Lecture 20: Segmentation (Matching, Advanced)

Contents

Reading

SH&B, 5.4–5.5

20.1 Matching

So far, we have talked about ways of segmenting general images (edges or regions) and for finding configurations of edges in parametrically-defined shapes (Hough transform). We'll now talk about segmenting where you know roughly what you're looking for in both intensity and configuration.

20.1.1 Overall Strategy

The overall strategy of template matching is easy: for every possible position, rotation, or other geometric transformation, compare each pixel's neighborhood to a template. After computing the strength of the match for each possibility, select the largest one, the largest n, or all that exceed some threshold.

20.1.2 Comparing Neighorhoods to Templates

Suppose that you already have a template for what you're looking for. One way of finding matches is *correlation*: shifting the template to each point in the image, point-wise multiplying each pixel in the neighborhood with the template, and adding the results. Essentially, this is the dot product of each neighborhood with the template. Remember that the dot product of two vectors is essentially the projection of one onto the other—the more they match, the larger the dot product.

There are other ways of measuring similarity of individual neighborhoods to a template, also based on vector-similarity metrics. The most common vector (or matrix) similarity metric is the L-norm of their difference. The L-norm of order p for a vector $\bar{v}$ is

$$\|\bar{v}\|_p = \left(\sum_i |\bar{v}_i|^p \right)^{1/p}$$

The most common forms of L-norms are the L_1 norm (sum of absolute values), the L_2 norm (square root of the sum of squares—i.e., Euclidean distance), and the L_∞ norm (maximum absolute value).

The C_2 metric (Eq. 5.39) in your text is the reciprocal of the L_1 norm of the difference between each neighborhood and the template. The C_3 metric (Eq. 5.40) in your text is the reciprocal of the L_2 norm squared. The C_1 metric (Eq. 5.39) in your text is the reciprocal of the L_∞ norm.

Picking the neighborhood with the largest match criterion (C_1, C_2, or C_3) is the same as choosing the neighborhood with whose respective norm (L_∞, L_1, or L_2) of the difference between the neighborhood and the template is the smallest.

20.1.3 Flexible Templates

Sometimes, what we're looking for might not be exactly the same in every image. One way to deal with this is to break the template into pieces and try to match each piece as if it was its own template. Position the entire template over the neighborhood, then search around the normal position of each subtemplate for the best match. The best combined match for all subtemplates gives the match for the overall template.

20.1.4 Control Strategies

Obviously, checking all possible transformations is computationally prohibitive in many cases. There are a number of ways of avoiding this, usually by only checking possible cases and then refining those that are "promising".

Hierarchical Matching (Pyramids)

One way to reduce the computational complexity of matching is to use the *pyramid* structure introduced in Lecture 3. By matching a coarser template to a coarser level of the pyramid, fewer comparisons must be performed. Once the strength of each coarser-resolution match is calculated, only those that exceed some threshold need to be evaluated/compared for the next-finer resolution. This process proceeds until the finest resolution (or at least a sufficiently fine resolution for the current task) is reached.

Iterative Refinement

Another way to perform matching is to use gradient-descent minimization techniques to iteratively tweak the transformation parameters until the best match is found. Unfortunately, these techniques require starting "near" the right solution. Still, this means that we don't have to check *all* possible transformations, just a sufficient number to allow us to find the matches through further refinement. For example, we might test rotations in 5- or 10-degree increments only. For each initial rotation, we then further refine the match by iteratively minimizing the difference from the template.

Chamfer Matching

Other approaches use distance maps to guide edge-based matches to corresponding edges. Remember that a *chamfer* is a map of the distance from each point to the nearest boundary. We might characterize the degree of mismatch for two curves (boundaries or other curve-based representations) by creating a distance map for one and then integrating the distance map for the first curve over all positions in the second curve. We can then iteratively adjust the position/orientation of the second curve until this integrated distance is minimized. This approach is known as *chamfer matching* and can be a very powerful technique.

Chamfer matching has also been used to match skeletal structures such as the medial axis.

20.2 Advanced Boundary Tracking

20.2.1 Tracking Both Sides of a Thin Object

Earlier, we discussed how to formulate boundary tracking as a graph-searching problem. This approach only tracks one boundary at a time. For some applications, we may want to simultaneously track both sides of a long, thin object (blood vessel, road, etc.). Simply tracking each side independently doesn't work when the object branches, crosses other edges, etc.

We can extend tracking of one contour to tracking two by constructing a graph that represents simultaneous movement along both edges. This can be done by constructing a separate graph for tracking each edge then "folding" one orthogonal to the other (like folding a piece of paper in half so that the two halves are perpendicular to each other) to create the basis for a three-dimensional graph. As we move from along the length of the shape, we simultaneously move side-to-side in each of the two other dimensions of the graph, which each correspond to a different side of the object. Your text does a nice job explaining a cost function that combines the strength of the edge on each side.

20.2.2 Surface Tracking

The concepts of boundary tracking can also be extended to surfaces in 3-d. The general form of this problem is actually quite hard: there isn't a two-dimensional surface analog to a one-dimensional minimum-cost path from one point to another. One can calculate the cost for a surface by adding all of the costs of the individual nodes, but what defines the "start" and "end" of the surface? Some approaches have tried to use 1-d ribbons or similar structures to "wrap" the surface. Other approaches use constrained forms of the problem.

One such constrained approach is described in your text. The idea is to construct the surface-search so that it goes from one side of a 3-d volume to the opposite side. As you progress through each slide through the volume, you can move in each of the other two dimensions by some predefined maximum transition, but always progressing from one face of the volume towards the opposite face. The problem can thus be set up as a dynamic-programming problem and implemented as such.

When we talked about using a top-to-bottom search graph earlier, we also talked about how you could wrap a resampled search grid around an already-approximated boundary. We can also do the same for surfaces. If the object's surface can be "unrolled" to a simple function of two search variables, we can find it using this surface-tracking technique.

Vocabulary

- Chamfer matching
- Surface tracking