A New Approach for Line Recognition in Large-size Images Using Hough Transform

Jiqiang Song, Min Cai, Michael R. Lyu, and Shijie Cai*
Dept. Computer Science & Engineering, the Chinese University of Hong Kong, Hong Kong, China
{jqsong, mcai, lyu}@cse.cuhk.edu.hk
* State Key Lab. of Novel Software Technology, Nanjing University, Nanjing, China
sjcai@netra.nju.edu.cn

Abstract

The application of Hough Transform (HT) has been limited to small-size images for a long time. For large-size images, the peak detection and the line verification become much more time-consuming. Many HT-based line detection methods are not able to detect line width. This paper proposes a new approach for detecting line segments using HT, which makes HT applicable to large-size images, especially for those applications whose line width is critical. Our approach applies a boundary recorder to eliminate redundant analyses, and employs an image-analysis-based line-verification method to overcome the difficulty of using a threshold to distinguish short lines from noise. It avoids the overlapping lines by removing the pixels of detected line segments, which is more robust than only clearing the NxN neighborhood. This approach could be easily extended to improved HT methods that perform the global accumulation. The experimental result shows that this approach is very time-efficient for large-size images.

1. Introduction

Hough Transform (HT) is a powerful tool for finding predefined features in digital images [1]. Since HT converts a difficult global detection problem in image space into a more easily solvable peak detection problem, it can deal with noise, gaps, and partial occlusion, even in complicated background. HT is capable of detecting straight lines, circles, ellipses and other curves in both binary and grayscale images. However, most reported applications of HT are limited to small-size images. The attempts of applying HT to large-size images are usually discouraged by the well-known weaknesses of HT: the time-inefficiency, the difficulty of choosing a proper threshold to distinguish short lines from noise, and the missing of the line width.

Generally, using HT to detect lines consists of three steps: accumulation, peak detection, and line verification. Typically, a pre-processing is necessary to extract feature points from the image to be transformed, usually medial points or edge points. The first two steps have been well investigated so far. An abundant number of improved HT methods, e.g. gradient-based HT [2], randomized HT [3], probabilistic HT [4] and sampling HT [5], have been proposed to accelerate the accumulation and to highlight the peaks greatly. These techniques can also be applied to large-size images. There are also many ways to detect peaks after the accumulation. The common way is to find the local maxima within an $N \times N$ neighborhood [6], where N is very critical: using too large N will suppress some real lines, while using too small N will yield overlapping lines. Princen *et al* [7] proposed an iterative global peak detection method. This method is more robust than clearing a rigid-size neighborhood, but it is very time-consuming for a large-size image due to the iterative accumulations. In fact, it is only applied to a sub-image in [7]. The line verification step is to get the exact location of line segments along the line. The basic method is sequentially checking the connectivity of feature points within the narrow strip area determined by the peak parameter (r,θ), the quantization interval Δr, and the sampling interval $\Delta\theta$. Since the line equation is calculated frequently and the feature points are searched iteratively, this step may be more time-consuming than the previous two steps for large-size images containing numerous lines. And, it cannot detect the line width. However, the improvement on this step is seldom addressed.

The motivation of this paper is to recognize lines in poor-quality scanned images of engineering drawings. These images usually contain noise and broken lines with rough edges. Popular graphics-recognition methods, including thinning-based ones, contour-based ones, and pixel-tracking-based ones, cannot handle these images well, since they all depend on the connectivity and parallel edges of the line image. However, HT has distinguished

1051-4651/02 $17.00 © 2002 IEEE

advantages for these cases. Therefore, we propose an efficient HT-based approach to verify lines and detect the line width, because the line-width information is critical to further processing of engineering drawings.

2. Line detection algorithm

As engineering drawings are usually stored and processed in binary format, we assume that the image is monochromatic (i.e., black for foreground and white for background). We predefine two thresholds, MIN_LW and MAX_LW, to indicate the minimum and maximum acceptable line widths, respectively. In the pre-processing, we perform horizontal and vertical run-length scans on the input image to extract the medial pixels of valid runs to be feature points. The length of a valid run must be between MIN_LW and MAX_LW. Considering the poor image quality, one-pixel-long gap does not break a run.

To ease the description, we choose the standard HT for straight line (see Equation 1 below) to explain the line detection algorithm.

$$r = x\times\cos\theta + y\times\sin\theta \qquad (1)$$

2.1 Boundary recorder

One important reason for the time-inefficiency of common line verification methods is that they do not know which part of the strip area contains feature points. Thus they have to either recalculate all feature points with the known θ to pick out those with the same (or similar) r, or check every position within the strip area in the image. Obviously, neither way is fast for a large-size image. Usually, only a small part of the strip area contains the feature points. According to this fact, we add a boundary recorder to each parameter cell, which only contains an accumulator before, to record the minimum scope that contains the feature points contributed to this parameter.

The boundary of each parameter is actually two feature points, called "up boundary" and "low boundary", which enclose all other feature points contributed to this parameter. Since the dimension of parameter space is large when the image size is large, one should be considerate to add any byte to the parameter cell. According to Equation (1), given r and θ, one dimension of the image coordinates can be calculated from another dimension. So we only need to record one dimension of the coordinates in the parameter cell. The choice of X- or Y- dimension coordinates depends on θ (Fig. 1). When $45°<\theta<135°$, the line in the image space is nearly horizontal, so X-dimension coordinates is chosen to record the boundary; otherwise, Y-dimension coordinates is chosen. The initialization and recording processes of the boundary recorder are shown in the following codes.

Initialization: for all parameter cells

```
Param[r][θ].accumulator = 0;
```

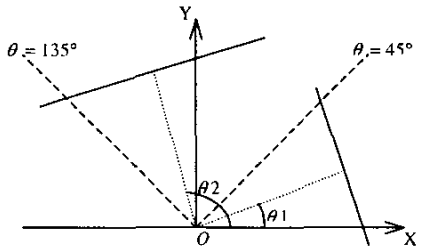

Figure 1. Choice of the recording dimension

```
Param[r][θ].low_boundary = max(X_MAX, Y_MAX);
Param[r][θ].up_boundary = min(X_MIN, Y_MIN);
```

Recording: when a point (x, y) contributes to a parameter

```
Param[r][θ].accumulator += 1;
IF (45°<θ<135°) {
  Param[r][θ].low_boundary=min(Param[r][θ].low_boundary, point.x);
  Param[r][θ].up_boundary=max(Param[r][θ].up_boundary, point.x);
}ELSE {
  Param[r][θ].low_boundary=min(Param[r][θ].low_boundary, point.y);
  Param[r][θ].up_boundary=max(Param[r][θ].up_boundary, point.y);
}
```

Where $\min(a,b)$ returns the smaller one of a and b, and $\max(a,b)$ returns the bigger one.

Consider an image of size $L\times W$, whose range breath of r is at most $\sqrt{L^2+W^2}$. The memory requirement for the 2D parameter array of HT can be calculated as follows:

$$Mem\ Req = \frac{\sqrt{L^2+W^2}}{\Delta r}\times\frac{180}{\Delta\theta}\times sizeof(param\ cell) \quad \text{bytes.}$$

We choose $\Delta r = 2$ and $\Delta\theta = 1°$ to keep both the accuracy of direction and the clustering effect. The parameter cell contains one accumulator, usually an integer (4 bytes), and two boundary recorders that are short integers (2 bytes for each). For a large image of A0-size engineering drawing scanned with 300 dpi, which is fine enough to digitize a line as thin as 0.003 inches, L is about 14,000 pixels and W about 9,900 pixels. Then, the memory requirement for the parameter array is about 12.3 Mega-bytes. This is obviously acceptable to current hardware condition.

2.2 Line verification

In this step, we introduce two more thresholds: MIN_LEN stands for the minimum acceptable line length, and MAX_GAP stands for the maximum acceptable gap length. They are preset according to the image type.

After all feature points are transformed, we detect the local maximal peak within a small 5×5 neighborhood, and all peaks higher than MIN_LEN are stored into a peak list in descending order of peak value. The line verification begins from the head of the peak list; thus, it can also take advantage of the global peak. Owing to the recorded boundary information, we only need to analyze the valuable part in the original image determined by the peak parameter to find the evidence of line segments and detect

the line width of each segment. Then, the pixels of the verified line segments are removed from the image, i.e., turning them from black to white. Since this line verification is based on image analysis, not on feature points, the overlapping lines are successfully avoided by removing the pixels of the verified line segments. Thus, it does not need the re-accumulation, and it is much faster than using the method of [7].

According to the boundary recording process defined in Section 2.1, the image coordinates of two boundary points can be calculated easily, denoted by P_{lb} and P_{ub}. The line verification analyzes the image along the straight-line direction from P_{lb} to P_{ub} sequentially. To avoid the heavy computation in solving the equation, we adopt the off-the-shelf rasterization method – Bresenham algorithm for straight line [8], which generates a straight-line path point with at most three additions – to generate the eight-connected path points from P_{lb} to P_{ub}, denoted by P_i (i=1..n), where $P_1=P_{lb}$ and $P_n=P_{ub}$. The detailed image analysis algorithm is as follows.

```
gap_count = 0; start_pos =1;
FOR (i = 1 TO n) {
  IF (Pi is black) {
    gap_count = 0;
    IF (start_pos == 0) start_pos = i;
    IF (i == n) {
      IF (VerifySegment(start_pos,i) == true) {
        Accept this segment;
        Remove the pixels of this segment;
      }
    }
  }ELSE {
    gap_count +=1;
    IF (gap_count == MAX_GAP or i == n) {
      IF (i-gap_count-start_pos > MIN_LEN) {
        IF (VerifySegment(start_pos,i-gap_count) == true) {
          Accept this segment;
          Remove the pixels of this segment;
        }
      }
      start_pos = 0;
    }
  }
}
```

This algorithm detects all segments from P_{lb} to P_{ub} that are longer than MIN_LEN and do not contain gaps longer than MAX_GAP. For each segment, it then calls VerifySegment(*start,end*) for verification by checking the line width. The line width of a line segment is voted by all local line widths detected at each black P_i(i=*start..end*). If the voted line width is larger than MAX_LW, this segment may be a part of intersecting line or other shapes so that it should be rejected. Since this algorithm does not depend on a single threshold for the decision purpose, it overcomes the difficulty of trading-off between short lines and noise. Thus, it can distinguish between true lines and a random alignment of points correctly.

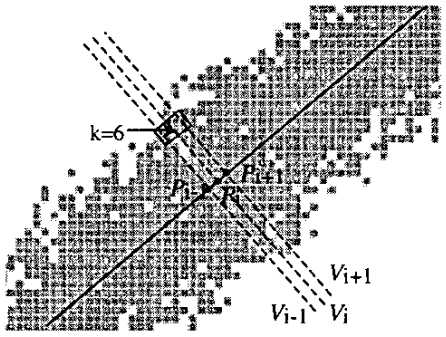

Figure 2. Local line width detection

Considering the poor quality of line image, we use a missing-pixel-tolerant approach to detect the local line width. For each P_i, we use P_{i-1} and P_{i+1} (if available) to help the decision (Fig. 2). V_i is the straight-line path passing P_i and perpendicular to the line $P_{lb}P_{ub}$, which is also generated by Bresenham algorithm. $V_i(k)$ (k=-MAX_LW..MAX_LW) is the point on the V_i path with k steps away from P_i, particularly, $V_i(0)= P_i$. The detection starts from k=0 and increases k by 1 iteratively until the number of black pixels among $V_j(t)$ (j=i-1..i+1, t=k..k+1) is less than 4, and then records the stopped k as k_{max}. Next, it decreases k from 0 with the same criteria to get k_{min}. Finally, the local line width is calculated as k_{max}-k_{min}+1. This approach can detect correct line width from poor quality images as well as high quality images. The verified line segments will be stored with three parameters: starting point, end point, and line width.

2.3 Line removal

After all line segments contributed to a peak have been verified, the pixels of these line segments should be removed from the image to avoid overlapping lines. It is easy to remove the pixels of a line segment within a rectangular area determined by the parameters (the long axis is from the starting point to the end point, and the length of short axis equals the line width). This is correct for an isolated line segment. However, if there are other under-detected line segments intersecting this line segment, their intersection parts will also be removed so that the under-detected line segment will be separated. This problem also exists in other line verification methods based on removing the feature points.

Instead, we use an intersection-preserving approach based on detecting the trends of branches at the intersection [9]. It simply removes those parts whose local line widths are less than or similar to the line width of this line segment as rectangular areas. For other parts, i.e. intersection parts, it detects the trend of branches toward the line segment to approximate the border for removing and for preserving purposes. This approach only removes

the pixels belonging to the verified line segments. Thus, the overlapping lines are avoided, and the entirety of under-detected line segments is kept as well.

3. Experimental results

We have implemented our approach based on the standard HT using VC++, and the experiment was performed on a PC with PIII500 CPU and 256M RAM. Figure 3 shows a fraction of the line recognition result on a real image. The left picture is the original image, and the right one shows the detected lines with their line widths displayed. Both the location and the line width are detected correctly. The vertical broken lines are also recovered, while the horizontal dashed lines are retained.

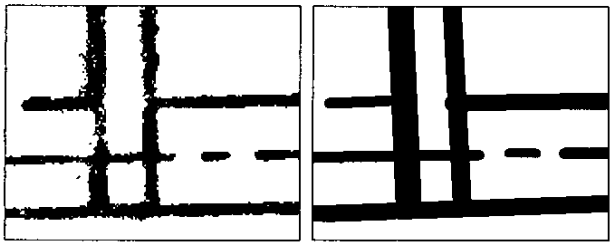

Figure 3. Line recognition result of a real image

The testing data include five real images, which are scanned from engineering drawings of size A4, A3, A2, A1 and A0, respectively. We set MIN_LW to be $0.01\times R$, MAX_LW to be $0.1\times R$, MIN_LEN to be $0.15\times R$, and MAX_GAP to be $0.03\times R$ in the experiment, where R is the scan resolution. Table 1 shows the performance of our approach, where **Time** is the whole processing time for the line detection, **T_LV** is the time spent on the line verification, **LS_Num** is the number of recognized line segments, **T/LS** is the average line-verification time for each line segment, and **Recog_Rate** is the recognition rate.

Image	A4	A3	A2	A1	A0
Size (Pixel^2)	3533 ×2527	4990 ×3537	6959 ×4990	10078 ×6959	13783 ×10078
Storage (MB)	1.05	2.11	4.18	8.41	16.78
Time (sec.)	17.4	39.3	81.5	152.7	303.6
T_LV (sec.)	3.8	8.9	18.3	41.6	87.1
LS_Num	547	1018	1778	3240	5218
T/LS (msec.)	6.95	8.74	10.29	12.84	16.69
Recog_Rate	0.89	0.90	0.91	0.88	0.87

Table 1. Performance over different image sizes

From the above experimental results, we conclude that the proposed approach is very time-efficient since **T_LV** is only a small fraction of **Time**. **T/LS** increases as the image size becomes larger since the average length of line segments becomes longer. The whole processing time is acceptable considering the image size. Actually, the standard HT can be replaced by some proper improved HTs to further accelerate the accumulation.

4. Conclusions

This paper proposes a new approach for detecting line segments using HT. The boundary recorder and image-analysis-based line verification make it very time-efficient. This approach enables HT to process large-size images, especially for those line-width-critical applications. It overcomes the difficulty of choosing a proper threshold to distinguish between short lines and noise, and it avoids the overlapping lines by removing the pixels of detected line segments, which is more robust than just clearing the $N\times N$ neighborhood. This approach could be easily extended to other global accumulation HTs to accelerate the accumulation step. Of course this approach can work with the hierarchical HT [7], which, however, cannot take its full advantages. Since Bresenham algorithm for circle is also very time-efficient, the similar idea can be applied to arc and circle detection, which will be useful to detect dashed arcs and circles with correct line width in a noisy environment.

Acknowledgement

The work described in this paper was fully supported by two grants from the Hong Kong Special Administrative Region: the Hong Kong Research Grants Council under Project No. CUHK4222/01E, and Innovation and Technology Fund, under Project No. ITS/29/00.

References

[1] J. Illingworth and J. Kittler, "A survey of the Hough transform", ***CVGIP***, 1988, vol.44, pp. 87-116

[2] T.M. van Veen and F.C.A. Groen, "Discretization errors in the Hough transform," *Pattern Recognition*, 1981, 14: 137-145.

[3] L. Xu and E. Oja, "Randomized Hough transform (RHT): basic mechanisms, algorithms, and computational complexities", *CVGIP: Image Understanding*, 1993, 57(2): 131-154.

[4] N. Kiryati, Y. Eldar, and A.M. Bruckstein, "A probabilistic Hough transform", *Pattern Recognition*, 1991, 24(4): 303-316.

[5] P.-K. Ser and W.-C. Siu, "Sampling Hough algorithm for the detection of lines and curves", in *Proceedings of IEEE International Symposium on Circuits and Systems*, 1992, vol.5, pp. 2497 -2500

[6] P.R. Thrift and S.M. Dunn, "Approximating point-set images by line segments using a variation of the Hough transform", ***CVGIP***, 1983, vol.21, pp. 383-394

[7] J. Princen, J. Illingworth, and J. Kittler, "A hierarchical approach to line extraction based on the Hough transform", ***CVGIP***, 1990, vol. 52, pp. 57-77

[8] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics: Principles and Practice.* Addison-Wesley, Reading, MA. 1990

[9] J. Song, F. Su, J. Cheng, and S. Cai. "A knowledge-aided line network oriented vectorization method for engineering drawings", *Pattern Analysis and Application*, 2000, 3(2):142-152.