

## (Updated) Lab Assignment 2 - CSE 337/564, Fall 2007

Due: Thursday, October 18, 2007, 11:59pm

In this lab you will implement some of the image processing routines you learned about in class, using matlab. As before, submit all your work in a zip file on blackboard. Matlab is available in the CS department's UG and G labs. See me if you need an account. If you run it at home, you will need the image processing toolbox as well. Check the lab page for some example images, but look for more on the web. Also make sure you look at the web page for descriptions of the image processing tool box routines, the link is given on the lab page. Please start early.

Here is what you need to submit:

- a) All .m files
- b) A report that shows, for each question, the respective matlab code, the appropriate output (numbers, plots, images, spectra), and a narration of these (that is, a discussion of your solution and your findings, and any observations you may have made). Any questions asked in the text below should also be answered in the report. This report will form the basis for grading. Be professional about it.

1. Write a function **gconv(image,sigma)** that applies Gaussian convolution to the 2D image for the given value of sigma, using a normalized gaussian. Matlab has a function *fspecial()* which you can use to create a 2D Gaussian. Note, for larger sigmas you need to specify a larger size. Just leave out the “;” to make sure that the filter matrix falls off sufficiently towards 0. For example, for  $g=fspecial('gaussian', [3\ 3], 2)$  the matrix is not large enough, but for  $g=fspecial('gaussian', [10\ 10], 2)$  it is. If you do not make the filter array large enough, you will not get the desired degree of blurring. Use the matlab operator *imfilter()* for the convolution. Display the result. Try this with different sigmas. Practical hint: A Gaussian with  $\sigma=0.5$  will do some smoothing, keep doubling the sigma for more and more smoothing. Remember to make the filter matrix large enough.

2. Now write a function **bconv(image,L)** that convolves the image with a box filter. Use the *fspecial()* function again, now with the ‘average’ filter, which is a normalized box. You can set different widths. Display the resulting images as well. Try this with different box sizes. A box filter of width = 3 will do a minimum amount of smoothing. Wider box filters will do more. Compare with the results you got with the Gaussian filter for a comparable amount of smoothing.

3. Now find the (2D) Fourier transforms of the original images, as well as of the blurred ones you computed above. You can use matlab's *fft2()* (2D Fast Fourier Transform) function for this. Here, use *fftshift()* to put the origin of the computed spectrum (the zero-frequency band, the DC component, the average term) into the center of the plot. Also, use the *abs()* function to compute the magnitude of each (complex) frequency term before plotting. Using *log()* will bring out smaller values better, or alternatively you could bracket the values using specialized parameters in the display command (see the matlab help files). Use the routine *imagesc()* for this display since your value range will likely be outside [0, 255]. In your report put the spectra images next to the corresponding spatial images.

Recall, from question 6 and 7 of the lab 1 assignment, the *sinc*-pattern (lobes separated by zeros) in the frequency spectrum of the box function. Do you observe a pattern in the frequency transforms of the box-smoothed images as well (as you use larger and larger boxes)? Where do these patterns come from? Do the Gaussian smoothed images have that? Compare the Gaussian-filtered images with the box-filtered images, both in the spatial and in the frequency domain. Can you relate what you see in the frequency domain with what you see in the spatial domain (look at artifacts and patterns there)?

4. Now let's have a look at images with spot/speckle noise. Smooth a noisy image using filter developed above and compare the result with that obtained using a median filter. Use matlab's function *medfilt2()* to achieve the latter. What is more effective here and why? Will median filtering be useful for general blurring? Note, *medfilt2()* expects a grey level image, so use *rgb2gray()* first.

5. Implement a routine **edgeDetect(image)** that performs edge detection with the (Sobel) mask described in the notes. You can use *fspecial()* to define this filter as well. You need to create 2 intermediate images, one for the x-derivative and one for the y-derivative (use the transpose operator ‘*'* for this, that is, if *sf* is your Sobel filter in *y*, then *sf'* is your Sobel filter in *x*). Now you need to combine the 2 (x and y edge) images using the absolute value mechanism (see notes). Use *imagesc()* to display the result, since the values are out of the [0, 255] bounds again.

Display all 3 images, the x-derivative, the y-derivative, and the combined image. Use the subplot feature of matlab to create this composite display. Note, matlab also has an *edge()* function, compare your results with the results obtained using that function. Read the matlab description for the *edge()* function and you will see that much more sophistication is at work here.

6. Implement a routine called **unsharpMask(image, sigma, alpha)** that does unsharp masking with different blurring factors sigma and different weighting factors alpha, as discussed in the notes. Use your own function, and *not* the function matlab provides. Do you think the images reveal more information than the edge images, and in what sense? Use *Google Image* to find interesting images that show off the strength of this function. Submit the full-resolution images with your report.

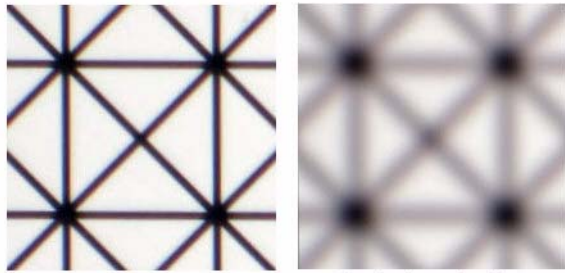
7. Compute the histogram of an image using matlab's *imhist()* function. Use a grey level image from this (or convert a color image before you go on). Take an image with a narrow intensity histogram (that is, an image that is too dark, too bright, or has little contrast). Then widen its intensity range using a simple transfer function that scales the dominant histogram intensity range  $[I_{min}, I_{max}]$  to the range  $[0, 255]$ . For this, use a linear function that maps  $I_{min}$  to 0 and  $I_{max}$  to 255 and all intensities  $I$  within that range to  $255 * (I - I_{low}) / (I_{high} - I_{low})$ . Now use this transfer function to map the pixel values of the image (representing the x-axis values of the function) to pixel values in the result function (representing the y-values of the function). This is easy in matlab, just write *out\_image=in\_image(transfer\_function)*. Use this function also for *windowing* for enhancing a certain selected intensity range in order to make small detail clearer in this range. Compare the result obtained using histogram equalization (again, use *Google Image* to find good images that show off the strength of the contrast enhancement and windowing. Submit the full-resolution images with your report.

Extra credit (+20%): Implement the multi-scale image enhancement framework discussed in class. First create the multi-scale gaussian derivative-filtered image pyramid. Then pass each detail image through an (non-linear) intensity transfer function to scale up low intensities and scale down high intensities. Then re-compose the processed details into an image. You should get an overall clearer image.

Some hints on expected results:

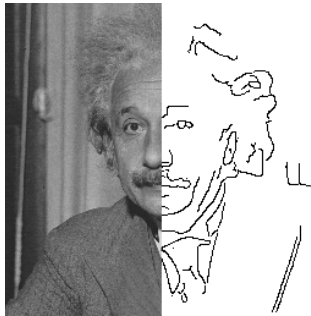
- (a) Larger boxes and larger sigmas should produce blurrier images and narrower frequency spectra (that is, the higher components are attenuated).
- (b) The blurriness and narrowing should be isotropic (that is, in all directions, not just along the x- or the y-axis).
- (c) In the edge-detected image you should see just the edges, not much else.

Some examples:

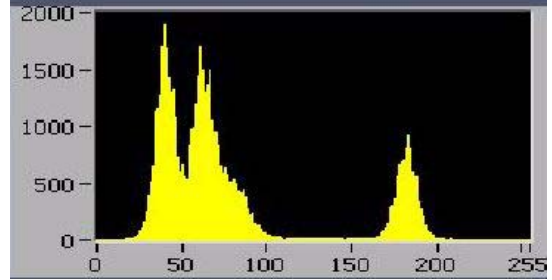
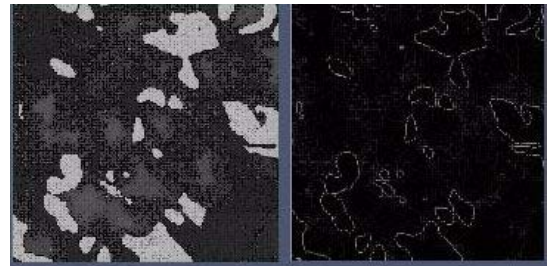


In focus      Simple Gaussian blur

Gaussian blurring



Edge detection / filtering



histogram and edge detection



unsharp masking (left: original, center and right: more detail added back in)