Fall 2016 - CS 477/577 - Introduction to Computer Vision

Assignment Four

Due: 11:59pm (*) Monday, October 17.

(*) There is grace until 8am the next morning, as the instructor will not grade assignment before then. However, once the instructor starts grading assignments, no more assignments will be accepted.

Note that this assignment has extra time relative to similarly sized ones because we have a midterm in this time period. I have budgeted for about 6 days for quiz preparation.

I recommend trying to do Part A (and perhaps Part B if you are a graduate student) before the midterm, as this might be helpful preparation for the midterm, and you probably do not want to leave the entire assignment until after the midterm.

Weight: 9 points

This assignment must be done individually

General instructions

Unless there is a good reason others, you should use Matlab for this assignment. **If you want to use any other language, you must discuss this with the instructor at lease one week before the due date**. If permission is granted, you will have to sort out any image handling and numerical library support on your own or as a group. (The IVILAB has support for C/C++ , with example files geared towards this course that can be made available on request).

You need to create a PDF document that tells the story of the assignment, copying into it output, code snippets, and images that are displayed when the program runs. Even if the question does not remind you to put the resulting image into the PDF, if it is flagged with ($), you should do so. I should not need to run the program to verify that you attempted the question. See

http://kobus.ca/teaching/assignment-instructions.pdf

for more details about doing a good write-up. While it takes work, it is well worth getting better and more efficient at this. A substantive part of each assignment grade is reserved for exposition.

Assignment specification

This assignment has five parts, two of which are required for undergrads, and four of which are required for grads. The rest are optional (modest extra credit is available). You can also do extra problems from the previos assignment for modest extra credit.

To simplify things, you should hard code the file names in the version of your program that you hand in. You can assume that if the grader needs to run your code, they will do so in a directory that has the files linked from this page.

Part A (Required for both undergrads and grads)

The following seven files

http://kobus.ca/teaching/cs477/data/4-1.tiff
http://kobus.ca/teaching/cs477/data/4-2.tiff
http://kobus.ca/teaching/cs477/data/4-3.tiff
http://kobus.ca/teaching/cs477/data/4-4.tiff
http://kobus.ca/teaching/cs477/data/4-5.tiff
http://kobus.ca/teaching/cs477/data/4-6.tiff
http://kobus.ca/teaching/cs477/data/4-7.tiff

are seven images taken of a Lambertian surface with the light at seven different directions. Those directions are list in order in this file:

http://kobus.ca/teaching/cs477/data/light_directions.txt

You can assume that the projection is orthographic, with the z axis being normal to the image plane. You may recall that this means that the point (x,y,z) is simply projected to (x,y,0). If we ignore issues of rotation and units, this is like an aerial photograph where the points are far enough away that the relief does not matter.

A possible point of confusion regarding data formats is that images are typically indexed by row (increases in the direction that you normally think of as negative Y), and column, (increases in the direction that you normally think of as positive X). It is best to keep this convention for this assignment. If you use a different one, such as using X as the horizontal axis, you will get a different solution that is more difficult to interpret.

Now the meat:

Use photometric stereo to compute the surface normals at each point of the image. Notice that the surface has different albedo in the four quadrants. The normals that you compute must be independent of this. Demonstrate that you have good values for the normals by creating an image of the surface as if it had uniform albedo, and though it was illuminated by a light in the direction of the camera (i.e., in direction (0, 0, 1). You can assume that the albedo/light combination is such that a surface that is perpendicular to the camera direction (i.e., the normal is in the direction (0, 0, 1) has pixel value (250, 250, 250).
Put this image into your writeup together with a nice readable explaination of how you got produced it. Do not forget an informative caption for the image **($)**.

Now compute a depth map of the surface, and make a 3D plot the surface z=f(x,y). Do this by integrating the partial derivatives along a path. You can specify any point you like to be the reference “sea-level” point with z=0. Put this plot into your writeup together with a nice readable explaination of how you produced it. **($)**.

Part B (Required for grad students only).

The following three files

http://kobus.ca/teaching/cs477/data/color_photometric_stereo_1.tiff
http://kobus.ca/teaching/cs477/data/color_light_colors_1.txt
http://kobus.ca/teaching/cs477/data/color_light_directions_1.txt

as well as the following three alternatives

http://kobus.ca/teaching/cs477/data/color_photometric_stereo_2.tiff
http://kobus.ca/teaching/cs477/data/color_light_colors_2.txt
http://kobus.ca/teaching/cs477/data/color_light_directions_2.txt

contain a color image, a matrix of light directions (one per row), and a matrix of light RGB (again, one per row). These lights were used to make an image of a surface (the color image). The setup is much like classic photometric stereo (e.g., Part A), except that **1)** the lights are colored and we know their RGB, and **2)** all lights were on at the same time. You further can assume that the albedo in each channel is unity—i.e., the surface is pure white. Your mission, should you choose to accept it, is to recover the surfaces and plot them (**$**). Dealing with points that are in the shadow of any light is tricky, and so I have chosen data that does not need you to do this. In both cases there is either has little or no shadowing, or it does not affect the result too much (at least in my implementation). As you might guess from the image, the second surface is the same one that I used to create the data for Part A.

I recommend thinking through the problem as best you can before consulting the hint below.

> *Hint. You might find it easiest to convert the lighting situation to an equivalent one consisting of three lights that are entirely red, green, and blue, respectively. This will demonstrate that you undertand the linearity of light aggragation and cast the problem into a more conventional photometric stereo one.*

Part C (Required for both undergrads and grads).

Background

This gzipped tar file

http://kobus.ca/teaching/cs477/data/color_constancy_images.tar.gz

contains a directory with image pairs where one of the images is taken under one light and the other image is taken under another light. Hopefully the organization is clear from the file names.
Note that some of these images are very dim because they were imaged so that specularities (if they exist) did not cause too much pixel clipping. A pixel is clipped if one of the channels would have a value of over 255, and is set to 255. You cannot trust the value of a pixel that is 255, or even close to that. However, the images were also taken in such a way to keep noise low. Hence you can scale them up for visualization, and even computation as long as you stay in a floating point representation.
These images also appear dim because they are linear images (not gamma corrected). This simplifies things because for the kind of question we want to ask of images it can matter if they are gamma corrected, and if they were we might need to linearize them first. Fortunately for us, this has already been done.

Inspection of the file

`macbeth_syl-50MR16Q.tif`

reveals that the "syl-50MR16Q" light (50MR16Q is just the Sylvania light bulb number), is reasonably close to what this camera expects, as the white patch (bottom left) looks neutral, and the R,G,B values are relatively even (not perfect, but good enough for our purposes). We will thus take the "syl-50MR16Q" as

our canonical light, and one interpretation of the color constancy problem is to make images of scenes look as though this light was shining on them (instead of the actual, unknown, light).

In color constancy we often do not care about the absolute scale of our estimate of the illumination (R,G,B), or the error in the estimated brightness of the image. We will assume that our application is inferring the *chromaticity*. To make things easier to grade, report the color of the illumination scaled so that the max value is 250. E.G., convert (125, 100, 75) to (250, 200, 150). We will consider two error measures for color constancy, both of which ignore overall brightness. In addition, you are asked to provide triplets of images: original, corrected, and canonical (target). This visual comparison is full RGB color, not chromaticity. In detail:

- **Angular error of illuminant estimate**. Here consider the illumination color, and the estimate thereof, as vectors in 3D space, and compute the angle between them in degrees. To get the angle recall that the cosine of the angle between two vectors is the dot product of them, divided by each of their magnitudes (or the dot product of unit vectors). Then you can use `acos()` to get the angle in radians and convert to degrees, or more conveniently use `acosd()`. If you do not divide by their magnitudes, then your argument to `acos()` can be greater than one in absolute value, and then the angle is a complex number.
- **Corrected image**. We can use the illumination estimate to convert an image (*original* image) to a different one which is an estimate of the image as if the scene where illuminated by the canonical light. To do this we use the diagonal transformation computed by the element-wise ratio of the canonical light RGB and the estimated light RGB. This is referred to below as the *corrected image*. If the algorithm is working well, and the diagonal model is OK, then the corrected image will be close to the image of the scene under the canonical light (*target* image). There is one such image for each scene, and they are provided for evaluation.
- **RMS (r,g) mapping error**. Having computed the corrected image, we can compute measures of how close it is to the canonical (target) image. We will focus on the RMS of the (r,g) mapping error. To compute this, you compute the *(r,g)* representation of the corrected image and the target image. Recall that $r=R/(R+G+B)$ and $g=G/(R+G+B)$). When we do this, we must exclude dark pixels. Because these images were taken carefully, we can use a relatively "dark" definition of dark. Lets exclude pixels where $R+G+B$ *i*s less than 10. (If you need to reduce this threshold, let me know by email, and/or in your writeup). The errors are the differences in *r* and *g* between the two images, pixel by pixel. The RMS error is square root of the average squared error over all the *r* and all the *g* (taken together) except where one of the original images was too dark.

Finally, we are ready to begin!

Problems

1) Average some of the pixels in the white patch in macbeth_syl-50MR16Q.tif, and then scale the result so the max is 250 to provide an estimate of the illuminant color (for the Sylvania light, which, as noted above, will be our canonical illuminant) **($)**.

> *Tip. One way to do this is to click two diagonally opposite points of a rectangle that fits comfortably within the white patch (the pixels near the edges of the patch are not reliable). Then take the average of all pixels within that rectangle*

2) Do the same for macbeth_solux-4100.tif. What is the color of the “solux-4100” light? **($)**

3) What is the angular error between the two light colors found in problems 1) and 2)? **($)**.

4) Now use the diagonal model computed from the illuminant RGB found in problems 1) and 2) to map the second (bluish) image to the one under the canonical light. This is the corrected image as described

above, where the "algorithm" is essentially an oracle because you happened to have a white patch in the image, and you (the human) found it. Display three images: The original (bluish) one, the (hopefully) improved (corrected) one, and the canonical one for reference. To make things easy to inspect, scale each of these images so that the max value in any channel is 250 **($)**.

5) Compute the RMS error in (r,g) of the pixels between A) the original image and the canonical (i.e., with no color constancy processing), and B) the (hopefully) improved corrected image and the same scene imaged under the canonical illuminant (i.e., using "oracle" color constancy) **($)**.

6) Using the MaxRGB algorithm, estimate the light color in the remaining solux-4100 images (there are three of them–apples2, ball, and blocks1). Report the angular errors between these three estimates (one for each scene) and the solux-4100 light color you measured using the macbeth color chart image **($)**.

7) Using the MaxRGB illumination estimate from problem 6), display image triplets: original, corrected image, and canonical (target) image for each of the three scenes (9 images total). For each scene this is the same process that you did in problem 4) for the Macbeth color chart using the human assisted oracle algorithm **($)**. Report the *(r,g)* RMS error for the mapped images **($)**. Is there good agreement between the general trend of the *(r,g)* RMS error and the angular error computed in problem 6) **($)**?

8) Finally, repeat the previous two questions using the gray-world method instead of the MaxRGB method **($, $)**. Which method is working better on this data **($)**?

Part D (Required for grads only).

Derive a formula for the best diagonal map between the (R,G,B) under one light and the (R,G,B) under another light using sum of squared errors as your definition of best ($). Based on your derivation, is this diagonal map guaranteed to give a better answer using the (r,g) measure than any algorithm you might invent **($)**?

Following the same pattern of investigation as in the previous part, provide before, after, and target images for the correction procedure you just derived **($)**. Provide (r,g) error estimates for this method as well **($)**.

Part E (Optional)

If you were keen on doing HW3 part E, but did not quite have time, it will be considered for grade as part of this assignment.

What to Hand In

As usual, the main deliverable will be PDF document that tells the story of your assignment as described above. Ideally the grader can focus on that document, simply checking that the code exists, and seems up to the task of producing the figures and results in the document. But you need hand in your code as well as follows.

If you are working in Matlab (recommended): You should provide a Matlab program named hw3.m, as well any additional dot m files if you choose to break up the problem into multiple files. Do not package up the files into a tar or zip file–this messes up the D2L conventions.

If you are working in C/C++ or any other compiled language you need to discuss this with the instructor at least one week before the due date: You should provide a Makefile that builds a program named hw4, as well as the code. The grader will type:

```
make    ./hw4
```

You can also hand in hw4-pre-compiled which is an executable pre-built version that can be consulted if there are problems with make. However, the grader has limited time to figure out what is broken with your build. In general a C/C++ solution will require nonstandard libraries, and you should discuss with the instructor how they can be provided as part of your submission, or assumed to exist on the system that is used for testing.

If you are working in any other interpreted/scripting language (again you need to discuss this with the instructor at least one week before the due date): Hand in a script named hw4 and any supporting files. The grader will type:

```
./hw4
```