Lecture 15: Segmentation (Edge Based, Hough Transform)

Last modified on February 23, 2000 at 2:00 PM

Contents

Reading

SH&B, 5.2.6
Castelman, 18.5.2.3

15.1 Introduction

Today we'll start talking about how to group isolated edge points into image structures. We'll come back later to talk about edge thresholding, edge relaxation, edge linking, and optimal edge determination using graph searching. All of these require some continuous path of edge pixels, nearby edge pixels, or edge-cost information.

The technique we'll be talking about today, called the *Hough Transform*, doesn't require connected or even nearby edge points.

15.2 Hough Transform: Basic Algorithms

15.2.1 Parametric Representations of Primitives

Let's first talk about finding edges that we know are straight lines. Consider a single isolated edge point (x, y)—there could be an infinite number of lines that could pass through this point.

Each of these lines can be characterized as the solution to some particular equation. The simplest form in which to express a line is the *slope-intercept* form:

$$y = mx + b \tag{15.1}$$

where m is the slope of the line and b is the y-intercept (the y value of the line when it crosses the y axis). Any line can be characterized by these two parameters m and b.

We can characterize each of the possible lines that pass through point (x, y) as having coordinates (m, b) in some slope-intercept space. In fact, for all the lines that pass through a given point, there is a unique value of b for m:

$$b = y - mx \tag{15.2}$$

The set of (m, b) values corresponding to the lines passing through point (x, y) form a line in (m, b) space.

Every point in image space (x, y) corresponds to a line in parameter space (m, b) and each point in (m, b) space corresponds to a line in image space (x, y).

15.2.2 Accumulators

The Hough transform works by letting each feature point (x, y) vote in (m, b) space for each possible line passing through it. These votes are totalled in an *accumulator*.

Suppose that a particular (m, b) has one vote—this means that there is a feature point through which this line passes. What if it has two votes? That would mean that two feature points lie on that line. If a position (m, b) in the accumulator has n votes, this means that n feature points lie on that line.

15.2.3 The Hough Transform Algorithm

The algorithm for the Hough transform can be expressed as follows:

```
1.    Find all of the desired feature points in the image.
2.    For each feature point
3.        For each possibility $i$ in the accumulator that passes
          through the feature point
4.            Increment that position in the accumulator
5.    Find local maxima in the accumulator.
6.    If desired, map each maxima in the accumulator
      back to image space.
```

Algorithm 15.1: Hough transform algorithm

For finding lines, each feature point casts a line of votes in the accumulator.

15.2.4 A Better Way of Expressing Lines

The slope-intercept form of a line has a problem with vertical lines: both m and b are infinite.

Another way of expressing a line is in (ρ, θ) form:

$$x \cos\theta + y \sin\theta = \rho \tag{15.3}$$

One way of interpreting this is to drop a perpendicular from the origin to the line. θ is the angle that the perpendicular makes with the x-axis and ρ is the length of the perpendicular. θ is bounded by $[0, 2\pi]$ and ρ is bounded by the diagonal of the image.

Instead of making lines in the accumulator, each feature point votes for a sinusoid of points in the accumulator. Where these sinusoids cross, there are higher accumulator values. Finding maxima in the accumulator still equates to finding the lines.

15.2.5 Circles

We can extend the Hough transform to other shapes that can be expressed parametrically. For example, a circle of fixed radius can be described fully by the location of its center (x, y).

Think of each feature (edge) point on the circle as saying, ”if I’m on the circle, the center must be in one of these places”. It turns out that the locus of these votes is itself a circle.

But what about circles of unknown size? In this case, we need a third parameter: the radius of the circle. So, we can parameterize circles of arbitrary size by (x, y, r). Instead of casting votes in a circular pattern into a two-dimensional accumulator, we cast votes in circles of successively larger size in a three-dimensional accumulator.

15.2.6 More Complicated Shapes

We can extend the Hough transform to find shapes of arbitrary complexity, so long as we can describe the shape with some fixed number of parameters. The number of parameters required to describe the shape dictates the dimensionality of the accumulator.

15.3 Implementation Details and Variations

15.3.1 Different Orientations

For the line-matching Hough Transform, the orientation of the line is one of the parameters. For the circular form of the transform, orientation is unnecessary. For other methods, you can choose whether or not to use orientation as one of your parameters.

If you don’t use an orientation parameter, you will only be able to find matches in a specific orientation.

If you do include an orientation parameter, you’ll have a larger-dimensional accumulator, but you’ll be able to find matches in various orientations (as many as you want to discretely parameterize).

15.3.2 Using Gradient Orientation

The basic Hough transform can be made more efficient by not allowing feature (edge) points to vote for all possible parametric representations but instead voting only for *those whose representation is consistent with the edge orientation*. For example, instead of casting votes for all lines through a feature point, we may choose to vote only for those with orientation θ in the range $\phi - a \leq \theta \leq \phi + a$ where ϕ is the gradient orientation. We can take this idea further by weighting the strength of the vote by some function of $|\theta - \phi|$.

15.3.3 Discrete Accumulator

This discussion is all well and good for continuous parameterizations, but we need to discretize the parameter space. The granularity with which we discretize the accumulator for the parameter space determines how accurate we’ll be able to position the sought-after target.

15.3.4 Smoothing the Accumulator

Because of noise, discretization of the image and accumulator, and factors inherent in the application itself, we sometimes want to allow a little tolerance in the fitting of lines or other shapes to the edge pixels. We can do this by allowing a feature point to vote note just for a sharp line or curve in the accumulator, but to cast fuzzy votes for nearby possibilities. In essence, this votes not just for all lines or other shapes that pass through the feature point but also for those that pass close by.

This casting of votes for nearby possibilities can be done directly during the voting phase of the transform or afterwards through a post-processing blurring of the accumulator. The way in which you blur the accumulator depends on what nearby possibilities you want to allow.

15.3.5 Finding Relative Maxima in the Accumulator

If you're just looking for one thing, simply pick the largest value in the accumulator and map it back to the image space. If you're looking for other shapes, you need to find all relative maxima.

As a further test, you may want to threshold the maxima that you find by the number of points that voted for it. Relative maxima with few votes probably aren't real matches.

You'll find that this part is the real trick of getting the Hough transform to work.

15.3.6 Grey-level Voting

The discussion so far has been of binary feature points casting binary votes. A more effective way of doing this is to use the full range of grey-level feature detection (i.e., gradient magnitude) and cast votes proportional to the strength of the feature.

15.4 Comparison to Template Matching

Of course, one could also use template matching (correlation) to find lines, circles, etc. However, for an $N \times N$ image and and $M \times M$ template, the computational complexity is $O(N^2 M^2)$.

The Hough Transform, by matching only image edge points to target contour points, requires much less computation. In particular, the number of edge points goes up only linearly with N, not by N^2. Likewise with the number of target contour points. Thus, the complexity of the Hough Transform is only $O(NM)$.

15.5 The Generalized Hough Transform

Some shapes may not be easily expressed using a small set of parameters. In this case, we must explicitly list the points on the shape.

Suppose that we make a table that contains all of the edge pixels for our target shape. We can store for each of the pixels its position relative to some reference point for the shape. We can then feature point "think" as follows: "if I'm pixel i on the boundary, the reference point must be at ref$[i]$."

This is called the *Generalized Hough Transform* and can be expressed as follows:

```
1.    Find all of the desired feature points in the image.
2.    For each feature point
3.        For each pixel $i$ on the target's boundary
4.            Get the relative position of the reference point from $i$
5.            Add this offset to the pisition of $i$
6.            Increment that position in the accumulator
7.    Find local maxima in the accumulator.
8.    If desired, map each maxima in the accumulator back to image
      space using the target boundary table
```

Algorithm 15.2: Generalized Hough transform algorithm

We can make this more efficient by incorporating the idea from Section 15.3.2. We can build a table that records for each point with edge tangent orientation θ the direction α and distance r to the reference point. Thus, when we find a point with edge tangent orientation θ, we have to vote only in the direction of r, α. Of course, depending on the complexity of the shape, there may be multiple such points with tangent orientation θ. We thus build our table to store (and vote for!) all such points:

$$\begin{array}{c}\theta_1 : (r_1^1, \alpha_1^1), (r_1^2, \alpha_1^2), \ldots \\ \theta_2 : (r_2^1, \alpha_2^1), (r_2^2, \alpha_2^2), \ldots \\ \theta_3 : (r_3^1, \alpha_3^1), (r_3^2, \alpha_3^2), \ldots \\ \vdots\end{array}$$

This is called an *R-table*.

15.6 Refining the Accumulator

In the Hough transform, feature points vote for *all* possibilities through that point. This can cause unwanted clutter in the accumulator and false matches.

A way to deal with this is to do a second voting phase. In this phase, each point examines the locus of points in the accumulator that it voted for and finds the maximum. It then casts a single vote into a second accumulator *only for this maximum*. In other words, it sees where there is agreement with its neighbors and then goes along with the crowd. Gerig (1987) has shown that this removes this unnecessary clutter and leadsto only a few isolated points in the accumulator.

As another approach, suppose that after we find the global maximum we remove all votes cast by feature points on or near the corresponding match in the image space. We can then continue the process by looking for the next largest global maximum, remembering it, and removing the votes from its points.

Vocabulary

- Hough Transform
- Accumulator
- Generalized Hough Transform