COMP 558 Assignment 2

Prepared by Profs. Siddiqi and Langer

Posted: Wed. Oct. 10th, 2018
Due: Fri. Oct. 26th, 2018 (midnight)

Introduction

This assignment covers the material from lectures 6 - 9.

In order to do the assignment, you need to know how to use Matlab and to be familiar with indexing conventions for matrices, with plots and with the various image processing commands. Use Matlab's documentation for guidance. In particular, typing "doc functioname" will give you detailed information about "functioname" and its variants. If you click on http://mathworks.com, you will also find webinars and tutorials for various computer vision and image processing tasks.

Please use mycourses->Discussions for questions related to this assignment. You are also free to discuss the questions with one another. *However, the solutions that you submit must represent your own work.* You are not permitted to copy code from each other, or from the internet.

Instructions

Submit a single zipped file **A2.zip** to the mycourses-> Assignment 2 folder. The zip file must contain:

- A PDF with figures and explanatory text for each question. We suggest that you do not spend time with fancy typesetting. Please just make sure the explanations are clear and the figures are easy to read. Also, make sure that all your figures are embedded in your PDF document so that the TAs can provide feedback.
- The Matlab code that you wrote for each part.
- The images that you used (e.g., for question 3).

In order to receive full points, your solution code must be properly commented, and you must provide a clear and concise description of your computed results in the PDF file. Please be concise but make sure you answer all parts of each question. The TAs will spend at most 30 minutes grading each assignment.

Late assignment policy: Late assignments will be accepted up to only 3 days late and will be penalized by 10 percentage points per day late, e.g. An 80 out of 100 would be reduced to 72 for being one day late. If you submit only a few minutes late, we reserve the right to treat this as "one day late".

Question 1: (Heat Equation - easy) 20 points

For this question we shall study the equivalence between blurring with a 2D Gaussian and the standard heat equation, as pointed out in the very nice paper by Hummel: “Reconstructions from zero crossings in scale space" (IEEE Transactions on Acoustics, Speech and Signal Processing 37(12), Dec. 1989). We shall use the image I(x, y) of the James Administration Building, “james.jpg”, shown below, which is provided with the assignment. You should use only one of its color channels, i.e., use R or G or B.

Referring to the lecture notes on scale spaces, you will see that there is a formal relationship between the σ of the 2D Gaussian used to blur I(x,y), and the total diffusion time used to run the heat equation) on I(x,y). Recall what that relationship is. Now recall that the heat equation is given by:

$\frac{\partial I}{\partial t} = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$ which can be discretized in time as follows: $\frac{I^{n+1} - I^n}{\Delta t} = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$.

This is a first order in time discretization, where Δt is a chosen time step parameter. The idea here is that you get I at iteration (n+1) from I at iteration n, via a simple numerical update. On the right side of the heat equation you would use central differences in space to estimate the partial derivatives at each iteration. You can do this by using the Matlab `gradient` function. If you run this discrete equation for n iterations, the effective total diffusion time will be n Δt.

First, blur the image of the James building to generate a family of smoothed images, with smoothing scales $\sigma = 4,8,12,16$. For this you can use standard Matlab functions. Second, using your own implementation of the heat equation, generate the same family of smoothed images. Compare these two results numerically, and discuss any differences you see, and explain why you think these discrepancies occur.

Question 2: (Image Pyramids and SIFT – more work) 40 points

For this question as well you will use the James Administration building image I(x,y) (again, using only one of its color channels). For all upsampling or downsampling steps you should use Matlab’s `imresize` function. This question has several subparts.

1. Generate 6 levels in a Gaussian pyramid, with each scale level given by 2^n, where n is the level, and there is a downsampling factor of 2^n in each direction. In other words, you should first have the original image which is (512x512, $\sigma = 0$). Then level 0 is the samesize, i.e., no downsampling, (512x512, $\sigma = 1$). Then you have level 1 (256x256, $\sigma = 2$), then level 2 (128x128, $\sigma = 4$) and so on, with the last image being at level 5 (16x16, $\sigma = 32$). Present your results in a composite figure.

2. Now, generate 6 levels in a Laplacian pyramid, using the same scale levels as in part 1. For this use the exact process described in the class notes, where in fact each level in the pyramid is generated by a difference of Gaussians at successive scales. To compute this difference, upsampling is required. Once again, present your results in a composite figure.

3. Why is it that the method we described to generate a Laplacian pyramid, *does not actually use convolutions with a Laplacian of Gaussian*? The answer is that there is a close relationship between convolving an image I(x,y) with a difference of Gaussians, G(x, y, k σ) − G(x, y, σ) (here k is a multiplicative factor), and convolving it with a Laplacian of a Gaussian.

The relationship was originally pointed out by Lindeberg in his discussion of scale-normalized derivatives, and was later picked up by David Lowe in his IJCV 2004 paper "Distinctive Image Features from Scale-Invariant Keypoints" in which SIFT keypoints were introduced. Referring to Lowe's paper: a) describe, mathematically, how a difference of Gaussians can approximate a Laplacian of a Gaussian, with a suitable multiplication factor and b) using Matlab's visualization functions, compare the two constructions for a fixed choice of σ and k. Qualitatively and quantitatively the two constructions should look similar but they won't be exactly the same.

4. You will now use the Laplacian pyramid you generated in part 2, to find SIFT keypoint locations. Recall that this is done by searching over all pyramid levels to find pixel locations (x,y) where the intensity I(x,y) has an extremal value (is a minimum or a maximum) both in space (at that level in the pyramid) and also in scale with respect to the levels above and below (slide 28 in lecture 9). To do the comparisons across scales you need upsampling and downsampling. For finding the extrema you can implement this strategy in a very local sense (use a 3x3 spatial neighborhood). Each keypoint should be represented by a 3-tuple (`x,y,`σ) and the full set of detected keypoints should be stored in an Nx3 matrix (assuming that you find N keypoints).

To demonstrate your detected keypoints, you can overlay them on the original image by using Matlab to draw circles at their locations on the original image I(x,y), with the radius of each such circle proportional to the scale σ of the keypoint. You could also use different colors for each scale.

Finally, to evaluate the quality of your SIFT, keypoint locator, create a slightly rotated and scaled version of the original image and find keypoints in it. You can do this by using the Matlab function `imrotate` and `resize`. (*You can restrict this evaluation to a cropped region in the original image.)* If your keypoint locations are indeed robust to scale and rotation, most keypoint locations in the first image should also be present at "corresponding" locations in the scaled and rotated version.

NOTE: We are not requiring you to build a full SIFT vector. This would also involve computing a histogram of oriented gradients and finding a characteristic orientation for each keypoint as well as storing entries for a chosen level of discretization in space and angles. We are simply asking you to evaluate the quality of keypoint location detection. You are free to use more elaborate strategies for

extrema detection if you like. We ask that in the PDF you explain what you did clearly and that you visualize your results.

Question 3: (RANSAC – some work) 40 points

In this question you will build on results that you obtained in assignment 1 question 2, where you wrote Matlab functions to detect edges, by thresholding at a chosen level of the magnitude of the image gradient. Recall that the pixels with the gradient magnitude above a suitable threshold were marked as edges, and also that the input image was chosen to have many linear structures, such as the facade of a building with rectangular windows, or an indoor office scene. For this question you will also use the image of the James Administration building, again working with only one of its color channels.

First, run your solution to assignment 1, question 2 on this image, using only one of its color channels. Use the Matlab `find` function to get the (x,y) locations of the edges. You can then use these locations to index into the gradient direction map to get the orientation at each edge location (x,y). Build a vector `theta` containing the orientation of each edge point.
Your task is to implement a RANSAC-like algorithm which takes as input these three Nx1 vectors x, y and theta, where N is the number of detected edge locations.
The template for your code should look as follows:

```
RANSAC( x, y, theta ){
    Repeat T times {     // where T is much less than the number of edges N
        Randomly sample an edge point E from the list of edge points;
        Define the line model M  through that edge point E by E's  edge location and image gradient direction;
        For every other edge point E* in the list, decide if E* is an inlier or outlier for that line model M (and count
        the number of such inliers);
                // You will need to come up with a sensible way of deciding this, and explain your idea.
        If the number of inliers for model M is greater than the number of inliers for any previous line model,
        then E is the new best edge, and so remember it and its model M and the number of inliers;
    }
}
```

Produce an XY plot of the edge points, using one color (red) to show inliers *of the best model* and another color (black) to show outliers. You do not need to fit a line to the set of inliers.

Hints:

- Be careful with the ordering of x and y indices. In Matlab, matrices are indexed by (row, column) which is standard in linear algebra. However, when the matrix is an image, the horizontal axis (x) is a column and the vertical axis (y) is a row, with lowest index at the top. So when you index an (x,y) position, you need to use index order (y,x) in the matrix, i.e. (row, column).
- For random sampling of indices of your edge points, you could use the built-in function `randi`.
- To come up with a suitable inlier criterion it you should keep in mind that an edge has both location and orientation and you need to consider both. It might be helpful to explore various distance measures use to compare locations, such as Euclidean distance, as well as to compare orientations.
- When your implementation is working, if there are multiple good models with roughly the same number of inliers, then multiple runs will give different results.