

## Linear Feature Extraction and Description

RAMAKANT NEVATIA AND K. RAMESH BABU

*Computer Science Department and Image Processing Institute, University of Southern California,  
Los Angeles, California 90007*

Received July 30, 1979; accepted October 24, 1979

A technique of edge detection and line finding for linear feature extraction is described. Edge detection is by convolution with small edge-like masks. The resulting output is thinned and linked by using edge positions and orientations and approximated by piecewise linear segments. Some linear features, e.g., roads and airport runways, are suitably described by "antiparallel" pairs of linear segments. Experimental results on aerial images are presented.

### 1. INTRODUCTION

The importance of effective early processing of visual input for a complete image understanding system is generally accepted. The early processing may consist of a description of image discontinuities, usually in the form of edges and lines, or segmentation into uniform regions. In this paper, we describe a technique for extracting linear features in an image by a process of edge detection and line linking and also of deriving higher-level descriptions from the extracted lines. These techniques are intended to be general. Applications to road detection and airport recognition tasks are described.

Edge detection and line finding techniques, of course, have been studied since the early days of this field and are described in textbooks [1,2]. However, in spite of the large amount of previous research in this area, and a number of comparative studies [3,4], the choice of algorithms suitable for complex imagery is not clear. Further, the comparisons are often limited to local edge detection only. Our research was motivated by the need for a software package for linear feature extraction, with adequate performance for higher-level processing, rather than by the desire to invent new edge and line detection algorithms. In particular, we found the output of the widely used Hueckel edge operator [5], followed by a linking step developed by one of us [6], to be not very satisfactory for complex aerial images which contain fine detail and texture.

Here, we describe an edge detection and line finding technique with superior performance on a wide variety of images. Many of the steps in this process are similar to, or extensions of, previous methods, but we believe that our integration and refinements are important and different. We hope that this presentation will be helpful to other researchers attempting to implement similar algorithms. Our software system is already being used by a number of researchers outside our institution.

Our process of line finding consists of determining edge magnitude and direction by convolution of an image with a number of edge masks, of thinning and thresholding these edge magnitudes, of linking the edge elements based on proximity and orientation, and finally, of approximating the linked elements by piecewise linear segments. Some objects of interest, e.g., roads and runways, are characterized by being bounded by nearly parallel line segments of opposing contrast, to be

known as "antiparallel" segments. Such descriptions are related to the generalized cone representation suggested by Binford [7]. Our algorithms are largely local in nature and can be applied to large images without difficulties of storage (but, of course, requiring proportionately larger computing time), and hardware implementation should be feasible. Experimental results of working programs are presented in the following sections.

## 2. EDGE DETECTION

The literature regarding local edge detection is rather large. A survey of the various techniques can be found in [1, 2, 8]. It has been difficult to evaluate the various edge detectors due to the lack of suitable mathematical models for images.

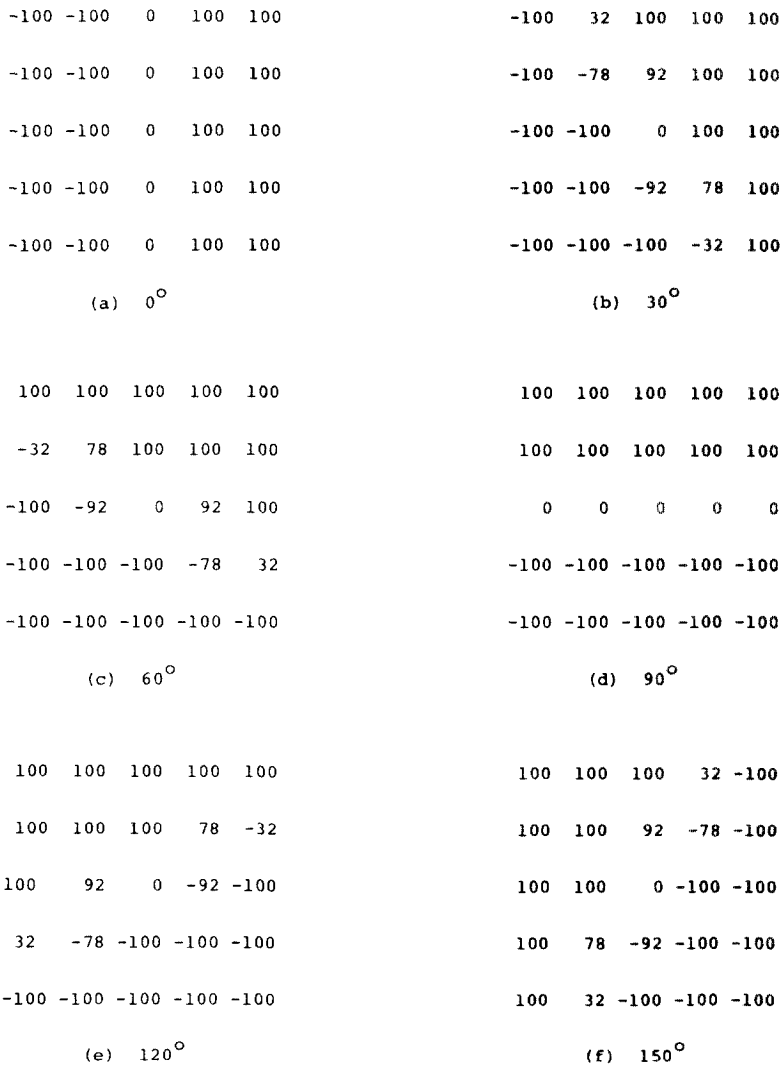


FIG. 1. Edge masks in six directions.

Some comparisons may be found in [3, 4]. Our choice of local edge detector was guided by these evaluations and other empirical observations.

Edge detection in our technique starts by convolving a given image with masks corresponding to ideal step edges in a selected number of directions. The magnitude of the convolved output and the direction of the mask giving the highest output at each pixel are recorded as edge data. (The edge data are two files, one containing the magnitude and the other a coded direction). Such edge detection is optimal if we view the process as detection of a known signal, embedded in a Gaussian noise field [4]. However, the assumptions are not always valid for real images; the desired edges are not necessarily step edges and undesired components are not necessarily modeled well as a Gaussian field. Herskovits and Binford have studied real edges in the limited domain of polyhedral scenes [9]. We are not aware of any similar studies for other domains.

The choice of the sizes of the edge masks in the chosen scheme is important and difficult. In general, the smaller masks are sensitive to noise, whereas the larger masks cannot resolve fine detail and may have difficulties if the texture elements are of similar size. The optimal mask size varies with the images and within a single image. Rosenfeld and Thurston [10] and Marr [11] have described techniques for using masks of various sizes and choosing an appropriate size based on their convolved outputs. Unfortunately, the larger masks can be comparable to the size of the image, and for large images, the computational cost is prohibitive. Further, the criteria for choosing among the various outputs is also unclear in the presence of texture. We have found six  $5 \times 5$  masks, as shown in Fig. 1, corresponding to edges at  $30^\circ$  intervals, to be suitable for a wide variety of images, though other sizes and different numbers of directions may be more suitable in some cases. The described techniques can be easily adapted for other masks. The weights associated with the masks shown in Fig. 1 correspond to the digital images of ideal step edges at given orientations. Note that the masks are not binary.

### *Thinning and Thresholding*

The simplest technique for thinning is to retain only those edges whose magnitude is a local maximum. In our technique we use the magnitudes and the directions of neighbors during thinning. The presence of an edge at a pixel is decided by comparing the edge data with some of the eight neighboring pixels. An edge element is said to be present at a pixel if:

1. The output edge magnitude at the pixel is larger than the edge magnitudes of its two neighbors in a direction normal to the direction of this edge. (The normal to a  $30^\circ$  edge is approximated by the diagonals on a  $3 \times 3$  grid.)
2. The edge directions of the two neighboring pixels are within one unit ( $30^\circ$ ) of that of the central pixel.
3. The edge magnitude of the central pixel exceeds a fixed threshold.

Further, if conditions 1 and 2 are satisfied, the two neighboring pixels are disqualified from being candidates for edges.

The selection of the threshold in step 3 above is of obvious importance. We have not used any automatic or adaptive techniques of threshold selection. Instead, we have set the threshold at a very low level, yielding a large number of edges. The

problem of choosing among these edges is left to the higher symbolic description levels. We have found the same threshold to be suitable for many test images. This threshold is the only parameter that is not fixed in our entire edge and line finding programs.

The decision as to the presence of an edge could be based on examining the shape of the profile of the convolution output, e.g., an ideal step edge should produce a triangle-shaped output. Such techniques have been used by Herskovits and Binford [9] and by Marr [11]. Our experiments with requiring the neighboring pixels to have edge magnitudes that are at least a certain fraction of the central pixel magnitude resulted in poor performance, perhaps due to variations caused by fine texture in the test images. More complex decision strategies hold promise for improved performance.

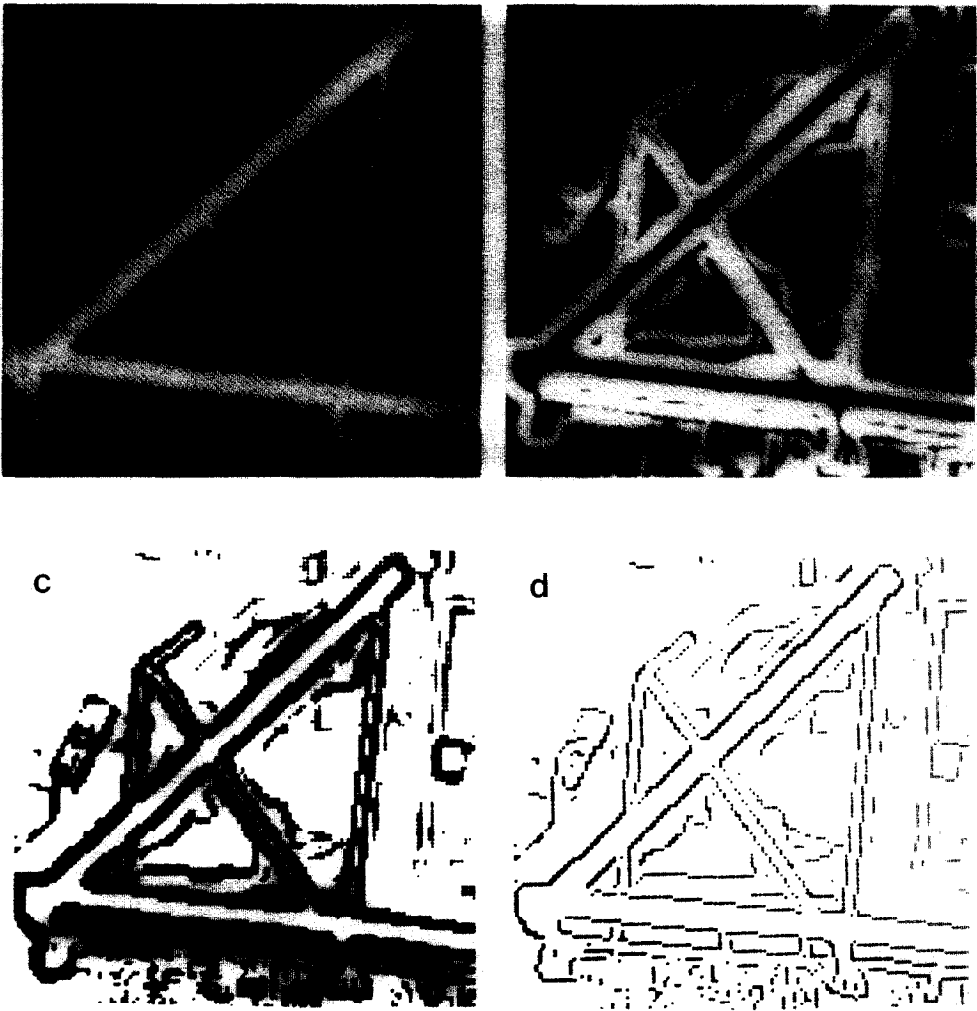


FIG. 2. Edge detection in an airport image: (a) digital image; (b) edge magnitude; (c) unthinned, thresholded edges; (d) thinned and thresholded edges.

Figures 2a through d show an aerial image of an airport, the magnitude of the convolved output, and the unthinned and the thinned edges detected in the image. (Note that the unthinned output is *not* an intermediate step in the algorithm.)

3. LINE LINKING AND LINEAR APPROXIMATION

Many techniques of computing boundary segments from isolated edge points have been described in the past. Some representative techniques may be found in [6, 12–15]. The edge detector used in our technique tends to extend an edge segment along its length and hence the output edges have good connectivity. Hence, adequate linking can be obtained simply by considering connections of an edge point with its 8-neighbors on a  $3 \times 3$  grid; search over extended neighborhoods to select optimal segments has not been used. Also, no attempt is made, at this stage, to separate the desired and the undesired boundary segments, such as by grouping edge elements according to some established criteria.

In our implementation, computing boundary segments consists of three parts: computing the neighbors that each edge is to be linked to, tracing boundaries using this linking information, and finally approximating the resulting boundaries by piecewise linear segments.

*Linking*

The main step in computing the boundary is to determine the neighbors on the boundary for each edge element. Typically each element has two neighbors along the boundary, known as a predecessor and a successor, except for end-elements, isolated elements, and where two boundary segments join or intersect.

For each edge element, we examine its 8-neighbors on a  $3 \times 3$  grid to determine its predecessors and successors. The neighbors normal to the central edge direction cannot be present, as they are eliminated during the thinning step. Thus, we have a choice of at most three edge elements each for predecessors and successors as shown in Figs. 3a and b. Another condition for two edges to be linked is that their orientations not differ by more than a certain value, set at  $30^\circ$  for the chosen masks.

Linking of two edges is further restricted so that each edge point has at most two predecessors and two successors, as described in detail below. The rules for linking were derived by examining all configurations of potential predecessors and successors. Many of the combinatorially possible configurations are eliminated due to the processing of the thinning step. The remaining configurations can be characterized by only a few cases. In cases of multiple successors or predecessors, choices are made such that the resulting lines have smooth continuation.

The following cases may occur:

1. Only one element is an acceptable successor (predecessor). No further choices are necessary in this case.



FIG. 3. Possible successor locations for two edges.



FIG. 4. Three instances of two successors: (a) nonneighboring successors; (b) successor directions differ by  $60^\circ$ ; (c) successors in same direction.

2. Two candidates are acceptable successors. If they are not 4-neighbors, a fork is present, as shown in Fig. 4a. If they are 4-neighbors, a fork exists only if their directions differ by more than 2 units ( $60^\circ$ ), as in Fig. 4b. Otherwise, no fork exists and the nearer of the two (using Euclidean distance), forms the successor (predecessor), as shown in Fig. 4c.

In the case of a fork, the successor with the larger magnitude is said to be the *primary* successor and the other successor is called a *secondary* successor.

3. Three candidates are acceptable successors. Figure 5 shows all possible such configurations for a vertical edge (no three successor configurations occur for  $30^\circ$  edges). In these cases, a fork exists. The main stream is formed by the nearer of the two edges having the same direction, and the other candidate with different direction forms the other branch.

At each pixel of the image, the predecessor and successor information is recorded in two files called the “*P*” and “*S*” files, each of the size of the original image. Each element of the *P* file contains the location of one of the predecessors of the corresponding element in the edge file, and the elements of the *S* file contain the locations of the successor elements. A nine-valued integer is sufficient to record this information, eight for each of the eight neighbors and an additional one to indicate no predecessor or no successor. In the case of multiple successors, to save storage in our implementation, we have chosen to record the primary successors and predecessors (as defined above) only and indicate the presence of another successor or predecessor by marking a fork bit in the *S* or the *P* files, respectively. The secondary successor of an element, *E*, that is not explicitly recorded in the *S* file can be found by searching the eight neighbors of *E* in the predecessor file for an element whose value is *E*. Similarly, finding a secondary predecessor requires a search of the *S* file.<sup>1</sup>

The predecessor–successor representation of the linked edge elements is in contrast to explicit lists of elements forming a connected segment. For larger images, not entirely resident in the core, it is more convenient to form predecessor and successor matrices first, as the processing requires only a sequential scan of the image file. Further, this representation is “iconic” and allows easier computation of certain geometric properties.

<sup>1</sup>This technique fails if the secondary successor of element *E* does not have *E* as its primary predecessor and hence is not recorded in the *P* file. A more complicated search could still be used to find the secondary successor. However, such an event is highly unlikely and we have found that it does not occur in practice, and hence, we have not implemented the more complex search. Further, losing this information in rare cases is not important for the resulting boundaries. Of course, the two successors can be recorded explicitly, requiring more storage if this problem is to be avoided completely.



FIG. 5. All instances of three successors for a vertical edge.

### *Tracing Boundary Segments*

Boundary segments containing ordered lists of edge elements are computed from the predecessor–successor files. Given a starting point, a boundary can be traced by following the successor elements until an element with no successors or an element already included in a previous boundary segment is reached. A binary file records the traversing of each edge element. Our boundary tracing algorithm works in three passes.

1. In the first pass, tracing begins at the edge points that have no predecessors. Note that the edge points belonging to closed segments will not be traced in this pass. At a point with multiple successors, the primary successor, i.e., the one marked in the  $S$  file, is chosen.

2. In the second pass, boundary tracing starts from the secondary successors of the edge points that are indicated to have forks, and proceeds as before.

3. The third pass traces circular segments. This pass begins arbitrarily at any edge point that has not been traversed so far, as determined by the binary file maintaining this information, and proceeds as before.

At this stage, one-pixel gaps in the resulting segments can be bridged by requiring that the pixel between the two segments to be bridged be a potential successor for one segment and a potential predecessor for the other. We have implemented such bridging. Bridging of longer gaps requires search and heuristic criteria of good continuation and is not used on our system.

### *Linear Approximation*

Each boundary segment is approximated by a series of piecewise linear segments. The algorithm used is a variation of the well-known iterative end-point fit algorithm (see [16, pp. 338–339]). The unmodified algorithm operates by approximating a segment by joining the end points. The point of maximum error determines a segmentation point and is used to form two approximating line segments. This process is repeated until the error is within an acceptable bound.

A straightforward application of this procedure can be inefficient. We employ a variation in which a smaller portion of the curve, say  $m$  points long, is processed at one time. For the next part of the processing, the curve begins at the farthest corner found and ends  $m$  points later, and so on, until the end of the original curve is reached. To avoid the possibility of the algorithm missing some corners because the end point of an  $m$ -long portion was at or around a genuine corner, we consider  $2m$ -long chains in case no corners are found, then  $3m$ -long chains, and so on, until either we find a corner or come to the end. A typical value for  $m$  is 32 elements.

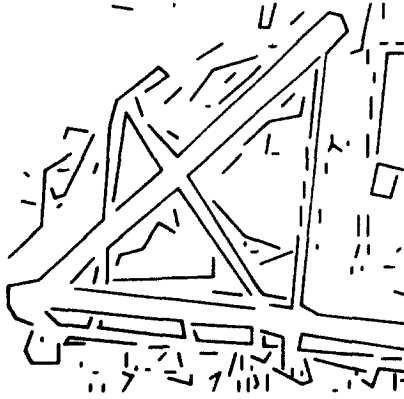


FIG. 6. Linear segments detected in the image of Fig. 2.

### Results and Computation Times

Figure 6 shows the linear segments resulting from the edges of Fig. 2d. The output contains not only the descriptions of the linear segments, but also their connectivity. (A connected list of segments is referred to as a "supersegment.") Our line finding algorithm has been applied to more than 100 images of wide variety containing natural and man-made objects and has produced output useful for higher-level matching; some applications are described in [17, 18]. Subjective evaluations indicate the performance to be superior to that obtained by the Hueckel edge operator followed by a linking step in images with fine detail and texture.

The computation times for various stages of processing, for a  $128 \times 128$  image, on a PDP-10, KL-10 processor, are as follows:

Convolution with edge masks	17 sec
Thinning and thresholding	2.3 sec
Linking ( $p$ and $s$ files)	2.3 sec
Segment tracing and linear approximation (maximum error—2 pixels)	4.8 sec

All computation times, except for linear segment fitting, scale linearly with the number of points to be processed. Also, except for the linear segment approximation, the storage requirements are limited to only a few lines of an image at a time, and we are able to handle images of large size such as  $2048 \times 2048$ . As much of the processing uses small image neighborhoods, inexpensive hardware implementation should be feasible.

### 4. OBJECT DESCRIPTION AND ROAD DETECTION

An important step in Image Understanding systems is to compute descriptions of objects useful for their recognition. Certain classes of objects, such as roads and airport runways, are characterized as being bounded by elongated parallel lines. These lines have opposite contrasts and we call them "antiparallel" lines, abbreviated as "apars." These apars are conveniently described by a medial line and the width of the pair.

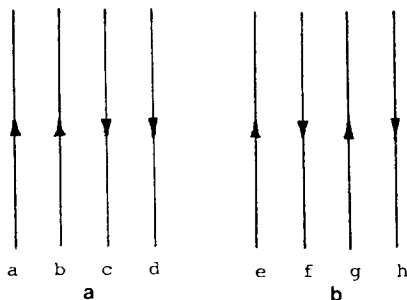


FIG. 7. Some antiparallels illustrating difficulties of object isolation.

These descriptions were motivated as being two-dimensional special cases of Binford's "Generalized Cone" representation [7]. A generalized cone is generated by sweeping an arbitrarily-shaped planar cross section along a space curve. The size and sometimes also the shape of the cross section varies according to given functions. While this representation has many desirable properties, its computation is difficult and the previous implementations [19, 20] have assumed perfect boundaries as input.

To find antiparallel pairs, the line segments are first sorted by their orientation. Sorting of segments is done simply by grouping the segments with the same orientations correct to the nearest integer. Then we search for segments whose angles are  $(180 \pm \alpha)$  degrees apart, where  $\alpha$  is a tolerance. Further, we require that the segments overlap and that they be within a certain distance of each other. An antiparallel pair is described as a 2-dimensional generalized cone, with an axis and a width and an additional attribute of relative brightness.

Proper choice of apars that correspond to objects can be difficult and is like resolving figure-ground relationships (e.g., see Figs. 7a and b). However, for many applications, such as road and runway detection, a choice of the closest pairs may suffice and may also be aided by knowledge of the desired objects being brighter or darker than the background. Also, the axes of the apars can be merged on the basis of collinearity to form larger apars.

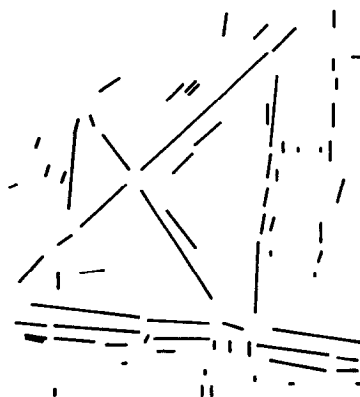


FIG. 8. Axes of antiparallel segments for the airport image (Fig. 2a).



FIG. 9. Aerial image of Stockton, California area.



FIG. 10. Linear segments detected in Fig. 9 (segment length  $\geq 4$  pixels).



FIG. 11. Axes of apars detected in the Stockton image.

Figure 8 shows the axes of the apars computed from the segments of Fig. 6. Only the closest pairs were retained. In the future, we hope to recognize structures from the spatial relationships of partial axes of the type shown here.

As an example of potential applications, the above description technique was used for the detection of parts of roads, as they are bounded by locally straight and parallel lines. Figure 9 shows an aerial image of the Stockton area in California. Figure 10 shows the detected straight segments (only segments longer than 4 pixels are displayed). Figure 11 shows the axes of the apars of width less than 10 pixels.

Note that many of the apars correspond to the roads in the image, but others correspond to narrow sections of rivers or other objects and some are simply formed by accidental alignment. Further, the roads are fragmented, sometimes due to the failings of the line finding algorithms, but also due to causes such as shadows and obstructions. Complete detection of roads under these conditions and distinguishing them from other narrow structures requires global analysis and is beyond the scope of this paper; some approaches may be found in [21]. However, even the partial roads have been useful in our experiments with matching of map-like models of an area to descriptions computed from an aerial image of the same area [18].

Some improvements can be obtained by connecting apars to form longer links. Two apars are connected if either segment of any apar is connected to a segment of a neighboring apar, i.e., if the two apars are formed by segments belonging to the same supersegment as defined in Section 3 (see Fig. 12). Figure 13 shows the axes

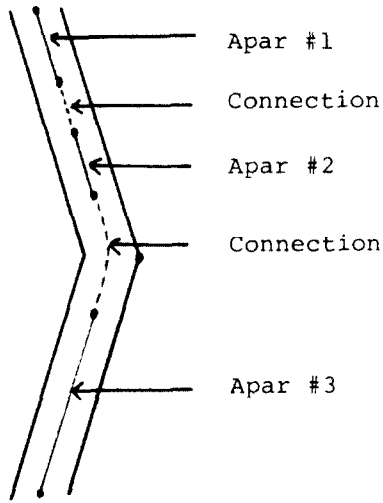


FIG. 12. Connecting apars.



FIG. 13. Connected apars in the Stockton scene.

of connected pairs for the Stockton image. Further improvements can be obtained by bridging one-pixel gaps between the boundary segments as described previously.

#### 5. CONCLUSIONS

We have described a technique for line finding that has been tested on a large number and variety of images, yielding good and robust performance. We have also described early attempts at using the detected lines to generate useful symbolic descriptions in complex images and have presented results for an application to road detection.

#### ACKNOWLEDGMENTS

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense under Contract No. F-33615-76-C-1203. Dr. Keith Price has been extremely helpful in many discussions and in providing image manipulation software.

#### REFERENCES

1. A. Rosenfeld and A. Kak, *Digital Picture Processing*, Academic Press, 1976.
2. W. K. Pratt, *Digital Image Processing*, Wiley, New York, 1978.
3. J. R. Fram and E. S. Deutsch, On the quantitative evaluation of edge detection schemes and their comparison with human performance, *IEEE Trans. Comput.* **C-24**, 1975, 616–628.
4. I. K. Abdou, Quantitative methods of edge detection, University of Southern California, USCPI Report 830, July 1978.
5. M. Hueckel, A local visual operator which recognizes edges and lines, *J. ACM* **20**, 1973, 634–647.
6. R. Nevatia, Locating object boundaries in textured environments, *IEEE Trans. Comput.* **25**, 1976, 1170–1175.
7. T. O. Binford, Visual Perception by a Computer, IEEE Conf. on Systems and Controls, Miami, Fla., December 1971.
8. L. S. Davis, A survey of edge detection techniques, *Comput. Graphics Image Proc.* **4**, 1975, 248–270.
9. A. Herskovits, On boundary detection, MIT AI Memo 183, 1970.
10. A. Rosenfeld and M. Thurston, Edge and curve detection for visual scene analysis, *IEEE Trans. Comput.* **20**, 1971, 562–569.
11. D. Marr, Early processing of visual information, MIT AI Memo 340, 1975.
12. L. G. Roberts, Machine perception of three-dimensional solids, in *Optical and Electro-optical Information Processing* (J. T. Tippett *et al.*, Eds.), pp. 159–197, MIT Press, Cambridge, Mass., 1965.
13. A. K. Griffith, Edge detection in simple scenes using a priori information, *IEEE Trans. Comput.* **C-22**, 1973, 371–381.
14. F. O’Gorman and M. B. Clowes, Finding picture edges through collinearity of feature points, in *Proc. 3rd International Joint Conf. on Artificial Intelligence*, pp. 543–555, Stanford, Calif., 1973.
15. U. Ramer, Extraction of line structures from photographs of curved objects, *Comput. Graphics Image Proc.* **4**, 1975, 81–103.
16. R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.
17. C. Clark *et al.*, High Accuracy Model Matching for Scenes Containing Man-Made Structures, in *Proc. SPIE Symposium on Digital Processing of Images*, Huntsville, Ala., May 1979.
18. R. Nevatia and K. Price, Locating objects in aerial images, University of Southern California Technical Report, in preparation.
19. G. Agin and T. O. Binford, Computer Description of Curved Objects, in *Proc. 3rd International Joint Conf. on Artificial Intelligence*, pp. 629–635, Stanford, Calif., 1973.
20. R. Nevatia and T. O. Binford, Description and recognition of curved objects, *Artificial Intelligence*, **8**, 1977, 77–98.
21. M. Fischler *et al.*, The SRI Road Expert: An Overview, in *Proc. ARPA Image Understanding Workshop*, pp. 13–19, Pittsburgh, Penn., November, 1978.