**Canny Edge Detector  
ECE 847 Assignment #3**

**Purpose:** In this assignment you implement the Canny edge detector. You will also use the detector to find an object by matching the edges of an image with the edges of the object’s template via the chamfer distance.

**Before you start:** Watch the Lecture 3 series.

**Background info:** As mentioned in the lecture, edge detection is a classic problem in computer vision, and the Canny edge detector is a classic approach to that problem. The Canny edge detector is an important algorithm because it is easy to implement, computationally efficient, and yields good results. Moreover, it remains a useful learning exercise, because it combines several important concepts, such as gradient computation, hysteresis thresholding (double thresholding), edge linking (a modified version of floodfill), and so forth.

Intensity edges are useful for many applications, one of which is detecting objects in an image via the chamfer distance. While this approach is not the most popular these days in the computer vision community, it nevertheless has important applications in machine vision and elsewhere. In fact, the algorithm you will implement in this assignment is very similar to how researchers at Daimler-Chrysler detect pedestrians from a forward-facing video camera in the car.

**Instructions:**

1. In this assignment you will write an application (that is, you will modify the code in *homework*) to perform Canny edge detection, as well as chamfer distance matching. The code should perform the following steps in order when it is run:
   1. Reads 2 or 3 command-line parameters, which we will call *sigma*, *filename*, and *template*. In other words, the third parameter is optional when calling your program. To properly call your program a user should either specify *sigma* and *filename* or *sigma*, *filename,* and *template*. Examples (note that *homework* is the name of the program):
      1. *homework 2.0 cat.pgm*
      2. *homework 1.5 cherrypepsi.jpg cherrypepsi\_template.jpg*
   2. Construct a 1D Gaussian convolution kernel and a 1D Gaussian derivative convolution kernel, both with the specified value for sigma. The length of each kernel should be automatically selected as explained in the notes.
   3. Compute the gradient of the image. Use the principle of separability to apply the 1D Gaussian and 1D Gaussian derivative kernels. Do not worry about image borders; the simplest solution is to simply set the border pixels in the convolution result to zero rather than extending the image, but extension is fine, too.
   4. Perform non-maximum suppression using the gradient magnitude and phase.
   5. Perform thresholding with hysteresis (i.e., double-thresholding). Automatically compute the threshold values based upon image statistics.
   6. Compute the chamfer distance on the edge image with the Manhattan distance.
   7. If the third parameter is specified (the “template”), then perform Canny edge detection on the template, too. Then perform an exhaustive search (for simplicity, only consider locations for which the template is completely in bounds) for the best location of the template in the image. If the image and/or template are color, then convert from color to grayscale before computing the edges.
2. Your output should look like this:
   1. One figure window should show the original image. Seven additional figures should show the x- and y- gradient components, the gradient magnitude and angle, the edges after non-max suppression, the final resulting Canny edges, and the chamfer distance, all in separate figures. Be sure to set the title of each figure to an appropriate human-readable string that indicates what is being displayed.
   2. Print the Gaussian and Gaussian derivative convolution kernels.
   3. If the third command-line parameter (“template”) is specified, then three additional figures should be displayed, showing the Canny edges of the template, the “inverse probability map”, and the original image with a rectangle overlaid indicating the most likely location of the template (i.e., the smallest value in the inverse probability map). The inverse probability map is computed by summing the distances to the edges. We call it the inverse map simply because lower values indicate higher probability, not because it is the actual inverse of the probability.
3. The grader will test your code on the images cat.pgm and cameraman.pgm, as well as other similar images.
4. For this assignment, you may ***not*** use any Blepo functionality contained or prototyped in ImageAlgorithms.h (e.g., Chamfer), and you may not use the Gauss\*, Grad\*, Convolve, Correlate, Smooth, etc. functions prototyped in ImageOperations.h.
5. Submit your code to the grader as described in the first assignment.