COMPUTER VISION COURSE (2021/22)
Master in Electrical and Computers Engineering
Laboratory Assignment 0
Matlab Image Processing Toolbox - Learn how to use it

The Bayer RAW Image Decoder

1 Course Presentation

Welcome to the Computer Vision course, 2021/22 edition.

Along the semestre this course will provide you an introduction to the challenging scientific domain of computer vision, including fundamentals of image formation, camera imaging geometry, feature detection and matching, multiview geometry including stereo, motion estimation and tracking, and classification. Along the course you will implement basic methods for vision-based applications that include extracting image features, image descriptors and matching, camera calibration, 3D reconstruction, etc...

The laboratory component of the course will comprise a set of Labworks, all of then develloped within the Matlab pipeline, that must be carried out in groups of two students.

2 Description and Learning Goals

The purpose of this first assignment is to introduce you to Matlab as a tool for manipulating images. For this, you will build your own version of a very basic *image processing pipeline*. As we will confirm along the course, most of the computer vision tasks involves some sort of *image processing pipeline* which is the sequence of steps taken to fulffill a certain major task.

In this assignment we propose you to implement your own *image processing pipeline* to simulate the stages that happens inside a Photografic camera, in order to convert a RAW image (roughly speaking, the values measured by the camera sensor) into a regular *8-bit* image that you can display on a computer monitor or print on paper - *The Bayer RAW Image decoder*

We provide you some ″Hints and Informations″ at the end of this document. Please read then carefuly.

In contrast to what will happens in the following labworks, we also provide a solution in the `./solution` directory of the homework ZIP archive - but we strongly encourage you to first try to solve the assignment on your own, before looking at the solution.

Throughout this assignment, you will use the file `banana-slug.tiff` included in the `./data` directory of the homework ZIP archive. This is a RAW image that was captured with a Canon EOS T3 Rebel camera [1]. The original RAW image has already submitted to a very slight pre-processing, in order to convert it to `.tiff` format. At the end of this assignment, the image should look something like what is shown in Figure 1. The exact result can vary greatly, depending on the choices you make in your implementation.

3 Introduction

In the fields of image processing and computer vision, researchers are often indifferent to the origins of the images they work with. The use of 8-bit intensity image or three-channel RGB

Figure 1: One possible rendition of the RAW image provided with the assignment.

image with 8 bits per channel is almost a standard for most applications. Sometimes, however, it is important to truly relate an image to the light of the scene from which it was captured. In those cases, it is important to know the entire processing chain that was applied to an image after being captured. If possible, the best image to deal with is the sensor data straight from the camera, the *'raw'* data.

'RAW' is a class of computer files which typically contain an uncompressed image containing the sensor pixel values as well as a large amount of meta-information about the image generated by the camera (the Exif data). RAW files themselves come in many proprietary file formats (`Nikon's .NEF, Canon's .CR2, Sony's .ARW, etc`) and at least one common open format, `.DNG`, which stands for Digital Negative.

When using commercial software to read and display a RAW file there are a number of processes that happen behind the scenes which are invisible to those who primarily work with the final product (the 8- bit-per-channel `RGB JPG/TIF/etc.`). In order to display the image on a screen, the raw image is being *linearized*, *demosaiced*, *color corrected*, and its pixel values are being *non-linearly compressed* to yield the image information most operating systems and display drivers expect. These image processing stages will be the focus of this assignment.

For additional readings about Processing 'RAW' images, please see [1], or browse `http://rcsumner.net`.

Initials Let's start

- Load the image into Matlab. Originally, it will be in the form of a 2D-array of unsigned integers.
- Check and report how many bits per integer the image has, and what its width and height is. Then, convert the image into a double-precision array. (See help for functions `imread`, `size`, `class` and `double`.)

Linearization. The resulting 2D-array is not a linear function with respect to the true energy each pixel receives. It is possible that it has an offset due to dark noise (intensity values produced even if no light reaches the pixel), and saturated pixels due to overexposure. Additionally, even though the original data-type of the image was 16`bits`, only 14 of those have meaningful information, meaning that the maximum possible value for pixels is 16383 (that's 2^{14}).

For the provided image file, you can assume the following:

- All pixels with a value lower than 2047 correspond to pixels that would be black, were it not for dark noise. All pixels with a value above 15000 are over-exposed pixels. (The values 2047 for the black level and 15000 for saturation are taken from the camera manufacturer).
- Convert the image into a linear array within the range [0,1]. Do this by applying a linear transform (shift and scale) to the image, so that the value 2047 is mapped to 0, and the value 15000 is mapped to 1.
- Then, clip negative values to 0, and values greater than 1 to 1. (See *help* for functions `min` and `max`.)

Identifying the correct Bayer pattern. Most cameras do not capture true RGB images. Instead, they use a process called *mosaicing*, where each pixel captures only one of the three color channels. The most common spatial arrangement of pixels in terms of what color channel they capture is called the *Bayer pattern*, which says that in each 2×2 neighborhood of pixels, 2 pixels capture green, 1 pixel captures red, and 1 pixel captures blue measurements (see Figure 2). The same is true for the camera used to capture our RAW image: If you zoom into the image, you will see the 2×2 patches corresponding to the Bayer pattern.

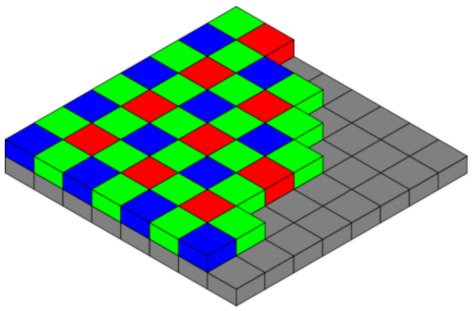

Figure 2: The Bayer pattern.

However, we do not know how the Bayer pattern is positioned relative to our image: If you look at the top-left 2×2 image square, it can correspond to any of the four red-green-blue patterns shown in Figure 3. Think of a way for identifying which version of the Bayer patterns applies to our image file, and report which version you identified. (See Hints and Information.)

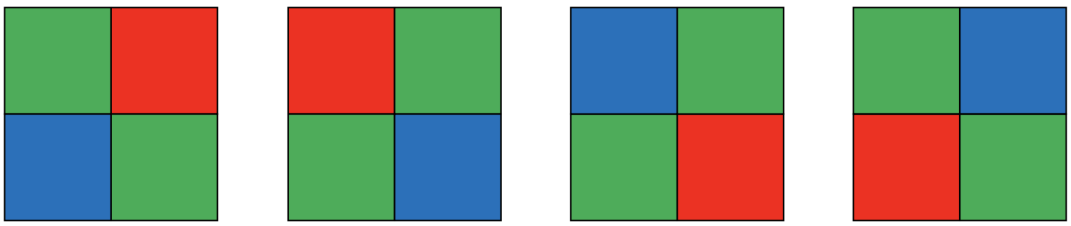

Figure 3: From left to right: ´grbg´ , ´rggb´, ´bggr´, ´gbrg´.

White balancing. Before converting the mosaiced image into a proper RGB images, cameras first apply a step called *white balancing*. Roughly speaking, this involves changing the

relative weights of the red, green, and blue measurements, to make sure that materials that are normally white also appear white in the image.

Two of the most common algorithms for white balancing use the so-called *gray world* and *white world* assumptions, and correspond to the transformations show bellow, where white balancing using the gray world assumption is shown on the left and the white world assumption is shown on the right.

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} \frac{G_{avg}}{R_{avg}} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{G_{avg}}{B_{avg}} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} \frac{G_{max}}{R_{max}} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{G_{max}}{B_{max}} \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{1}$$

- Implement both gray world and white world white balancing. At the end of the assignment, check what the image looks like under both white balancing algorithms, and decide which one you like best. (See help for function mean.)

Demosaicing. After white balancing, you want to demosaic the image. This means that you want to convert the partial red, green, and blue color channels you have available because of mosaicing, into three full-resolution color channels.

- Use bilinear interpolation for demosaicing, as shown in Figure 4. Do not implement bilinear interpolation manually! Instead, read the documentation to learn how to use Matlab′s `interp2` function for this purpose.

Brightness adjustment and gamma correction. You now have a 16-bit, full-resolution, linear RGB image. Because of the scaling you did at the start of the assignment, the image pixel values may not be in a range appropriate for display. As a result, when displaying the image, it will appear very dark.

- Brighten the image by linearly scaling it by some number.
- Select the scale as a percentage of the pre-brightening maximum grayscale value. (See help for `rgb2gray` for converting the image to grayscale). The correct percentage is highly subjective, so you should experiment with many different percentages and report and use what percentage looks best to you. Before having an image that can be properly displayed, the last step you need to do is tone reproduction, also known as *gamma correction*. This is a non-linear transformation of intensity values, which roughly speaking is used to better match sensor measurements to the response function of common displays. For this, implement the following non-linear transform, then apply it to the image:

$$C_{non-linear} = \begin{cases} 12.92 \cdot C_{linear}, & C_{linear} \leqslant 0.0031308 \\ (1 + 0.055) \cdot C_{linear}^{\frac{1}{2.4}} - 0.055, & C_{linear} \geqslant 0.0031308 \end{cases} \tag{2}$$

  where `C = R,G,B` is each of the red, green, and blue channels. This function may look completely arbitrary, but it comes from the `sRGB` standard. It is a good default choice if the camera′s true gamma correction curve is not known.

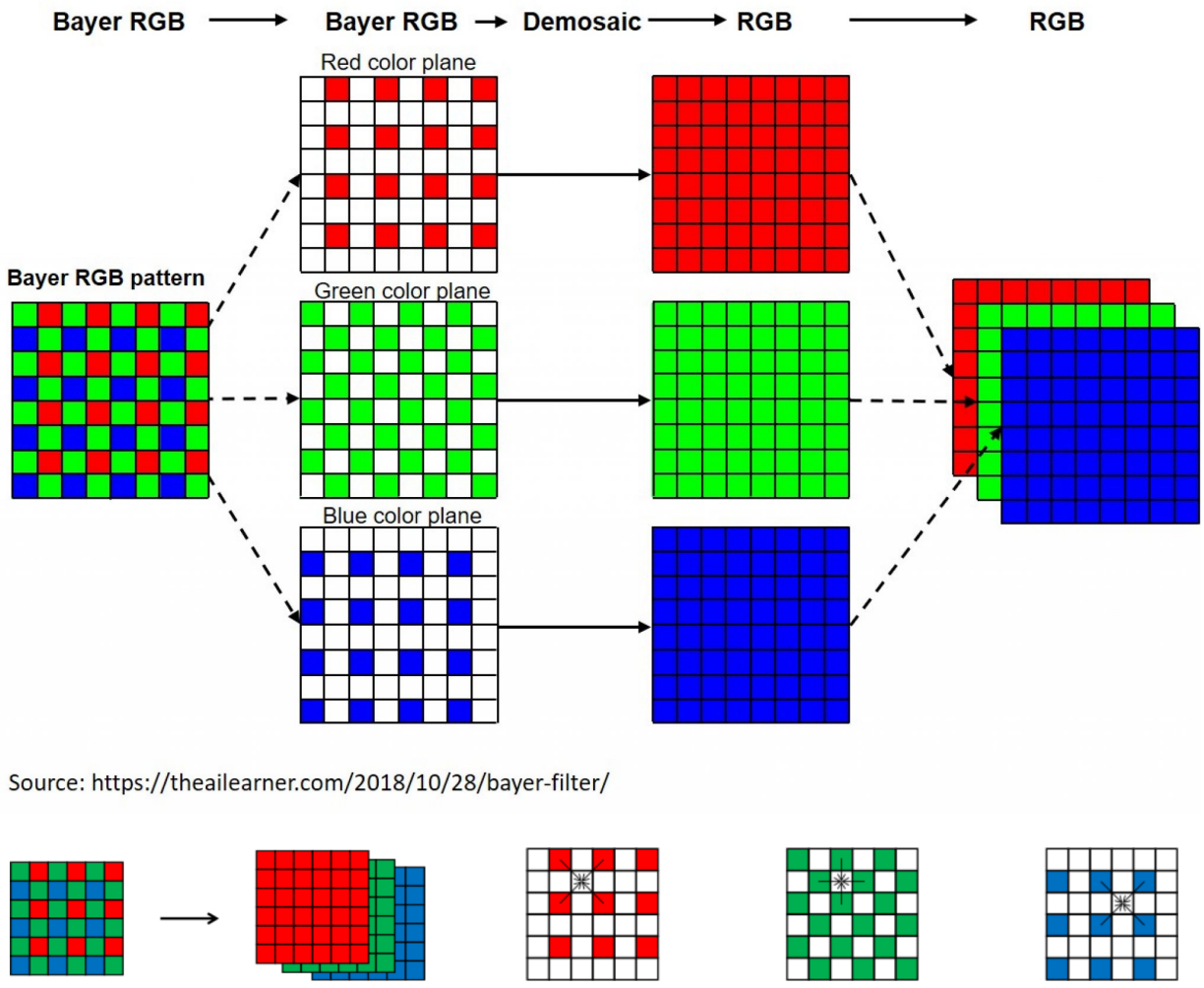

Figure 4: The demosaicing process. Demosaicing is the process of turning ′filling-in′ gaps in the three color channels, to obtain a full-resolution RGB image. This can be done by performing bilinear interpolation in each of the partial color channels measured by the sensor.

Compression. Finally, it is time to store the image, either with or without compression. Use the `imwrite` command to store the image in `.PNG` format (no compression), and also in `.JPEG` format with quality setting 95%. This setting determines the amount of compression.

- Can you tell the difference between the two files?
- The compression ratio is the ratio between the size of the uncompressed file (in bytes) and the size of the compressed file (in bytes). What is the compression ratio?
- By changing the JPEG quality settings, determine the lowest setting for which the compressed image is indistinguishable from the original. What is the compression ratio?

4 Hints and Information

- To get help on a particular function in Matlab, type `help <function>`. To get a list of functions related to a particular keyword, use the `lookfor` function. We will be making

extensive use of the *Image Processing Toolbox*, and a list of the functions in that toolbox is generated by typing `help images`. Print your results (to a printer or to a file) using the `print` command, and when making hardcopies please save space by using subplot whenever possible.

```
print -dpng output.png % output a .PNG image
```

- You will find it very helpful to display intermediate results while you are implementing the image processing pipeline. However, before you apply the brightening and gamma correction, you will find that displayed images will look completely black. There are several ways to deal with this issue. Suppose you want to display an intermediate image `im_intermediate`:

  ```
  Figure; imshow(im_intermediate,[]); % using this option the imshow function scales automatically the dynamic range of the image
  ```

  or

  ```
  Figure; imshow(min(1,im_intermediate*5));  % using this option image brightness is increased by 5.
  ```

  Figure 5 shows what you should see using this command after the linearization step.

Figure 5: Left: The linear RAW image (brightness increased by 5). Right: Crop showing the Bayer pattern.

- The colon operator : allows to form arrays out of subsets of other arrays. In the following example, given an original image `im`, it creates three other images, each with only one-fourth the pixels of the originals. The pixels of each of the corresponding sub-images are shown in Figure . You can also use the function `cat` to combine these three images into a single 3-channel RGB image.

```
% extract RED, GREEN and BLUE channels
img1=img(1:2:end,1:2:end);
img2=img(1:2:end,2:2:end);
img3=img(2:2:end,1:2:end);

% combine the above images into an RGB image, such that img1 is
% the RED, img2 is the GREEN, and img3 is the BLUE channel.
```

```
im_rgb = cat(3,img1,img2,img3);
```

Im1	Im2	Im1	Im2	Im1	Im2
Im3		Im3		Im3	
Im1	Im2	Im1	Im2	Im1	Im2
Im3		Im3		Im3	
Im1	Im2	Im1	Im2	Im1	Im2
Im3		Im3		Im3	

5 References

[1] Rob Sumner, **Processing RAW Images in MATLAB**, 2014;