Homework Assignment 1

15-463/663/862, Computational Photography, Fall 2017
Carnegie Mellon University

Due Thursday, Sep. 14, at 11:59pm

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 (without denoising and color transform steps), to convert a RAW image into an image that can be displayed on a computer monitor or printed on paper. There is a "Hints and Information" section at the end of this document that is likely to help.

1. Implement a basic image processing pipeline (90 points)

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 problem, 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 of the image processing pipeline.

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

Initials (5 points). 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 (5 points). The 2D-array is not yet a linear image. As we discussed in class, it is possible that it has an offset due to dark noise, 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 $2^{14} - 1$). 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 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 (20 points). As we discussed in class, most cameras use the Bayer pattern in order to capture color. The same is true for the camera used to capture our RAW image.

We do npt know, however, the exact shift of the Bayer pattern relative to our image: If you look at the top-left 2x2 square of the image file, it can correspond to any of four possible red-green-blue patterns, as shown in Figure 2.

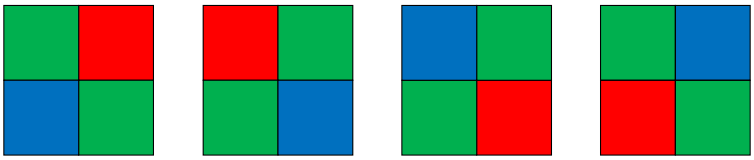

Figure 2: From left to right: '`grbg`', '`rggb`', '`bggr`', '`gbrg`'.

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 (20 points). After identifying the correct Bayer pattern, we want to do white balancing. Implement both the white world and gray world automatic white balancing algorithms, as discussed in class. 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 (25 points). After white balancing, you want to demosaic the image. Use bilinear interpolation for demosaicking, as discussed in class. 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. Additionally, as we discussed in class, the image is not yet gamma-corrected. 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 (gamma correction). 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 more detail in class during the color lecture, but you are welcome to read up on it on Wikipedia.

Compression (5 points). 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?

2. Bonus: Perform manual white balancing (10 points)

As we discussed in class, one way to do manual white balancing is by: 1) selecting some patch in the scene that you expect to be white; and 2) normalizing all three channels using weights that make the red, green, and blue channel values of this patch be equal.

Implement this manual white balancing algorithm, and experiment with different patches in the scene. Show results for using different patches, and discuss which patches work best. (See help for functions `ginput` and `impixelinfo`.)

3. Bonus: Capture and process your own images (10 points)

Use your own camera (or one that you borrowed) and take a RAW photograph. If you only have a phone camera, you can store RAW images using the Lightroom application (select the option to save as `.DNG`).

Depending on what camera you use, the RAW file will have that is camera-dependent. According to Wikipedia, this extension can be any of the following: `.IIQ` (Phase One), `.3FR` (Hasselblad), `.DCR`, `.K25`, `.KDC` (Kodak), `.CR2` (Canon), `.ERF` (Epson), `.MEF` (Mamiya), `.MOS` (Leaf), `.NEF` (Nikon), `.ORF` (Olympus), `.PEF` (Pentax), `.RW2` (Panasonic), `.ARW`, `.SRF`, `.SR2` (Sony), or `.DNG` (if you use Lightroom).

You cannot directly load these images into Matlab. Instead, you will need to convert them into `.tiff` files using a command-line tool called `dcraw`[1]. You can do the conversion by calling `dcraw` as follows.

```
dcraw -4 -D -T <RAW_filename>
```

4. Bonus: Learn to use dcraw (5 points)

Beyond converting RAW files to `.tiff`, `dcraw` provides options to emulate all steps in the image processing pipeline, including steps that are camera-dependent.

Inside the `./data` directory of the homework ZIP archive, you will find the file `banana_slug.CR2`. This is the original RAW file captured by the camera, before we converted it to `.tiff` for the main part of the assignment. Read through `dcraw`'s documentation, and figure out what the correct flags are in order for `dcraw` to perform on the `.CR2` file all the image processing pipeline steps you implemented in Matlab.

Deliverables

As described on the course website, solutions are submitted through Canvas. Your solution should be an archive (e.g., a ZIP file) that includes the following:

- A PDF report explaining what you did for each problem, including answers to all questions asked throughout Problem 1, as well as any of the bonus problems you choose to do. The report should include any figures and intermediate results that you think may help. Make sure to include explanations of any issues that you may have run into that prevented you from fully solving the assignment, as this will help us determine partial credit. The report should also explain any `.PNG` files you include in your solution (see below).
- All of your Matlab code, including code for the bonus problems, as well as a `README` file explaining how to use the code.
- At least two `.PNG` files, showing the final images you created in Problem 1 with the two different types of automatic white balancing. You can also include `.PNG` files for various experimental settings if you want (e.g., different brightness values).
- If you do Bonus Problem 2: `.PNG` files, showing the result of your manual white balancing for different patch selections.
- If you do Bonus Problem 3: The original RAW files you used, as well as corresponding `.PNG` files after processing them.

[1] https://www.cybercom.net/~dcoffin/dcraw/

- If you do Bonus Problem 4: The `.PNG` file produced by `dcraw`. If you do Bonus Problem 4, make sure to mention in the PDF report the exact `dcraw` flags you used.

Please organize your solution submission using the following file structure:

```
.zip
├─ .pdf ..... The PDF report.
├─ src/ ..... Contains all Matlab M-files and the README file explaining how to use the code.
└─ data/ ..... Contains all image and other data files.
```

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
```

- 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);
```

- 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 3 shows what you should see using this command after correctly performing the linearization step.

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	

Figure 3: Left: The linear RAW image (brightness increased by 5). Right: Crop for showing the Bayer pattern.

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.