Homework Assignment 1

15-463/663/862, Computational Photography, Fall 2019
Carnegie Mellon University

Due Friday, Sep. 13, at 11:59pm

This assignment consists of two parts. The purpose of the first part 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). You will use this to turn the RAW image into an image that can be displayed on a computer monitor or printed on paper. The purpose of the second part is to introduce you to very basic principles of image formation and exposure control. For this, you will build your own simple *camera obscura*, which is basically a fancy term for describing a pinhole camera.

There is a "Hints and Information" section at the end of this document that is likely to help.

1 Developing RAW images

For this problem, you will use the file `banana_slug.CR2` 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. As we discussed in class, RAW images do not look like standard images before first undergoing a "development" process[1]. The developed 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.

1.1 Implement a basic image processing pipeline (80 points)

RAW image conversion (5 points). The RAW image file cannot be read directly into Matlab. You will first need to convert it into a `.tiff` file. You can do this conversion using a command-line tool called `dcraw`[2]. After you have downloaded and installed `dcraw`, you can call it to perform the conversion as follows:

[1] The verb "develop" comes from the analogy with the process of developing film into a photograph. As discussed in class, the same process is often also called "rendering" the RAW images

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

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

Matlab 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 pixel the image has, its width and its height. 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 over-exposure. 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 transformation (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 not know, however, the exact shift of the Bayer pattern. 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.

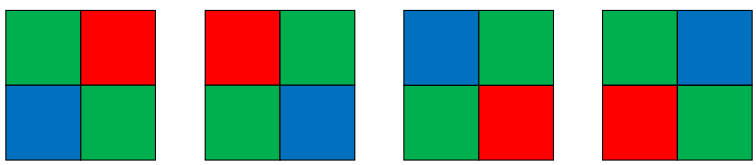

Figure 2: From left to right: '`grbg`', '`rggb`', '`bggr`', '`gbrg`'.

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 (10 points). After identifying the correct Bayer pattern, we want to perform 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 (15 points). Once white balancing is done, it is time 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.

Brightness adjustment and gamma correction (15 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. Moreover, the image is not yet gamma-corrected. As we discussed in class, this means that when you display 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.) What the best percentage is is a highly subjective judgment call, so you should experiment with many different percentages and report and use the one that looks best to you.

You are now one step away from having an image that can be properly displayed. The last thing you need to take care of is tone reproduction (also known as gamma correction). For this, implement the following non-linear transformation, 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 odd-looking function is not arbitrary: it corresponds to the tone reproduction curve specified in the *sRGB standard* [1]. It is a good default choice if the camera's true tone reproduction 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?

1.2 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 algorithm, and experiment with different patches in the scene. Show the corresponding results, and discuss which patches work best. (See help for functions `ginput` and `impixelinfo`.)

1.3 Learn to use dcraw (10 points)

Besides converting RAW files to `.tiff`, `dcraw` provides options to emulate all steps in the image processing pipeline, including steps that are camera-dependent. Read through `dcraw`'s documentation, and figure out what the correct flags are in order for `dcraw` to perform all of the image processing pipeline steps you implemented in Matlab. Make sure to report the exact `dcraw` flags you used.

2 Camera Obscura

A camera obscura is essentially a dark box with a pinhole on one face (hence also "pinhole camera"), and a screen on the opposite face. Light reflecting off an object is directed through the pinhole to the screen, and an inverted image of the object forms on the screen. All it takes to make such a camera is having a paper box and something you can use to pinch small holes on it. The caveat is that it will be hard to see the image formed with your naked eye. Instead, you will need to attach to the pinhole camera a digital camera with a long exposure time. Figure 3 shows a schematic of a pinhole camera constracted in the way described above.

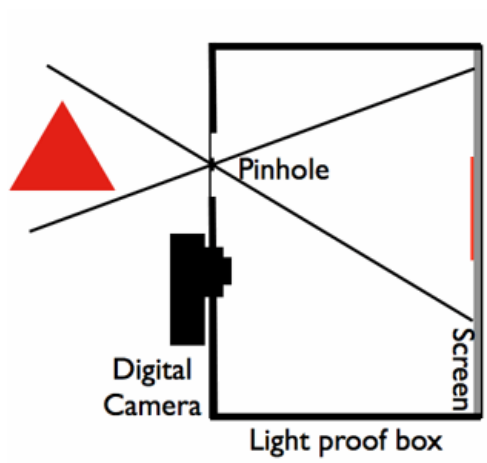

Figure 3: Schematic of a camera obscura.

2.1 Build the pinhole camera (70 points)

To design a basic easy-to-use pinhole camera, we recommend the following steps:

- Get a cardboard box. It does not have to be too big, but it does have to ultimately be "lightproof". A shoebox will do. The cardboard box should be such that the distance between the pinhole and the screen (i.e., the focal length) is longer than the minimum focus of the digital camera. Otherwise, all of the images you will capture will be blurry.
- Obtain a digital camera that allows for long exposure times, around 15-30 seconds. (See Hints and Information.)
- Determine which face of the box should be your screen. Cover the screen with white paper (printer paper geerally works fine). Cover the rest of the faces on the inside with black paper. If you prefer, you can use white and black paint instead of sheets of paper.
- On the face opposite the screen, cut a large hole. This hole should be bigger than the pinhole, e.g., about 1 inch in diameter.
- Take another piece of black paper, and punch the pinhole into that. Then, tape this piece of paper over the bigger hole on the box, so that light only enters through the pinhole. The advantage of this is that you can use multiple pieces of paper to test pinholes of varying diameters. Make your design so that you can easily switch between papers with different pinhole diameters.
- Next to the hole for the pinhole-carrying paper, cut a hole for the digital camera. The size of the hole should be such that you can attach the digital camera's aperture to it in a light-tight way—no light should enter through this hole. Additionally, the digital camera's hole should not be too far from the pinhole, as otherwise the digital camera's field of view may not be wide enough to capture the screen's image. You may need to angle the digital camera a bit towards the pinhole. At the same time, you should make sure that the digital camera is not blocking the light path inside or outside the box.
- Make sure that your digital camera is charged and has a memory card. Then attach it to the box. While attaching it, make sure that it is still possible to change its settings (e.g., focus, exposure, ISO) without destroying all of your construction. Turn off autofocus and adjust the focus of the camera manually so that it is focused on the screen.
- Duct tape your box all over! You really want to make it light proof.

Once everything is done, submit a few photos of your constructed pinhole camera. Make sure to also report on all your design decisions, such as screen size, focal length, and field of view.

2.2 Use your pinhole camera (30 points)

It is now time to capture some images with your pinhole camera. We leave it up to you to identify interesting scenes. Once you have found what you want to photograph, point the pinhole towards it. Figure out the appropriate settings for the digital camera (see Hints and Information), then set it to capture an image for 16-30 seconds.

You should capture at least three scenes, each with three different pinhole diameters, for a total of nine photographs. Some suggested pinhole diameters are 0.1 mm (really just a pinprick), 1 mm , and 5 mm. These diameters are suggestions: in reality, your pinhole diameter should be about $1.9\sqrt{(f\lambda)}$, where f is the focal length, and λ is the wavelength of light (550 nm on average, for visible light). If you use this formula, then also go a few millimeters up and down, in order to have three pinhole diameters in total.

Report what pinhole diameters you use, and discuss the differences you observe for the different pinholes.

2.3 Bonus: Camera obscura in your room (20 points)

Instead of using a cardboard box, you can build a camera obscura using a room with a window. Use thick black paper to cover the window, and pinch a small hole at its center. Then, on the wall opposite the window, you will see an inverted projection of the view outside your room. Use a digital camera to capture an image showcasing this projection. See Abelardo Morell's Camera Obscura gallery for inspiration and some further instructions. Antonio Torralba and Bill Freeman also wrote a fun computer vision paper about room-sized pinhole cameras [2], which is definitely worth a read.

3 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 Problems 1 and 2, 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.
- For Problem 1.1: At least two `.PNG` files, showing the final images you created 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).
- For Problem 1.2: `.PNG` files, showing the results of manual white balancing for different patch selections.
- For Problem 1.3: The `.PNG` file produced by `dcraw`.
- For Problem 2.1: At least two photographs of your constructed pinhole camera.
- For Problem 2.2: At least nine photographs captured with your pinhole camera (three different scenes, each captured with three different pinholes).
- For Bonus Problem 2.3: A photograph of your room's "screen" with the projection of the view from outside, as well as a photograph of your covered window "pinhole".

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.
```

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. 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 4. 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	

- 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));
```

  Figure 4 shows what you should see with this command after the linearization step.

- In many point-and-shoot cameras, as well as in most manual-control cameras (DSLR, rangefinder, mirrorless, and so on), exposures up to 30 seconds are readily available. If this exceeds your camera's settings, you can reduce exposure times by using a large aperture size for your lens, as well as a large ISO setting (possibly up to 1600).

  Modern smartphone cameras typically have a hardware limit on maximum exposure time at around 4 seconds. However, many phones nowadays support a "long exposure" mode, where longer exposure

Figure 4: Left: The linear RAW image (brightness increased by 5). Right: Crop showing the Bayer pattern.

times are simulated by capturing a sequence of images and summing them all up at the RAW stage. This article provides some information for both iOS and Android phones.

If you decide to use your smartphone, you should also use an app that enables manual control of camera settings (especially focus, exposure time, and ISO). Additionally, make sure to disable the flash.

- Below are some tips that will hopefully make capturing images with your pinhole camera easier:
  1. We strongly recommend that you go outside during a sunny day. Capturing images indoors will require extremely long exposures, which is impractical.
  2. Make sure to place your pinhole camera on a surface that is absolutely steady throughout the exposure time. Definitely do not hole it with your hand.
  3. When focusing your digital camera, do so with the aperture of your lens fully open (even if you later decide to stop down the aperture while capturing images.
  4. If your photographs are blurry, it could be because either your camera is not focused correctly on the screen, or the pinhole is too large. To make sure that it is not a focusing issue, place a piece of paper with text on the screen and take a photo (or look at the viewfinder of your digital camera) with your box open. Make sure the camera is focused so that the text appears sharp. If the issue is the size of your pinhole, try another one that is smaller.

5 Credit

The RAW image and some inspiration for the first part of this assignment came from Robert Sumner's popular guide for reading and processing RAW files. Some inspiration for the second part came from James Hays' and Gordon Wetzstein's computational photography courses, at Brown and Stanford respectively.

References

[1] International Electrotechnical Commission and others. IEC 61966-2-1:1999. *Multimedia systems and equipment–Colour measurements and management–Part 2-1: Colour management–Default RGB colour space–sRGB*, 1999.

[2] A. Torralba and W. T. Freeman. Accidental pinhole and pinspeck cameras: Revealing the scene outside the picture. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 374–381. IEEE, 2012.