Homework 3
Computer Vision CS 4731, Fall 2011
Due Date: Oct 13, 2011
Total Points: 28

Note 1: Both the analytical problems and the programming assignments are due at the beginning of class on Oct. 13, 2011. The analytical problems should be turned in as a hard copy at the beginning of class, and the programming assignments should be posted to the "Homework 3 Dropbox" folder on CourseWorks by the beginning of class. Please start working on your assignment early and note that there is no credit for late submissions.
Note 2: Read the guidelines for the programming assignments carefully. They are available on CourseWorks at Class Files/Shared Files/Programming Guidelines.

Problem 1: You have to smooth an MxN image (M = width, N = height) using the 3x3 averaging mask

$$\begin{bmatrix} 1/16 & 2/16 & 1/16 \\ 2/16 & 4/16 & 2/16 \\ 1/16 & 2/16 & 1/16 \end{bmatrix}$$

For simplicity, assume that the image wraps around itself, i.e. the row above the first row in the image is the last row in the image; the row after the last row is the first row in the image; the column after the last column in the image is the first column; and the column before the first column in the image is the last column. (This assumption makes your life easy as you do not have to worry about boundary conditions at the edges of the image).

- If we did this averaging naively (direct computation), how many total additions and multiplications do we need? (1 point)
- But we can surely do better than above. Show that you can compute the smoothed image using 4MN additions and 6MN multiplications. (If you can compute the final image using fewer operations, even better). (2 points + 1 bonus)

Note: When counting operations: additions and subtractions are both counted as additions; multiplications and divisions are both counted as multiplications.

Problem 2: Show that if you use the line equation $xsin(\theta) - ycos(\theta) + \rho = 0$, each image point (x, y) results in a sinusoid in (ρ, θ) Hough space. Relate the amplitude and phase of the sinusoid to the point (x, y). Does the period (or frequency) of the sinusoid vary with the image point (x, y)? Why or why not? (4 points)

Programming Assigment

Your task is to develop a vision system that recognizes lines in an image using the Hough Transform. Such a system can be used to automatically interpret engineering drawings, etc. We will call it the "line finder". Three images are provided to you (available on courseworks as hw3data.zip): **hough_simple_1.pgm**, **hough_simple_2.pgm**, and **hough_complex_1.pgm**.

Each program must accept arguments in the format specified as:
program_name {1st argument} {2nd argument} ... {Nth argument}

a. First you need to find the locations of edge points in the image. For this you may either use the squared gradient operator or the Laplacian. Since the Laplacian requires you to find zero-crossings in the image, you may choose to use the square gradient operator. The convolution masks proposed by Sobel should work reasonably well. Else, try your favorite masks. Generate an edge image where the intensity at each point is proportional to the edge magnitude. In the README, write down which mask(s) you used and why.

   p1 {input image} {output edge image}

   (5 points)

b. Threshold the edge image so that you are left with only strong edges. We will not use edge orientation information as it is generally inaccurate in the case of discrete images. Next, you need to implement the Hough Transform for line detection. As discussed in class, the equation $y = mx + c$ is not suitable as it requires the use of a huge accumulator array. So, use the line equation $xsin(\theta) - ycos(\theta) + \rho = 0$. What are the ranges of possible θ values and possible ρ values (be careful here)? You can use these constraints to limit the size of the accumulator array. Also note them down in the README file.

   The resolution of the accumulator array must be selected carefully. Low resolution will not give you sufficient accuracy in the estimated parameters, but very high resolution will increase computations and reduce the number of votes in each bin. If you get bad results, you may want to vote for small patches rather than points in the accumulator array. Note that because the number of votes might exceed 255, you should create a new 2d array and store the votes in this array. Once you're done voting, copy over the values into an output image, scaling

values so that they lie between 0 and 255. This image should be of the same resolution as your hough array, which will be different from the input image resolution, in general. In the README, write down what resolution you chose for your accumulator array (and why), what voting scheme you used (and why), and what edge threshold you chose.

p2 {input edge image} {edge threshold} {output thresholded edge image} {output hough image}

(6 points)

c. To find "strong" lines in the image, scan through the accumulator array looking for parameter values that have high votes. Here again, a threshold must be used to distinguish strong lines from short segments (write down the value of this threshold in the README; it can be different for each image, as noted in the Makefile). After having detected the line parameters that have high confidence, paint the detected lines on a copy of the original scene image (using the line() function provided in the vision utilities). Make sure that you draw the line using a brightness value that is clearly visible in the output image.

p3 {input original scene image} {input hough image} {hough threshold} {line-detected output image}

(5 points)

d. Note that the above implementation does not detect end-points of line segments in the image. Implement an algorithm that prunes the detected lines so that they correspond to the line segments from the original image (i.e. not infinite). Briefly explain your algorithm in the README.

p4 {input original scene image} {input hough image} {input thresholded edge image} {hough threshold} {cropped line-detected output image}

(5 points)

Notes:

- The data for the programming assignment can be downloaded from Courseworks as hw3data.zip. This zip file contains the test images and a Makefile which contains targets for building the programs (you will need to fill some values, such as the names of your files, thresholds, etc.), target for testing it out ('make test' to test all programs, 'make test1' to test only program p1, 'make test2' to test only program p2, etc.), and a target for submitting ('make submit').
- You should also download the vision utilities, available from courseworks. This contains useful functions for reading and writing images, accessing image data, and drawing lines. Be sure to copy the files **vision_utilities.c** and **vision_utilities.h** to your working directory so that you can compile and use them in your program.