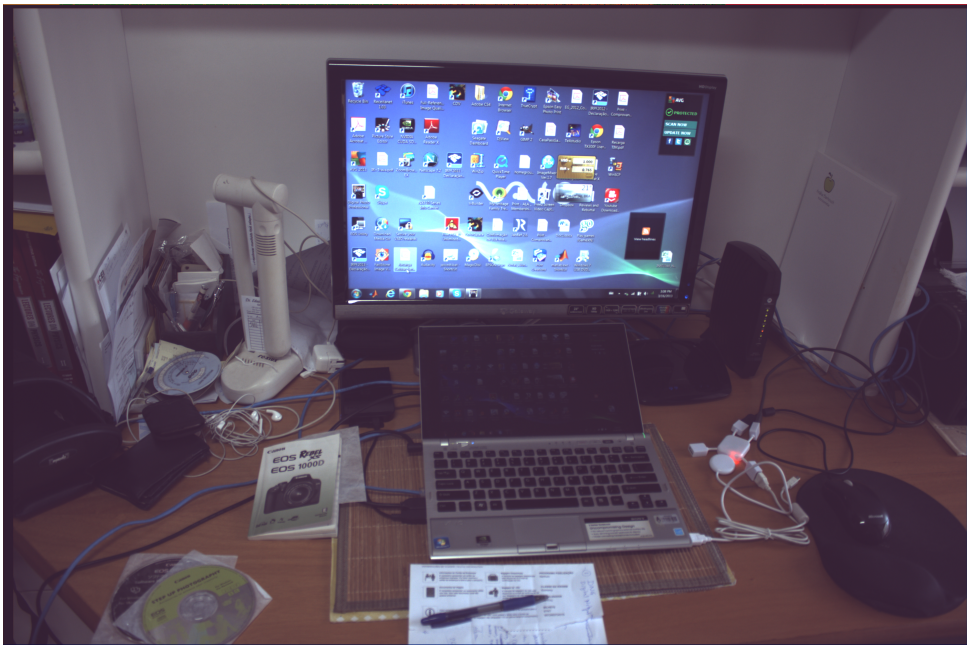

Figura 4: White balance using adapter pixel (1844x3027).

Figura 5: White balance using paper pixel (2413x1691).

The results made me curious to see how would be the image that an automatic white balance method would produce. Furthermore, I also implemented the Gray World method, which assumes that the color in each sensor channel averages to gray. In this method, the R channel is scaled by α and the B channel by β, following:

$$\alpha = G_{avg}/R_{avg}, \quad \beta = G_{avg}/B_{avg}. \tag{3}$$

The image produced with the Gray World method is shown in Figure 6.

Figura 6: White balance using Gray World method.

2.4 Step 4 - Gamma encoding

Finally, gamma encoding is applied to the color balanced image in order to account to the non-linear perception of color by the human visual system. Each value of intensity is powered by γ, thus stretching the representation of lower values, since humans detect changes in darker intensities more easily.

The image in Figure 6 (white-balanced with Gray World method) was chosen to demonstrate the effect of different values of γ on gamma encoding. The final resulting images are shown in Figures 7, 8 and 9.

Figura 7: Gamma encoding with $\gamma = 1/1.2$.

Figura 8: Gamma encoding with $\gamma = 1/1.7$.

Figura 9: Gamma encoding with $\gamma = 1/2.2$.

We can observe that the lower the value of gamma, the brighter the resulting final image, as the intensities of the pixels get closer to 1. This result does not seem very realistic to me, i.e., I found the image to look better before gamma encoding was applied.

3 Conclusions

This assignment allowed me to better understand how pictures are generated in photographic cameras and to learn the motivation behind each step of the pipeline of raw image decoding. Each of these steps were successfully implemented.

By looking throught the available material, I also realized how automatic white balance methods constitute a very large field of research. They can get very complex, besides the method I implemented (Gray World) is a relatively simple one. I find myself curious to know which method of automatic white balance the Canon camera used to produce the final JPG image provided by the professor.

Finally, I believe that the most important thing that I learned during this assignment is that a *correct* image does not exist. The process of transforming a raw image obtained from camera sensors to a full-color image is much of an art than it is a science. Nevertheless, most methods were developed by taking into account the properties of human vision.