A line segment detection algorithm based on statistical analyses of quantified directions in digital image

Liang Jia, Nigang Sun*

School of Information Science & Engineering, Chang Zhou University, 213164, China

Received 12 June 2014, www.tsi.lv

Abstract

Line segment detection is a typical image processing problem with constantly evolving solutions. Following the line segment detect (LSD) by Grompone von Gioi, two branches of algorithms merged. The first branch aimed to improve its speed at the cost of lower accuracy; the second applied in the opposite way. We investigated the philosophies of these methods, and attempt to develop a line segment detection algorithm based on statistical analyses of quantified directions (LSDSA) to achieve better accuracy and faster speed. We utilize a statistical approach estimating the distributions of pixels with direction values approximating the direction changes when traversing along the edges given by any edge detector. It efficiently reduces the dimension of the input data, and incurs limited increasing in computation time for validation process. The simulation results show that the proposed algorithm achieves better performance compared to the existing typical LSD algorithms. The experiment using industrial data in noisy cases also exhibits excellent performance.

Keywords: Line segment detection, Hough transformation, Image processing, Pattern recognition

1 Introduction

Straight lines, such as straight roads, horizons and the walls, are basic visual elements in the world. They are represented by line segments in digital images. As mobile devices and digital cameras became popular, processing images are serviced as a common daily task for many people, and the number of digital images has increased heavily.

As a basis of the image processing algorithms, line segment detection are useful for various high-level image processing tasks such as crack detection in materials [1], robot-navigation [2] and many others [3, 5, 6, 4]. Roughly, there are three sets of typical algorithms for line segment detection [7]:

(1) Algorithms based on geometric duality [9, 11, 12, 10] such as Hough transformation (HT) [8] and Gaussian kernel-based Hough Transform (KHT) [13]. They usually suffer from the expensive computational costs of implementing geometric duality and the low detection accuracy. Although KHT made a great improvement of the voting procedure introduced by HT, it totally depends on the pre-processing procedure composed of algorithms proposed in [14] and [15] to provide the input data.

(2) Algorithms based on the analysis of the gradient orientations. Following the typical LSD of none-parameter-turning features [16], Akinlar proposed a line segment detector based on edge drawing algorithm (EDLines) [7]. EDLines is about 10 times faster than LSD, while its accuracy just approximates LSD. Yang proposed a line segment detector using two-orthogonal direction image scanning (TODIS) [22], which achieves better detection accuracy compared to LSD, but it consumes about 8 times of the computational time than LSD.

(3) Algorithms using line geometrical properties and the relative positions of pixels. They extract the line segments by traversing along the edge pixels given by the edge detectors. As an example, the algorithm proposed in [20] tries to find the blurred lines in a grey level image and its prototype is reported in [19]. Although the algorithm detects segment accurately, it also generates lots of positive false [7, 17, 22].

Generally, algorithms faster than LSD such as EDLines suffer from lower accuracies; algorithms with higher accuracies than LSD such as TODIS consume more computation resources. In this paper, we attempt to explore a method that could preserve higher accuracy and higher speed, namely, faster than TODIS and more accurate than EDLines

Inspired by the strategy introduced in [18] and direction value processing method proposed in [23], we develop an algorithm called LSDSA maintaining statistical records about direction values found in steps of the traverse and it dynamically decides whether the current traverse should continue or cease in each step based on the records. Once LSDSA finds the traverse leads to a line whose direction values differ a lot from the traversed line segment, the traverse terminates and the line segment is stored. After all found segments are validated, the distorted ones are split to shorter segments based on a statistical computation of the records. Finally,

* ***Corresponding author*** e-mail: ngsun@cczu.edu.cn

LSDSA checks the possibility of linking the segments of the similar direction values by appropriately extending the segments.

Basically, LSDSA needs two parameters: sample size and the minimal length of a line segment. Since there is no record initially, we sample a number of pixels as the first parameter. The minimal length filters the found segments based on their lengths. Actually, we can combine these two parameters into one parameter and automatically determine it by the dimensions of the image space if necessary. The experiment shows that LSDSA runs at least 2 times faster than TOIDS and its accuracy is higher than EDLines.

The rest of this paper is organized as follows. Section 2 introduces the work related to LSD. Section 3 presents LSDSA. The experimental results are reported in in Section 4. Finally, we conclude the paper in Section 5.

2 Background

In this section, we focus on two issues related to LSDSA, namely, inner border tracing [23, 24] and foot-of-normal method [27].

2.1 INNER BORDER TRACING

The border tracing is used to find the inner border of a region in a binary image. Typical border tracing methods includes versions for 4-connectivity and 8-connectivity. They label the positions in the neighbourhoods of the different connectivity's by using *direction values* tied to the directions in the image plane. The direction values of 8-connectivity as shown in Fig. 1 reflect more directions than 4-connectivity.

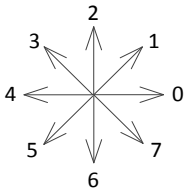

FIGURE 1 Direction values of 8-connectivity

The border tracing algorithm of 8-connetivity is shown in a diagram named activity diagram [25, 26] which satisfies the standards of Unified Modelling Language (UML) as shown in Fig. 2. The mod operations associated with the estimation of odevity of the *dir* yield values lying in a fixed range when *dir* can only be one of the values shown in Fig. 1. Hence there is a mapping between the input and output values of *dir*. The mapping can be represented by a matrix, i.e., a look-up table and the computations of the mod operations which actually are replaced by simple searches are accordingly reduced to $O(1)$.

The anti-clockwise search for the non-zero pixel is implemented by updating the variable *dir* after each check of a pixel in the neighbourhood, and stops once a non-zero pixel is found. The search will lead the centre of the searching to move to the found pixel and repeat. A continuous series of searches is called *tracing*. The tracing always starts at the first non-zero pixel in the upper left corner of the region and ends at the same pixel, no matter what shape of the checked region is used. Non-zero pixels of the inner border may be record more than once if the region is not closed, e.g., a one-pixel-width curve.

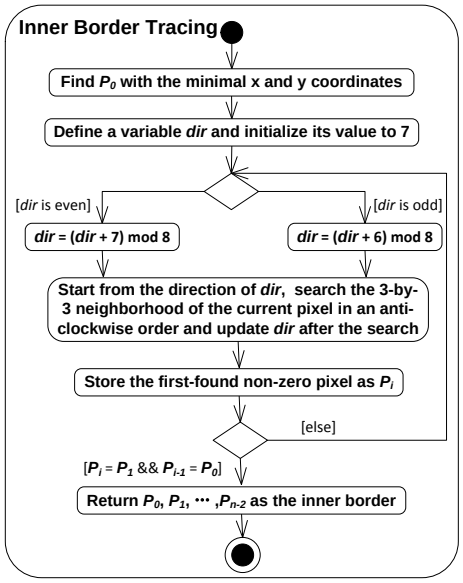

FIGURE 2 General structure of inner border tracing

2.2 INNER BORDER TRACING

The algorithms based on Hough transformation (HT) usually have high computation cost. Their most expensive step is the procedure called *voting* to change the values of points (cells) in the parameter space according to pixel coordinates s in the image foreground. Its computational complexity is $O(m \bullet n^2)$ where m denotes the degree of the parameter space discretization and n denotes one dimension of an image.

The foot-of-normal algorithm could reduce the computational complexity of the voting to $O(n^2)$ based on the fact that one line can only have one intersection with the normal which crosses the origin of the coordinate system. The intersections just are defined as votes as shown in Fig. 3.

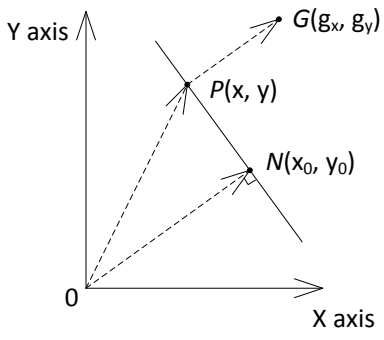

FIGURE 3 Foot-of-normal method

The origin is a fixed reference point. Assuming P is a point on the line and line $\overline{ON}$ is the normal of line $\overline{PN}$, from the facts that vector $\overrightarrow{ON}$ is perpendicular to vector $\overrightarrow{PN}$ and $\overrightarrow{ON}$ parallels to gradient vector (g_x, g_y) given by a Sobel operator, we get

$$g_x / g_y = y_0 / x_0 \text{ and } (x - x_0)x_0 + (y - y_0)y_0 = 0. \quad (1)$$

Solving the above formulae for x_0 and y_0, we obtain the formula of the voting point,

$$x_0 = v \cdot g_x,\ y_0 = v \cdot g_y, \text{ where } v = \frac{x \cdot g_x + y \cdot g_y}{g_x^2 + g_y^2} \quad (2)$$

Davies [27] analysed the line estimation error and found the image should be subdivided to reduce the error. The basic steps of the foot-of-normal method consist of subdividing the image, computing (x_0, y_0) in each sub-image whose origin is its centre instead of the upper-left corner, and finally making a vote at (x_0, y_0). Fig. 4 shows the voting results of a real-word image. The original image is divided to 20-by-20 and 50-by-50 sub-images indicated by white lines. The squares in the resulting images denote the voting points. Obviously, the more the image is subdivided, the more lines can be detected.

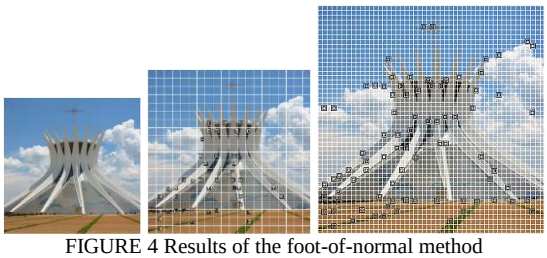

FIGURE 4 Results of the foot-of-normal method

3 The Proposed Algorithm

The basic idea of LSDSA is to dynamically link the pixels in the foreground of an image by using a statistical approach with low computational complexity. We develop a novel approach to represent the geometrical properties of line segments. Here, 8 infinite directions could be represented by direction values defined in section 2.1. The number of directions is determined by the resolution of the input image and the size of the neighbourhood employed. Among the neighbourhood of small sizes, a 3-by-3 neighbourhood reflects an adequate range of directions.

The general structure of LSDSA is shown in Fig. 5. It requires a pre-processing to generate the binary edge image based on the input image. The pre-processing is denoted by a solid rectangle labelled by the text "Detect Edge". Any edge detector can be employed in the pre-processing.

In Fig. 5, the largest dotted rectangle denotes the body of LSDSA. In this rectangle, three subroutines are designed to (1) find roughly straight-line segments, (2) find and break distorted line segments, and (3) link adjacent line segments with similar directions. They are represented by rectangles marked Subroutine 1, Subroutine 2 and Subroutine 3 in Fig. 5 and the sub steps of Subroutine 2 and Subroutine 3 are shown in their rectangles respectively.

When the detection of line segments is completed, the detected segments can be directly returned or processed by a post-processing to generate the global straight lines. As shown in Fig. 5, the two possible choices are denoted by two branches below the rectangle of the proposed algorithm. By combining the foot-of-normal method and the inverse HT algorithm [28, 29], the global lines can be obtained as the final result.

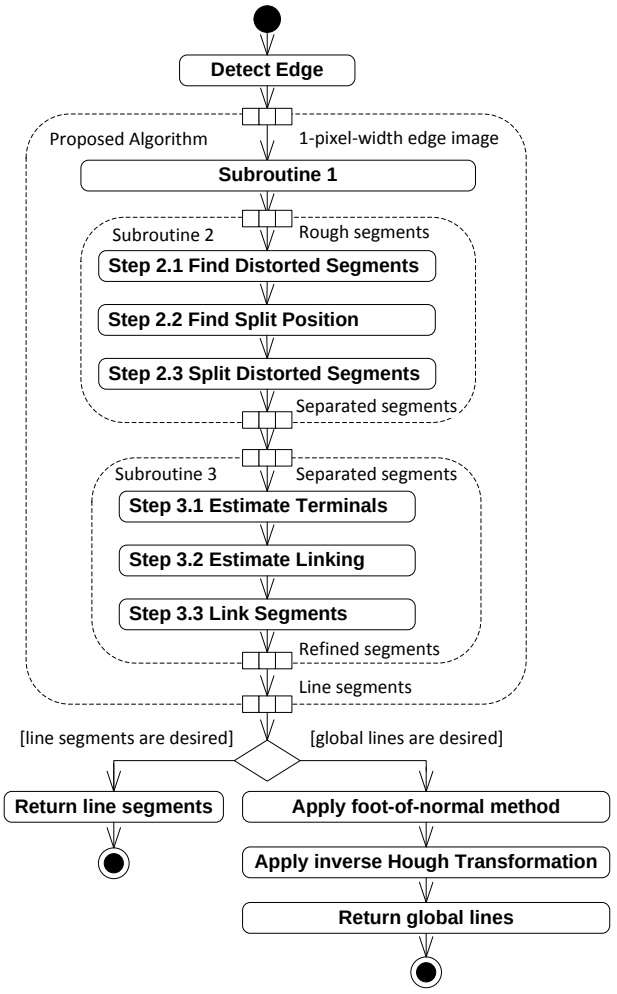

FIGURE 5 General procedure

3.1 FINDING ROUGHLY STRAIGHT LINES

Subroutine 1 performs a scan of the edge image and conditionally tracks each continuous curve (edge). It is used to rapidly find roughly straight-line segments without using any voting process. The following

subroutines after Subroutine 1 do not need to scan the whole image, and the size of the data is reduced to segments found by Subroutine 1. T the relatively sophisticated processing can be integrated to the following subroutines without drastically decreasing the speed of the algorithm. The details of Subroutine 1 are shown in Fig. 6.

In Fig. 6, *m* and *minLength* represent the sample size and the acceptable minimal length of the found line segment respectively, which are configured manually as input arguments. Variable *toleranceVal* is employed to distinguish the line segments of difference directions, e.g., if *toleranceVal* is set to 2, then direction values 0 and 1 are envisaged as representing the same direction because their difference is less than *toleranceVal*. The difference between two direction values is defined to be the smaller number of sectors between the direction values shown in Fig. 1. Namely, for direction values 0 and 3, the number of sectors can be anti-clockwise counted as 3 or clockwise counted as 5, and the difference is 3. The difference between two direction values such as *dir1* and *dir2* can also be expressed as

$$\text{if } |dir1 - dir2| > 4 \text{ then } difference = 8 - |dir1 - dir2|; \text{ else } difference = |dir1 - dir2| \quad (3)$$

When a non-zero pixel is found during Subroutine 1 scan, the scan is temporarily paused and the track operation is inserted. The track operation first tries to locate the right end of the curve. To locate the right end, the subroutine only checks the positions coinciding with direction values of 6, 7 and 0 of Fig. 1 in the neighbourhood of a non-zero pixel, moves the searching centre to it and repeat. The last-found pixel is envisaged as the right end of the line segment and the sampling starts at it. The searching procedure may split a line segment into several short segments, for example, if we apply the right-end searching to the right segment in Fig.7, it will stop before the pixel labelled 5 is reached because this pixel is in the direction 5 instead of 6, 7 or 0. This drawback may be fixed by merging segments in Subroutine 3.

Segments, *Directions* and *Occurences* in Fig.6 respectively are the set of found line segments, the set of the direction values and the set of statistical information of direction values associated with each segment. Variables *tempSegments*, *tempDirections* and *tempOccurences* contain the temporary data of *Segments*, *Directions* and *Occurences* respectively. The data of three temporary variables can be arbitrarily overwritten.

To illustrate these steps clearly, we depicts a simple binary image whose foreground contains two line segments in Fig. 7. Table 1 illustrates the result of its sampling procedure. The sampling procedure is denoted by the solid rectangle marked "Sample *m* pixels" in Fig. 6.

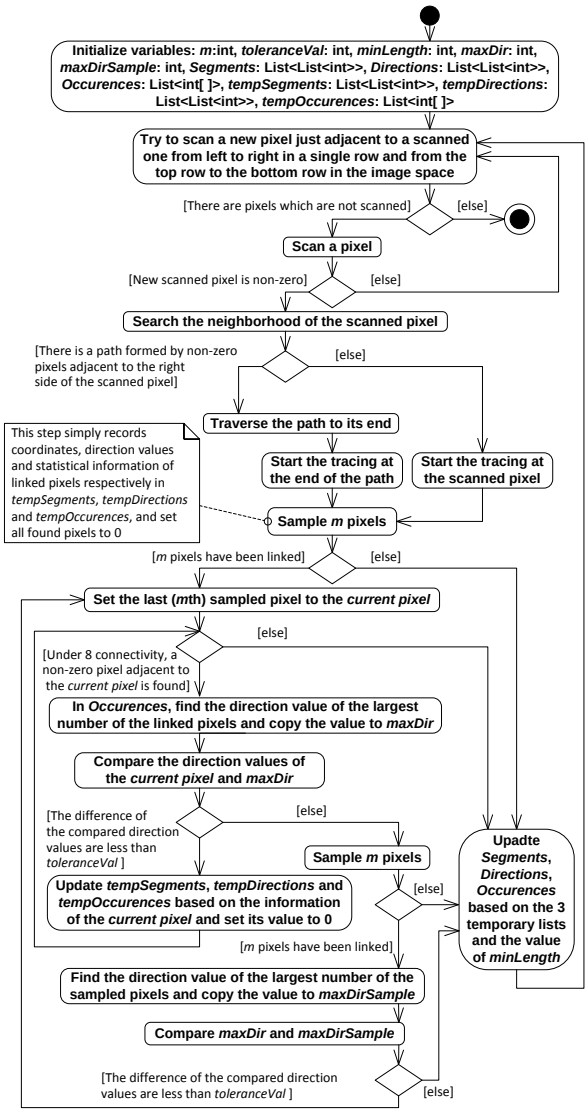

FIGURE 6 Details of Subroutine 1

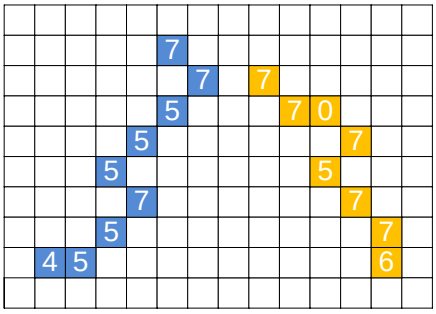

FIGURE 7 Representation of a binary image by lattices

TABLE 1 Values of *Segments*, *Directions* and *Occurrences*

Member	Value								
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Segments[0]	22	39	54	69	84	101	116	131	130
Segments[1]	41	58	59	76	91	108	125	141	Ø
Directions[0]	7	7	5	5	5	7	5	5	4
Directions[1]	7	7	0	7	5	7	7	6	Ø
Occurrences[0]	0	0	0	0	1	5	0	3	5
Occurrences[1]	1	0	0	0	0	1	1	5	7

3.2 DETECTING AND BREAKING DISTORTED LINES

In order to detect and break distorted lines, we launch three steps in Subroutine 2: (1) detect distorted segments; (2) determine positions of split; (3) splits distorted segments.

We detect distorted segments based on *Occurrences*. We find that the distribution of the direction values of a roughly straight line segments is similar with a uni-modal Gaussian distribution. Any multi-modal distribution implies the corresponding segment is distorted. Once the multi-modal is found, the difference between the direction values of the largest and the second largest modals is estimated. As shown in Fig. 8, we first detect the direction value of the second largest number of pixels which corresponds to a modal in the distribution in *Occurrences*, and then estimate the ratio and the difference of the maximal direction value and the found value which are respectively denoted by the variables *maxDir1* and *maxDir2*. Only when the difference exceeds *torleranceVal* and the ratio is larger than the parameter *thresholdRatio* whose value is experimentally set to 0.8, the corresponding segment is confirmed distorted.

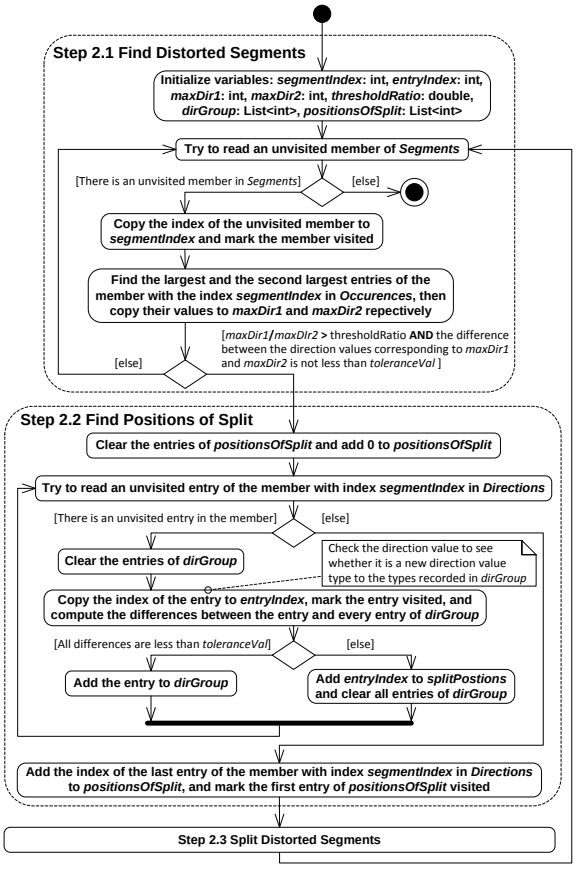

FIGURE 8 Details of the two steps of Subroutine 2

Next, in Step 2.2 we attempt to find the pixels where the main direction of a segment drastically changes. The strategy is assuming the segment is formed by several parts associated with specific direction value groups represented by *dirGroup* in Fig. 8. For each group, the differences among all elements are less than *toleranceVal*. Initially, *dirGroup* is empty and then the direction value of the first pixel encountered in the checking is added to it. The direction values of the following pixels are compared with the value in group. If the difference is adequate, the pixel is taken into the part, otherwise it is marked as a possible position in the segment to split and recorded by *positionsOfSplit* in Fig. 8.

Step 2.3 shown in Fig. 9 tries to split the segment according to *postionsOfSplit*. The lengths of different parts of a segment are compared with *minlength*. Only a segment with parts of lengths exceeding *minLength* is split and causes *Segments*, *Directions*, and *Occurences* to be updated. The updating procedure is denoted by four solid rectangles just above the thick horizontal bar in Fig. 9.

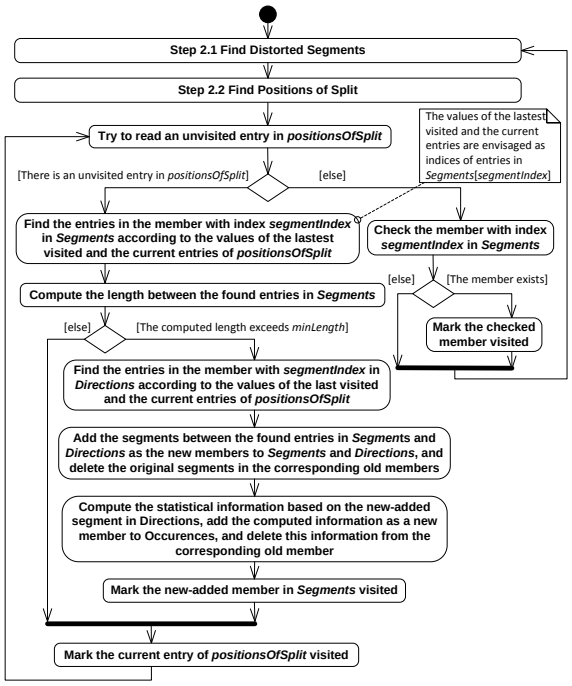

FIGURE 9 Details of the last step of Subroutine 2

3.3 LINKING ADJACENT LINES WITH SIMILAR DIRECTIONS

In Subroutine 3, we attempt to merge segments by appropriately extending the segments and comparing the directions with other segments found adjacent to the extensions. It composes three steps: Step 3.1 estimate terminals, Step 3.2 estimate linking and Step 3.3 link segments as shown in Fig. 5.

Step 3.1 locates the geometrical ends of segments for the subsequent extending operation. Although Subroutine 1 starts the traces from the right ends of segments, the first saved pixels may not be the right end because Subroutine 2 split segments and some of the first saved pixels of these segments are not the right end. The details

of finding geometrical ends are shown in Fig. 10. The ends can be easily found by estimating the directions of segments, i.e., if the maximal direction value is one of 5, 6, 7 and 0 (1 <= *maxDIrection* <= 4), the first recorded pixel is the geometrical right terminal; otherwise the first one is the left terminal.

After identifying all ends, we will estimate the path for extending in Step 3.2. Since the values of all entries in *Directions* range from 0 to 7 and these entries can reflect the geometrical shapes of segments basically, the entries may be envisaged as strings of letters 0 to 7 and their patterns can thus be found using approximating string matching [21]. Here we consider a simple but efficient strategy. We observe that the distribution of direction values around the middle point of a segment always follow a certain pattern. Therefore, we construct a path by repeatedly copying the middle direction values as shown in Fig. 11.

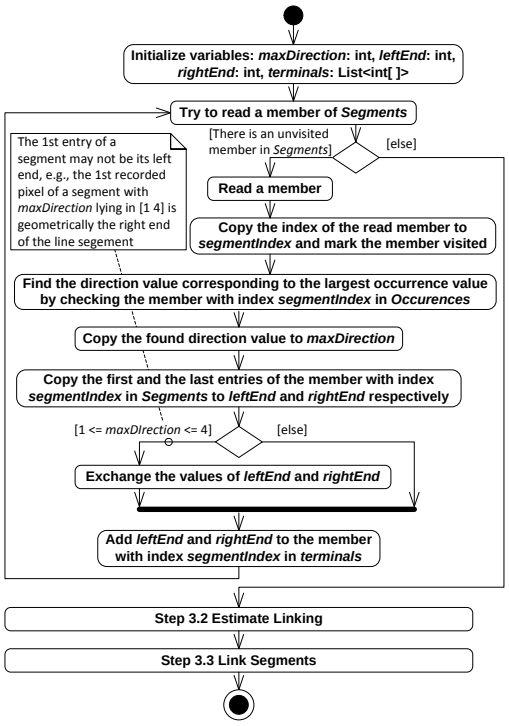

FIGURE 10 Details of Step 3.1 in Subroutine 3

Since Subroutine 1 traverses from one end of a line segment to the other, the recorded direction values reflects the pattern along the direction approximated by maximal direction value. Hence, the found path is only appropriate for one end, and inverses for the other. For instance, two red lattices in Fig. 12 indicate two ends; the colourful lattices between the ends denote the body of a line segment and the grey pixels form the two paths. The lattices of the segment except the paths are marked by their direction values obtained by tracing from the geometrical right terminal. The yellow pixel lies in the middle of the segment. The path adjacent to the left terminal, denoted by the variable *leftPath* in Fig. 11, is obtained directly by copying the direction values of lattices around the middle lattice. The path adjacent to the right terminal is obtained by reversing the direction values of the left path consisting of direction values 5, 3, 5, 4 and 4; the right path contains the corresponding inverse direction values 1, 7, 1, 0 and 0. The inverse path is denoted by *rightPath* in Fig. 11.

When moving along a path, we could check the neighbourhood of the moving centre to find other segments but it is computationally expensive. We employ an alternative approach by checking the ends of segments whose identities lying in a range with the identity of the current segment as centre and parameter *detectRadius* in Fig. 11 as a radius. This is because the difference between identities of two segments in *Segments* partially reflects their geometrical distance in image space. The *detectRadius* is set to 20 in our experiment.

After the connection information of all segments is collected by Step 3.2, Step 3.3 showed in Fig. 12 checks each member of *Connections* to perform the merging. If the statuses of ends indicate valid linking, it will simultaneously merge the associated segments with the current one and checks the statuses of the merged segments to see whether they can be further merged with other ones. This iterative procedure will stop until the statuses of both ends are null. For instance, segment 1, 3 and 4 will be merged together according to *Connections.*

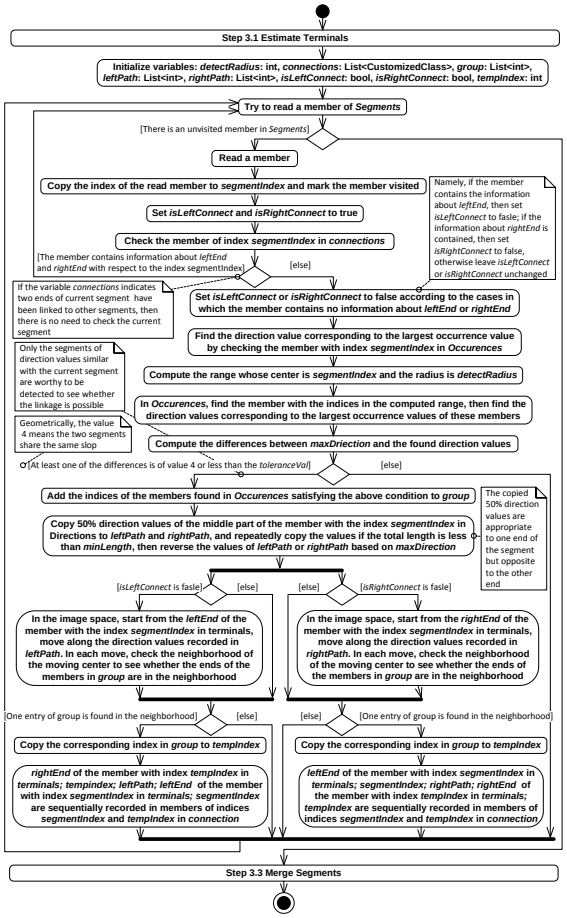

FIGURE 11 Details of Step 3.2 in Subroutine 3

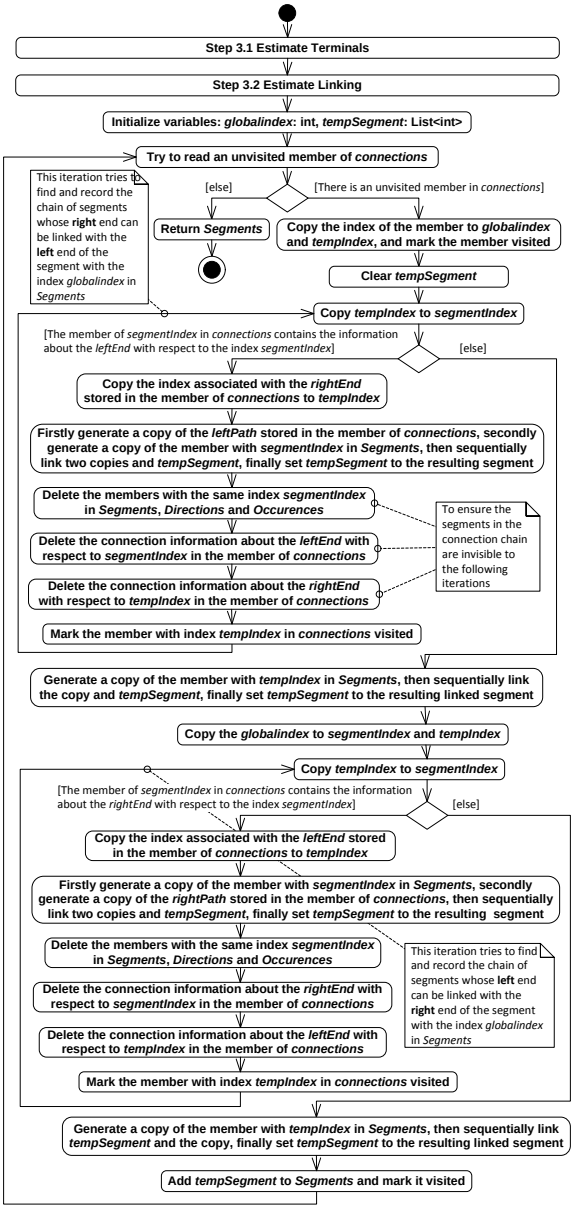

FIGURE 12 Details of Step 3.3 in Subroutine 3

4 Experimental Results and Discussion

In order to evaluate the performance of LSDSA, we test KHT [13], EDLines [7], TODIS [22] and LSDSA using two corpora of artificial and the real-world images. We developed an application package [30] for edge detection using C#. In general, LSDSA achieves performance similar to TODIS, which is a better result than KHT and EDLines. However, the computation cost of LSDSA is obviously lower than TODIS.

4.1 CASE COMPARISONS

Figure 13 lists some samples of line detections using LSDSA and KTH. We use colourful lines to display the segment results of LSDSA to distinguish the distinctive parts of a continuous curve. For example, the circle on the top of the building in the sub image labelled as H10 of Fig. 15 is indicated in H12. At the first glance, the indicated circle may seem to be a false detection. Actually, the detected circle in H12 is denoted by two kinds of colours (blue upper and purple lower). Thus, two roughly straight line segments are detected by the algorithm instead of a circle. The values of parameters *m* and *minLength* are set to 4 and 30 respectively according to the resolution of the test images.

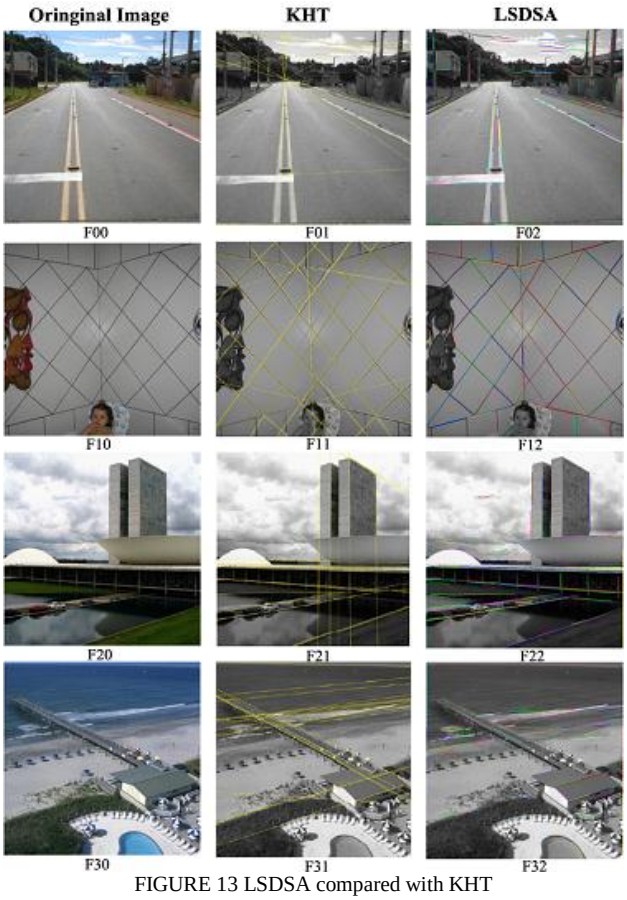

FIGURE 13 LSDSA compared with KHT

Fig. 14 compares the lines detected by LSDSA and EDLines. The parameters *m* and *minLength* of LSDSA are set to 3 and 8 respectively. All line segments detected by EDLines are marked by solid lines in the second column. Although the difference between LSDSA and EDLines is not as large as the difference between KHT and LSDSA, LSDSA can discover more significant line segments that are ignored by EDLines, such as the seams on the face of the building in G10 and the edges of the front doors of the house in G20. We also note that LSDSA can detect a line segment about the lady's right arm in G32, which is the border between the highlight and the lowlight areas of the arm surface, and its curvature continuously changes. However, EDLines fails to detect it.

Although TODIS is the slower than EDLines, KHT and LSD, it exhibits the best accuracy. Generally, LSDSA achieves accuracy as good as TODIS as shown in

Fig. 15. Almost all line segments detected by TODIS in H01 are indicated by LSDSA in H02 as well, except the broken vertical line on the left. Similar results are shown in H11 and H12. LSDSA successfully detects the traces in the right corner of H10 and the edges of the circle on the top of the building. Note, the detected edges of the circle are indicated by two colours (the blue upper/lower curves). The two curves approximate two roughly straight line segments. It says that LSDSA can detect roughly straight line segments, and further example can be found in G32 of Fig. 14. The detection of LSDSA is clearly more accurate than TODIS inasmuch as the edges of windows in the top of the skyscraper are detected in H22 but not in H21. The edges of the white circular lights on the top of the hallway are indicated in H32 and ignored by TODIS in H31. It suggests that LSDSA can detect the roughly straight line segments correctly.

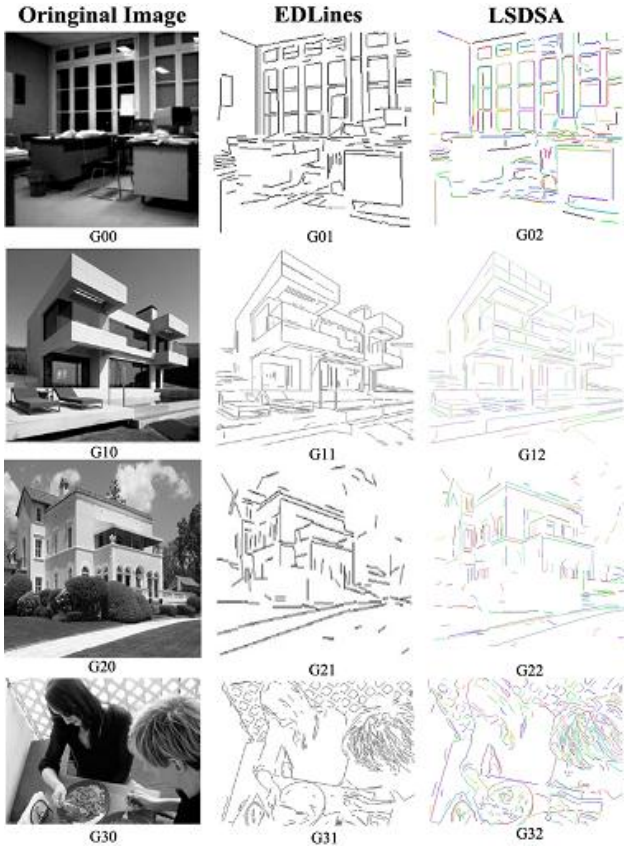

FIGURE 14 LSDSA compared with EDLines

4.2 COMPUTATIONAL COMPLEXITY

The computation cost of LSDSA is determined by Subroutine 1 for finding roughly straight lines. The time complexity of this part is $O(l \cdot s)$ where l is the largest length of a line segment and s denotes the number of segments. Unlike TODIS with $O(n^2)$ where n represents one dimension of the image space, LSDSA consume less time obviously when original images are large. This is because the first subroutine only performs the tracing when the current pixel is non-zero and set all traced pixels to 0.

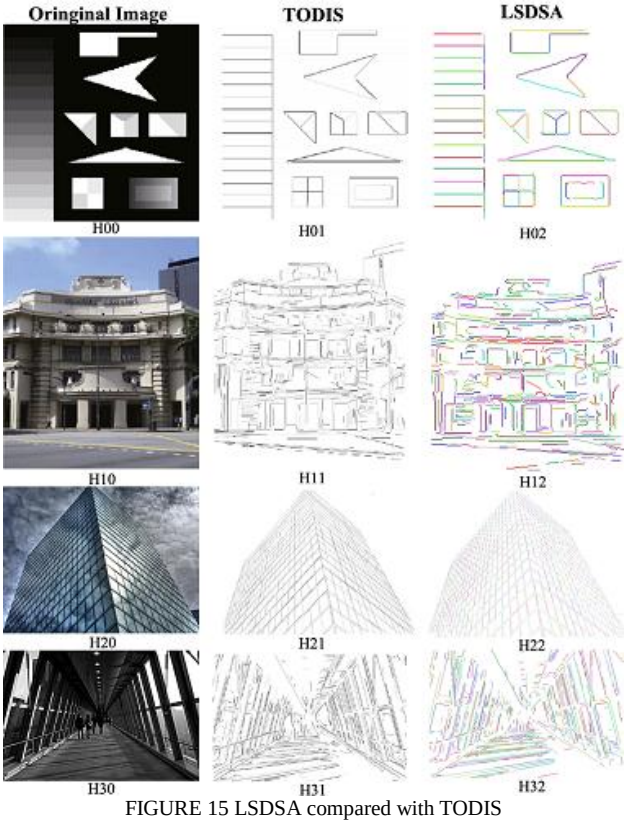

FIGURE 15 LSDSA compared with TODIS

Furthermore, we find a close relation between l and s. If the length of a line segment l is very large, then the number of segments s must decreases. Since the longer the segments are, the more space in the image they will occupy, and less space are left to the rest segments. It suggests that fewer segments can be released in this concise space. Conversely, if there are many segments, then their lengths tend to be short. Therefore, a balance exists between l and s.

The complexity of Subroutine 2 is $\text{Max}(O(s), O(l \cdot d))$ where d is the number of the distorted line segments found in this subroutine by performing a searching of complexity $O(s)$. Since d can't exceed s, the worst case of the second subroutine is $O(l \cdot s)$ as same as the first subroutine.

The third subroutine is of complexity $O(s)$.

Theoretically, the time complexity of the proposed algorithm is $O(l \cdot s)$, while the complexity of TODIS is at least $O(n^2)$ according to the analysis of the BU-Scan procedure [22], which contributes only a half of the computational cost of TODIS. Practically, we compare the time consumed by KHT, EDLines, LSDSA and TODIS under different conditions. The time consumed by algorithms do not consist the pre-processing as edge detection and post-processing as inverse Hough Transformation.

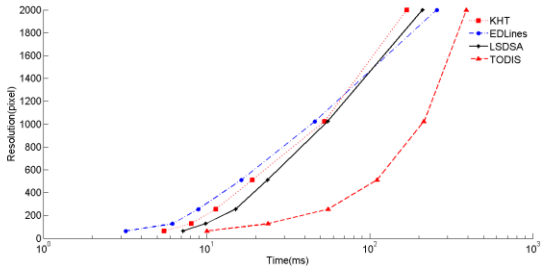

FIGURE 16 Time consumptions with respect to few lines

Fig. 16 shows time spent on processing the image containing few lines when the resolution ranges from 0-by-0 to 2000-by-2000 in pixel. Note the unit of x-axis is logarithm of the time, hence the crooked curve indicates a roughly straight line in a plane with non-logarithm axes. In Fig. 16, although TODIS consumes more time than other algorithms, its curve implies the ratio of resolution and time is linear (even the slop is very large). KHT, EDLines and LSDSA all share a similar pattern when resolution is lower than 1200 pixel in Fig. 16, but their curves lead to different destinations when the resolution reaches 2000 pixel. This implies their consumed time may be quite different when the input data becomes very large. In the level of 2000 pixel, LSDSA consumes less time than EDLines and TODIS. This illustrates the capability of LSDSA to reduce the dimension of input data.

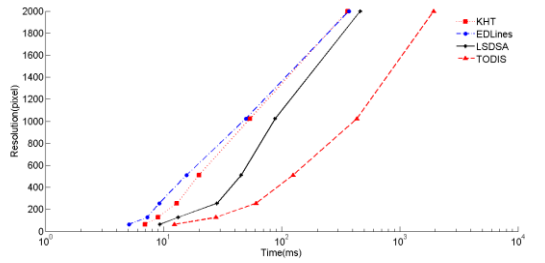

FIGURE 17 Time consumptions with respect to moderate lines

Fig. 17 shows the time consumed to process image moderate lines when resolution is changing, As in Fig. 16, TODIS apparently consumes more time than other algorithms, and EDLines and KHT share a similar pattern. The curve of LSDSA follows the pattern when resolution is higher than 1000 pixel, but still ends behind EDLines.

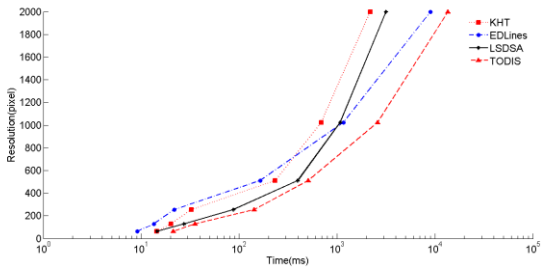

FIGURE 18 Time consumptions with respect to many lines

Fig. 18 illustrates the time when the processed image contains lots of lines. All algorithms even TODIS follows a similar pattern when resolution is lower than 600 pixel. As the resolution increases, EDLines and TODIS are exceeded by KHT and LSDSA. In level of 2000 pixel, there is a large gap between the group of algorithms with capability of reducing input dimension, i.e., KHT and LADSA, and the group of algorithm with no such capability, i.e., EDLines and TODIS.

Generally, when the processed image is small, there are little difference among the time consumed by KHT, LSDSA, EDLines and TODIS. As the resolution increases, the number of lines contained by image becomes an important factor affecting the consumed time. If lines are few, the difference may still remain small even when resolution increases. If lines are many, there will be an obvious gap between the time consumed by algorithms with or without capability of reducing input dimension. Hence, the feature of LSDSA illustrated by Fig. 16 to Fig. 18 is the capability of fast processing images with high resolution and complicated content.

5 Conclusions and Future Work

In our proposed method, LSDSA employs a statistical tracing strategy to reduce the dimension of the input data and distinguish the distorted segments by analysing the distributions of direction values which are approximate quantified values of geometrical directions in image space. LSDSA collects statistical data of quantified directions, so it is able to achieve higher speed under the condition of good accuracy. We report the experiment results of LSDSA using test images and industrial images, and compare its performance with typical LSD algorithms such as KHT, EDLines and TODIS. It indicates that the accuracy of LSDSA is clearly better than KHT and EDLines, and it is as good as TODIS. But LSDSA consumes much lower computation cost than TODIS does.

In order to further increase the accuracy of LSDSA, we plan to explore more refined direction values and large neighbourhood in the following research. At the same time, we will investigate how to simplify its processing procedure and proposed algorithm.

Acknowledgments

This work is supported by National Natural Science Foundation of China under Grant 61103172. The authors would like to thank Geyong Min of Bradford University in UK and Shiren Ye of Changzhou University in China for their comments and insightful suggestions.

References

[1] Mahadevan S, Casasent D P 2001 Detection of triple junction parameters in microscope images *Proc. SPIE* **4387** 204-14

[2] Kahn P, Kitchen L, Rieseman E M 1990 A fast line finder for vision-guided robot navigation *IEEE Trans. Pattern Anal. Mach. Intell.* **12**(11) 1098–102

[3] Tupin F, Maitre H, Mangin J F, Nicolas J M, Pechersky E 1998 Detection of linear features in SAR images: application to the road network extraction *IEEE Trans. Geosci. Remote Sens.* **36**(2) 434–53

[4] Zhu Y, Carragher B, Kriegman D J, Milligan R A, Potter C S, et al 2001 Automated identification of filaments in cryo-electron microscopy images *Journal of Structural Biology* **135**(3) 302–12

[5] Yu X, Lai H C, Liu S X F, Leong H W 2005 A gridding Hough transform for detecting the straight lines in sports videos *In Proc. Int. Conf. on Multimedia and Expo., Amsterdam, The Netherlands, 6-8 Jul. 2005* 45–8

[6] Zheng Y, Li H, Doerman D 2005 A parallel-line detection algorithm based on HMM decoding *IEEE Trans. Pattern Anal. Mach. Intell.* **27**(5) 777-92

[7] Akinlar C, Topal C 2011 EDLines: A real-time line segment detector with a false detection control *Pattern Recognition Letters* **32**(13) 1633-42

[8] Hough P V C 1962 *Method and Means for Recognizing Complex Patterns* U.S. Patent No. 3069654

[9] Aguado E, Montiel A S, Nixon M S 2000 On the intimate relationship between the principle of duality and the Hough transform *Proc. Roy. Soc. A* **456**(1995) 503-26

[10] Wright M, Fitzgibbon A, Giblin P J, Fisher R B 1996 Convex Hulls, Occluding Contours, Aspect Graphs and the Hough Transform *Image Vision Comput.* **14**(8) 627-34

[11] Bhattacharya A, Rosenfeld A, Weiss I 2003 Geometric and Algebraic Properties of Point-to-line Mappings *Pattern Recogn.* **36**(2) 483-503

[12] Bhattacharya A, Rosenfeld A, Weiss I 2002 Point-to-line Mappings as Hough Transforms *Pattern Recogn. Lett.* **23**(14) 1705-10

[13] Fernandes L A F, Oliveira M M 2008 Real-time Line Detection Through an Improved Hough Transform Voting Scheme *Pattern Recognition* **41**(1) 299-314

[14] Pope A R, Lowe D G 1994 Vista: A software environment for computer vision research *In Proceedings of Computer Vision and Pattern Recognition, Seattle, WA, USA* 768–72

[15] Lowe D G 1987 Three-dimensional object recognition from single two-dimensional images *Artificial Intelligence* **31** 355–95

[16] Burns J B, Hanson A R, Riseman E M 1986 Extracting Straight Lines *IEEE Trans. Pattern Anal. Mach. Intell.* **8**(4) 425-55

[17] von Gioi R G, Jakubowicz J, Morel J M, Randall G 2010 LSD: A fast line segment detector with a false detection control *IEEE Trans. Pattern Anal. Mach. Intell.* **32**(4) 722-32

[18] Etemadi A 1992 Robust Segmentation of Edge Data *In Proc. Int. Conf. on Image Processing and Its Applications, Maastricht, the Netherlands, 7-9 Apr 1992* 311–4

[19] Debled-Rennessona I, Feschet F, Rouyer-Degli J 2006 Optimal Blurred Segments Decomposition of Noisy Shapes in Linear Time *Computers & Graphics* **30**(1) 30-6

[20] Kerautret B, Even P 2009 Blurred Segments in Grey Level Images for Interactive Line Extraction *In Proc. Int. Conf. on Combinatorial Image Analysis, Playa del Carmen, Mexico, 24-27 Nov. 2009* 176-86

[21] Skiena S S 2008 *Set and String Problems* In The Algorithm Design Manual, 2nd ed. NY: Springer, Ch. 18, sec. 4, 631-6

[22] Yang K, Ge S S, He H 2011 Robust Line Detection Using Two-orthogonal Direction Image Scanning *Computer Vision and Image Understanding* **115**(8) 1207-22

[23] Shapiro V 2006 Accuracy of the Straight Line Hough Transform: The non-voting approach *Computer Vision and Image Understanding* **103**(1) 1-21

[24] Sonka M, Hlavac V, Boyle R 2008 *Image Processing, Analysis, and Machine Vision* 3rd ed. CT, USA: Cengage Learning

[25] Jacobson I, Booch G, Rumbaugh J 2005 *Unified Modelling Language User Guide* 2nd ed. MA, USA: Addison-Wesley

[26] Rumbaugh J, Jacobson I, Booch G 2010 *Unified Modelling Language Reference Manual* 2nd ed. MA, USA: Addison-Wesley

[27] Davies E R 2005 *Machine Vision: Theory, Algorithms, Practicalities* 3rd ed. CA, USA: Morgan Kaufmann

[28] Anastasios L, Kesidis A L, Papamarkos N 1999 On the inverse Hough transform *IEEE Trans. Pattern Anal. Mach. Intell.* **21**(12) 1329-43

[29] Kenzie D S, Protheroe S R 1990 Curve description using the inverse Hough transform *Pattern Recogn.* **23**(3-4) 283-90

[30] Jia L, Sun Y, Wang M, Gu Y 2011 A Research on Implementation of Image Scattergram by Using C# *In Proc. Int. Conf. on System Design and Data Processing, Tai Yuan, China, 26-28 Feb. 2011* 353-5

Authors

Liang Jia

Current position, grades: is a faculty member at Changzhou university of Jiangsu province in China.
University studies: MS degree in computer science from the Nanjing university of Science and Technology in 2009 and the BS degree in computer science from Beifang university of Nationalities in 2004.
Scientific interests: image processing, computer vision, service-oriented software development.

Nigang Sun

Current position, grades: joined East China University of Science and Technology in 2007 and Changzhou University in 2010. Now he is an associate professor in the department of the Computer Science and Technology.
University studies: Ph.D. degree in information security, University of Chinese Academy of Sciences, 2007.
Scientific interests: formal methods, system modelling and analysis, information security.