Computer Science 664
Fall 2003

Assignment 1

Due Date: Monday October 6 by 5pm. Hardcopy to 4114 Upson, electronic to dph "at" cs.cornell.edu.

Material: Slides for lectures 1-4.

In this problem set you will implement a flexible template matcher and experiment with it for the task of detecting stop signs in images. You are to use the flexible template matching algorithm for a model with a central reference part, covered in the slides for lecture 4. For models of the individual parts you are to use the integral images approach covered in lecture 1, for fast detection of rectangles of various sizes and colors. Your stop sign model will be a collection of rectangles at certain locations with respect to a central reference rectangle, each rectangle having a preference for certain colors.

You are to submit an electronic copy of your source code and an executable (for Windows, Solaris or Linux). Your code should clearly indicate which part of the code is for which problem.

You are also to submit a hardcopy writeup that answers the questions posed in each problem, describes how to run your code (parameters, inputs, etc), and describes any issues you ran into in solving each problem. Again your writeup should clearly indicate which part is for which problem.

You may use the image libraries provided on the course web site, as well as the implementation of the L_2^2 distance transform covered in class (the distance transform code uses the image libraries).

1. In this problem you are to implement matching cost computations for a simple detector for a single rectangular part. This detector has a number of limitations, for instance it won't do a good job if the color balance of the images changes substantially. For this detector, a part will be specified as the 6-tuple (w, h, p_r, p_g, p_b, b), where w and h are the width and height of the rectangle in pixels, p_r, p_g and p_b are the desired proportion of red, green and blue in the rectangle (each as a number between 0 and 1) and b is the brightness of the rectangle (again as a number between 0 and 1).

   You should implement a procedure **integral** that takes as argument an RGB image and returns an RGB image that is the integral of each color plane of the input image. That is, a given pixel of the output for the R-plane contains the sum of all pixel values in the input above and to the left of that pixel (including the pixel itself). Similarly for the G- and B-planes.

You should implement a procedure **cost** that takes as arguments an RGB integral image produced by the **integral** procedure and a 6-tuple describing a rectangle. This procedure should return two arrays that specify for each location (x, y) each of two match costs c and i for placing the specified rectangle with its upper left corner at that image location.

The first match cost c is based on the difference between the observed proportion of red, green and blue in the rectangle at that location in the image and the proportions specified by the rectangle model:

$$c = 100(|s_r/s_t - p_r| + |s_g/s_t - p_g| + |s_b/s_t - p_b|)$$

where s_r is the sum of the R-plane pixel values in the image rectangle, s_g is the sum of the G-plane pixel values in the image rectangle, s_b is the sum of the B-plane pixel values in the image rectangle and $s_t = s_r + s_g + s_b$.

The second match cost i is based on the difference between the observed intensity in the rectangle and the maximum intensity (all white):

$$i = 100|(s_t/(w * h * 255)) - b|.$$

2. Using the code you implemented in the first problem you should create a very simple detector for a stop sign which just searches for a rectangle that has a high degree of red and a moderate degree of intensity. You should experiment with various combinations and settings of color and overall intensity. This detector should show the k best matching locations (those with lowest cost) by outputting a PPM file that has the original image with the rectangles superimposed at each of the k best locations (with k a parameter of the method).

   For this part hand in some example images that show both strengths and weaknesses of your detector. Make sure to describe all parameter settings that you use. Discuss what kinds of issues arose in designing this simple detector and choosing parameter values.

3. In this problem you are to modify the L_2^2 distance transform code (or write your own if you choose not to use that code). The change is to add an argument s that is used to scale the distance function. That is, rather than using the distance $\|x - y\|^2$ use the distance $s\|x - y\|^2$ in computing the distance transform.

4. In this problem you are to implement a more sophisticated detector for a stop sign, which searches for a set of rectangles that are in a particular spatial configuration. This detector will use the flexible template method discussed in Lecture 4, where all parts are positioned with respect to a common reference part.

   Implement a procedure **flexmatch** which takes as input a number of parts, n, and three vectors of length n, P, O and S. The vector P should contain the parts, each represented as a rectangle as specified in problem 1. The first element of P should be the reference part. The vector O should contain the ideal (x, y) offset of each part with respect to the reference part. The first element of O should be assumed to always be $(0, 0)$ as the first part is not offset with respect to itself. The vector S should contain for each part a scaling constant that is used as the parameter s for the distance transform function from the previous problem (again for the first part this value is ignored). These scaling parameters should be set to balance the match cost versus the deformation cost. If they are set to different values for each part then certain parts can have their positions more constrained than others.

Your **flexmatch** procedure should compute a cost array m_j for each part j, using the cost function that you developed in problem 2. Then (as described in lecture) for each part other than the reference part, compute the distance transform D_{m_j} of m_j.

To compute the overall match cost for each (x, y) location, sum the value of m_0 (the reference part match cost) at (x, y) with the value of D_{m_j} at the location (x, y) offset by $O[j]$, for each part $j > 0$ (the other parts). That is, for each location (x, y) of the reference part, consider the value of the match cost m_0 at that location together with the value of the match cost m_j for each other part, at the ideal relative location of that part (as specified in $O[j]$).

In experimenting with this matcher you will need to balance the match and deformation costs by setting appropriate values of S. As the elements of S all approach zero, the deformation costs are zero and only the match costs are considered. As the elements of S get large the deformation costs dominate and the image matching costs are considered less strongly.

As with the detector in part 2, this detector should output a PPM file that has the original image with the best k matches super-imposed on it. In this case a match should be indicated by drawing just the reference part and not the other parts.

You should construct a stop sign model composed of several parts, at least one part for detecting a red rectangle, one part for detecting a smaller white rectangle within the red one (where the text is), and four parts for detecting non-red rectangles in the background around the sign. If you are able to design a better model than this, describe it and why it is better, but start with a model of this form.

As in problem 2, for this part hand in some example images that show both the strengths and weaknesses of your detector. Again make sure to describe all parameter settings that you use and to discuss what kinds of issues arose in designing this simple detector and choosing parameter values. You should also hand in a description of your model, what parts it has, and the part parameters and costs.

5. (Extra Credit) Extend your matcher to efficiently handle changes in overall scale (size) of the object being detected in the image. Allow the scale range to be specified, e.g., .5 to 2x the size of the model, etc.