

Spring 2019 - CS 477/577 - Introduction to Computer Vision

Assignment Three

Due: 11:59pm (*) Monday, February 04.

[Originally planned for January 30, but we did not cover enough by Jan 23 to make this reasonable]

(*) 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.

Weight: Approximately 5 points

This assignment must be done individually

General instructions

You can use any language you like for this assignment, but unless you feel strongly about it, you might consider continuing with Matlab.

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 two parts, each of which has a sub-parts that is required for both grads and undergrads, and a sub-part that is only required for grads.

To simplify things, you should hard code the file names in the version of your program that you hand in.

Learning goals

- Understand solving problems by optimization using line fitting as an example
- Additional practice presenting results
- Being able to differentiate local and global optimization (grad students)
- Become more familiar with perspective by analyzing images to judge if they are in perspective
- Practice making an argument about perspective images

Part A1

Overview. In part A you will explore fitting lines to data points using two different methods, and consider the difference between them. Recall that when there are more than two points, we do not expect a perfect fit even if the points come from a line because of noise. Fitting the line to many points mitigates the effect of noise (similar to averaging), providing that the noise is well behaved (no outliers). However,

we need to be clear about what line we are looking for (i.e., what is the definition of the “best” line). There are a number of possibilities, and we explored two of them in class.

The file

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

is an ASCII file containing coordinates of points that are assumed to lie on a line. Apply the two kinds of least squares line fitting to the data. You need to implement the fitting yourself based on formulas given in class.

There may be Matlab routines that simplify this question (e.g. REGRESS), but you need to use slightly lower level routines to show that you understand how things are computed. However, you may find it interesting to compare with the output of REGRESS or like routines.

Your program should fit lines using both methods, and create a plot (\$) showing the points and the fitted lines. For each method you should output the slope, the intercept, and the RMS error of the fit **under both models** (four numbers per fit, eight numbers total) (\$). Comment in your writeup how you expect that the error under the one model to compare with the error of the line found by the alternative model, and vice versa, referring to your results as an example (\$).

Part A2 (required for grad students only)

Regarding your comment or observation regarding how the errors associated with the two models fair against each other in the two cases, make a formal argument that shows that what you found to be the case must be the case (\$).

Both our least squares provide the true, *global* minimum for each of their respective problems. However, in most interesting computer vision and machine learning problems finding the global minimum is intractable. Hence we often have to settle for a *local* minimum. Does your argument above hold if our two least squares methods **only** guaranteed local minimum (\$) ? If yes, can you make a formal argument? If no, can you explain further (\$) ?

Part B1 (Required for both undergrads and grads)

Below is a link to an image. Dump it into a drawing program, and draw enough lines over the image to make a case that the image is either approximately in perspective or not. (Have a look at the building examples in the lecture notes if this is not making sense). You need to consider both tests covered in class with respect to the building example. You should understand and state your assumptions, and explain your reasoning.

Your deliverables for this part of the assignment is entirely within the your PDF (no code). You will provide a image with lines drawn on them and an explanation of what you conclude and why (\$). The quality of the argument is what matters most for this question. Since no image is in absolutely perfect perspective, you need to make some judgments, and these need to be articulated in your writeup.

<http://kobus.ca/teaching/cs477/data/building.jpeg>

Part B2 (Required for grads ONLY)

This question is essentially same as the previous one, except the image is harder to analyze. To get you started, you can assume that that the chandelier is perfectly symmetric.

For this image, it will also be less clear how you draw the lines, so you must also put small circles (i.e., "dots") that make it clear where the lines come from. If you color code the points and/or lines, this will help provide points of reference in your explanation. Because the analysis is a bit harder and less decisive than for the previous image, the quality of the explanation matters even more.

<http://kobus.ca/teaching/cs477/data/chandelier.tiff>

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: 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 Python: You will provide a Python program named `hw3.py` as well any additional dot py 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 should provide a Makefile that builds a program named `hw3`, as well as the code. The grader will type:

```
make ./hw3
```

You can also hand in `hw3-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.