Chapter 2
Segmentation and Vectorization

The segmentation and vectorization of line drawings is well studied[1] (see Chap. 1 for related work), however, no analytic solution has been proposed to date, and therefore, the existing methods are ad hoc and based on heuristics. Most methods also need some amount of tweaking for the specific kind of drawings to be analyzed. We present our approach to these problems here, and this development has been done in the context of the semantic analysis of scanned images of mechanical CAD engineering drawings.

2.1 Segmentation

We assume that a binary image of the drawing is available, and that the line drawing part is the foreground (i.e., pixel value is 1, and usually displays as the color white; note that in this book we will show the foreground pixels as black in figures for better visual clarity). The processing sequence is shown in Fig. 2.1. By segmentation, we mean to divide the skeleton of a connected component into significant pieces that contribute to the intended stroke structure of the component. The input is the image of an individual connected component (e.g., produced by Matlab's *bwlabel*), and the final result is a set of segments from the skeleton. Figure 2.2 shows the set of segments from the image of the digit 3. Each segment consists of a sequence of skeleton pixels (called the *segment path*) that starts and ends with either an *endpoint*, *branchpoint*, or a *virtual point* of the component. In the figure, the endpoints are shown with star shapes, while the single branchpoint is shown as an annulus. Endpoints and branchpoints can be found from simple computations on the neighborhood of a pixel (e.g., see Lam et al. [66]). Matlab's *bwmorph* function can be used to identify these types of points. However, note that *bwmorph* can produce branchpoints that one would not always typically consider

[1] Some parts of this chapter are contributed by Chimiao Xu based on her MS thesis.

T.C. Henderson, *Analysis of Engineering Drawings and Raster Map Images*,
DOI 10.1007/978-1-4419-8167-7_2, © Springer Science+Business Media New York 2014

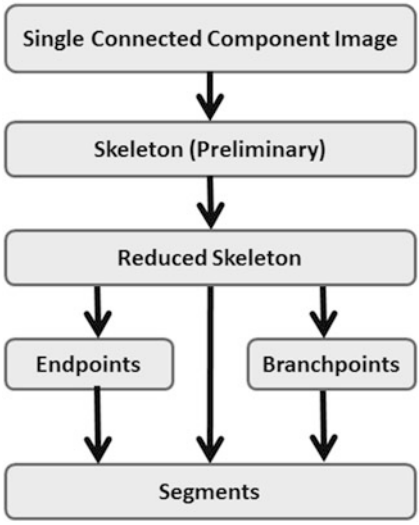


Fig. 2.1 The segmentation process

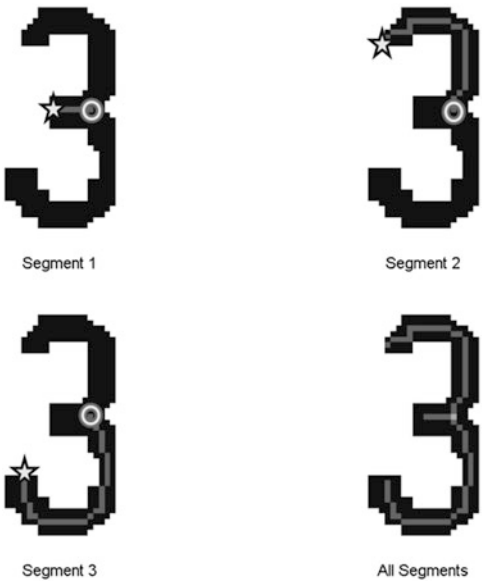


Fig. 2.2 Segments of digit "3"

to be branchpoints. Figure 2.3 shows on the left a set of branchpoints produced by *bwmorph* and on the right our corrected version. In order to eliminate false branchpoints, it suffices to check the following condition on each one: first find the pixels which are three neighbors distant from the branch point—if this set forms three or more distinct connected components, then the branchpoint is valid. *Virtual endpoints* are needed when there is a single cycle (or pixel loop) as will be found in

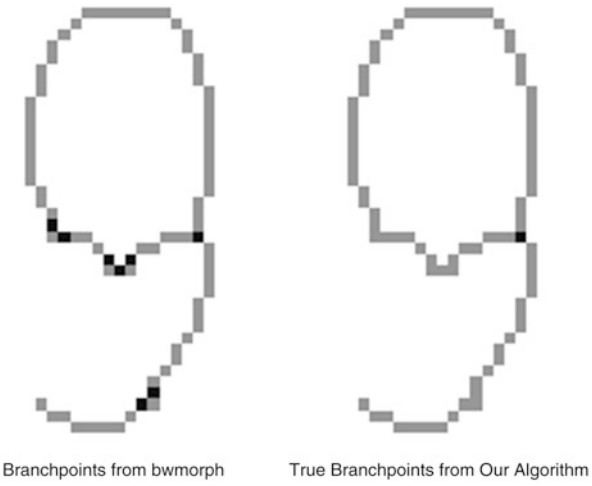


Fig. 2.3 False branchpoints produced by *bwmorph*

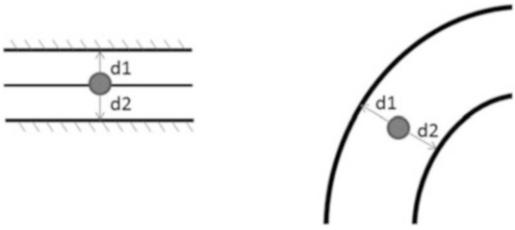


Fig. 2.4 Inside a straight or smoothly bending corridor. The two minimal distance directions to the background are 180 degrees apart

the character "O" and there are no distinguishing points; the *virtual endpoint* is any point of the cycle and is used as both endpoints of the segment path.

Before finding the segment paths, we first eliminate spurious points from the skeleton and produce the *reduced skeleton*. To achieve this, we take advantage of the following observation. Since the line drawings we analyze here are for the most part composed of simple strokes, the vast majority of skeleton pixels will be situated in one of the following contexts:

Context 1: *Inside a straight or smoothly bending corridor* (see Fig. 2.4). In this context, there will be two directions with about the same range value, and these will be the minimum of all range values in a 360 degree scan. These two minima will be about 180 degrees apart. The sum of these two distances is the width of the corridor. Finally, the line defined by the two minimal directions is perpendicular to the direction of the corridor; the skeleton pixels should be running along the middle of the corridor.

Context 2: *Inside a right angle turn in the corridor* (see Fig. 2.5). In a right angle turn, the two minimal range directions are about 90 degrees apart. However, the minimal sum of two opposite directions (180 degrees apart) gives the corridor width and will usually be in the diagonal direction as shown in the figure.

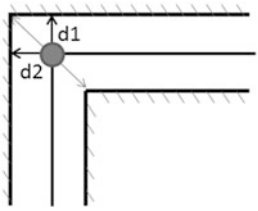


Fig. 2.5 Inside a right angle turn in a corridor

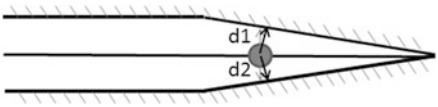


Fig. 2.6 At the end of a corridor

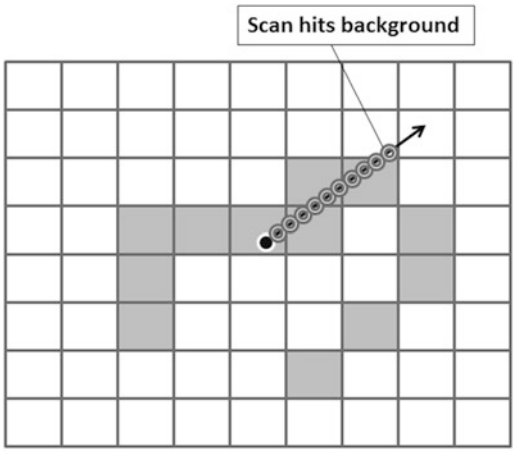


Fig. 2.7 The sub-pixel resolution range scan

Context 3: *The end of a corridor* (see Fig. 2.6). At the end of a corridor, the directions of minimal range to the background will no longer be in opposite directions, but the medial axis will be the bisector of those two directions. The minimal sum of opposite range values will generally be perpendicular to the skeletal axis. This feature distinguishes the right angle turn from the end of corridor and straight or smoothly bending corridor. Note that the end of a corridor may taper together like a pencil point or be more like a the end of a rectangle.

In order to analyze these contexts, we have developed a few image analysis tools. One very useful function is the virtual range finder, *range_scan*. This function determines the distance to the background from a given pixel, starting at zero degrees, rotating in the positive direction (in a right-hand frame; i.e., counterclockwise) in $\delta\theta$ increments until 360 degrees. Our scans are usually in one degree steps. Then in the direction of the scan, we step δx pixels (usually set to 0.001) until a background pixel is reached. Figure 2.7 shows how this works. A sub-pixel length step is made in the scan direction until the step location is in a background pixel. The range map produced by this function is called the *Pseudo-Range Map* (PRM). (Note that it is also possible to obtain a more exact range scan by

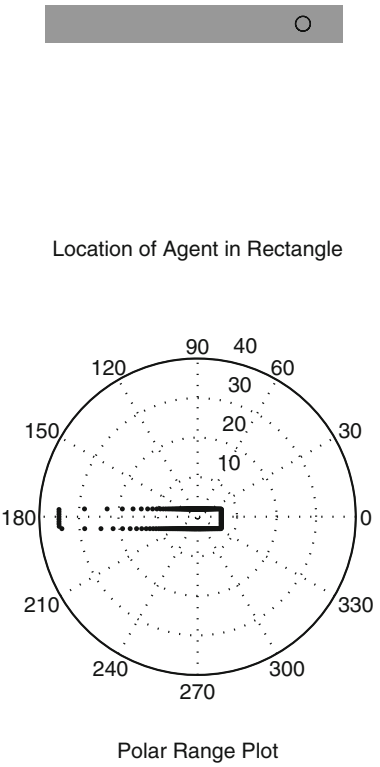


Fig. 2.8 The non-uniform nature of equal-angle sampling in the PRM

intersecting the scan ray with the column and row lines that it crosses, and then give the exact pixel boundary where the change from foreground to background occurs.) The use of robot mapping techniques as a means to understand engineering drawings was the topic of the Masters thesis of Xu [131] who showed that the foreground image could be treated as a floor plan that a small mobile robot navigates in order to segment the components into line segments, endpoints and branchpoints. It was demonstrated there that accurate and robust segmentations could be achieved using this approach which views drawn lines and symbols as hallways and rooms, and the mobile agent is placed *inside* the component and located with sub-pixel accuracy. The Pseudo-Range Map is used by the robot agent to survey the component. (See also Henderson and Xu [51].)

An important issue is whether an equal angle scan should be performed, or a scan that achieves an equal step distance on the boundary. Since the equal angle sample is easier to implement, we have used that, however, in some cases it is better to use the alternative approach since, for example, the center of mass will be highly skewed when most of the scan points are at points near the agent, and far points are sampled less frequently spatially. Figure 2.8 shows a set of scan points and how they are distributed in space (they are plotted as points so that they are more visible). The set of scan points also influences the computation of derived features, such as the

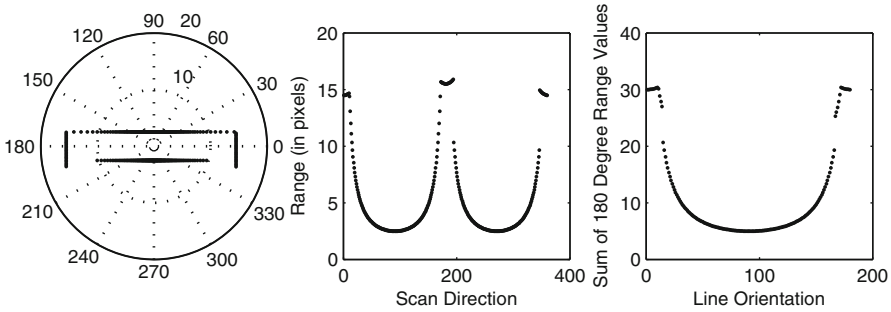


Fig. 2.9 Range scan inside a straight corridor

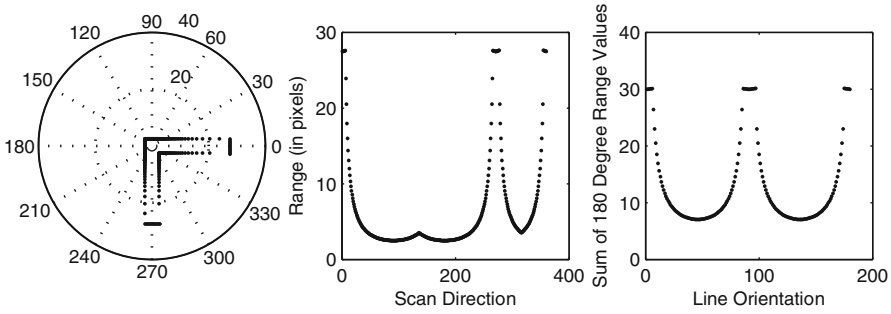


Fig. 2.10 Range scan in a right turn of a corridor

normal at each scan point. As mentioned, another important issue is the resolution of the step in the direction of the scan. The scan distance is found as follows:

- start at the given sub-pixel location
- move δx in the scan direction to a new point x
- stop once x lies in a background pixel.

If δx is too small, then this may be computationally expensive, whereas if it is too large, then background pixels may be skipped over. We have found that $\delta x = 0.001$ provides good pseudo-range scans.

Let's see how this helps determine the context of a skeleton pixel in a corridor. Figure 2.9 shows (on the left) a polar plot of the range to the background for an in corridor pixel; in the center is a plot by direction (0–360 degrees), and on the right is the sum of the two opposite directions of the lines from 0 to 180 degrees. As can be seen the two minimal directions are 90 degrees and 270 degrees (across the corridor) and the minimum sum of opposite directions is at 90 degrees. Figure 2.10 shows the same information for a right turn, and Fig. 2.11 shows the results for the end of a corridor. To produce good skeletons, we proceed as follows. First, *bwmorph* is used to produce a first set of skeleton pixels. Figure 2.12 shows the skeleton produced for the digit four. As can be seen there are a few spurious paths. Figure 2.13 gives the

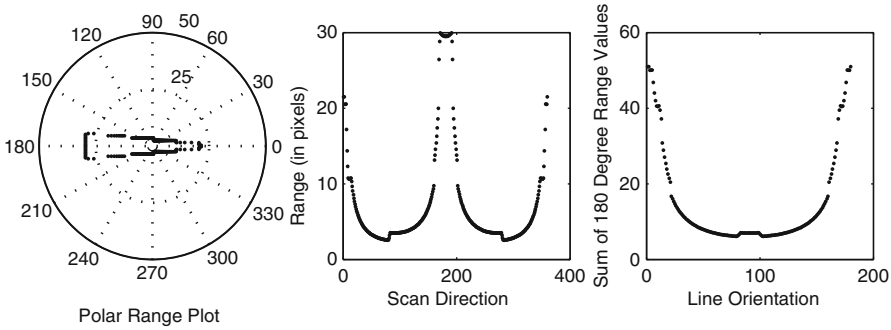


Fig. 2.11 Range scan in the end of a corridor

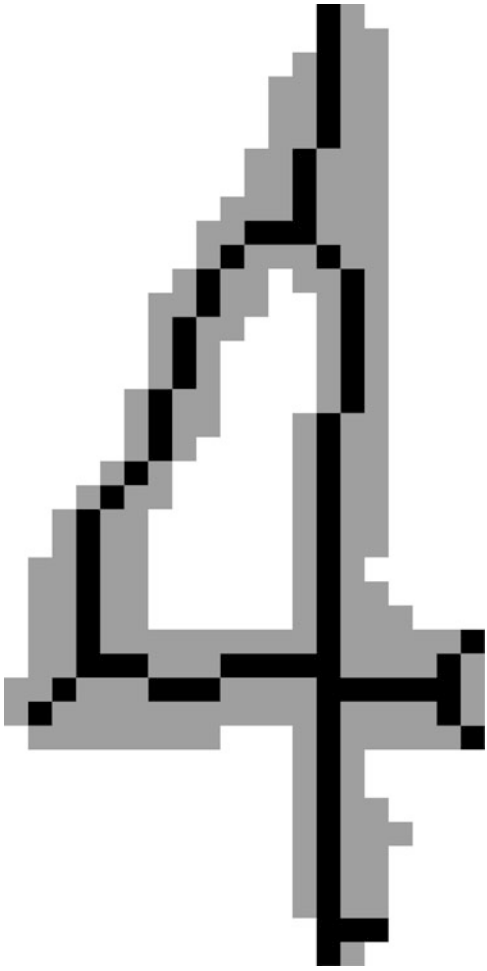


Fig. 2.12 Skeleton produced by *bwmorph*

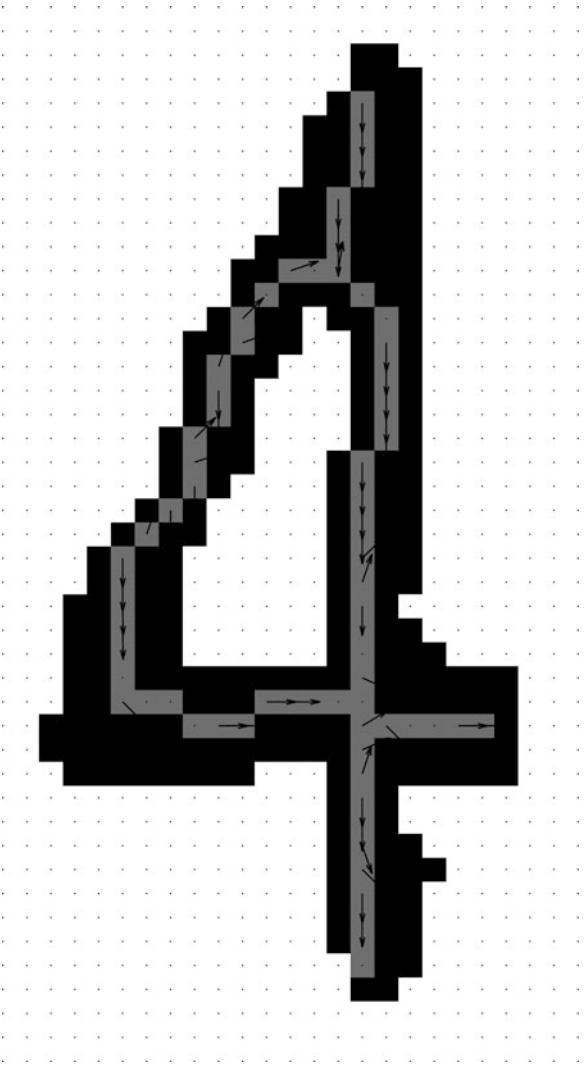


Fig. 2.13 Reduced skeleton produced by pseudo-range map techniques

reduced skeleton produced using the methods described above. Note that the figure also shows the estimate of the corridor direction produced for each pixel. Figure 2.14 shows the skeletonization result on some images of the ten digits, while Fig. 2.15 shows the same for the first ten letters of the alphabet. Figure 2.16 shows the results on the arrow part of a dimension set.

Xu studied two important aspects of the engineering drawing analysis problem: point feature analysis and linear feature analysis. The PRM was used pixelwise to identify:

1. *endpoints*: the terminal part of a line segment.

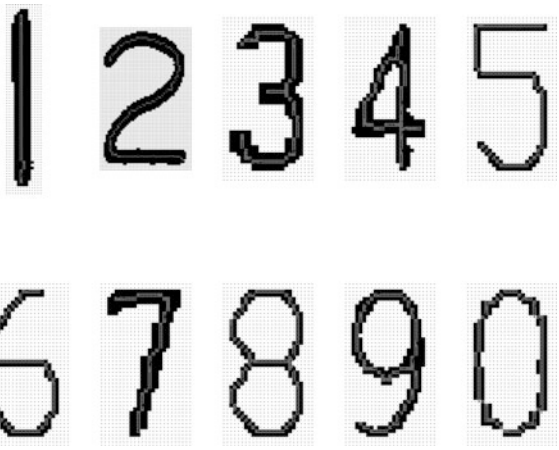


Fig. 2.14 Skeleton produced for digits

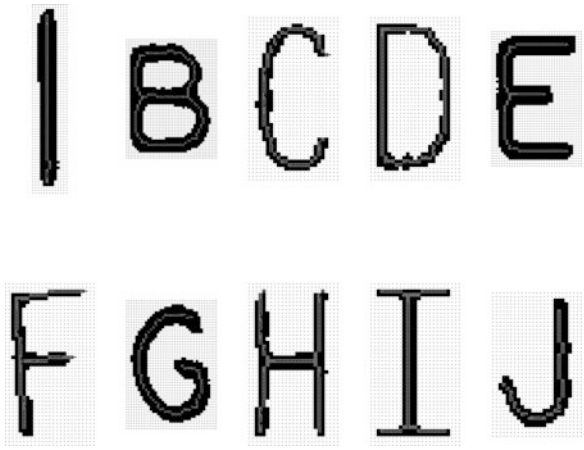
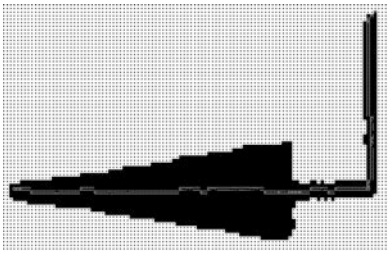


Fig. 2.15 Skeleton produced for letters

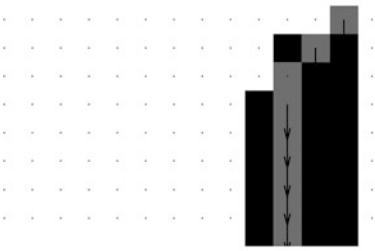
2. *interior corridor points*: two directions of travel are possible, but not a corner.
3. *corner points*: two directions of travel possible, but at significantly different angles.
4. *multibranch points*: more than two directions of travel possible.

Methods were developed to identify these four types of point features, and a decision tree method was used to classify pixels. A set of attributes of the pixel and its PRM were selected, and at each nonterminal node of the tree an attribute value is used to pick a branch to follow, and each leaf node provides a point feature decision (i.e., one of the four possible types). An information theoretic approach was used to inform the construction of the decision tree (see [101]). Attributes used in building the decision tree included:

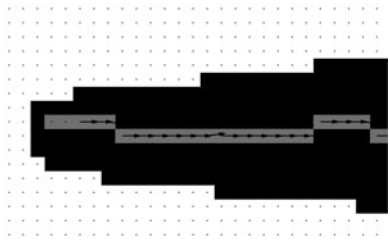
- **Attributes 1:7**: Hu invariant moments [57] of the PRM.
- **Attribute 8**: Area of the PRM region.



Skeleton for an Arrow



Detail of Upper Part of Arrow



Detail of Point Area of Arrow

Fig. 2.16 Skeleton produced for arrow of dimension set

- **Attribute 9**: Perimeter of the PRM region.
- **Attribute 10**: Sum of ranges of PRM.
- **Attribute 11**: Total distance of the points in the PRM perimeter to the point that is used to compute the PRM.
- **Attribute 12**: Sum of the absolute distance of the rows and columns of the PRM region points to the row and column of the point used to compute the PRM, divided by the total number of points in the PRM region.
- **Attribute 13**: Number of branches on the PRM.

Table 2.1 The *recall* and *precision* results for the non-PRM, PRM and decision tree methods

	F_end	F_corr	F_corn	F_bra	
Non-PRM	91.94	99.26	64.32	83.10	
PRM	98.59	98.73	99.93	98.96	*Recall*
D-Tree	98.92	86.38	73.41	93.80	
Non-PRM	45.62	100.00	43.12	40.13	
PRM	52.14	100.00	41.10	49.98	*Precision*
D-Tree	33.38	100.00	34.16	35.65	

The attributes with the most discriminatory power for each feature type were: (1) endpoints: Attribute 11, (2) corridors: Attribute 2, (3) corners: Attribute 12, and (4) branches: Attribute 11.

The overall system performance was evaluated in terms of the quality and computational complexity demonstrated over various image datasets. Since noise occurs in the images, there are extraneous as well as missing objects. In order to objectively measure how well the proposed system analyzed digitized engineering drawings, we compared the results over a dataset for which ground truth was available. The following steps were taken:

- A testbed benchmark set of five images and ground truth were established.
- A non-PRM classifier was run on the images.
- A hand-made PRM classifier was run on the images.
- A decision tree classifier was developed using training data and run on the dataset.

To measure the performance, we use *recall* to mean the ratio of correct features found to the total number of relevant features, and *precision* to mean the ratio of the number of correct features found to the total number of features found. Table 2.1 gives the results of the performance analysis.Although the static PRM feature classification method works well as far as *recall* is concerned, good precision can only be achieved by applying a post-processing step.

Xu also developed methods for linear feature extraction. As described in Chap. 1, much work has been done on this topic, and our method can be viewed as an extension of the *Zig-Zag* method of Dori [30]. The mobile robot mapping approach exploits the Generalized Voronoi Diagram (GVD) [12]. After constructing the GVD, linear segments are found by using the differential GVD curvature.

In the free space of a plane with a set of obstacles, the Voronoi diagram (VD) [24] is defined as a collection of line segments or convex polygons that partition free space into n cells so that each cell contains one obstacle. Any point on the edge of two neighboring cells is equidistant to the two particular obstacles in the two neighboring cells. When the obstacles extend to any object, the VD is called the generalized Voronoi diagram (GVD) for these objects. In the robotics path planning community, the GVD is used as a roadmap through a planar set of obstacles. The planar space is divided into *free space*, or a set of points through which the robot may pass, and the *obstacle points* through which the robot may not pass. Roadmaps

provide a structure completely describing the topology of the workspace, including: accessibility, connectivity and departability. Robot motion planning consists of:

1. Find a way to get to the GVD.
2. Find a path along the GVD to the neighborhood of the goal point.
3. Find a way from the GVD to the goal point.

The GVD is more formally considered a deformation retract, defined in terms of a continuous map from the free space of the GVD. In our problem domain, we are only interested in the GVD points found in the corridors (i.e., curvilinear segments); at segment ends, we want the points to go straight to the end of the line segment rather than splitting and going to the corner points as occurs with the medial axis transform.

The modified GVD (MGVD) points, or perhaps more appropriately, the center line points, are characterized by being midway between two scan points that are 180 degrees separated in the range scan, are a minimally distant pair of 180 degree separated points, and have inward pointing normals. The fundamental idea of the segmentation algorithm introduced by Xu is based on the fact that every foreground pixel must belong to a linear segment or multiple segments if the pixel is either a corner or branch point. The PRMs of most points in a given linear segment share similar features captured by the PRMs. Such points are usually interior corridor points. The algorithm takes the following steps to extract a MGVD path as a linear segment.

1. Find an unexpanded interior corridor point **P**.
2. Compute **P**'s PRM and the longest forward direction **FDIR**.
3. Compute a MGVD path along **FDIR** and its opposite direction **BDIR** until it hits the background pixel boundary.

The set of MGVD nearest neighbor points form a linear path through the segment. Hence the linear segment to which **P** belongs has been extracted. By iterative application of the above steps on a connected component foreground object, the algorithm extracts all linear segments of the connected component. Performance measures for the MGVD extraction method were developed and tested on several engineering drawing images. Correspondence to ground truth segments is made based on:

1. The distance between segment endpoints **gpt1** and **pt1**, and distance between the other segment endpoints **gpt2** and **pt2** must both be less than the segment's width.
2. The absolute difference between **GDIR** and **DIR** must be less than 5 degrees out of 360.

   - **gpt1** and **gpt2** are the two endpoints of a ground truth segment, and **pt1** and **pt2** are the two endpoints of an extracted segment.
   - **GDIR** is the ground truth segment's direction, and **DIR** is the extracted segment's direction.

The MGVD method was compared to a Hough-based method on eight test images, and the average correctness result for MGVD was 99.63% whereas the Hough method was at 82.98%.

In summary, Xu's work explored the use of robot navigation and mapping techniques in the analysis of engineering drawings. Contributions include:

- *Definition and Analysis of the Pseudo Range Map (PSM)*: For each foreground point (to sub-pixel accuracy), the distance to the background is computed for a selected set of angles and to sub-pixel accuracy. The PRM has been shown to provide a robust and stable basis for mapping analysis.
- *Point Feature Analysis*: Given the PRM at a specific location we have demonstrated that the shape of the PRM permits classification of the location into one of the categories: endpoint, corridor point, corner point or branch point. The correct classification rates are very high compared to conventional techniques.
- *Linear Feature Analysis*: Given the ability to move an agent in the foreground, we have demonstrated the extraction of an approximation to the Modified Generalized Voronoi Diagram (MGVD) and can exploit the MGVD to segment the drawing into 1D curves and straight line segments.

2.2 Vectorization

Once a set of segments is identified, the next step in the process is to produce a higher-level approximation to the segments; recall that a segment consists of a set of pixels running from an endpoint to an endpoint, or an endpoint to a branchpoint, a branchpoint to a branchpoint, or a virtual endpoint to itself. For the application at hand, we use a straightforward line fitting algorithm which tracks from one segment endpoint to the other taking in new pixels until a fitting threshold is exceeded. After a piecewise polyline is produced, the endpoints are adjusted for a better fit for the two vectors to either side.

Figure 2.17 shows the vectors produced for the digits.

2.3 Connected Component Information

Although the segments and vectors are the core information about each connected component, it is useful to put together more information about the component which will make further analysis easier. This information includes:

- *image*: original image
- *branchpoint image*: specific branchpoint pixels
- *skeleton*: center line points through component
- *segments*: the segments as described above; for each segment:

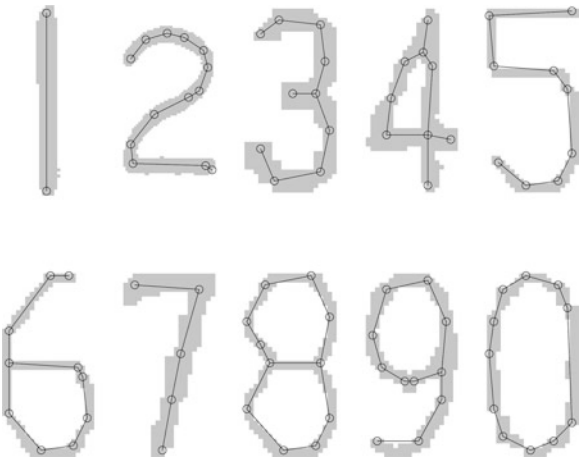


Fig. 2.17 Vectors produced for digits

  - *path*: row, col array from one endpoint to the other
  - *num_rows*: number of rows in the original image
  - *num_cols*: number of columns in the original image
  - *endpoints*: endpoints of the segment; these are actual termination points (i.e., not branchpoints or virtual points)
  - *branchpoints*: branchpoints of the segment
- *vectors*: the vectors as described above; for each vector:
  - *points*: row, col array of points in the vector
  - *num_pts*: number of points in the vector
  - *index 1*: segment path index for first point of vector
  - *index 2*: segment path index for last point of vector
  - *error*: error for vector fit to segment
  - *endpoint 1*: row, col of first point
  - *endpoint 2*: row, col of second point
  - *theta*: orientation of vector
  - *num_rows*: number of rows in original image
  - *num_cols*: number of cols in original image
- *segment neighbors*: s by s array of segment neighbors
- *vector endpoint info*: for each endpoint, first and second element are row, col of endpoint, and third element is the number of vector neighbors
- *vector neighbors*: $2v \times 2v$ array of vector endpoint neighbors
- *cycles*: cycles found in the component
  - *paths*: list of vector endpoint sequence that defines the cycle
- *vector tree*: tree of vector connectivity; each node has:
  - *parent*: index of parent node

  - *index*: vector endpoint index
  - *children*: indexes of children nodes
- *paths*: endpoint to endpoint paths (starting vector is indexes 1 and 2)
  - *path*: sequence of vector endpoint indexes from one endpoint to the other

These items are useful when trying to recognize characters, arrowheads, graphics, etc. For example, the digits “0”, “4”, “6”, and “9” have one cycle (unless it is broken due to noise or touching another component); moreover, the digit “8” has three cycles: the upper and lower two circles as well as one around the outer periphery. The path information allows one to more easily find, e.g., the correct one path through the segments which are produced for the character “2” (alternative paths may be created by spurs and noise). The exploitation of this information will be further elucidated in following chapters.

http://www.springer.com/978-1-4419-8166-0

Analysis of Engineering Drawings and Raster Map Images

Henderson, Th.C.

2014, XII, 241 p. 214 illus., Hardcover
ISBN: 978-1-4419-8166-0