Assignment 1: white balance

In this assignment your task is to design and implement a solution for white balancing a photo using a flash/no-flash photo pair, and along the way to experiment with the different color adaptation models that were mentioned in the lecture. You may use either Matlab or Python.

Flash/no-flash white balance

The following instructions assume that you have access to a camera where you can control the flash, aperture, exposure, white balance settings, and which can save RAW images. To get you going, you may use a set of images that we provide (in .CR2 or .tiff formats), which were captured as described below. Note that nowadays all you need to capture a RAW image is a mobile phone. You can use the Adobe Photoshop Lightroom application (available for both Android and iOS), which can save RAW images (in DNG format) and lets you manually control exposure and white balance settings. So, even if you only have a mobile phone, you are still required to capture and use several of your own input flash/no-flash pairs, in addition to the input that we provide.

1. Measure the chromaticity coordinates of your camera's flash (done only once): The chromaticity coordinates of a particular flash unit may be measured by photographing a gray card using only the flash as the source of illumination. In other words, turn on the flash in your camera, and take a picture of a gray card (does not need to be a professional one) in an otherwise completely dark room. Set your camera's white balance to some specific setting (e.g., daylight) and NOT to the automatic (AWB) setting.

2. Capture a flash/no-flash image pair: Use your camera to take two photos of a scene: one without the flash, and another with the flash. Do this in a scene where there's only one dominant illuminant, aside from the flash (such as daylight coming from the window, or light from a ceiling lamp, but not both). Use a tripod (or some other means of ensuring the camera does not move between the two shots) to avoid the need to register the two images. It will also be necessary to make sure the camera uses the same aperture and shutter settings in both photos, so a camera with a manual mode would work better. Using the camera RAW images is likely to work better than using the non-linear JPEG versions. It is important that you use **the same white balance setting** that was used to measure the flash chromaticities.

3. Propose and implement a method that uses the measured flash chromaticities in order to perform white balance on the no-flash photograph. You may assume the simple image formation model that was also used in the spatially varying white balance work that was presented in class:

$$I = R\,(k_1 L_1 + k_2 L_2)$$

Note that unlike that work, you only know one of the two light source's chromaticities (that of the flash). But on the other hand you have two images at your disposal, rather than just one, making the problem different (and much simpler to solve).

In order to work on camera RAW images you need first to convert them into a format that Matlab or Python is able to read. One simple way of doing this is by using the `dcraw` program (source code is available here). You should run it as follows: `dcraw -T -w -o 0 -4 filename.CR2`

(replace CR2 with the appropriate suffix for your camera's RAW format) and it will write the output into filename.tiff, which is a 16-bit linear TIFF file. Note that the resulting TIFF image will likely be very dark, because it linear (not gamma-corrected) and because it does not span the entire range provided by the 16-bit TIFF format, as the camera RAW file typically only use 12 or 14 bits. One way to display the image in a nice way is: (a) convert the image into doubles; (b) scale the image values such that the minimum is 0 and the maximum is 1; (c) apply gamma correction.

Experiments and food for thought

The simplest approach is to operate directly on the RGB channels of the input images. A more principled approach involves first converting the RGB values into the CIE XYZ color space, and then to LMS responses, applying the correction on the LMS channels, and then converting back to XYZ and then to RGB. If you use dcraw to convert the input photos, you can ask dcraw to output the 16-bit TIFF directly into XYZ (use the -o 5 option, instead of -o 0). Another option is to get the images directly from the camera in sRGB (or some other color profile), and convert to XYZ in matlab using `makecform` and `applycform`. There are also various tools for doing such a conversion in Python.

As we saw in the lecture, several different XYZ-to-LMS transforms (adaptation matrices) have been proposed, some of the common ones are XYZ scaling (identity matrix), von Kries, and Bradford. Experiment and compare operating directly on RGB with operating on LMS (after converting from XYZ to LMS using each of the above three methods). Try to establish which approach produces the best results. You have to implement the conversions from XYZ to LMS yourself (it is easy enough, and you will then know exactly what you are doing), so do NOT use existing Matlab or Python tools for this. You may only use such tools for converting between sRGB and XYZ.

Your solution should be robust. But despite your best efforts it will likely work better on some images and less well on others. Identify and discuss the limitations of your approach.

Submission and Grading

Submit by midnight on April 30 (via moodle) a ZIP or RAR archive that contains your code, a document (pdf or HTML, no doc/docx!), which clearly describes your solution and reports your results, draws conclusions, and discusses limitations. In addition to the document and the code, the ZIP should include all of your input and output images (except the test input images that we provided). Do not submit huge image files (use reasonable resolution and compression settings).

Grading: exercises which will do a good job with all of the above will get up to 95. Higher grades might go to those who will go beyond: either come up with a particularly original solution, present a particularly insightful analysis, implement a particularly flexible and user-friendly tool, etc.