1-1-1985

Contour tracing for precision measurement

K. Dunkelberger

Robert Mitchell
University of Missouri--Rolla

Follow this and additional works at: http://scholarsmine.mst.edu/faculty_work
Part of the Electrical and Computer Engineering Commons

Recommended Citation
http://scholarsmine.mst.edu/faculty_work/1641

This Article - Conference proceedings is brought to you for free and open access by Scholars' Mine. It has been accepted for inclusion in Faculty Research & Creative Works by an authorized administrator of Scholars' Mine. For more information, please contact weaverjr@mst.edu.
CONTOUR TRACING FOR PRECISION MEASUREMENT

Kirk A. Dunkelberger and O. Robert Mitchell

School of Electrical Engineering, Purdue University
West Lafayette, Indiana 47907

Abstract

Current methods of contour tracing are based on chain code concepts which require movement from the center of a pixel to the center of a four- or eight-connected neighbor. They are basically serial processes requiring memory and decision structures based on the connectedness of the objects and the direction of the trace. A chain code method is proposed here which moves along the edge midpoints of pixels, increasing the number of contour samples. This new tracing process needs no memory, lends itself to fast table lookup implementation, uses only two tables for all connectedness rules and trace directions, and permits parallel preprocessing. The resulting chain code is nearer the true contour than any previous digital contour tracing method. Methods for conversion between the standard chain code and the new edge chain code are also presented.

Introduction

Most computer vision systems which are designed for robot vision or inspection require extraction of the contour of the boundary of objects as a step in recognition. Many early binary vision systems utilized the SRI feature set which includes information relating to the boundary such as the perimeter, the ratio of perimeter squared to area, and the minimum, maximum, and average excursion of the boundary from the center of area. Many industrial vision systems consider these features significant enough to be included in their analysis variable set. Shape recognition methods based on Fourier descriptors also require a description of the object boundary as input, usually in the form of an ordered set of boundary coordinates. Precision measurement methods of grey level images based on threshold decomposition and using Fourier boundary descriptors shows the variability in contour measurements caused by thresholding a grey level image at different values.

Problems in Standard Chain Code Measurements

The most common contour tracing algorithm moves from the center of a pixel to the center of an eight-connected neighbor. This results in the horizontal and vertical moves having a length of 1 pixel and the diagonal moves having a length of \(\sqrt{2}\) pixel. But a choice must be made as to which pixels to follow. One can follow the pixels just outside the object, or follow those which lie just inside the object. Shown in Fig. 1 is a binary silhouette with traditional center-to-center inner and outer boundaries and a crack code boundary which will be discussed in the next section. We will discuss the errors encountered in estimating perimeters and areas and also the difficulties in the implementation of tracing using standard chain codes.

![Figure 1. Silhouette with inner and outer center-to-center chain coded contours (dotted lines) and the new crack coded contour (solid line).](image-url)
for the two lower pixels to be “1” and the two upper pixels “0”, (the criteria for a horizontal chain code link to be generated) the true contour must pass through the line segment (0,0)–(0,1) and also the line segment (1,0)–(1,1). The chain code approximation of the contour can then include an error area $\Delta -1 < \Delta < 0$ for the inner contour I and for the outer contour $O$.

(Only the center region in Fig. 2 is considered in calculating the area error since the area outside this region depends on other chain code links for straight line segments.)

![Figure 2. Model for error analysis of a horizontal contour move.](image)

Assuming uniform densities for the line segment intercepts, $0 \leq r < 1$ and $0 \leq p < 1$, and that $r$ and $p$ are independent, the expected error in area for an inner trace would be $-1/2$ square pixels per horizontal or vertical link in the contour. The outer contour would likewise have a mean error of $+1/2$ square pixels.

$$f(r) = \begin{cases} 1 & \text{if } 0 \leq r < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$f(p) = \begin{cases} 1 & \text{if } 0 \leq p < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$\bar{e}_h = -1/2 \int_{0}^{1} \int_{0}^{1} f(r)f(p) \, dp \, dr = -1/2$$

The diagonal moves of the contour can be represented in the same manner with the coordinate system rotated by 45 degrees as in Fig. 3. Once again, in order to obtain the binary representation of the object shown, we assume the line segment approximation of the contour must pass through both the segments $(0,0)–(0,\sqrt{2})$ and $(\sqrt{2}/2,0)–(\sqrt{2}/2,\sqrt{2}/2)$ and will generate uniform probability densities for $0 \leq r < \sqrt{2}$ and $0 \leq p < \sqrt{2}/2$. The expected area error for the inner trace is then $-1/2$ and the corresponding error for a diagonal move of an outer trace is $+1/2$ square pixels.

$$\bar{e}_{id} = -\sqrt{2} \int_{0}^{1/2} \int_{0}^{\sqrt{2}/2} f(r)f(p) \, dp \, dr = -1/2$$

Note this error is in the area estimate along straight line paths. At corners additional errors are introduced which are not analyzed here. These errors have a negative value for the inner trace and a positive value for the outer trace so that in both cases the total error is increased when corners are considered.

Errors in the perimeter estimate involve two factors. One is that chain code approximations to a straight line are always longer than or equal to the line itself. The integrals for expressing the mean error are best solved by numerical means and have the form

$$\bar{E}_{th} = \int_{0}^{1} \int_{0}^{1} \sqrt{1 + (r - p)^2} f(r)f(p) \, dp \, dr$$

$$= 0.076836$$

$$\bar{E}_{id} = \int_{0}^{\sqrt{2}/2} \int_{0}^{\sqrt{2}/2} \sqrt{2 + 4(r - p)^2} f(r)f(p) \, dp \, dr$$

$$= 0.369340$$

These dimensionless quantities indicate the average additional perimeter per pixel error caused by fitting chain code to a straight line. The second source of error is corners, where the perimeter estimate is reduced.
by amounts ranging to 2 pixels for inner traces and increased by similar amounts for outer traces. Note that in the case of the inner trace, these two types of error have opposite signs and partially cancel each other.

The actual implementation of the center-to-center trace depends upon memory of where it has been. The next contour link direction cannot be determined uniquely directly from the bit pattern of the trace neighborhood. Thus memory is required in the trace routine to determine the next direction to go.

Implementation of such a trace in parallel does not seem practical. What results is a set of variables too large for a manageable lookup table, and four different hard-wired decision trees to account for all the possible combinations of connectedness and trace direction. For instance, to determine the next move in a clockwise outer trace, one looks in the most clockwise possible direction from the previous direction, looks for a transition from background (or previously visited location) to object in the clockwise direction, and increments the direction counter-clockwise until this transition is found. Besides being difficult to describe, this algorithm is slow in execution due to the multiple conditional tests. We are required to access 3x3 windows of data along the contour resulting in a strictly serial process.

**Crack Code Tracing**

Defining a new coordinate system with twice the pixel resolution with origin at the upper left of a pixel and x increasing towards the right and y increasing down, integer values now correspond to the cracks in between pixels at intersections and midpoints. The intersections themselves will be illegal ending locations creating the restriction that for any (x,y) coordinate of a crack code point, x and y cannot be both odd or even. (See the crack code boundary in Fig. 1.) Also the restriction will be imposed that a pixel can never be directly crossed. For instance, from a point (2i+1,2j), moves to (2i+1,2j-1) and (2i+1,2j+1) are not allowed.

There are two types of locations in crack coordinates. If the point is of the form (2i+1,2j), the location is in between two pixels vertically and the next move can be determined directly from a 2 x 3 window. Similarly, for a point of the form (2m,2n+1), the location is between two pixels horizontally and the next move is determined by a 3 x 2 window. The crack code algorithm generates a unique trace for a given silhouette and it is guaranteed by construction that it never traces back on itself until the entire contour has been traced. The vertical and horizontal moves will have magnitude length of 1, while the diagonal moves will have magnitude length $\sqrt{2}/2$.

The definition of outer and inner boundaries are not needed for the crack code boundary since both are equivalent. Solving in a manner similar to the center-to-center chain code trace, we have the bounds for the area error in crack code tracing as

$$-0.5 < \Delta_C \leq 0.5$$

and continuing for the mean area error

$$\overline{\Delta_{CA}} = \frac{1}{2} \int_0^1 \int_0^{1/2} f(r) f(p) \, dp \, dr = 0$$

$$\overline{\Delta_{CD}} = \frac{1}{2} \int_0^1 \int_0^{1/2} (\sqrt{2} - 1/2) f(r) f(p) \, dp \, dr = 0$$

The crack code contour provides an unbiased estimate of the area under straight line sections of a contour. The error due to corners is not analyzed here but is much smaller than that of the center-to-center chain code.

Theoretical analysis of perimeter errors is more complex and will not be presented here. However, empirical results presented in the next section show that the crack code contours give better perimeter estimates.

**Experimental Verification of Mean Error Values**

It is difficult to completely specify the error of contour tracing methods. A silhouette could be generated from a statistical model, but, although mathematically pleasing, in practice this does not generate natural, real-life shapes. The applications for precision contour tracing lie in the measurement of man-made objects built from basic geometric figures. It is for this reason that experiments were performed on squares.

The error arises from two sources. There is the sampling error in converting the object located with a sub-pixel precision to a digitized binary image and the error in the representation of the contour of this binary image by a form of chain code. Errors presented here will be in the form of a signed percentage difference from the parameter of the original object. Algorithms were written to generate squares using the assumptions that a square sampling aperture is used, and that a pixel will be on when more than half its area is covered by the object. Several parameters of the object were fixed while allowing one to vary, and an attempt made to fairly evaluate the differences between the original object area and perimeter values and the computed values from the crack coded contour.

A square has three degrees of freedom: center location, side length, and rotation with respect to sampling lattice. In Fig. 4 is an example plot of area error versus side length for an orientation of 0° with Table 1 showing the mean error values for several different orientations.

As seen in Table 1, the inside chain code consistently produced a large (negative) area error, while for all angles, the crack code produced very small area errors centered around 0. A square whose sides are
parallel to the sampling lattice gives results which are the most easily understood. The periodicity of the plot arises from the turning on of large groups of pixels as their centers are included inside the square. The square was located at the center of a pixel so the discontinuities occur at even integer pixel side lengths. The decreasing magnitude of the extrema is due to the decreasing sampling error as the area of the square increases, e.g., the sampling lattice is better able to describe the shape. A square at 45 degrees gives similar results due to its regular growth pattern as the area increases. Here the discontinuities occur every \( \sqrt{2} \) pixel increase in side length when another group of pixels is added to the binary representation.

Another experiment was run in which the side length was fixed and the angle of orientation varied linearly through 45 degrees. An example area error versus orientation plot is shown in Fig. 5, with the mean error values for several side lengths shown in Table 2. The discrete nature of the plot shows the edge angle quantization quite well. For short side lengths, there are very few distinct angles for which the square produces a unique binary image. As the side length is increased, more values are possible. Once again, the chain code produced area errors significantly larger than the crack code, always negative (calculated area too small). Crack code errors are small and centered about zero, pointing to the validity of the theoretical calculations.

**Crack Code Tracing Implementation**

It is in the implementation of crack code tracing that its advantages are fully realized. With only two sets of binary windows with 6 pixels each, an integer from 0 to 63 can be formed having bits set according to the value of each pixel in the window. Note that this calculation of the lookup table index can be done in any parallel hardware which allows convolution. Of these 128 windows, only 64 are useful. In order to be on the edge of an object, one of the center pixels must be 1 and the other 0; both cannot be the same value eliminating half the possible windows.
Some simple conventions aided in the development of the lookup tables for counter-clockwise tracing. In the 3x3 window, if the lower center pixel belonged to the object, only moves to the left (links 3, 4 and 5) were allowed since we must be above the object. Likewise, if the upper center pixel was belonged to the object, only moves to the right (links 7, 0, and 1) were allowed. For the 3x2 window, moves were up for the object to the left, and down for the object to the right. One pixel wide regions of an object are no longer a problem since the trace will never double back on itself. Tagging is only necessary if a higher level program needs to know if a contour has already been traced.

Once the tables for tracing four-connected objects counter-clockwise are developed, the tables for tracing eight-connected objects clockwise are not needed. It is known that if the object is four-connected, the background must be eight-connected. In crack code, since there is a unique contour for a given silhouette, tracing the object four-connected is equivalent to tracing the background eight-connected in the opposite direction. To switch the mode, the mask index need only be one-complemented before lookup. Hence, only one more set of tables is needed, which will serve for tracing four-connected objects clockwise and eight-connected objects counter-clockwise.

Run-time Efficiency Experiments

A thermal image of some laser drilled holes was used for speed comparisons. Two routines were written with the only difference being the tracing method used. For the crack code worst case, the window values were calculated in a serial manner along the path of the contour and were not retained for later calls. In a second experiment, preprocessing to calculate the crack code leaving links for each location in the binary picture was performed which took negligible time since it was done in parallel using a Grinnell 275 image processing board. The same calling routines were used, with the results in the form of CPU time listed in Table 3.

<table>
<thead>
<tr>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routine</td>
</tr>
<tr>
<td>crack()</td>
</tr>
<tr>
<td>crack()</td>
</tr>
<tr>
<td>ctout()</td>
</tr>
</tbody>
</table>

The serial implementation of crack code tracing ran slightly faster than the conventional chain code trace while producing longer, more accurate, chains. When the full potential of the parallelism of crack code is used, the tracing runs 8.5 times faster.

Informational Equivalence and Conversion between Chain Code and Crack Code

With some consideration, it can be seen that a crack code can be converted to a standard chain code since both represent of identical objects in a binary image. For simplicity, assume that we start with a counter-clockwise crack code, and a center-to-center chain code which traces object pixels is desired. The rules governing crack code generation determine the boundary of the object directly when a crack coordinate location and the link departing it are known. A sequence of boundary pixels can be easily built into an inner trace sequence since we are guaranteed that the new sequence will be of equal length or shorter and only 0 length chains must be dropped.

The process of converting inside chain code to crack code is more complicated. The length of the resulting crack code varies with respect to the concavity of the chain code trace at any given point. Where a counter-clockwise trace has a positive total angle change in a local region, (meaning the object is convex through that region,) the resulting crack code will be longer and enclose more area than the chain code through that region. In the regions where the chain code trace has a negative total angle change, (concave regions,) the number of crack code links will be equal to that of the chain code.

Once again, the portion of the conversion which collapses out points, locally concave, is straightforward. Only the portions which are convex fall into four special cases dependent on the relative angle change of the two chain code links involved. The crack code rules can be applied to the local binary picture restored from the two chain code links to produce the desired extra crack code links.

The implications of these conversions are wide reaching. It is shown here that there is a one-to-one correspondence between crack and inner chain codes since there is a uniquely invertible transform between the two. Therefore, they must include the same information. In the case of crack code, the intuitive methods of adding the lengths of the individual links to estimate the perimeter and counting the pixel area falling inside the crack coded contour to estimate silhouette area produce best estimates directly. This does not apply to chain code. Rather, chain code links must be considered in pairs which describe the local concavity of the object, adjusting the perimeter and area accordingly.

The fact that crack code and chain code can both be used to calculate the same perimeter and area does not detract from the attractiveness of crack code. As shown above, the benefits received by computing crack code directly, the parallel implementation in particular, still point to crack code as an excellent alternative in contour description. With the conversion process known, crack code can easily be used in existing
software packages with the small overhead of the conversion.

Conclusions

Crack coded contours can give directly an unbiased estimate of the area of a silhouette. In a serial machine, the crack code algorithm executes at roughly the same speed, but when parallel or convolution hardware is available, it executes 8.5 times faster. The actual code for the algorithm is compact and easily modified. Crack code can be substituted directly into most applications. An invertible transformation has been developed which allows one to change at will between versions of contour coding.

References

