Assignment 1: Camera Pipeline - Lab session 1 -

1 Introduction

The purpose of this assignment is to learn about the in-camera image processing pipeline. You will build your own version of a very basic image processing pipeline, which will transform a RAW image, as captured by the camera sensor, into an image that can be displayed on a computer monitor or printed on paper. Figure 1 shows the basic camera pipeline that you will implement.

RAW images do not look like standard images before undergoing a rendering process first. The exact result can vary greatly depending on the choices you make in your implementation of the image processing pipeline. In general, every camera uses different algorithms and the processing order may vary. Most of this information is proprietary and it is usually not disclosed. In this assignment, we ask you to implement some standard techniques for every stage of the camera pipeline so you familiarize with all the processing an image undergoes after it is captured by the sensor.

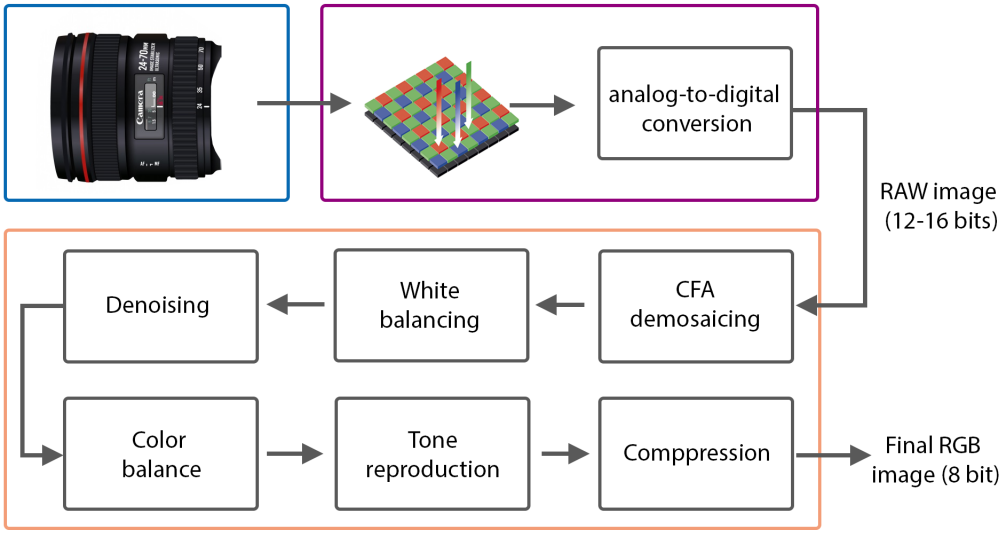

Figure 1: The in-camera image processing pipeline. A sequence of image processing operations by the camera's image signal processor (ISP) to convert a RAW image into an image ready to be displayed or printed.

2 Importing your image

For this assignment, we provide you with a set of different images in both RAW and TIFF formats. All the RAW images have been captured with a Canon EOS T6i Rebel camera. Select one image that you like. You will use this image to showcase the results of the different steps you implement in this assignment. The image used to illustrate the different steps throughout most of the figures in this report is `IMG_0596`, but you can use a different one.

2.1 Reading the image

Load the image in TIFF format into MATLAB or PYTHON. 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.
- Check its width and its height.
- Convert the image into a double-precision array.
- See help for functions `size`, and `double` (to get help on a particular function in MATLAB, type `help <function>`).

Python notes: If you decide to use Python instead of MATLAB, you may need to use external libraries such as numpy, scipy, Pillow[a] or matplotlib. They are fair game, as long as you implement the same parts that you would have to implement in MATLAB (e.g., you cannot do demosaicing calling an external function).

[a] https://pillow.readthedocs.io

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 very dark (remember, these are linear images, which match poorly the non-linear manner in which humans perceive light and color).

You can use the sequence of commands `figure; imshow(img);` to display an intermediate image contained in variable `img`. Please, read `imshow`'s documentation in MATLAB to get familiar with the expected type of data and range of values (hint: the expected range for data of type `double` is within [0, 1], whereas the expected range for data of type `uint8` is within [0, 255]).

2.2 Linearization

The 2D-array is not yet a linear image. For instance, 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 bits have meaningful information, meaning that the maximum possible value for pixels is 16383 (that is $2^{14} - 1$). For the provided image file, you can assume the following:

- All pixels with a value lower than 1023 correspond to pixels that would be black (if it was not for dark noise).
- All pixels with a value above 15600 are over-exposed (these values for the black level and for saturation are taken from the this particular camera 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 1023 is mapped to 0 and the value 15600 is mapped to 1. Then, clip negative values to 0, and values greater than 1 to 1. See help for functions `min` and `max`. Figure 2 shows what the image should look like if you display it after the linearization step (remember the previous code snippet for displaying intermediate images).

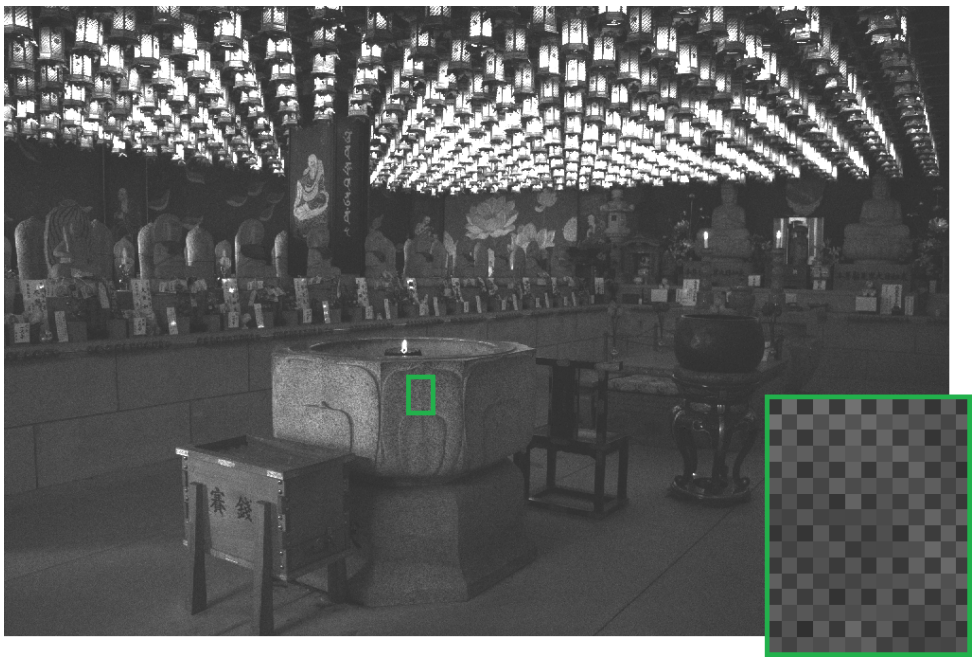

Figure 2: TIFF image after linearization. The inset shows a zoom of the image region marked in green, where the Bayer pattern can be seen.

3 Demosaicing

Once you have imported the image, it is time to demosaic the image. MATLAB has a built-in `demosaic()` function to do the debayering process. Instead of using it, in this session you are going to write your own debayering algorithm.

Most cameras use the Bayer pattern in order to capture color. The same is true for the camera used to capture your image. However, we do not know 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 3.

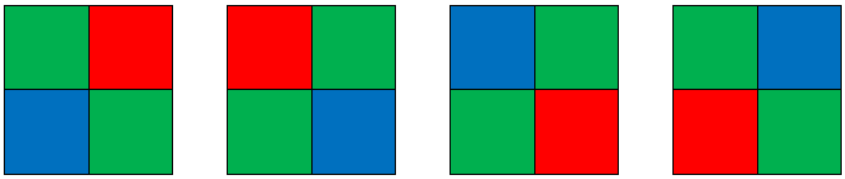

Figure 3: Bayer patterns. From left to right: 'grbg', 'rggb', 'bggr', 'gbrg'.

Once you have identified your pattern, you can demosaic the image. To implement this step, you will need to separate each of the three color channels by considering the pattern you have identified and then interpolate the missing pixels. In this session, we ask you to implement two simple techniques for interpolating the missing pixels: i) ***nearest neighbor*** and ii) ***bilinear interpolation***.

The use of one technique or another depends on the arrangement of the mixing pixels. In your report, comment on which pixel arrangements each interpolation techniques seems to be more appropriate.

After performing the demosaicing, you should see that your image already has three color channels, similar to the example shown in Figure 4. You will notice that the colors do not look

balanced (probably the image will look greenish). You will take care of this issue in the next step.

Figure 4: Color image after demosaicing.

4 White balancing

After demosaicing the image, you will perform white balancing. White balancing is the process of removing color casts so that colors that we would perceive as white are rendered as white in the final image. Cameras nowadays come with a large number of presets. You can select the kind of light under which you are acquiring the images and the appropriate white balancing is applied. Most cameras use sophisticated histogram-based algorithms in order to perform white balancing. In this assignment, you need to implement one of two simple automatic white balancing techniques: either i) the ***white world*** white balancing; or ii) ***gray world*** white balancing.

Additionally, you need to implement a **manual white balancing** as follows:

- Choose an object in the photograph that you think is neutral, somewhere between black and white in the real world.
- Compute scale factors S_R, S_G, S_B that map the object's (R, G, B) to neutral (R=G=B), i.e. $S_R = \frac{R+G+B}{3R}$ for the red channel, etc.
- Apply the scale factors to all pixels in the image.

The eventual appearance of R=G=B and, hence, of your chosen object, depends on the color space of the camera. The color space of most digital cameras is sRGB and the reference white for sRGB is D65 ($6500°K$) thus, white balancing on an sRGB camera forces your chosen object to appear $6500°K$ (blueish white). If you trust your object to be neutral, this procedure is equivalent to finding the color temperature of the illumination.

At the end of the assignment, check which image looks best to you among the images resulting from the white balancing methods. In Figure 5, you can see examples of the gray-world, white-world, and manual balancing methods. In this case, because the image has a very particular illumination, the manual white balancing is more successful than the other two methods.

(a) Gray world assumption (b) White world assumption (c) Manual white balancing

Figure 5: Example results after performing different white balancing methods.

5 Denoising

While capturing a scene, we may end up with a significant amount of noise. This noise is typically more prominent in low light conditions and it can have different sources, as we discussed in class.

The goal of denoising algorithms is to remove the noise from a given image while retaining high-frequency details. The underlying idea behind many denoising techniques is to take into account information from a number of similar pixels (in the simplest case, average them) in order to reduce noise.

In this assignment, we ask you to implement three simple denoising techniques (1 point each): i) ***mean filtering***, ii) ***median filtering***, and iii) ***Gaussian filtering***. For mean and median filter, check the lecture notes.

The Gaussian filter is a non-uniform low-pass filter which convolution kernel coefficients are sampled from a 2D Gaussian function

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where σ is the standard deviation of the distribution. Note that the mean is assumed to be zero and that the exponent is unitless, thus, the units of σ are the same as the units used for x and y. Figure 6 shows an example of one of the provided images denoised using a median filter.

TIP: If the image you are processing does not display noticeable noise in any region, artificially add noise using `imnoise(I,"gaussian")` or `imnoise(I,"poisson")` in MATLAB.

TIP: When discussing and showing the effect on the report, crop a region of the image where the noise is noticeable and show a side-by-side comparison (such as in Figure 6).

6 Color balance

The colors captured by the camera depend on the particular camera sensor spectral sensitivity. At this stage, the camera applies a color space transform that maps the white-balanced raw

(a) Before denoising (b) After median denoising

Figure 6: Example result after denoising an image using a median filter.

color values to a perceptual color space. This mapping is different for every camera and, often, it is not known.

In this assignment, you are only going to saturate the colors of your image so it looks nicer. To complete this, you are first going to transform your RGB image to the HSV color space (check the MATLAB function `rgb2hsv()`), boost the S (saturation) channel, and then transform back your image to the RGB color space (`hsv2rgb`).

7 Tone reproduction

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 the display. Moreover, the image is not gamma-corrected yet, which means that when you display the image, it will appear very dark, as you probably have noticed when visualizing the intermediate steps.

Brighten the image by linearly scaling it by some number. One option is to 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 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:

$$C_{non-linear} = \begin{cases} 12.92\, C_{linear}, & C_{linear} \leq 0.0031308 \\ (1+0.055)\, C_{linear}^{1/\gamma} - 0.055, & C_{linear} > 0.0031308 \end{cases} \quad (1)$$

where $C = \{R, G, B\}$ is each of the red, green, and blue channels. This function is not arbitrary: it corresponds to the color transfer function specified in the sRGB standard [1]. It is a good default choice if the camera's true tone reproduction curve is not known. The standard sets $\gamma = 2.4$. However, because in this assignment we are performing very simple color and tone correction operations, it is possible that this value produces very dark, or very washed-out images. You can experiment and try different values for γ that work well for your image. If your images still look too dark, you can adjust their exposure (α) before the gamma correction

step, using the formula $C_{adjusted} = C_{linear} * 2^{\alpha}$. In Figure 7 you can see an example final result at the end of the image processing pipeline that you have implemented.

Figure 7: Example final result.

8 Compression

Finally, it is time to store the image, either with or without compression. In MATLAB you can 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?

9 EXTRA: Full pipeline with a different image

In this part you will select a different image and start the pipeline with a RAW file. RAW files themselves come in many different proprietary file formats (e.g., Nikon's NEF, or Canon's CR2), and, therefore, often the RAW image file that your camera outputs cannot be read directly into MATLAB. You will first need to convert it into a different file format.

For this assignment, you only need to convert the RAW file to a file that can be read from MATLAB. However, there are many software alternatives that provide tools for reading, converting, and editing RAW files. Examples include: Adobe Camera Raw, and Adobe Lightroom (commercial); or RawTherapee, and Irfanview (free). In this case, you are going to use a decoder called `dcraw` (you can download it for free at https://www.easyhdr.com/download/dcraw). DCRAW can convert from a wide variety of different RAW file formats to TIFF image format,

which can be easily read into MATLAB. You can do this conversion using the following command line on a Windows command window `cmd`:

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

where the `-4` indicates that the output will be linear (16 bit), `-D` indicates that the output image will be totally raw, without scaling, and `-T` is for writing a tiff image instead of ppm. You can see all the different options that DCRAW offers by typing `dcraw` in the command window.

10 Deliverable

Your deliverable should be an archive file (e.g., a ZIP file) with the following items:

1. All your MATLAB code, including commented code for all the pipeline stages, as well as a README file explaining how to use the code. Comment your design choices in the code.
2. An up-to three-page (plus figures) PDF report with all the intermediate and final results obtained.
3. Size and bits per pixel of the image(s) you have selected.
4. Equation you have used to linearize the image(s). Include the parameters used, and discuss their effect.
5. Report the Bayer pattern you have identified and used to demosaic the image(s).
6. Clearly report which white balancing method has worked best for your image(s) and the reason why you think that is the case.
7. The intermediate and final images you have generated. Name the files using the source image name and the figure (and possibly subfigure) number they are shown on, e.g. `cityhall_figure_1a.jpg`, `cityhall_figure_2.jpg`. To save any intermediate image, you can simply use the `imwrite` command at any point in the pipeline. If the image files do not fit into the maximum submission size allowed by Moodle, upload them to a shared folder (e.g. Google Drive, OneDrive) and include the folder link in the README file. Make sure the folder can be accessed using any unizar.es account.

11 Evaluation

All tasks, including the "EXTRA: Full pipeline", sum up to 12 points. The maximum score is 10 points:

Importing image (Section 2)	1
Demosaicing (Section 3)	2
White balancing (Section 4)	1
Denoising (Section 5)	3
Color balance (Section 6)	1
Tone reproduction (Section 7)	1
Compression (Section 8)	1
EXTRA: Full pipeline (Section 9)	2
Total	12

12 Acknowledgments

This assignment is adapted from Ioannis Gkioulekas' Computational Photography course (Carnegie Mellon University), and strongly inspired from Processing RAW Images in MATLAB from Rob Sumner (Department of Electrical Engineering, UC Santa Cruz, 2014), and from Gordon Wetzstein's Computational Imaging and Display (Stanford, EE367/CS448I). Some of the provided sample RAW images have been obtained from: https://www.imaging-resource.com/PRODS/canon-1dx/canon-1dxA7.HTM

References

[1] "Multimedia systems and equipment. Colour measurement and management-Part 2-1: Colour management. Default RGB colour space-sRGB". In: *International Electromechanical Commission* (1999). URL: https://ci.nii.ac.jp/naid/10021143238/.