

Final Project

As previously announced, the last homework assignment of this course is for you to work on a small project of your own. You have about two weeks to complete it, so do not be overly ambitious. Do something that amounts to about two and a half times the work of a standard homework assignment (the half comes from the fact that you have been repeatedly warned to start thinking about this).

The list below repeats the suggestions made in previous homework assignments, and adds a few more. Many of the project topics involve programming, but some of them do not.

I will be glad to help you with advice as to whether your ideas are sound or feasible. However, I will not tell you what to do. This is because the main point of this assignment is to invite you to try to formulate a problem or a direction of interest of your own. You should have enough of a feel for computer vision by now to be able to do this.

Here is what you need to do, all by electronic mail:

- Send me (tomasi@cs.duke.edu) email by Wednesday, April 7th, stating what you are going to do for the project. Just write one or two paragraphs describing the topic and the deliverable: this could be a piece of code, experiments with someone else's code (whose?), a survey, or a mathematical argument. This email is worth 15 percent of the project grade.
- By Wednesday, April 15, send me first results. If you write code, show your code running (perhaps with poor performance) on at least one input. If you run experiments, show me two experiments that compare something: perhaps the running time with or without some improvement; or a before/after comparison of the quality of some result (it is not important that the "improvement" actually makes things better; it is more important that you understand what is going on; negative results are ok). If you write a survey, send me an outline, the bulk of a draft, and a complete list of references (two or three papers at least). If you come up with a mathematical argument, send me a sketch of the argument, with some numerical values to show that the basic idea makes sense. This account (a Word or PDF document: do not send me code) is worth 35 percent of the project grade.
- By Thursday, April 22, send me a final report. This is a clear description of what you set out to do (more or less a repetition of your first email), and what happened: results, how they compare with your initial expectations (again, negative results are ok), a discussion of what your results tell you, and a brief account of what you would do if you had much more time to address the problem you chose. In the case of a survey, "results" are conceptual conclusions you draw from reading about other researchers' work, and "future work" may be an indication of research directions that appear promising to you. This final report need not be longer than two or three well written and thought-out pages. It is worth 50 percent of the project grade.

Possible Project Topics

These are only suggestions. The last five are new. You can make up similar ones on different topics.

1. Write C code, callable from MATLAB, that median-filters any input image with windows of any odd size.

For smaller windows, the algorithm used in `immedian.c` is most efficient. For bigger windows, redundancy should be exploited as illustrated in homework 1. It is your responsibility to determine when to switch from one method to the other. One way to find out is to estimate the number of operations as a function of w , and determine the break-even point theoretically. Alternatively, or, better, in addition, an empirical evaluation can be used (use the MATLAB `tic` and `toc` functions to time your program on large images, or on many images). It is useful to compare the performance of your code with that of the MATLAB image toolbox function `medfilt2`.

In addition, your code should do something better at image boundaries than just leaving them black. For instance, you could shrink the window size as you approach the boundaries. If you only shrink as much as strictly necessary, your windows become rectangular, so you may want to keep that in mind when you start writing your code.

Your report would include an explanation of your code and of your calculation of the break-even point, a diagram that shows your experimental results for the break-even point, examples of inputs, outputs, and running times on one or more images with different window sizes, and a plot of running times as a function of window size for a few different image sizes. If you use different algorithms for computing medians, list the appropriate bibliographic references.

2. The paper “Bilateral Filtering for Gray and Color Images” by C. Tomasi and R. Manduchi (*Proceedings of the Sixth International Conference on Computer Vision*, Bombay, India, pp. 839–846, January 1998) discusses a method for smoothing images while preserving edges. The paper is available through the class web page. Read the paper, and write an efficient function callable from MATLAB that implements the method.

There are no immediately obvious ways to exploit redundancy in this method, so efficiency would be achieved essentially by good programming (presumably in C) and perhaps a judicious use of table lookups.

A literal implementation, although simple, would be slow, so there is ample room for improvement here. Your implementation would fill a real, practical need, since the bilateral filter has encountered quite a bit of interest, and I do not know of a very efficient implementation.

The format of your project report would be similar to that for the previous project, with an additional section that explains the bilateral filter in your own terms. Also, the effects of filter parameters on the result are less understood than for the median filter, so your input/output examples would come with a more detailed discussion of strengths and drawbacks of the filter.

3. Write an edge linker. From homework 2 you may have learned that linking edgels from a standard edge detector may not be such a bad idea after all. The linker would take a binary edge map as its input, and output a list of edgel lists. Your code would be structured by-and-large after the thresholding stage of Canny’s edge detector. Additional code would ensure that chains of edgels are listed in the proper order from one endpoint to the other. An algorithm to bridge gaps under certain circumstances is optional.
4. Write a complete feature tracker based on the Sum of Squared Differences (SSD). Rather than following the Newton-Raphson approach of Lucas and Kanade, you could search exhaustively for the minimum of the SSD function, as done in the `brute_force` function for this homework. Your task would be to add code for feature selection, and to test the tracker through a sequence (several sequences are available on the web). Feature selection could proceed by computing eigenvalues of the moment matrix discussed in the report on tracking handed out in class. Alternatively, and more in line with a “pure black-box” approach to tracking, you could define feature points as points in the image where the minimum of the SSD function is deep enough, that is, has high enough curvature. Curvature can be estimated numerically by finite differences. Please ask me for more details if you are interested. Your method could be compared with Lucas and Kanade (available on the class web page) both in terms of quality of the results and processing speed.
5. Propose better ways to initialize EM or k-means than what is done in the code provided on the web page. Explain your rationale, and show experiments with random clouds of points. This is open-ended, and somewhat risky unless you come up with some idea.
6. Compare EM with a standard numerical optimization method like conjugate gradients. Questions of interest concern the number of iterations, the number of function evaluations, and differences in where each method converges to. I recommend doing this only if you are familiar with the theory of numerical optimization.
7. Replace k-means with EM for color segmentation. Now a pixel can be painted with a convex linear combination of the colors in the color map, according to the values of the membership probabilities $p(k|n)$. There are several issues in doing this. One is the cost of the representation: if you need to store K real numbers per pixel, you lose. However, the probabilities can be themselves compressed (how about using k-means?).
Another issue concerns what to do with the results. If we have a tree in front of a background of a different color, a color map based on EM allows you to cut the tree and paste it onto a different background using the membership probabilities as transparency values, thereby obtaining a more realistic “cut and paste.”
8. Use EM for motion segmentation. You can use Lucas and Kanade for motion measurements. I recommend very textured images, so you have many motion measurements.
9. The Marr-Poggio algorithm for stereo uses a numerical technique called relaxation to optimize a function that contains three terms: an excitation term that enforces the heuristic that surfaces are continuous, an inhibition term that enforces match uniqueness, and a third term (inhibition or excitation depending on how you formulate it) that encourages matches between similar pixels. One reference for the algorithm is D. Marr and T. Poggio, “Cooperative Computation of Stereo Disparity,” *Science*, 194, pages 283–287, October 1976. However, you may be able to find other descriptions through Google. Or you may want to dissect the code given on the class web page to understand what it does. One possible project is to understand what needs to happen in order to extend this algorithm to real images, rather than random-dot stereograms. Eric Grimson tried this already, so it is fine to read his paper and discuss its strengths and

weaknesses. Or write code if you prefer. Grimson's paper is "Computational Experiments with a Feature Based Stereo Algorithm," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(1), pages 17–34, January 1985.

10. Another project on Marr-Poggio is to try and see if you can use dynamic programming to optimize essentially the same function they use. You may have to modify the latter somewhat.
11. Quite a bit has happened in stereo since Poggio or Grimson. However, new developments have mostly to do with algorithms, rather than with a fundamental understanding of the problem itself. Some conceptual progress has come from reformulating the problem in probabilistic terms. See for instance P. N. Belhumeur and D. Mumford, "A Bayesian Treatment of the Stereo Correspondence Problem Using Half-Occluded Regions," *IEEE Conference on Computer Vision and Pattern Recognition*, pages 506–512, June 1992. Other advances were obtained by using more than two cameras. Look at some of these approaches and analyze them critically in comparison with the older work. What has improved, what has not, and what do you think needs to happen to make stereo work.
12. Use Google to look for hardware that computes stereo disparities. This may be commercial hardware or systems built by research groups. Survey what is available, and what you think performance is from reading the inventors' claims and looking at whatever results they publish. Be critical, although you need not be negative. There must be something good in all this work!