CSE190 – Image Processing – Homework #2

Instructor: Prof. Serge Belongie. T.A.: Josh Wills.
`http://www-cse.ucsd.edu/~sjb/classes/cse190`
Due (in class) 1:25pm Wed. Jan. 23, 2002.

Reading

- GW 10.0-10.1.
- GW 4.0-4.2, 4.6 (except for 4.6.6).

Written exercises

1. GW, Problem 10.5.
2. GW, Problem 10.8.
3. Compute $\nabla^2 f$ for the function $f(x,y) = e^{-(x^2+y^2)/2\sigma^2}$. Show your work.
4. GW, Problem 10.11.
5. GW, Problem 4.1.
6. GW, Problem 4.4.
7. GW, Problem 4.6. Explain how this relates to the Matlab function `fftshift`.

Matlab exercises

1. The Binomial Approximation

   The binomial coefficients are commonly used as a discrete approximation to the Gaussian function. Recall that the expression for a Gaussian in 1D is given by

   $$G_\sigma(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-x^2/2\sigma^2}$$

   To produce a discrete approximation to a Gaussian with effective width σ^2, use the Nth row of Pascal's triangle, where $N = 4\sigma^2 + 1$. The corresponding normalization factor is $2^{-(N-1)}$.

   (a) Make plots comparing the binomial kernel to the continuous Gaussian kernel for $\sigma^2 = 1.0, 2.5, 4.0, 5.5$. In each plot, use `stem` to plot the binomial kernel and `fplot` to plot the corresponding Gaussian on the interval $[-3\sigma, 3\sigma]$. You will need to use `hold` to put both functions on the same plot. Arrange your plots on a single sheet of paper using `subplot`.

   Things to turn in:

   - Printout for part 1a.

2. Canny Edge Detection

   (a) Examine the `gradient` function and explain how it is implemented in terms of convolution. (The commands `which` and `type` may be useful for this purpose. Alternatively, you can apply `gradient` to an impulse and inspect the result.)

(b) Implement the simplified version of the Canny edge detector as described in lecture. The syntax of your function should be as follows: `[E,M,A]=canny(I,sig,tau)`, where `E` contains the detected edges, `M` contains the smoothed gradient magnitude, `A` contains the gradient angle, `I` is the input image, `sig` represents σ for the smoothing filter, and `tau` is the threshold τ. You do not need to implement hysteresis thresholding, but you do need to implement oriented nonmaximal suppression. For extra credit, implement hysteresis thresholding with edge tracking.

(c) Run your edge detector on Figure 10.4(a) using $\tau = 5$ and the following three values for σ^2: $0.5, 1$, and 3. (Note: to save ink, invert `E` before printing it.) Discuss how the choice of σ effects the results.

(d) Apply your edge detector to Figure 1.14(c), adjusting σ and τ as you see fit. Display the resulting edges and the parameter settings used.

Things to turn in:

- Written answers to parts 2a and 2c.
- Printouts of results for parts 2b, 2c, and 2d.
- Code listing for `canny.m` in part 2b.

3. Laplacian of Gaussian Edge Detection

(a) Use `fspecial` with the `'LoG'` option to construct a 15×15 isotropic Laplacian of Gaussian kernel with `SIGMA=2`. Call the kernel `H`. Make a mesh plot of `-H`.

(b) Using `conv2` with the `'same'` option, apply `H` to Figure 10.15(a) and call the result `F`. Display the original image, the filtered image, and the filtered image thresholded at zero.

(c) Display the zero crossings of `F` as a contour plot superimposed on the raw image. (Hint: use `contour` with level set parameter `[0 0]`.)

(d) The contour plot in the previous step displays the zero crossings regardless of edge strength. We can use the gradient magnitude to suppress contours for weak edges as follows. Compute the smoothed gradient magnitude `M` as in the previous exercise, using $\sigma = 2$. Set all pixels in `F` for which `M<tau` equal to `NaN`. Pick a value of τ in the interval $[2, 6]$. Now reproduce the superimposed contour plot with this threshold in place.

You will notice that there is no choice for τ that finds all the meaningful edges in the image while suppressing the spurious ones. This is a reminder that edge detection is a *low level* operation; without high level knowledge of the objects that are meaningful in this image, one cannot hope to detect and localize the boundaries correctly.

Things to turn in:

- Printouts for parts 3a-3d.

4. Computing 1D Discrete Fourier Transforms

Compute and display the DFT of the following 1D signals. The Matlab function for the 1D DFT is `fft`. For each DFT, use `stem` to display the following three plots: (1) original signal, (2) DFT magnitude (using `abs`), (3) DFT phase (using `angle`). In the latter two plots, use `fftshift` to place the DC component at the middle. Label the axes and put a title on each plot. Set the y axis range on the phase plots to $[-\pi, \pi]$.

(a) $\delta(x - x_o)$ for $x = 0, 1, \ldots, 7$ with the following values of x_o: $0, 1, 4$.

(b) $\cos \omega_o x$ for $x = 0, 1, \ldots, 7$ with the following values of ω_o: $0, \pi/4, \pi/2, \pi$.

(c) The first difference kernel, padded with zeros to length 8: `[0.5 0 -0.5 0 0 0 0 0]`.

(d) The binomial kernel with the following values of σ^2: $1, 2, 3$. In each case, pad the kernel with zeros to make the total length 64.

Things to turn in:

- Printouts for steps 4a-4d.

5. Computing 2D Discrete Fourier Transforms

(a) Compute and display the magnitude of the DFT of Figures 4.3(a) and 4.4(a). The Matlab function for the 2D DFT is `fft2`. Use `fftshift` to move the DC component to the center, and use the log transformation so that your results look like Figures 4.3(b) and 4.4(b), respectively. In both cases, use `imagesc` with `CLIM=[5 13]` to display the DFT log magnitude.

(b) Compute and display the DFT log magnitude for an image of your own choosing, e.g. taken from the internet or your own digital camera. If the image is color, use `rgb2gray` to convert it to monochrome.

Things to turn in:

- Printouts for parts 5a and 5b.