16-385 Computer Vision, Spring 2018

Homework Assignment 0
The Image Processing Pipeline

The purpose of this 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 see later in the class, the image processing pipeline is the sequence of steps that happens inside a 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. There is a "Hints and Information" section at the end of this document that is likely to help. Uniquely to this assignment, we also provide a solution in the `./solution` directory of the homework ZIP archive—but you should really, really, *really* first try to solve the assignment on your own, before looking at the solution.

Throughout this problem, 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. We did some very slight pre-processing to the original RAW file, 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.

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

Initials. 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 vallue 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.

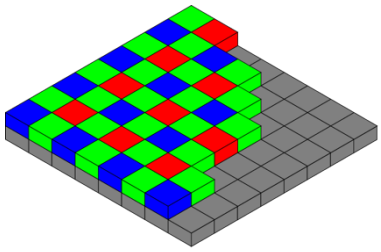

Figure 2: The Bayer pattern.

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.

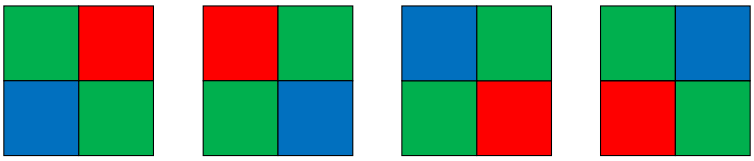

Figure 3: From left to right: '`grbg`', '`rggb`', '`bggr`', '`gbrg`'.

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.)

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.

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

Figure 4: White balancing using the gray world assumption (left) and the white world assumption (right).

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 in Figure 4. 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 demosaicking, as shown in Figure 5. 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 (20 points). 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.

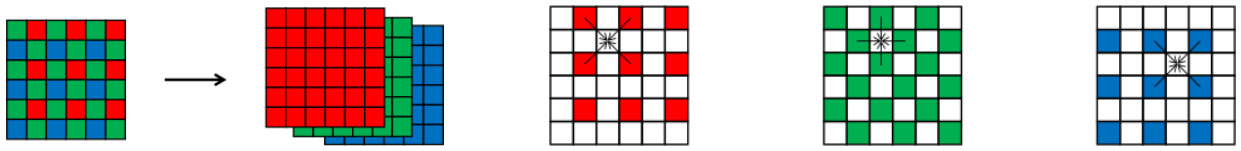

Figure 5: Demosaicing is the process of turning "filling-in" gaps in the three color channels, to obtain a full-resolution RGB image. This can be done by performing bilinear interpolation in each of the partial color channels measured by the sensor.

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:

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

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. We will discuss sRGB in class, but you are welcome to read up on Wikipedia.

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?

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. As an example, the following Matlab script loads three images, displays them in a figure and prints the figure to a PNG file.

```
% read three images from current directory
im1 = imread('image1.tiff');
im2 = imread('image2.tiff');
im3 = imread('image3.tiff');

% display images in a figure, side-by-side
figure;           % create a new figure
imshow(im1);      % display an image
title('Image 1'); % add a title

% print the displayed figure a PNG file. You can also print from the figure menubar.
```

```
print -dpng output.png
```

- 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. To be able to see something more meaningful, you can use the following command to display an intermediate image `im_intermediate`.

  ```
  figure; imshow(min(1, im_intermediate * 5));
  ```

  For example, Figure 6 shows what you should see using this command after the linearization step.

Figure 6: 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.

  ```
  % create three sub-images of im as shown in figure below
  im1 = im(1:2:end, 1:2:end)
  im2 = im(1:2:end, 2:2:end);
  im3 = im(2:2:end, 1:2:end);

  % combine the above images into an RGB image, such that im1 is the red,
  % im2 is the green, and im3 is the blue channel
  im_rgb = cat(3, im1, im2, im3);
  ```

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	

Credit

The RAW image used in this assignment, and some inspiration for the questions, came from Robert Sumner's popular guide for reading and processing RAW files.