Computational Photography - Assignment 1.

March 5, 2008

1 Introduction

The objective of this assignment is for you to gain an understanding of the imaging pipeline that converts the raw output of a digital camera's sensor into the `JPEG` image that you view your computer monitor. This pipeline consists of a number of non-linear operations that are designed to give a visually pleasing image under normal shooting conditions.

In this assignment we will consider the following processing steps that were discussed in today's class:

1. Demosaicing - Converting the Bayer pixel pattern into a one where each pixel has red, green and blue color channels.
2. White Balancing - Adjusting the relative levels of the red, green and blue channels so that white objects appear white.
3. Gamma correction - Applying a non-linearity to the intensities in the image to compensate for non-linearities in the display, yielding a perceptually linear tone scale.
4. Histogram clipping - A few very dark or very bright pixels in the image will mean that majority of pixels are forced to use a very narrow part of the 0 to 255 range. By clipping these, we allow the majority of pixels to make use of the full 255 intensity values per channel.

The assignment has two main parts:

- (i) Write Matlab code that converts the raw output of the sensor to a standard `JPEG` file. This involves manually implementing the four steps above.

- (ii) Given a set of `JPEG` 's from some unknown camera, infer the non-linear mapping that the raw sensor readings must have undergone. Loosely speaking, this is the inverse of the first part.

2 Requirements

You should perform this assignment in Matlab. Please turn in all code and result figures/images. The code should be clearly commented, explaining what each line is doing.

If you are not familiar with Matlab, I suggest you go through some of the tutorials posted on the course web page.

3 RAW to JPEG conversion

In the `.zip/.tgz` file accompanying the assignment, the file `office.CRW` contains the raw unprocessed output image file from the sensor of a Canon DSLR . For your convenience, I have converted this to format that you can read into Matlab using `imread`, namely `office.pgm`[1].

You should write a set of functions taking the file as input and performing each of the four stages described above. The output should be a `.jpg` file written to disk. There should be an overall function that reads in the image, passes it through each of the four stages and then writes the result image to disk (using `imwrite`).

Points to note: remember to convert the integer pixel values to floating point values (using `im2double`) before you perform manipulations on them. Similarly, convert back to a `uint8` before saving.

3.1 Demosaicing [15%]

If you read in the image using `imread` and then display it using `imshow` or `imagesc`, you will see a strange grayscale image. This is because you are seeing the raw Bayer pattern (zoom in to see it more clearly). With the Canon DSLR's the pixels are arranged in the following pattern: `[R G ; G B]`.

[1] I did this with the program `dcraw` (freely available on the Web, just Google for it), with options `-4 -D`.

Write a function to demosaic the image. The output image should be the same dimensions as the original image, but with three color planes instead of just one. You may find the `interp2` function particularly useful.

Pick an interesting patch of the image and show the image before and after demosaicing.

See if you can find a region with spurious color artifacts resulting from this process. Show how a median filter (`medfilt2`) can reduce these.

3.2 White Balancing [15%]

White balance the image by displaying the image and getting the user to select a point, or preferably a region, which is supposed to be white in the scene. Functions such as `getpts` or `ginput` may be useful for this. Incorporate a check to warn the user if any part of the region is saturated. Anything within 95% or so of the maximum is probably unreliable. Average over this region to obtain the relative scale factors for each channel. Apply them to the image to correct the white balance.

Incorporate an option to automatically white balance the image. Make sure you describe how it works in your code comments.

Pick an interesting patch of the image and show the image before and after white balancing.

3.3 Gamma correction [15%]

The pixel intensity values in the image are currently linearly related to the number of photons hitting the sensor. However, CRT's are non-linear in their operation, yielding a non-linear tone mapping curve. Modern displays, to ensure backward compatibility, mimic this non-linearity. Thus, to ensure a perceptually linear tone mapping we must apply the inverse of this non-linearity to our image. This is known as *gamma correction*. A good approximation to the true non-linear function, widely used, is to alter the image by raising the intensity to the power $1/\gamma$, where $\gamma = 2.2$. The `.^` command should be useful for this.

However, we must be careful not to alter the relative brightness between color channels, thus you should first compute the overall intensity (grayscale) image using `rgb2gray`. Then compute the ratio of each color channel to the grayscale image. Finally, gamma correct the grayscale image and reconstruct the color channels using the new grayscale image and the original ratios.

Pick an interesting patch of the image and show the image before and after gamma correction.

3.4 Histogram clipping [15%]

The image may now look somewhat "washed out". This is because not all the dynamic range is being used effectively. This may be seen by plotting a histogram of the image using the command `hist(im(:),255);` (`im` being the variable holding your gamma-corrected image). A few very dark and bright pixels mean that the majority not able to full use the full range of values (0 to 1 if your image is a `double`).

Therefore we want to take the vast majority of pixels lying in the range $[l, u]$ and rescale them so that they are in the range $[0, 1]$. Any pixels $< l$ should be set to 0 and similarly, any $> u$ should be set to 1. Use your judgement to set l and u so that not too many pixels are excluded but good use is made of the available dynamic range.

Pick an interesting patch of the image and show the image before and after histogram clipping. Also show histograms before and after the operation.

4 Inferring the tone response curve [40%]

Cameras typically introduce non-linearities in addition to the operations you have just implemented in part 1. In this part of the assignment you will determine the overall relationship between the intensity of light falling on the sensor and the pixel values in a `JPEG` image. You will implement the approach described in the paper "Recovering High Dynamic Range Radiance Maps from Photographs" by Paul Debevec and Jitendra Malik. The paper is available on the course web page. You should read it through so you understand clearly what you are trying to do. Note that you will only be implementing the part that estimates the tone response curve, not the part that combines the multiple images to form a single high-dynamic range image.

The approach uses a set of aligned images of a scene, each taken with a different (but known shutter speed). By seeing how the pixel values vary as the ammount of light hitting the sensor varies, the overall relationship between the two can be determined.

In the `.zip/.tgz` file accompanying the assignment, there are 5 `.jpg` files, `office2_1.jpg` through `office2_5.jpg`. The shutter speeds used to capture

these images was $\{1/8, 1/16, 1/32, 1/64, 1/128\}$ of a second respectively. Also included in the `.zip/.tgz` file is the function `gsolve.m` which implements the regularized least squares optimization described in the paper. Read the comments at the top of the file to understand the format of its arguments and try to understand how it implements the equations in the paper.

Write a function which reads in the five images and plots the tone response curve for each color channel. The plot should have pixel values 0 to 255 on the y-axis and sensor exposure (linear, not logarithmic scale) on the x-axis. You should also produce a second plot where you gamma correct the sensor exposure. This should reveal a roughly (but not totally) linear plot. Both of these plots should be turned in.

The function should be structured as follows:

- Read in images.
- Create weighting function. A triangle shaped function, being zero at 0 and 255 and 128 at 128 will work well.
- Randomly pick a set of points in the image.
- Pass the pixel values at these points (for a single channel); the weighting function; the log of the shutter speeds and a regularization weight (try λ=0.1) to the `gsolve` function.
- Plot the resulting curve. Be careful to use a linear rather than logarithmic scale.
- Repeat for all color channels.

You can make the curves smoother by repeating the whole procedure many times and averaging over the resulting curves.