CVGIP: IMAGE UNDERSTANDING
Vol. 57, No. 2, March, pp. 131–154, 1993

Randomized Hough Transform (RHT): Basic Mechanisms, Algorithms, and Computational Complexities*

LEI XU†

Lappeenranta University of Technology, Department of Information Technology, Box 20, 53851 Lappeenranta, Finland; Department of Mathematics, Peking University, Beijing, Peoples' Republic of China

AND

ERKKI OJA

Lappeenranta University of Technology, Department of Information Technology, Box 20, 53851 Lappeenranta, Finland

Received February 13, 1990; accepted May 14, 1992

Recently, a new curve detection approach called the randomized Hough transform (RHT) was heuristically proposed by the authors, inspired by the efforts of using neural computation learning techniques for curve detection. The preliminary experimental results and some qualitative analysis showed that in comparison with the Hough transform (HT) and its variants, the RHT has advantages of fast speed, small storage, infinite range of the parameter space, and high parameter resolution, and it can overcome several difficulties encountered with the HT methods. In this paper, the basic ideas of RHT are further developed into a more systematic and theoretically supported new method for curve detection. The fundamental framework and the main components of this method are elaborated. The advantages of RHT are further confirmed. The basic mechanisms behind these advantages are exposed by both theoretical analysis and detailed experimental demonstrations. The main differences between RHT and some related techniques are elucidated. This paper also proposes several improved algorithms for implementing RHT for curve detection problems in noisy images. They are tested by experiments on images with various kinds of strong noise. The results show that the advantages of RHT are quite robust. Moreover, the implementations of these algorithms are modeled by a generalized Bernoulli process, allowing probability analysis on these algorithms to estimate their computational complexities and to decide some important parameters for their implementations. It is shown quantitatively that the complexities are considerably smaller than those of the HT.

*This work was initialized and partly done at Lappeenranta University of Technology, supported by Tekes Grant 4196 under the Finsoft project (Finland).

† Present address: Division of Applied Sciences, G-14 Pierce Hall, Harvard University, Cambridge, MA 02138.

1. INTRODUCTION

The Hough transform (HT) was proposed by Hough in 1962 [6] and was brought to the attention of the mainstream image processing community by Rosenfeld [7]. Duda and Hart [8] introduced the polar parameterization to HT, which makes HT more efficient for line detection; they also demonstrated circle detection. Kimme, Ballard, and Sklansky [9] made circular curve detection significantly more effective by using the gradient information of local pixels. Furthermore, Merlin and Faber [10] showed how the HT could be generalized to detect an arbitrary shape at a given orientation and a given scale, and Ballard [11] eventually generalized the HT to detect curves of a given arbitrary shape for any orientation or any scale by making use of the gradient information of local pixels. Up to now, hundreds of papers have been published on the issues related to HT. These issues include a lot of the applications, variants, and extensions of HT. A recent comprehensive survey on these developments of HT was published by Illingworth and Kittler [3]. These issues also involve many further theoretical studies on various aspects (especially on noise sensitivity) of both HT and the generalized HT [11]. Some of these studies can also be found in [3]. The further interesting studies on the noise sensitivity of the generalized HT were provided recently by Grimson and Huttenlocher [22, 23] and a number of earlier studies on the noise sensitivity of the generalized HT were given in the references provided by [22, 23].

Recently, when extending the Kohonen self-organizing neural network for curve detection [1] and inspired by its fundamental principles, the authors proposed a basic idea

1049-9660/93 $5.00

which introduces a random sampling mechanism and a converging mapping mechanism into the conventional HT methods. In [2], the idea was developed further and a novel HT-like approach was proposed, called the randomized Hough transform, RHT, for detecting curves from a binary image. Such images are usually obtained from grey-level images by conventional edge-detection techniques (e.g., the Canny operator). In comparison with the conventional HT and its variants as well as extensions (which are all deterministic methods), the preliminary experiments and some qualitative analysis in [2] showed that the RHT has several advantages like fast speed, small storage requirements, infinite scope of the parameter space, and high parameter resolution, and it can overcome some major difficulties (see Section 1 in [2]) met by the conventional HT methods.

In this paper, the basic ideas of RHT are further developed into a more systematic and theoretically supported new method for curve detection. The fundamental framework and the main components of this model are further elaborated. The deep mechanisms behind the above-mentioned advantages of RHT are revealed by both theoretical analysis and detailed experimental demonstrations. It is shown that these advantages are due to the consistent combinations of *random sampling, converging mapping, score accumulation, and stepwise implementation* and the appropriate selection of *accumulator structure*. Especially the first two of these play key roles and are most responsible for the effectiveness and novelty of RHT.

Moreover, we propose several improved algorithms for implementing RHT for curve detection problems in noisy images. It is shown by several experiments as well as an application example of form processing in document analysis that these algorithms can robustly maintain the advantages of RHT on images with strong random noise, quantization errors, and non-random interference pixels. Furthermore, the implementations of these algorithms are modeled by the *generalized Bernoulli process*, and probability analysis on these algorithms is given for estimating their computational complexities and deciding some important parameters for their implementations. As an example of the complexity results, it is shown that RHT can have the time or/and storage complexity much lower than an upper bound of order $O(n_t N^n / n_{\min}^n)$, considerably smaller than $O(N N_a^{n-1})$ and $O(N_a^n)$, which are the time and storage complexities of HT. There N, N_a are the sizes of the image and accumulation arrays, respectively, $n_{\min}$ is the length of the shortest curve in the image, and n_t is a small number. An important feature is that the complexities of RHT are dependent on the complexity of the image. For simple images, RHT algorithms can perform the tasks with very fast speed and small storage. In contrast, even for the simplest image, containing only one line without noise, HT still needs the complexities of $O(N N_a^{n-1})$ and $O(N_a^n)$ (where $n = 2$) to find it, while RHT only needs the time complexity of $2 \sim 3$ and the storage complexity of 1.

The contents of the sections are as follows. In Section 2, we will first give a refined description of the key ideas of RHT given in [2]. The fundamental framework and the main components of this method will be further elaborated. Then in Section 3, through theoretical analysis as well as experimental demonstrations, we will reveal each of the basic mechanisms which give RHT several favorable advantages. The main differences between RHT and some related techniques, e.g., the technique of *trading off work in parameter space for work in image space* [11, 13] and RANSAC [14], will also be elucidated. In Section 4 and Appendix A, several algorithms of RHT (including the earlier one given in [2] and some recent improvements [12]) are summarized in more concise forms, and the characteristics of each version are discussed. In addition, we will also show experimental results of an application example in document analysis. In Section 5 and Appendix B, we will model the implementation of the RHT algorithms by the generalized Bernoulli process. The algorithms are analyzed probabilistically to estimate the computational complexities and to decide the proper values of some important parameters in their implementations. Finally, we make some conclusions in Section 6.

2. THE BASIC IDEAS OF RHT METHODS

2.1. *The Curve Detection Problem*

The task of curve detection is conducted on a binary edge image which may be obtained from grey-level images by either simple thresholding operations or by some standard edge detection techniques (e.g., the Canny operator). Suppose the curves to be detected can be expressed by a parametric function $f(\mathbf{a}, \mathbf{d}) = 0$ with $\mathbf{a} = [\alpha_1, \ldots, \alpha_n]^t$ containing n independent parameters and $\mathbf{d} = \mathbf{d}(x, y)$ being the coordinates of a "white" pixel.[1] Roughly speaking, the task of curve detection is to solve the parameters of every curve expressed by this function.

Since n pixels may define a solution of $f(\mathbf{a}, \mathbf{d}) = 0$ (e.g., any two different pixels define a line and any three not collinear pixels define a circle), an image with N pixels will have at most C_N^n (the factorial n over N) curves expressible by the parametric function. However, many such curves may have only a few pixels and usually are not regarded as a real curve in practice. That is, a parameter vector $\mathbf{a}$ is not enough for defining a real curve. Some other parameters such as the total number of pixels lying

[1] A white pixel means a pixel that has binary value 1; in the rest of this paper, we call a white pixel simply a pixel.

on a curve and/or the largest length of connected segments of a curve are needed to define a real curve. Moreover, in a digital image from real applications, the pixels are usually not exactly located on a curve because of sensor noise and quantization. Instead, pixels are located in the digital neighborhood areas of a curve.

In order to make the presentation of RHT as precise as possible and to facilitate the latter theoretical analysis, we first formalize the curve detection problem by the following definitions:

DEFINITION 1a. Assume as given a curve c_i: $f(\mathbf{a}_i, \mathbf{d}) = 0$. As shown in Fig. 1(a), a δ-*band* of c_i is defined as the following subset of R^2: $B(c_i) = \{(x, y) | r_{\min}(x, y) \leq \delta\}$, where $\delta \geq 0$ and $r_{\min}(x, y)$ is the minimum of $\sqrt{(x-u)^2 + (y-v)^2}$ subject to $f(\mathbf{a}_i, \mathbf{d}(u, v)) = 0$.

DEFINITION 1b. Assume as given a number $m_{\min} > n$ (n is the number of curve parameters) and a predefined value of δ. A curve c_i: $f(\mathbf{a}_i, \mathbf{d}) = 0$, as shown in Fig. 1b, is called a *true curve* under error δ if there are $n_i \geq m_{\min}$ pixels $\{\mathbf{d}_1, \mathbf{d}_2, \ldots \mathbf{d}_{n_i}\} \subseteq B(c_i)$ (i.e., the n_i pixels fall in the δ-band of c_i), and a *pseudo curve* when $n_i < m_{\min}$. We also call n_i the *length* of c_i. Furthermore, in the special case of $\delta = 0$, we say that true curves and pseudo curves are ideal, and we call them *ideal true curves* and *ideal pseudo curves*, respectively.

DEFINITION 1c. Assume as given a specific curve function $f(\mathbf{a}, \mathbf{d}) = 0$, $\mathbf{a} \in R^n$, and a set S_{nc} of m_{nc} pixels with $m_{nc} \geq n$. If there is no n-tuple of pixels in S_{nc} among all the $C^n_{m_{nc}}$ n-tuples, giving an $\mathbf{a}' \in R^n$ such that its n pixels fall in a given δ-band of the curve $f(\mathbf{a}', \mathbf{d}) = 0$, then we call the pixels of S_{nc} *non-curve pixels* under error δ. In the special case of $\delta = 0$, we call the pixels in S_{nc} ideal *non-curve pixels*.

(Note: When the curve function expresses a line, the set of non-curve pixels S_{nc} is always empty. However, for a curve function with more than two independent parameters, S_{nc} may be non-empty. E.g., for the curve function expressing a circle, if there are no other pixels in an image except a set S_l which consists of pixels located on a line, then $S_{nc} = S_l$. We emphasize that non-curve pixels are defined subject to a specific curve function. For the just-mentioned example, the pixels of S_l are not non-curve pixels subject to the curve function of a line.)

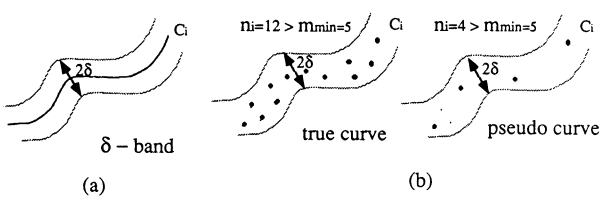

FIG. 1. True curves and pseudo curves: (a) A δ band of curve c_i is a band of width 2δ with c_i as the median axis. (b) A true curve c_i has at least $m_{\min}$ pixels falling in a given δ band; otherwise c_i is a pseudo curve.

DEFINITION 2. Assume as given a specific curve function $f(\mathbf{a}, \mathbf{d}) = 0$, $\mathbf{a} \in R^n$, and the predefined values of δ and $m_{\min}$. Subject to the given curve function, a *model binary picture* (MBP) is defined as an N-pixel binary image which contains m_t *true curves* c_i, $i = 1, \ldots, m_t$ and m_{ps} *pseudo curves* c_i, $i = m_t + 1, \ldots, m_t + m_{ps}$, as well as a set S_{nc} of *non-curve pixels* under the given error δ, where $m_t \geq 1$, $m_{ps} \geq 0$, and S_{nc} is possibly a null set. Furthermore, for the special case that $\delta = 0$, i.e., the true curves, pseudo curves, as well as non-curve pixels, are ideal, we call an MBP an *ideal model binary picture* (IMBP) subject to the curve function.

DEFINITION 3. Assume as given a binary image, a specific curve function $f(\mathbf{a}, \mathbf{d}) = 0$, $\mathbf{a} \in R^n$, as well as the predefined values of δ and $m_{\min}$. The whole task of curve detection consists of the following three interwoven parts:

(1) Find out the number m_t, i.e., how many true curves there are in the image.

(2) Solve the parameter vector $\mathbf{a}_i$ of each c_i of the m_t true curves under the given error δ.[2]

(3) For each pixel in the image, either classify it into one of the m_t true curves or regard it as a *non-true-curve pixel*.

By the above definitions, and given a specific curve function, any binary image encountered in practical curve detection problems can be modeled using the MBP model. However, it follows from the above definition that the task of curve detection is only a partial task of modelling a binary image into an MBP. In the sequel, we use the MBP model to guide our design of the implementing algorithms of RHT as well as our analysis of RHT. Especially, the IMBP model which is the ideal version of MBP will be used later to facilitate our theoretical analysis on the performance of RHT.

2.2. *The Basic Ideas of RHT*

Assume now that we have a binary image and we have decided on a specific curve function $f(\mathbf{a}, \mathbf{d}) = 0$, $\mathbf{a} \in R^n$ of the type that we want to detect (e.g., a line or a circle). Assume also that there are predefined values of δ, $m_{\min}$. A description of the basic principles of the RHT method is given in the following. In comparison with the earlier description of the basic ideas given in [2], there are some differences in the present one. The earlier one in [2] gave only one specific version of RHT, while the present one is more general.

[2] In the sequel, we will omit "under the given error δ" in places where no confusion occurs.

• The implementation of RHT is a series of simple *random trials*. The whole series consists of a number of *epochs*. Each epoch contains one or several *accumulation periods*, and each accumulation period consists of a certain number of random trials.

• At each random trial, n pixels $\mathbf{d}_i = d(x_i, y_i)$, $i = 1, \ldots, n$, are *randomly sampled* from the image with each pixel being picked with equal probability, and then are mapped using a *converging mapping* into *one point* $\mathbf{a} \in R^n$ in the parameter space by solving the set of n equations $f(\mathbf{a}, \mathbf{d}_i) = 0$, $i = 1, \ldots, n$. The candidate $\mathbf{a}$ is put into a *storage* P in the following way: if there exists an element $p \in P$, $p = [\mathbf{param}(p), \text{score}(p)]$ with its $\mathbf{param}(p)$ being regarded as the same as $\mathbf{a}$ subject to a given *error criterion*, then we let $\text{score}(p) := \text{score}(p) + 1$; while if none exits, we insert into P a new element $p = [\mathbf{param}(p), \text{score}(p)] = [\mathbf{a}, 1]$.

• An accumulation period is finished after a *sufficient number of trials*. Then among all the elements which have been accumulated in P during this period, one or several p_i, $i = 1, \ldots, r$ are taken as candidate curves if the $\text{score}(p_i)$ of each p_i can be regarded as *large enough* in some way (e.g., by checking if $\text{score}(p_i)$ is either a local maximum or among one or several global maxima).

• For each candidate p_i, it is *verified* whether $\mathbf{param}(p_i)$ represents a true curve by Definition 1(b); if yes, this $\mathbf{param}(p_i)$, together with those pixels falling in its given δ-band, are decided as the solution of one true curve; if not, p_i is discarded. If all the candidates are discarded, another accumulation period is undertaken using the present P after removing from it all these candidates. Otherwise, we say an epoch is completed.

• After a completed epoch, P is reset to *null*, all the pixels of the detected curves are removed from the image, and then a new epoch is started.

• The whole series is stopped when it is known that there is no true curve still remaining in the image.

The core of RHT is the combined use of *random sampling* in the image space, *score accumulation* in the parameter space, and *converging mapping* as the bridge between the two spaces. Around this core, some issues can be implemented in different ways, which results in several variants of RHT with different features. E.g., the earlier version given in [2] gives significant advantages when used on a low-noise image; the revised versions given in [12] are more suitable for applications with various types of noise although they may occupy more storage in less noisy applications. In the above descriptions, we try to include some of these versions (but not completely). Thus, some terms and issues appearing in the above description have to be further interpreted. We give such interpretations in the sequel.

(1) *Random sampling*. Although one can sample randomly n pixels directly from an image, a more efficient way is that at the very beginning all the "white pixels" of the image are scanned into a data set D. Each pixel being picked with equal probability means then the following: at each trial one randomly picks n points $d_1, \ldots, d_n$ out of D in such a way that all points of D have an equal probability to be taken as $d_1, \ldots, d_n$ out of D in such a way that all points of D have an equal probability to be taken as d_1, then all points of $D - \{d_1\}$ have an equal probability to be taken as $d_2, \ldots$, etc.; finally, all points of $D - \{d_1, d_2, \ldots, d_{n-1}\}$ have an equal probability to be taken as d_n.

(2) *Converging mapping*. To efficiently realize the *converging mapping*, we hope that the parameterization of a curve (i.e., the given curve function) is such that the n equations $f(\mathbf{a}, \mathbf{d}_i) = 0$, $i = 1, \ldots, n$, are analytically solvable. As indicated in [2], all curves with the following expression are linearly solvable;

$$\alpha_1 z_1 + \alpha_2 z_2 + \cdots + \alpha_n z_n + c = 0, \tag{1a}$$

where z_i, $i = 0, \ldots, n$, only depend on the coordinates x, y of a pixel, and c is constant. That is, the expression is linear with respect to the parameters. All the usual curves such as a line and all quadratic curves can be expressed in the form of Eq. (1a) (see [2]). Moreover, a line with polar parameterization

$$\rho = x \cos(\theta) + y \sin(\theta) \tag{1b}$$

is also easily solvable; given two pixels $d_1 = d(x_1, y_1)$, $d_2 = d(x_2, y_2)$, the solution can be directly obtained by[3]

$$\theta = \arctan\left(\frac{x_1 - x_2}{y_2 - y_1}\right), \quad \rho = x_i \sin(\theta) + y_i \cos(\theta), \quad i = 1 \text{ or } 2. \tag{1c}$$

In [2], it was also shown that a circle with the parameterization $(x - \alpha_1)^2 + (y - \alpha_2)^2 = \alpha_3^2$ can be solved through two coupled linear equations.

However, it may be that a unique solution does not exist. One possibility is that there exist several solutions like in circle detection indicated in [2]. In such a case we can just store each of them into P in the same way as storing a unique solution. Another possibility is that there exists no solution (e.g., three collinear pixels are picked for circle detection) or there exist an infinite number of solutions (e.g., when the n pixels happen to make the n equations not independent). In these cases, we simply discard the n pixels and randomly re-sample another set of n pixels.

[3] In a noisy image, it is better to use $\rho = ((x_1 + x_2)/2) \sin(\theta) + ((y_1 + y_2)/2) \cos(\theta)$. To avoid computing $\cos(\theta)$, $\sin(\theta)$, one directly computes $\rho = 0.5\,(((x_2 - x_1)(y_1 + y_2) - (y_2 - y_1)(x_1 + x_2))/\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2})$.

(3) *Different types of storage* P. Theoretically, P could be a non-ordered set. However, practically, P should be such that its elements can be effectively accessed. There are several types of P that can be used. First, P can be the usual accumulation array used in the HT methods [3]. In this case, before any sampling, all the elements of P are placed at the bins of a discretized hypercube window in the parameter space, and the scores of all the elements are set to zero. When putting a candidate **a** into P, we only need to discretize **a** by the same quantization into the coordinates of a bin p and let score(p) := score(p) + 1. Now storing **param**(p) is not necessary since it is implicitly represented by the coordinates or indices of each bin. Second, as used in [2], P can be either a dynamic linear list or tree structure. When a list is used, the search is made simply from the beginning of the list to the end of the list, and each new element p = [**param**(p), score(p)] = [**a**, 1] is appended at the end of the list. A tree structure allows ordering of the elements by the coordinates of **a**, and some management is needed for the operations of searching and inserting. Third, as suggested in [12], P can also have a structure which combines a *hashing table* and linear list.

The different types of P have different characteristics which produce both advantages and disadvantages, as well be discussed in the sequent subsections. Especially in Sections 3.4 and 4.1, we will further discuss P in detail.

(4) *The error criterion for regarding two elements being the same.* When P is the usual accumulation array, two elements are regarded as the same as they are discretized into the same coordinates. Now the error criterion is implicitly in effect through the quantization rate. When P is of other types such as a list, we have two alternatives. One is [2] that two elements are regarded as the same if we have dis(**a**, **param**(p)) $< \varepsilon$ under a given distance measure dis(**x**, **y**) (e.g., dis(**x**, **y**) = $\|\mathbf{x} - \mathbf{y}\|$) and a tolerance ε. The other is [12] that all the elements are quantized and stored at a given resolution rate and two elements are regarded as the same if they have the same quantized value. The way is almost the same as that used when P is an accumulation array except here we still store the quantized value as **param**(p), instead of transforming it into the coordinates of a bin. The former one is selected if one has some previous knowledge to choose an appropriate distance measure. Otherwise, the latter one is suggested to be used since it is simple and easy to handle.

(5) *How many trials are enough for finishing an accumulation period?* There are several possibilities. In this paper, we only consider two simple methods. In the first one, an accumulation period stops when the number k of random trials in the period reaches a previously given limit $k_{\max}$. In the other one, we stop when there is a $p \in P$ with its score(p) reaching a predefined threshold n_t; i.e., after each update score(p) := score(p) + 1, we check whether score(p) = n_t; if yes, we finish the accumulation period. We can also combine the two ways such that a period is finished whenever one of the two happens first. In section 5.2, we will analyze how to appropriately select the values of $k_{\max}$ and n_t.

(6) *How do we know that no true curve remains in the image?* The task is easy if we know initially how many true curves of the specific curve type there are in the given image. In this case, we only need to check how many true curves have been already detected at present. If we have no such previous knowledge, as pointed above in Definition 3, the task is strongly interwoven with the task of detecting out curves. In this case, what we can do is that after a completed epoch we just run another epoch as if there were some true curves remaining in the image. If the new epoch is still not completed when the number ku of accumulation periods that finished within the epoch reaches a given limit $ku_{\max}$, then we assume that there is no true curve left in the image. In Section 5.2, we will analyze the problem of how to appropriately predefine $ku_{\max}$.

3. THE NEW MECHANISMS IN RHT AND THEIR ADVANTAGES

The core of the conventional HT is basically the combined use of *exhaustive enumeration* of the pixels in the image space, *score accumulation* in the parameter space and a *diverging mapping*[4] bridging the two space. In RHT methods, two new mechanisms, namely *random sampling* and *converging mapping*, replace the roles of exhaustive enumeration and diverging mapping, respectively. The combined use of these two new mechanisms produces some favorable characteristics which not only give RHT the advantages of considerably increased computing speed and more reliable detection of maxima in the accumulation array, but also provide the possibilities for using an effective *stepwise procedure* to assist *curve verification* and for using *storage of other types* for *score accumulation*. These measures can further speed up the computation of RHT and give the advantages of small storage, infinite parameter space, and high resolution. In the sequel, we will show these characteristics through theoretical analysis and experimental demonstrations.

3.1. *Two New Functions of the Converging Mapping*

In the development of the HT and its variants, the function of converging mapping has been explored by several authors [11, 13]. It has been found that in the implementation of HT one can use more than one pixel simultaneously to vote a hypersurface with a reduced dimension or even a

[4] That is, HT maps one pixel in the image space into all the points on a hypersurface in the parameter space. We call this mapping shortly the *diverging mapping*.

single point in the parameter space, and thus the computation in the parameter space can be reduced. However, for an n-parameter curve function, it was found that in an image of N pixels one needs the total of C_N^n enumerations to implement the mapping of a whole HT process. C_N^n is of the order $O(N^n)$, and the complexity of HT is of the order $O(N\ N_a^{n-1})$ with N_a being the size (on one dimension) of the accumulation array used. As a result, the function of converging mapping has been regarded as *trading off work in parameter space for work in image space,* according to which of N, N_a is larger. However, here we like to show that the converging mapping has two other favorable properties which have remained undiscovered before.

First, the *converging mapping can considerably sharpen the local peaks (maxima) in the accumulation array.* Thus, it can ease the difficulty of finding maxima encountered in HT and make curve detection more reliable. Consider the IMBP model given by Definition 2 in Section 2.1. It is not difficult to figure out that the local peaks of the stores in the accumulation array are generated by pixels of ideal true curves and that the sharpness of a peak is affected by the relative difference between the scores of the peak and the scores generated by ideal pseudo curves, whose parameters are located in the neighborhood of the peak in the parameter space. Thus, the ratio of the scores of a local peak generated by an ideal true curve to the scores generated by any one of the ideal pseudo curves could be used as a tool for describing the sharpness of a local peak. The larger the ratio is, the sharper the peak.

Assume that there is an ideal true curve c_1 of n_1 pixels and an ideal pseudo curve c_2 of n_2 pixels with $n_1 > m_{\min} > n_2 \geq n$. For the diverging mapping used in the HT, the ratio of the scores of the local peak generated by c_1 to the scores by c_2 are given by $R(n_1, n_2, 1) = n_1/n_2 > 1$. On the other hand, for the converging mapping, there will be C_{n1}^n and C_{n2}^n combinations of n-tuples for being mapped into the scores corresponding to c_1 and c_2, respectively. Thus after enumerating all the possible n-tuples in an image, the ratio of the scores of the local peak generated by c_1 to the scores generated by c_2 is given by

$$R(n_1, n_2, n) = \frac{C_{n1}^n}{C_{n2}^n} = \frac{n_1(n_1 - 1)\cdots(n_1 - n + 1)}{n_2(n_2 - 1)\cdots(n_2 - n + 1)} > R(n_1, n_2, 1) = \frac{n_1}{n_2}. \tag{2}$$

Denote $n_1/n_2 = \gamma$; then we have approximately $R(n_1, n_2, n) = \gamma^n$ as long as n_1, n_2 are much larger than n. Recalling $\gamma > 1$ (since $n_1 > n_2$), we see that $R(n_1, n_2, n)$ has been considerably increased. In other words, the local peaks have been considerably sharpened.

Second, the *converging mapping provides a possibility of replacing the accumulation array by the storage* P *of other types.* For the diverging mapping used in the HT, all the parameter points on a hypersurface are involved after mapping each pixel. In this case, the storage used for the accumulation in parameter space has to be able not only to include all the involved parameter points, but also to explicitly retain all the geometrical neighborhood relations between these points so that every point on a hypersurface is easily accessed through the given curve function. The accumulation array used in HT satisfies this requirement. However, as will be pointed out in Section 3.4, this kind of array has also some quite unfavorable characteristics. For the converging mapping, each map only involves one point in the parameter space. The geometrical neighborhood relations between the stored parameter points are no longer of key importance although the relations may still be useful (e.g., for finding the local maxima). Thus, there exists the possbility of using other types of storage P instead of accumulation array. This opens up the possibility to give RHT several advantages through combining other mechanisms and appropriately selecting the storage P, as shown in later sections.

3.2 *Random Sampling Can Speed Up the Computation*

In spite of its favorable functions, the mere use of the converging mapping has an expensive computation cost of order $O(N^n)$. However, *the high cost can be significantly reduced by combining its use with random sampling.*

Consider again the IMBP model and assume that we could reduce the total number of pixels N into N/r_d in such a way that the length of each ideal true and pseudo curve is also reduced from n_i into n_i/r_d, with $r_d > 1$ the reduction rate. Then, the ratio given in Eq. (2) will change into

$$R\left(\frac{n_1}{r_d}, \frac{n_2}{r_d}, n\right) = \frac{n_1/r_d(n_1/r_d - 1)\cdots(n_1/r_d - n + 1)}{n_2/r_d(n_2/r_d - 1)\cdots(n_2/r_d - n + 1)} = \frac{n_1(n_1/r_d - 1)\cdots(n_1/r_d - n + 1)}{n_2(n_2/r_d - 1)\cdots(n_2/r_d - n + 1)} \tag{3a}$$

$$R\left(\frac{n_1}{r_d}, \frac{n_2}{r_d}, 1\right) = \frac{n_1/r_d}{n_2/r_d} = \frac{n_1}{n_2}. \tag{3b}$$

Since $n_1 > n_2$, as long as $(n_1/r_d - n + 1) > 1$, i.e., $r_d < n_1/n$, it follows that

$$R\left(\frac{n_1}{r_d}, \frac{n_2}{r_d}, n\right) > R\left(\frac{n_1}{r_d}, \frac{n_2}{r_d}, 1\right) = R(n_1, n_2, 1) = \frac{n_1}{n_2}. \tag{3c}$$

Since n_1 could be the number of any ideal true curve and n_2, the number of any ideal pseudo curve in the IBMP model, in comparison with Eq. (2), we can see that curves in an IBMP model of N pixels can be detected through its proportionally reduced version of N/r_d pixels in such a way that:

(1) When using the *diverging mapping,* the ratios of scores of ideal true curves to those of ideal pseudo curves remain unchanged. In other words, the local peaks in the accumulation array on the reduced model are as sharp as they are on the original model.

(2) When using the *converging mapping,* the ratios of scores of ideal true curves to those of ideal pseudo curves are still higher than those obtained on the original model by the diverging mapping. In other words, the local peaks in the accumulation array on the reduced model are still sharper than those obtained on the *original model by the diverging mapping.*

At the same time, the computing complexity of the diverging mapping will be reduced by r_d times, and the computing complexity of the converging mapping is now of the order of $O(N^n/r_d^n)$. That is, it has been reduced by the amount of

$$\frac{C_N^n}{C_{N/r_d}^n} = r_d^n \frac{(N-1)\cdots(N-(n-1))}{(N-r_d)\cdots(N-r_d(n-1))} \geqslant r_d^n,$$

when $r_d \gg 1$.

Obviously, this is a significant speed up on the computation of the converging mapping.

However, to realize the proportional reduction of an IMBP model of N pixels into one of N/r_d pixels, we have to know which pixel belongs to which one of the ideal true curves or ideal pseudo curves. This is precisely the task we are trying to accomplish. There is a vicious circle which is difficult to break in a deterministic way. Fortunately, in the average sense the random sampling can realize the task of proportional reduction. If we randomly pick N/r_d pixels of N pixels in such a way that each pixel has equal probability to be picked, then each picked pixel has probability $p_i = n_i/N$ to be a pixel of curve c_i with length n_i. On an average, when randomly sampling N/r_d times, among these picked pixels there are approximately $p_i(N/r_d) = (n_i/N)\ (N/r_d) = n_i/r_d$ pixels belonging to curve c_i.

The above analysis suggests that by randomly sampling N/r_d pixels and then using the converging mapping on these pixels we can detect curves with a considerably reduced complexity of computation while retaining the higher reliability for finding maxima. This explains why the RHT method uses the combination of *random sampling* and *converging mapping* as its core ingredient. The random picking of N/r_d pixels first and then using them for the converging mapping is equivalent to making N/r_d random trials and randomly sampling n pixels for the converging mapping in each trial.

The advantages of the combined use of *random sampling* and *converging mapping* discussed above are also verified by the experimental results given in Fig. 2 and Fig. 3. In comparison with the standard HT, Fig. 2 shows that even in an image disturbed by very strong outlier noise, the use of these two new mechanisms can reduce the computation by more than a hundredfold and at the same time make the local peaks A, B obviously sharper.[5] Figure 3 shows that these advantages are still retained when the two mechanisms are used on a more complicated image, disturbed not only by strong outlier noise but also by Gaussian noise and quantization errors, although the computations spent by both the HT method and the combined use of the two new mechanisms are larger than on the image in Fig. 2a. In [2], these advantages were shown by the experiments on two images with low noise levels. Here Fig. 2 and Fig. 3 show that the advantages are robust with respect to various types of random noise (outlier, quantization errors, Gaussian noise). In Section 3.3 and Figs. 4 and 5, we will also show that the advantages can be further enhanced by cooperatively using the measures of *stepwise implementation and curve verification.* Moreover, later in Section 4.2 the advantages will again be shown by the application of detecting lines on a document image with strong coherent (non-random) noise.

So, we have seen that the combined use of *random sampling* and *converging mapping* in the RHT methods are far beyond the well-known measure of *trading off work in parameter space for work in image space.* In Section 5, we will theoretically show that the computation complexity of the RHT method is in fact not of the order of $O(N^n)$ but in the average sense an order much lower than $O(N^n/n_{\min}^n)$, where $n_{\min}$ is the length of the shortest curve in an image. This is considerably smaller than the $O(N\ N_a^{n-1})$ complexity of the HT method and variants.

From Eq. (3c), one may find that even without the converging mapping, some speedup could be obtained by merely using *random sampling* to replace *exhaustive enumeration* in the core of the conventional HT (i.e., the combination of *exhaustive enumeration, diverging mapping,* and *score accumulation*). In fact, Fischler *et al.* [15] proposed such a replacement. They suggested that the computation of the conventional HT method can be speeded up by using in parallel several machines for picking pixels randomly and then by mapping them with the diverging mapping into the parameter space for score ac-

[5] The number of *flops* in this and the following tests is counted by the MATHWORK Inc.'s PROMATLAB software, by which one real number addition or multiplication is defined as one flop.

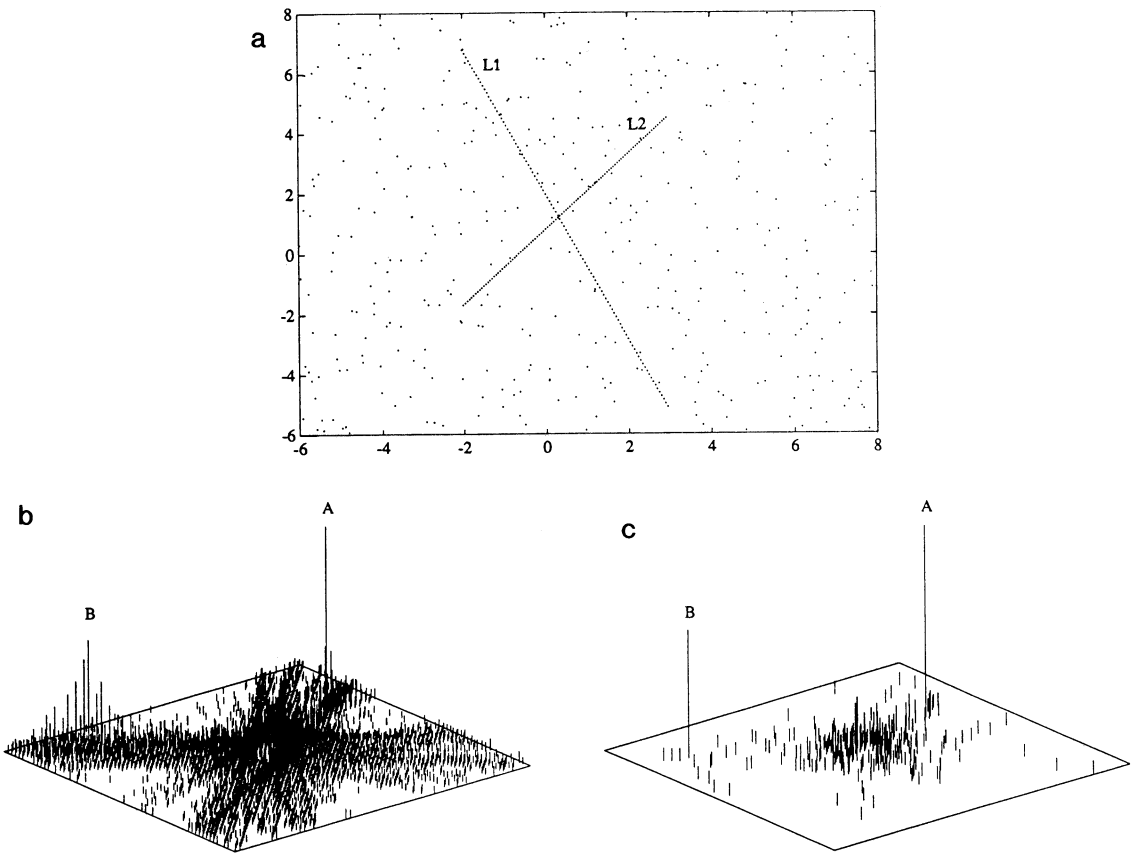

FIG. 2. A comparison experiment on an image with strong outlier noise. An accumulation array of size 128 × 128 is used on a window $[-2, 2] \times [-2, 2]$ in the parameter space under the parameterization $a_1x + a_2y = 1$: (a) An image which contains two lines L_1 and L_2; each has the length of 100 pixels and these pixels are buried among 400 random outlier points. (b) The resulting 2D histogram by the standard HT. The maxima *A*, *B* denote the two lines detected. The computation used 957,587 *flops*. (c) The resulting 2D histogram by using *Random sampling and converging mapping*. The result is obtained after 400 random samplings, and the computation used 8773 *flops* which is 108.7 times faster than that used by the HT. In addition, one can observe that the peaks *A*, *B* in (c) are much sharper than they are in (b). For a clarity, the displays in (b), (c) are sampled from the real ones in such a way that every 2 × 2 bin is replaced by one bin with its score being the largest of the four. In addition, any scores are set to zeros if they are lower than 2.5% of the maximal score in the whole accumulation array.

cumulation. Our analysis above has shown that even a single machine with a sequential implementation can speed up the computation. However, the use of random sampling only, without combining it with the converging mapping, has a limited advantage. It can only reduce the computing complexity of the HT from $O(N\ N_a^{n-1})$ to $O((N/n_{\min})\ N_a^{n-1})$. Also, it cannot share other advantages of the RHT.

Interestingly, in their good robust method for parametric modelling—RANSAC [14], Fischler *et al.* did use the combination of random sampling and converging mapping. The main difference between RANSAC and RHT is that RANSAC does not use *scores accumulation* in the parameter space. RANSAC is a *guess-and-test method* but not a HT-like method. In addition, RANSAC is proposed for modelling or curve fitting in the existence of ouliers. Although it may also be possible to extend RANSAC for the use of curve detection in an image, we will show elsewhere that its computation complexity is considerably larger than that of RHT. In addition, how RANSAC performs for finding multiple curves in the global optimal sense is also an interesting question to be answered, since it has no accumulation array for globally managing the parameter space and accumulating evidence.

3.3 *Further Reductions in the Computation by Using Stepwise Procedure*

Let us again consider the IMBP model. The pixels on an ideal true curve are usually not merely on the true curve but also possibly on other pseudo curves. Due to the contributions of these pixels, in the accumulation array the scores of pseudo curves may be comparable to or may even bury the local peaks formed by short ideal true curves. As a result, it is hard or impossible to find

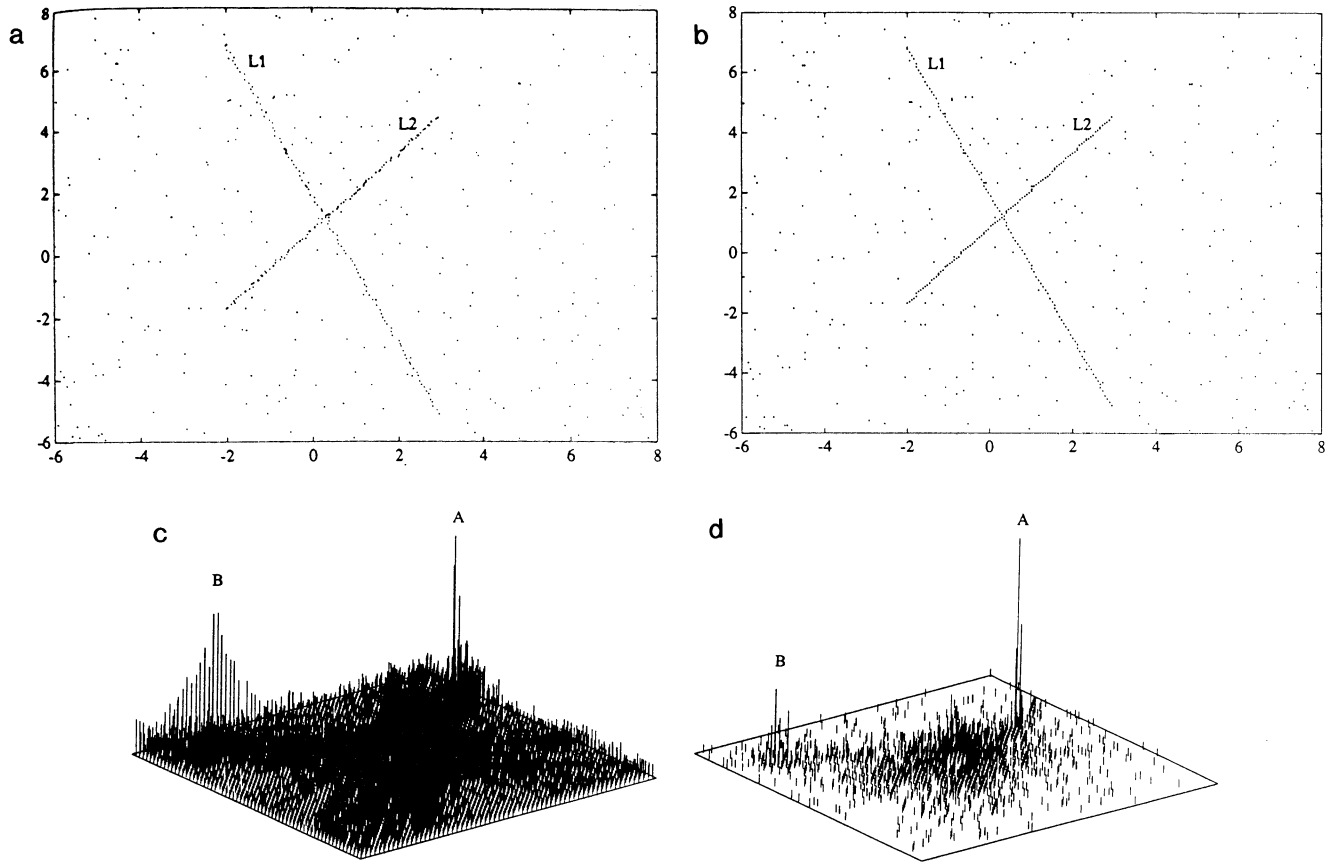

FIG. 3. A comparison experiment on an image distorted by Gaussian noise, quantization errors, and strong outlier noise. An accumulation array of size 256 × 256 is used in the same window and under the same parameterization as Fig. 2: (a) In the image, there are two lines L_1 and L_2 and each has the length of 100 pixels. Each pixel of the lines is disturbed by Gaussian noise, and the pixels are buried among 300 random outlier points. (b) The image is obtained by quantization of that of Fig. 3a into a 256 × 256 array. The lines are reduced to 99 pixels and the outlier set to 299 pixels, and the lines have become stair-shaped. (c) The resulting 2D histogram by the standard HT. The computation used 1,586,341 *flops*. (d) The resulting 2D histogram by using *random sampling and converging mapping*. The result is obtained after 79,438 flops which is 19.6 times faster than that used by the HT. Again one can observe the considerably sharpened peaks *A*, *B* in (c). The displays in (c), (d) are processed in the way similar to those used in Fig. 2, except that here every 4 × 4 bin is replaced by one bin with its score being the largest one.

out these short ideal true curves. In [5], Risse gave an approach for tackling the problem of HT. The idea is to first find out one (or more) or local maxima (e.g., the global maximum) from the accumulated array and then to decrease the accumulated array in such a way that each pixel on the curve specified by the detected maximum decreases all the accumulated bins located on the hypersurface specified by the pixel. The effect of Risse's approach is equivalent to using the HT on a reduced image in which the pixels of the detected curves are removed. Thus the contributions of all these pixels to other pseudo curves are eliminated and the short true curves become easy to detect. The cost of the approach is that the computation complexity is increased from $N\ N_a^{n-1}$ to $N\ N_a^{n-1} + N_c N_a^{n-1}$ with N_c being the total number of pixels of true curves in the image.

Random sampling can provide the same effect as Risse's approach; however, now the computation cost is not increased but can be in fact considerably decreased. For using the HT, one can start to make a decision (i.e., find the local maxima in the accumulation array) only after all the pixels in an image have been *exhaustively enumerated*. In other words, the HT can only work in the batch way, and the decision based on the accumulated evidence from the partial pixels of an image can be totally reversed after some more pixels are used for accumulation. In contrast, the use of *random sampling* can allow the decision to be made in an adaptive or incremental way. The decision based on the evidence accumulated during the previous trials of random sampling is gradually enforced or improved after some more trials are made. As a result, one can make a decision as long as the evidence for detecting one (or more) of the true curves is enough. With no need to wait until all the true curves can

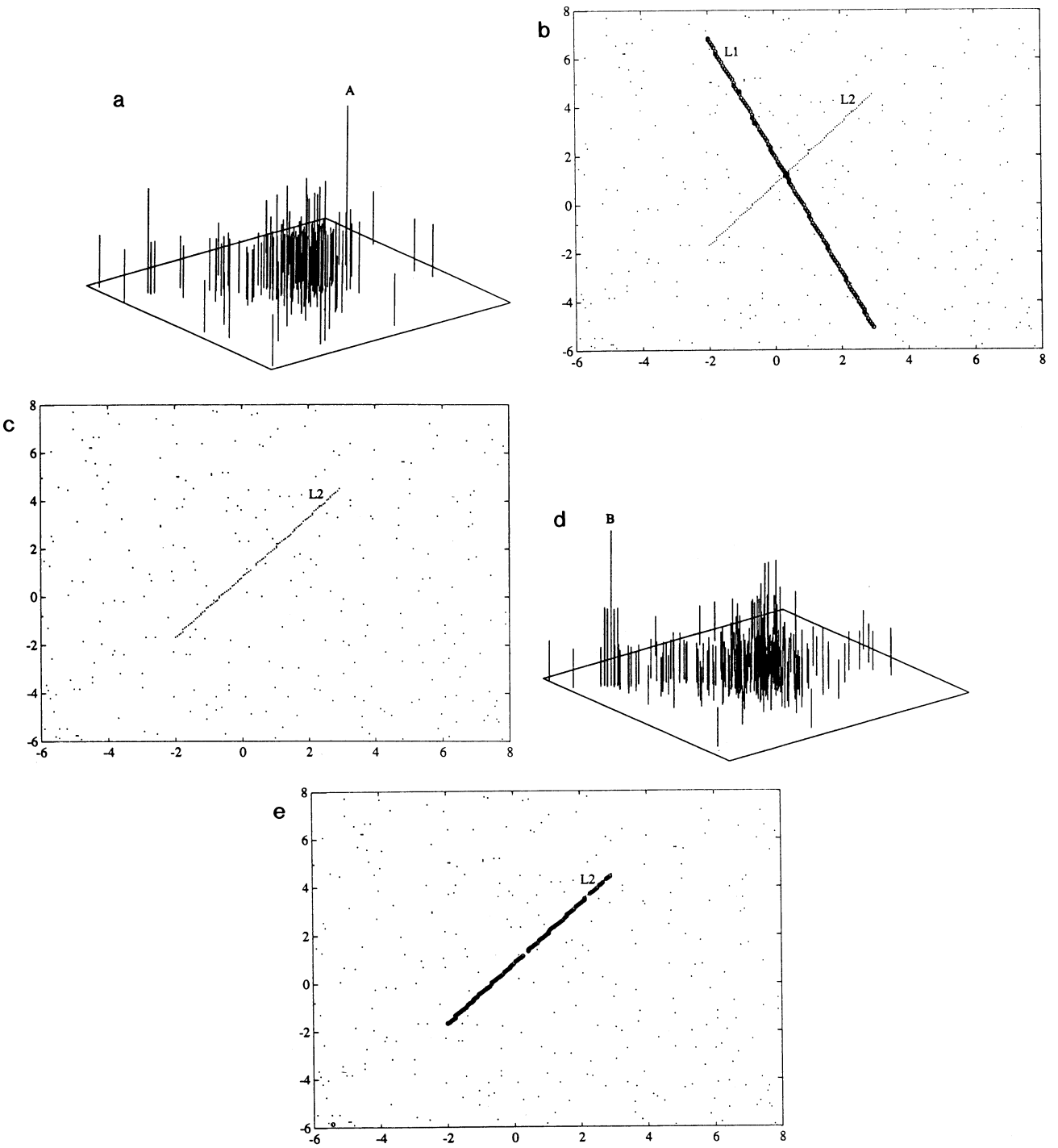

FIG. 4. The experiments for showing the advantage of detecting one line at one epoch: (a) The resulting 2D histogram by using *Random sampling and converging mapping* on the image given in Fig. 3(b). Only after 200 random samplings with the computation of 3984 flops, the first line is detected by the goal maximum A. (b) There are 105 pixels falling within a δ-band of the line specified by the parameter given by the maximum A, where $\delta = 1$ *pixel*. These pixels are taken out as the pixels of line L_1. (c) The remaining pixels after removing the detected 105 pixels of L_1. (d) The resulting 2D histogram on the image given in Fig. 4(c). After 350 random samplings with the computation of 6943 flops, the second line is found by the goal maximum B. (e) 94 pixels are detected as the pixels of L_2 because they fall within the band of $\delta = 1$ of the line specified by the parameter given by the maximum B. In this experiment the total computation was 3984 + 6943 = 11,927 flops which is 6.7 times less than that used for forming Fig. 3(b) and 133 times less than that used by the HT.

be correctly detected, an accumulation period epoch can be regarded as completed as long as one (or more) of the true curves can be correctly detected. After removing from the image all the pixels of the detected true curve (or curves) and resetting to zero all the bins in the accumulation array, we start another epoch on the reduced image for detecting the remaining curves.

This kind of stepwise procedure can further reduce the computation cost. One reason is that the number of random trials needed to obtain enough evidence for detecting all the true curves in an image is usually much larger than the number of random trials needed to obtain enough evidence for detecting one true curve in the image. The other reason is that after removing from the image all the pixels of the detected true curve, not only is the length of a large number of pseudo curves decreased but also the probability of a true curve being sampled at each trial is increased, since the length of a true curve usually remains unchanged but the total number of pixels in the image is obviously decreased due to the removal of a detected curve.

These characteristics are also shown by the experimental results in Fig. 4. The experiments were conducted on the image given in Fig. 3b which contains two lines under the disturbances of strong outlier and Gaussian noise and quantization errors. The use of the converging mapping is combined into the above *stepwise procedure* for using *random sampling*. The first epoch consists of 200 random samplings. After this epoch the first line is detected by the maximum A. The second epoch is completed after 350 random samplings with the second line found. In this experiment, the stepwise implementation accelerates the computation speedup 6.7 times in comparison with the batch implementation of random sampling and converging mapping, and by 133 times in comparison with the standard HT.

The key problem for implementing the above stepwise procedure is how to decide when an epoch has been completed. The solution is to regard an epoch as further consisting of one or several accumulation periods. Each accumulation period is considered finished when the number of random trials within the period reaches a given limit $k_{\max}$, and the curve (or curves) presently found are then regarded as candidates. A test called *curve verification* is made on each candidate to verify whether it is really a true curve by Definition 1b. If not, then in the accumulation array only reset to zero the bin that gives the candidate, and then random trials are continued for a new accumulation period. If yes, an epoch is considered as completed with some true curves detected, and we start the next epoch after removing from the image all the pixels of the detected true curve (or curves) and resetting to zero all the bins in the accumulation array. Furthermore, an alternative way for considering an accumulation array as finished is to give a predefined threshold n_t and finish a period if in the accumulation array there is one bin that has reached the score n_t. The appropriate values of $k_{\max}$ and n_t will be further studied in Section 5.

The inclusion of *curve verification* will usually make the computation more effective. As an example, Fig. 5 gives the experimental results made on the same image as used in the experiments of Fig. 4. The above alternative way is used with $n_t = 2$. As shown in Fig. 5a, the first period is completed only after nine random trials and the resulting

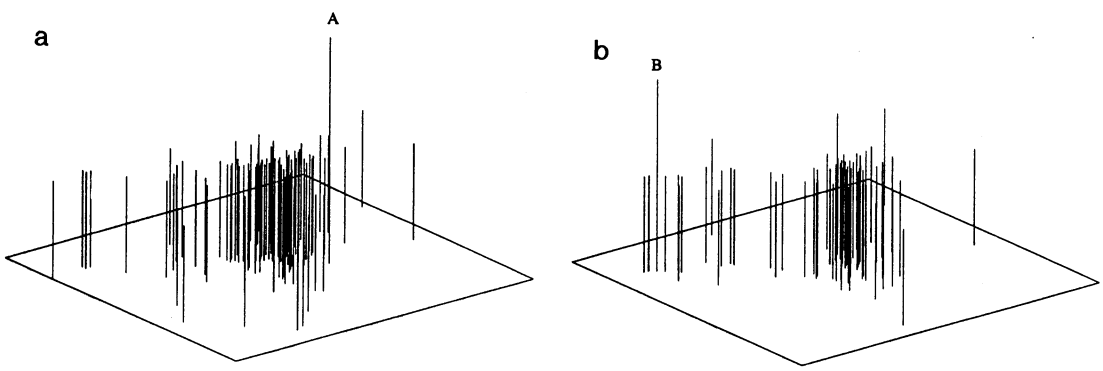

FIG. 5. Further speedup by adding *curve verification* into the stepwise implementation of *random sampling and converging mapping* with $n_t = 2$: (a) On the image given in Fig. 3(b), the first period is completed after only nine random trials; however, the consequent verification found less than $m_{\min} = 20$ pixels within the band of $\delta = 1$ of the line specified by the parameter of the bin with scores n_t. Thus the second period is continued without fully resetting the accumulation array. The period completed the first epoch after 108 random trials, and the first line L_1 is detected by the bin A with the maximal scores n_t. Together, 119 random samplings used 2332 flops. (b) After removing the detected 105 pixels of L_1, Fig. 4(c) is again obtained. On this image, another period completed the second epoch after 90 random samplings with the second line being found by the bin B with the maximal scores n_t. This epoch used 1087 flops. The total computations of this experiment are 2332 + 1087 = 3419 flops which is 3.4 times less than that used in the experiments of Fig. 4, 23.2 times less than that used for forming Fig. 3(b), and 464.0 times less than that used by the HT.

candidate has failed to pass the verification. The first line is detected on the second period which spent 108 random trials. Thus, the first epoch consists of two periods which together spent 117 trials. In Fig. 5b, the third period (or the second epoch) is completed after 90 trials with the second line being found. In this experiment, the computation has been further speeded up by 3.4 times in comparison with that in Fig. 4, by 23.2 times in comparison with the batch implementation of random sampling and converging mapping, and by as much as 464 times in comparison with the standard HT.

3.4 *The Advantages of Using Other Storage Types*

As pointed out in the end of Section 3.1, a storage of the accumulation array type has the feature of retaining explicitly all the geometrical neighborhood relations between the points in the parameters spaces and is thus suitable for satisfying the requirement of the diverging mapping used in the standard HT method. However, the use of this kind of accumulation array has some unfavorable features. First, the accumulation array can only represent a small window of parameter space; if the parameters of some curves fall outside the window, they cannot be detected. Therefore, one must appropriately predefine such a window. However, in practice, the location and the size of such a window is difficult to predefine unless prior knowledge is available about the curves in an image. Second, in order to minimize the chance for some curves to fall outside the window, one usually has to let the size of the window be quite large. But under a given quantization rate, a large size of window means a large N_a which will make the computation cost $N\ N_a^{n-1}$ of HT considerably high. Third, for some high resolution image (e.g., 512 × 512 or more), one would also like to expect the parameters of detected curves to be as accurate as possible. In this case, one has to use high resolution on the parameter space too, and thus the high resolution quantization will make N_a quite large even for a window of small size. Again the computation increases considerably.

As pointed out in the end of Section 3.1, the use of the *converging mapping* provides a potential possibility of replacing the accumulation array by some other storage type since the problem of retaining the geometrical neighborhood relations between the parameter points is no longer of key importance. The simplest way is to use a storage P of linear list structure to replace the accumulation array, with each element in the list consisting of not only the scores received by a point in the parameter space but also explicitly the coordinate values of the point, i.e., an element $p \in P$ is given by $p = [\mathbf{param}(p), \text{score}(p)]$. *The biggest advantage of such a replacement is that one needs not predefine the size and the location of a window in the parameter space.* In fact, now P can accommodate all the possible curves in an image, in other words, P *implicitly represents the infinite parameter space* R^n. Moreover, **param**(p) could be stored either using quantized discrete values or the real values under any accuracy, thus *it is possible to obtain arbitrarily high resolution in the parameter space for high resolution images* without obviously increasing the size of the storage.[6]

However, in the case of using the converging mapping only, the replacement of the accumulation array by a storage P of linear list structure is not very useful. First, the size of P will be very large since the *exhaustive enumeration* of C_N^n n-tuple will put a great many elements in P. Second, when appending a new element in P, one needs to check whether there is already an element in P which is the same as the new one under the given criterion. This will require considerable searching time if the size of P is large. Third, after each pixel in the image has been enumerated, one can not select candidate curves by choosing the local maxima in P since the geometrical neighborhood relations between the parameter points are no longer retained in P. Instead, one can only choose one global maximal score (or the first several global maximal scores) in P. As a result, some curves will remain undetected. Although one can detect the remained curves in a way similar to Risse's approach discussed above, it means a further increase of the computation complexity.

Fortunately, the above problems can be solved again when the converging mapping is cooperatively used with random sampling, stepwise procedure, and curve verification. First, as shown in Section 3.3, because of random sampling, one can at each accumulation period select one candidate by choosing the maximum score of P in the stepwise procedure with the computing costs not increasing but decreasing. Second, the size of P can be drastically reduced because the random sampling and stepwise procedure let each epoch only involve a very small portion of all the possible n-tuples, instead of exhaustively enumerating the whole C_N^n combinations. For example, one can observe from Figs. 2b, c and Figs. 3c, b that the number of the occupied bins is considerably reduced after using *random sampling* in cooperation of *converging mapping*. A more drastic reduction of the number of the occupied bins can be seen from Figs. 4a, d and Figs. 5a, b due to the further combined use of the stepwise procedure and curve verification. E.g., in Fig. 5b, there are only less than 100 bins occupied, while the total number of bins in the accumulation array is 256 × 256 which is 655 times larger than the occupied number, and the total

[6] E.g., for the IMBP model, the maximum size of storage P is $m_t + m_{ps}$ for any quantization rate. However, for images with large quantization errors, the resolution rate used in the parameter space does influence the size of P. A low rate will give a small size because segments of curves with small differences will be considered as only one curve and $m_t + m_{ps}$ becomes smaller.

number of C_N^n is $C_{497}^2 = 123{,}256$ which is more than 1232 times larger than the actually occupied number.

Now we are ready to replace the accumulation array by a storage P of other types. These types have a common point that an element $p \in P$ is given by $p = [\textbf{param}(p), \text{score}(p)]$ which explicitly stores the values of a mapped parameter point. Different data structures (e.g., linear list, tree structure, or Hashing table) and accessing methods could be used according to the situations in real applications. When an image has low noise levels and small quantization errors, one can simply let **param**(p) be real values stored in a linear list, as used in [2], where we spent only a storage size around 30 for getting a higher parameter resolution than the standard HT which used an accumulation array of $256 \times 256 = 65{,}536$. However, for a noisy image especially with rather large quantization errors, the way used in [2] is not good since the size of a line list may increase considerably. As a result, not only the storage is largely increased but also the search time for accessing an element in the list becomes large. One solution is to use an ordered tree or other structure for managing an effective search. The other solutions, which are more simple and probably better, are given in [12] and will be briefly introduced in Section 4.1.

4. RHT ALGORITHMS AND AN APPLICATION IN DOCUMENT PROCESSING

4.1. *Several Algorithms for Implementing RHT*[7]

(1) ALGORITHM RHT.b. This is the basic form of RHT, which is obtained by simply using *random sampling* and *converging mapping* to replace the roles of *exhaustive enumeration* and *diverging mapping* in the standard HT. The accumulation array is still used for *score accumulation*. In comparison with the HT, RHT.b has the advantages of fast computation and more reliable location of local maxima. As shown in Figs. 2 and 3, RHT.b is suitable not only for low-noise images but also for images with strong noise of various types.

Initialization. Given $k_{\max}$, let all the bins in accumulation array $A(i_1, i_2, \ldots, i_m)$ be zero.
10 **for** $k = 1$ to $k_{\max}$ **do**
20 randomly pick a n-tuple from D;
30 do converging mapping to get its solutions $\mathbf{a}_i$, $i = 1, \ldots, r$
($r = 1$ if only one solution; $r = 0$ if no solution or infinite many solutions.)
40 **for** i = 1 **to** r **do**
50 quantize $\mathbf{a}_i$ into a coordinate index $i_1, i_2, \ldots, i_m$;
60 $A(i_1, i_2, \ldots, i_m) = A(i_1, i_2, \ldots, i_m) + 1$;
70 **endfor**
80 **endfor**
90 find the coordinates of all the local maxima in A;
100 **for** each of these maxima **do**
110 transform its coordinates into a real parameter vector;
120 check whether it represents a true curve by *Definition 1* (b); if yes, take it as a solution;
130 **endfor**

---

(2) ALGORITHM RHT.bsa. The only difference from RHT.b is that it is implemented as a *stepwise procedure* with *curve verification* conducted after each accumulation period. The accumulation array is still used. Here, ".bsa" is the combination of the first characters in the words "basic," "stepwise," and "accumulation array." RHT.bsa has the same advantages as RHT.b, but as shown in Figs. 4 and 5, these advantages are considerably enhanced.

Initialization. Given $k_{\max}$, n_t, $ku_{\max}$. Let $I = 0$, $k = 0$, $ku = 0$, *stop* := *false*, *periodend* = *false*, *epochend* = *false*; and let all the bins in the m-dimensional array A be zero.
10 **repeat until** *stop* = *true*
12 **repeat until** *epochend* = *true* **or** *stop* = *true*
15 **repeat until** *periodend* = *true*
20 randomly pick a n-tuple from D;

---

[7] If not specially indicated, the symbols and terminologies used in this subsection have the same meanings as those in Section 2.

```
30          do converging mapping to get its solutions a_i, i = 1, . . . , r
               (r = 1 if only one solution; r = 0 if no solution or infinite many solutions.)
40          for i = 1 to r do
50             quantize a_i into a coordinate index i_1, i_2, . . . , i_m;
60             A(i_1, i_2, . . . , i_m) = A(i_1, i_2, . . . , i_m) + 1;
62             if A(i_1, i_2, . . . , i_m) = n_t then
64               let I_s = [i_1, i_2, . . . , i_m];
66               let A(i_1, i_2, . . . , i_m) = 0 and periodend = true;
68             endif
70             endfor
72             k = k + 1;
74             if k ≥ k_max then
76               find the coordinates I_s = (i_1, i_2, . . . , i_m) of the global maximum in A;
78               periodend = true
79             endif
80        endrepeat { an accumulation period is finished }.
82        transform I_s into a real parameter vector a_f according the quantization rate.
84        check whether a_f represents a true curve by Definition 1(b); if yes, epochend = true;
90        if epochend = false then ku = ku + 1
91        if ku ≥ ku_max then stop = true;
93        periodend = false and k = 0;
94     endrepeat { an epoch is completed };
95     I = I + 1; if I = m_t then stop = true;
       (where m_t is the total number of true curves in an image.)
96     if stop = false then
97       let a_f as a solution; remove from D all the pixels falling in the given δ-band of a_f;
100      let ku = 0, epochend = false, and all the bins in array A be zero;
102    endif
110 endrepeat { all the true curves have been detected }.
```

(3) OTHER ALGORITHMS. In Appendix A of this paper, we also give three other algorithms: RHT.srl, RHT.sql, and RHT.sqh. RHT.srl is just our earlier version of RHT algorithm given in [2]. RHT.srl shares with RHT.bsa two advantages: fast computation and better reliability in maximum finding (i.e., candidate finding). As shown in the experiments in [2], RHT.srl also has the advantages of having infinite parameter space, using small storage and having high resolution in comparison with the standard HT. However, not as RHT.bsa, RHT.srl generally is not well suitable for images with strong noise. The reason is that RHT.srl stores **param**$(p) = \mathbf{a}_f$ in real values. In the case of a noisy image, the size of P will increase, which results in the increase of both the storage and the search time of the linear list.

RHT.sql not only retains all the advantages of RHT.srl, but also avoids its disadvantage of being only suitable for low-noise images, since RHT.sql also uses a linear list as the storage (i.e., like RHT.srl) and RHT.sql stores **param**$(p) = \mathbf{a}_f$ in a quantized form (like RHT.bsa). Thus, as RHT.bsa, RHT.sql is also suitable for images with strong noise of various types. As a result, RHT.sql combines the advantages of both RHT.srl and RHT.bsa.

RHT.sqh has the advantage that its computing time is much less than that of RHT.sql, since the linear list is replaced by a mixture structure of two hash tables and one linear list. Due to the *hash table*, the elements in P can be accessed much faster. However, the minimal size of P is the size of the two fixed hash tables. In the case of less noisy images, the minimal size may be larger than the size of P used by RHT.sql. Thus, in such cases, RHT.sqh may take up more storage than RHT.sql.

4.2. *An Application in Document Processing: Detecting Lines in Forms*

Recently document image analysis has become a popular and growing research area, and a good survey about the present state of this area can be found in [16]. Among various kinds of documents, the form or table (e.g., checks, bills, income tax returns, and various application forms) is an important type. During the process of automatic form analysis, the detection of straight lines plays an important role in removing the skew and extracting form items [17]. Here, we give an example of using RHT for detecting lines in a form image.

Figure 6 shows an image scanned from a form. The form contains five lines which are distorted and discon-

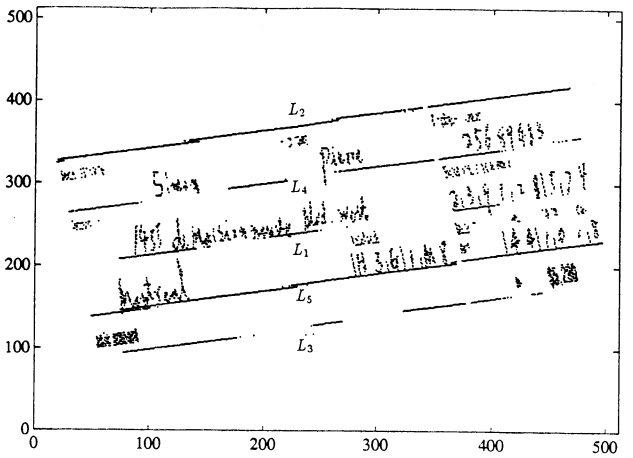

FIG. 6. Line detection for form processing—part (1). A form is scanned into a 512 × 512 binary image with 3826 white pixels. There are five straight lines which are divided into several segments due to scanning noise and quantizing errors. In addition, there are also a lot of pixels produced by the characters and other items of the form. These pixels act as non-random interferences for the task of line detection.

nected by sensor noises. On this form, the conventional HT only detected out four lines. Due to the influence of non-random noise pixels, the middle line (denoted by L_1 in Fig. 6 was not detected. The local peak for this line was not found in the accumulation array of HT since the peak was buried by the scores contributed from noise pixels. Here, the polar parameterization was used (see Eq. (1b) in Section 2.2), and the 256 × 256 accumulation array was located on the window $[0, \pi] \times [0, 512\sqrt{2}]$ which is the conventional way of selecting the window for the polar parameterization. It has taken very long computing times (about 8,578,712 *flops*) for the conventional HT to detect even the other four lines.

In contrast, all the five lines have been detected by using either RHT.sql or RHT.sqh. For the sake of comparing with the results of the conventional HT, here we give the experimental results by RHT.sqh (see Appendix A for details) as an example.

The same quantization rates as used by the HT were used for RHT.sqh, i.e., $\Delta\theta = \pi/256$, $\Delta\rho = 512\sqrt{2}/256$. The lengths of Hash Table 1 and Hash Table 2 are both 2000. The obtained results were further given in Fig. 7. As shown in Figs. 7a and b, the first epoch is completed by only one accumulation period with line L_3 in Fig. 6 detected. The epoch spends about 440 random samplings. Then, each of the following three epochs also contains only one accumulation period with line L_2, L_4, L_5 detected in sequel, and each epoch spent about 400 random samplings. Finally, the line L_1 is detected by the fifth epoch which consists of eight accumulation periods and it spent more than 3000 random samplings. The detection of L_1 is much harder than the other four, because after removal of pixels of the other four lines, one can see from Fig. 7c that the pixels of non-random noise take up a major portion of the whole image. While the HT is not able to find L_1, RHT.sqh can still find it after verifying eight candidates obtained by eight accumulation periods.

RHT.sqh has taken 310,122 *flops* for detecting the five lines, and only about 93,600 *flops* for detecting the first four lines. Thus, we see that RHT.sqh is 8,578,712/93,600 = 91.6 times (for detecting the four lines L_2, L_3, L_4, L_5) or 8,578,712/310,122 = 27.7 times (with the advantage of being able to find L_1) faster than the conventional HT.

Figure 7d shows the length of the dynamic linear list. Its maximum is 26, and its average is 11. This means that RHT.sqh has taken the storage of 2000 + 2000 + 11 = 4011 in average and 2000 + 2000 + 26 = 4026 in maximum, while the conventional HT has taken the storage of 256 × 256 = 65536, which is about 16.3 times larger than that taken by RHT.sqh.

5. QUANTITATIVE ANALYSIS ON STOPPING PARAMETERS AND COMPLEXITIES OF COMPUTATION

In the sequel, by using the IMBP model given by *Definition 2* of Section 2.1, we make some quantitative analysis on two important issues. One is how to appropriately select the parameters $k_{\max}$, $ku_{\max}$, and n_t in the RHT algorithms given in Section 4.1. We call them the stopping parameters of these algorithms since they decide when the implementation of a RHT algorithm should stop. The other issue is what are the average complexities of computing times and storages for each of the algorithms given in Section 4.1.

5.1. *The Generalized Bernoulli Trials and the Implementing Process of RHT Algorithms*

The *generalized Bernoulli trial* (GBT) is a probabilistic event that has $m + 1$ possible outcomes: m different types of "successes" $T_1, T_2, \ldots, T_m$ with probabilities $p_1, p_2, \ldots, p_m$, respectively, and one type of "failure" T_f with probability $p_f = 1 - (p_1 + p_2 + \cdots + p_m)$. A sequence of identical and mutually independent GBTs constitute a random process called the *generalized Bernoulli process* (GBP) which has some characteristics given as follows:

(1) In a sequence of n GBTs, let $\xi_1, \xi_2, \ldots, \xi_m$ denote the numbers of successes for $T_1, T_2, \ldots, T_m$, respectively. The joint probability distribution of $\xi_1, \xi_2, \ldots, \xi_m$ is the *multinomial distribution* [21]

$$P(\xi_1 = k_1, \xi_2 = k_2, \ldots, \xi_m = k_m) = n! \frac{p_1^{k_1} p_2^{k_2} \cdots p_m^{k_m} p_f^{k_f}}{k_1! k_2! \cdots k_m! k_f!} \quad (4.a)$$

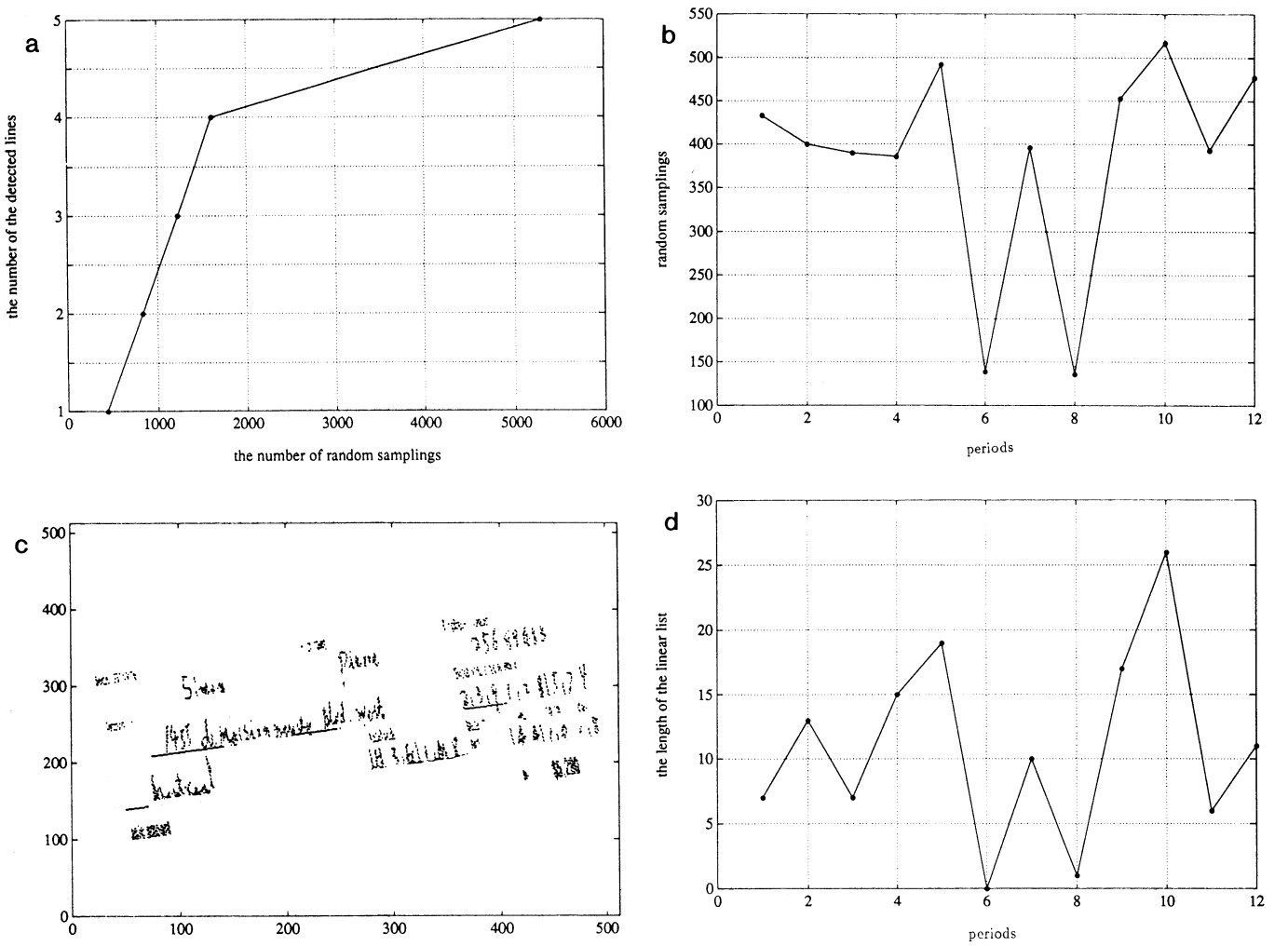

FIG. 7. Line detection for form processing—part (2): (a) the number of detected lines versus the used time. (b) The times spent by each accumulation period. The first four lines were detected in the first four periods. The fifth line was detected in the twelfth period. (c) The remaining parts after removal of pixels of the first detected four lines. (d) The number of the used units in the dynamic linear list by each accumulation period. The average number is 11.

for each $k_i = 0,1, \ldots, n$ and with constraint $\Sigma_1^m k_i = n - k_f$. Its univariate marginal distribution for ξ_i is the *binomial distribution* [21], i.e., for $i = 1, \ldots, m$,

$$P(\xi_i = k_i) = C_n^{k_i} p_i^{k_i}(1 - p_i)^{n-k_i}, \quad k_i = 0, 1, \ldots, n, \tag{4.b}$$

with the expectation and variance being

$$E(\xi_i) = np_i, \quad \sigma^2(\xi_i) = np_i(1 - p_i). \tag{4.c}$$

(2) In an unlimited sequence of GBTs, let η_i denote the total number of trials which precede (and include) the occurrence of the rth "success" of type T_i. Its probability distribution satisfies the *negative Binomial distribution* [21], namely

$$P(\eta_i = k_i) = C_{k_i-1}^{r-1} p_i^r (1 - p_i)^{k_i - r}, \quad k_i = r, r + 1, \ldots, \tag{4.d}$$

with its expectation and variance being

$$E(\eta_i) = \frac{r}{p_i}, \quad \sigma^2(\eta_i) = r\frac{1 - p_i}{p_i^2}. \tag{4.e}$$

We can easily find that each of the simple random trials in the implementing process of an RHT algorithm is just a GBT since the event of "randomly picking n pixels from D and mapping them by the converging mapping into a $p \in P$"[8] has the probabilities p_i, $i = 1, \ldots, m_t + m_{ps}$,

[8] I.e., one execution of lines 20 and 30 for each of the algorithms in Section 4.1.

of having the outcomes O_i, $i = 1, \ldots, m_t + m_{ps}$, respectively, and $p_f = 1 - \sum_{i=1}^{m_t+m_{ps}} p_i$ of having the outcome O^f. The outcomes O_i and O^f have the following interpretations: (1) for $i = 1, \ldots, m_t$, O_i means "the n pixels all come from a true curve c_i"; (2) for $i = m_t + 1, \ldots, m_t + m_{ps}$, O_i means "the n pixels all come from a pseudo curve c_i"; (3) O^f means "the n pixels fail to be mapped into a $p \in P$ since they are non-curve pixels."

The whole implementing process of a RHT algorithm falls into one of two types of combinations of many GBTs described above. The first type, produced only by RHT.b, is a stationary GBP which consists of identical and mutually independent GBTs with the constant probability parameters given by

$$p_i = \frac{n_i(n_i - 1) \cdots (n_i - n + 1)}{N(N - 1) \cdots (N - n + 1)}, \qquad i = 1, \ldots, m_t + m_{ps},$$

and

$$p_f = 1 - \sum_{i=1}^{m_t+m_{ps}} p_i. \tag{5.a}$$

The second type, produced by the other four algorithms in Section 4.1, is more complex. It consists of a number of connected segments of different GBPs. Each segment corresponds to an epoch. Although each epoch may consist of more than one accumulation period,[9] the GBTs within one epoch belong to a same stationary GBP with their probability parameters remaining unchanged since no pixel will be removed out of D until a period is completed. However, after an epoch is finished, the pixels of the present detected curve will be removed out of D. As a result, the probability parameters are changed in the next epoch, which thus produces a different stationary GBP.

In an IMBP with m_t true curves, there will be m_t segments of different stationary GBPs in the whole implementing process of an RHT algorithm. Let epoch(j) denote the jth GBP with its probability parameters $p_i^{(j)}$ and $p_f^{(j)}$ for outcomes O_i and O^f, respectively. We have then

$$p_i^{(j)} = 0, \quad i = 1, \ldots, j - 1,$$

and

$$p_i^{(j)} = \frac{n_i(n_i - 1) \cdots (n_i - n + 1)}{N^{(j)}(N^{(j)} - 1) \cdots (N^{(j)} - n + 1)}, \qquad i = j, \ldots, m_t + m_{ps}, \tag{5.b}$$

$$p_f = 1 - \sum_{i=1}^{m_t-j+1+m_{ps}} p_i^{(j)}, \tag{5.c}$$

where $N^{(j)} = N - \sum_1^{j-1} n_i$ is the total number of pixels in D during epoch(j), with $N^{(1)} = N$. In addition, it is assumed that the true curve detected at the end of epoch(j) is the true curve with the same index j (i.e., c_j). This does not lose any generality since we can re-index all the true curves according to the order in which they are detected.

Before closing Section 5.1, we would like to give some notes which may be useful for understanding how the parameters in Eqs. (5) change with j:

(1) In an IMBP, there may be some pixels which can be considered as belonging to several curves if they are located at the intersection points of these curves. For facilitating later analysis, when we initially count the number n_i of pixels of each curve c_i, each intersection pixel is included if c_i is a pseudo curve, and is not included if c_i is a true curve, although such a pixel will give some positive contribution to the detection of the true curve. This means that we chose the most conservative way for analysis, so that any of the conclusions later obtained will not be influenced by how to count the intersection pixels.

(2) After epoch(j) is finished, the removal of pixels of the detected curve c_j will not only cause a reduction of the number of pixels in D (i.e., $N^{(j+1)} = N^{(j)} - n_j$), but also a reduction of the number of pixels of some pseudo curves since some of the removed pixels may be the intersect pixels of these curves. Thus, the actual values of $p_i^{(j)}$ for $i > m_t$ are smaller than those given by Eq. (5.b).

(3) As j increases, the probability for finding the undetected true curves will increase due to the reduction of $N^{(j)}$. However, for pseudo curves, although the reduction of $N^{(j)}$ may also increase probabilities $p_i^{(j)}$, $i = m_t + 1, \ldots, m_{ps}$, the reduction of the number of intersect pixels may decrease them. So, as the net effect they will not be increased as much as the probabilities for the true curves.

5.2. *Quantitative Analysis on RHT.b Algorithms*

We start our analysis on the basic form of RHT algorithm, namely RHT.b. Obviously, its storage complexity C_s is still N_a^n (i.e., the same as that spent by the HT), and its computing time complexity C_t is $k_{\max} t_{rm}$, where N_a is the size (in one dimension) of the accumulation array and t_{rm} is the average computing time for a random trial (i.e., one excution of lines 20 and 30). Since t_{rm} is independent of both N_a and N (i.e., the size of an image), we see that the selection of $k_{\max}$ is here the key problem which will

[9] For all the algorithms in Section 4.1 except RHT.b, an epoch corresponds to one execution of the loop from line 12 to line 94, and an accumulation period corresponds to one execution of the loop from line 15 to line 80.

not only determine when to stop the algorithm but also the time complexity C_t.

To reduce C_t, we naturally want $k_{\max}$ to be as small as possible. However, $k_{\max}$ should be larger than some value such that the accumulated scores of true curves are certainly greater than those of pseudo curves. Roughly speaking, it should be possible to find such a value since it is assumed that the length of any true curve is larger than any pseudo curve. But what we deal with are $k_{\max}$ random trials which makes the appropriate value of $k_{\max}$ also a random variable, which can only be selected subject to certain stochastic variations.

From Eq. (4.c), we see that for curve c_i the average of its score, $score_i$, accumulated during $k_{\max}$ trials is $k_{\max}p_i$ with p_i given by Eq. (5.a). Thus, regardless of the value of $k_{\max}$, the average scores of true curves will be greater than those of pseudo curves under the assumption that the length of any true curve is larger than any pseudo curve. However, one should remember that this average $k_{\max}p_i$ is subject to the standard derivation $\sqrt{k_{\max}p_i(1-p_i)}$. That is, in a special implementation of RHT.b, there is some probability that the accumulated score of a true curve may be less than that of some pseudo curve, which will produce some difficulty for detecting the true curve. To reduce such kind of probability, we need to suitably select $k_{\max}$ in consideration of the standard derivation $\sqrt{k_{\max}p_i(1-p_i)}$.

From probability theory, we know that score_i will most probably vary within the interval

$$\Delta\text{score}_i = [k_{\max}p_i - c\sqrt{k_{\max}p_i(1-p_i)},\ k_{\max}p_i + c\sqrt{k_{\max}p_i(1-p_i)}], \quad (6.a)$$

where $c > 1$ is a constant. The larger c is, the more unlikely it is that score_i will fall outside Δscore_i.

Since RHT.b is implemented in the batch way, it is not difficult to see that in an IMBP model the true curve $c^t_{\min}$ is most difficult to detect since it has the minimum number $n_{\min}$ of pixels among all the true curves. Denote by $c^{ps}_{\max}$ the pseudo curve that has the maximum number $n^{ps}_{\max}$ of pixels among all the pseudo curves. Let $p_{\min}$ and $p^{ps}_{\max}$ denote the obtained probabilities by replacing n_i in Eq. (5.a) with $n_{\min}$ and $n^{ps}_{\max}$, respectively, and let $\Delta\text{score}_{\min}$ and $\Delta\text{score}^{ps}_{\max}$ denote the corresponding intervals obtained by Eq. (6.a). In some conservative sense, we would like to select $k_{\max}$ such that $\Delta\text{score}_{\min}$ and $\Delta\text{score}^{ps}_{\max}$ are not overlapping; i.e.,

$$k_{\max}p_{\min} - c\sqrt{k_{\max}p_{\min}(1-p_{\min})} > k_{\max}p^{ps}_{\max} + c\sqrt{k_{\max}p^{ps}_{\max}(1-p^{ps}_{\max})} \quad (6.b)$$

and it gives

$$\sqrt{k_{\max}} > \frac{c(\sqrt{p_{\min}(1-p_{\min})} + \sqrt{p^{ps}_{\max}(1-p^{ps}_{\max})})}{(p_{\min} - p^{ps}_{\max})}. \quad (6.c)$$

Further from Eq. (5.a), by assuming that $n_{\min}$ is several times larger than $n^{ps}_{\max}$, i.e., $n_{\min} > \kappa n^{ps}_{\max}$ with $\kappa \gg 1$,[10] we can have approximately

$$k_{\max} > c^2\frac{1-p_{\min}}{p_{\min}} \approx c^2\frac{N^n - n^n_{\min}}{n^n_{\min}}. \quad (6.d)$$

According to the binomial distribution given in Eq. (4.b), we know that for $c > 3 \sim 10$ it will be very rare for score_i to fall out of Δscore_i. Thus, in practice, we could let

$$k_{\max} = (10 \sim 100)\frac{N^n}{n^n_{\min}}. \quad (6.e)$$

In summary, we can conclude that:

For implementing the algorithm RHT.b, *the stopping parameter* $k_{\max}$ *can be selected according to Eqs.* (6d) *and* (6e). *The storage complexity of* RHT.b *is of the order* $O(N^n_a)$, *and it follows from Eq.* (6.e) *that the computing time complexity is approximately of the order* $O(N^n/n^n_{\min})$, *where* $n_{\min}$ *is the minimum number of pixels of a true curve. Usually,* $O(N^n/n^n_{\min})$ *is considerably smaller than* $O(N\,N_a^{(n-1)})$, *the computing time complexity of the conventional* HT, *especially when the relative numbers of pixels of true curves in an image remain constant with N.*

5.3 *Quantitative Analysis on RHT.bsa Algorithms*

As RHT.b, the storage complexity of RHT.bsa is still $C_s = N^n_a$ since it also use an accumulation array. What needs to be analyzed are its computing time comlexity C_t and how to select its stopping parameter $ku_{\max}$. In the algorithm RHT.bsa given in Section 4.1, the implementation is stopped by either line 90 (i.e., by the stopping parameter $ku_{\max}$) or line 95 (i.e., by externally knowing the total number of true curves). In the sequel, we first study its time complexity C_t by assuming that the implementation is stopped by line 95 only (i.e., let $ku_{\max} = \infty$). Then, we further discuss how to stop the implementation using $ku_{\max}$.

The analysis of C_t on the implementing process of RHT.bsa is much more complex than that of RHT.b. As indicated in Section 5.1, the process of RHT.b is just a stationary GBP with constant probability parameters given by Eq. (5.a), while the whole process of RHT.bsa consists of a number of epochs with each epoch(j) being

[10] This is usually true; otherwise it is not reasonable to say that one curve is a true curve and another is a pseudo curve.

a segment of stationary GBP with probability parameters given by Eqs. (5.b) and (5.c). Furthermore, each epoch(j) again may consist of more than one accumulation periods (for convenience, here we use period(j, r) to denote the rth period of epoch(j)).[11]

Let random variables $\eta^{(j)}$, $\eta^{(j,r)}$ denote the total number of GBTs (i.e., the simple random trials) contained in epoch(j) and period (j, r), respectively,[12] and let random variable $n_{ep}^{(j)}$ denote the total number of accumulation periods in epoch (j). Then, on the average we have

$$C_t = t_{rm} \sum_{j=1}^{m_t} E(\eta^{(j)}) + \sum_{j=1}^{m_t} T_v^{(j)},$$
$$T_v^{(j)} = E(n_{ep}^{(j)}) \left(N - \sum_{k=1}^{j-1} n_k \right) t_{ch}, \tag{7}$$

where $E(.)$ denotes expectation, t_{rm} is the same as in Section 5.2, and t_{ch} is the computing time used for checking whether a given pixel belongs to a given curve (i.e., whether it falls in the δ-band of a given curve function, see Section 2.1). Like t_{rm}, t_{ch} is independent of both N and N_a. Moreover, $(N - \sum_{k=1}^{j-1} n_k)t_{ch}$ is the total time needed for checking whether a candidate curve is a true curve under the condition that there have been already $j - 1$ true curves found, and $T_v^{(j)}$ is the total computing time spent on verifying candidate curves for detecting the jth true curve c_j.

It follows from Eq. (7) that the key for estimating C_t is how to estimate $E(\eta^{(j)})$ and $E(n_{ep}^{(j)})$. For clarity, in the sequel we divide the task of estimating C_t into three steps:

(1) *Estimating* $E(\eta^{(j)})$. For a given number k, we can see that the event "$\eta^{(j)} = k$" means "in the kth GBT one of outcomes O_i, $i = 1, \ldots, m_t$ reaches its n_tth success." Let random variables ξ_i be the scores for O_i, and let k_i be a specific value of ξ_i. We have

$$\begin{aligned} P(\eta^{(j)} = k) &= \sum_{i=1}^{m_t} P(k\text{th } GBT \rightarrow O_i) P(\text{in past } k-1 \text{ GBTs}, \\ &\qquad \xi_i = n_t - 1 \text{ and } \forall j \neq i, \xi_j \leq n_t - 1) \\ &= \sum_{i=1}^{m_t} p_i^{(j)} \sum \cdots \sum P(\xi_1 = k_1, \ldots, \xi_{i-1} = k_{i-1}, \\ &\qquad \xi_i = k_i, \xi_{i+1} = k_{i+1}, \ldots, \xi_{m_t} = k_{m_t}), \end{aligned}$$

Where "$\rightarrow$" means "outcomes." Moreover, k_i is fixed (i.e., $k_i = n_t$) and the sums $\sum \cdots \sum$ are over all $k_1 \leq n_t - 1, \ldots, k_{i-1} \leq n_t - 1, k_{i+1} \leq n_t - 1, \ldots, k_{m_t} \leq n_t - 1$ which satisfy the constraint $\sum_{i=1}^{m_t} k_i + k_f = k - 1$.

Each term $P(\xi_1 = k_1, \ldots, \xi_{i-1} = k_{i-1}, \xi_i = k_i = n_t, \xi_{i+1} = k_{i+1}, \ldots, \xi_{m_t} = k_{m_t})$ is described by the *Multinomial Distribution*, i.e., eq. (4.a). So we have the distribution of $\eta^{(j)}$ as follows

$$P(\eta^{(j)} = k) = \sum_{i=j}^{m_t} (k-1)! \frac{p_i^{(j)n_t}}{n_t!} \left(\sum \cdots \sum \frac{p_j^{(j)k_1} \cdots p_{i-1}^{(j)k_{i-1}} p_{i+1}^{(j)k_{i+1}} \cdots p_{m_t}^{(j)k_{m_t}} p_f^{(j)k_f}}{k_j! \cdots k_{i-1}! k_{i+1}! k_{m_t}! k_f!} \right) \tag{8.a}$$

where the sums $\sum \cdots \sum$ are over all $k_j \leq n_t - 1, \ldots, k_{i-1} \leq n_t - 1, k_{i+1} \leq n_t - 1, \ldots, k_{m_t} \leq n_t - 1$ which satisfy the constraint $\sum_{i=j}^{m_t} k_i + k_f = k - 1$, and $p_i^{(j)}$, $i = j, \ldots, m_t$, are given by Eq. (5.b).

Let S_i denote the sum $\sum \cdots \sum$ in the big parentheses of Eq. (8.a), and let S_i' be the same sum but after removing the constraint $\sum_{i=j}^{m_t} k_i + k_f = k - 1$. Then, we have $(k - 1 - n_t)! S_i < (k - 1 - n_t)! S_i' = [1 - p_i^{(j)}]^{k-1-n_t}$. As a result, we can rewrite Eq. (8.a) into

$$\begin{aligned} P(\eta^{(j)} = k) &= \sum_{i=j}^{m_t} p_i^{(j)n_t} \frac{(k-1)!}{(k-1-n_t)! n_t!} S_i \\ &< \sum_{i=j}^{m_t} C_{k-1}^{n_t} p_i^{(j)n_t} (1 - p_i^{(j)})^{k-1-n_t}. \end{aligned} \tag{8.b}$$

The upper bound in Eq. (8.b) tends to 0 when $k \rightarrow \infty$. Thus $P(\eta^{(j)} = k) \rightarrow 0$ when $k \rightarrow \infty$; i.e., epoch(j) has probability zero to be of infinite length. This means that **each epoch is terminable.** Consisting of a finite number of epochs, the whole RHT process then also has probability zero to be of infinite length; in other words, **the implementation process of the RHT process is terminable.**

Noting that $C_{k-1}^{n_t} p_i^{(j)n_t} (1 - p_i^{(j)})^{k-1-n_t}$ in Eq. (8.b) has the same form as that given by Eq. (4.d), we can obtain an upper bound $E(\eta^{(j)}) = \sum_{k=1}^{\infty} kP(\eta^{(j)} = k) < \sum_{i=j}^{m_t} (n_t / p_i^{(j)})$. However, this bound is not so good. In the following, we will find a tighter upper bound in a different way.

For $i = j, \ldots, m_t$, let $\eta_i^{(j)}$ denote the total number of trials which precede (but include) the occurrence of the n_tth success of O_i, and note that $\eta^{(j)}$ is the total number of trials which precede (but include) the occurrence of the n_tth success of anyone of O_i, $i = j, \ldots, m_t$. We then always have $\eta^{(j)} \leq \eta_i^{(j)}$, $i = j, \ldots, m_t$, or $\eta^{(j)} = \text{Min}\{\eta_i^{(j)}, i = j, \ldots, m_t\}$. Moreover, assuming that $E_{\min}(j) = \text{Min}\{E(\eta_i^{(j)}), i = j, \ldots, m_t\}$ and $E_{\min}^2(j) = \text{Min}\{E(\eta_i^{(j)^2}), i = j, \ldots, m_t\}$, we have

$$E(\eta^{(j)}) \leq E_{\min}(j), \quad \sigma^2(\eta^{(j)}) < E(\eta^{(j)^2}) \leq E_{\min}^2(j). \tag{9.a}$$

[11] In the algorithm RHT.bsa given in Section 4.1, a period (j, r) is considered to be completed jointly by line 66 (i.e., there is one score reaching a given number n_t) and by line 78 (i.e., the length of the period reaches a given limit $k_{\max}$). We have already discussed the selection of $k_{\max}$ in Section 5.2. For simplicity, here we only consider the cases that period (j, r) is completed solely by line 66, i.e., we let $k_{\max} = \infty$ in the algorithm given in Section 4.1.

[12] We sometimes also call $\eta^{(j)}$, $\eta^{(j,r)}$ the length of epoch(j) and period (j, r), respectively.

Since each $\eta_i^{(j)}$ satisfies the *negative binomial distribution*, from Eqs. (4.d) and (4.e), we have

$$E(\eta_i^{(j)}) = \frac{n_t}{p_i^{(j)}}, \quad E(\eta_i^{(j)^2}) = n_t \frac{(1 - p_i^{(j)})}{p_i^{(j)^2}} + \frac{n_t^2}{p_i^{(j)^2}}. \quad (9.b)$$

Substituting them into Eq. (9.a) and letting $p_{\max}^{(j)} = \text{Max}\{p_i^{(j)}, i = j, \ldots, m_t\}$, $n_{\max}^{(j)} = \text{Max}\{n_i, i = j, \ldots, m_t\}$, and $n_{\min} = \text{Min}\{n_i, i = 1, \ldots, m_t\}$, it follows from Eq. (5b) that

$$E(\eta^{(j)}) \le \frac{n_t}{p_{\max}^{(j)}} = n_t \prod_{l=0}^{n-1} (N - l - \sum_{1}^{j-1} n_i) / \prod_{l=0}^{n-1} (n_{\max}^{(j)} - l)$$

$$\ll \frac{n_t N^n}{\prod_{l=0}^{n-1} (n_{\min} - l)} \sim O\left(\frac{n_t N^n}{n_{\min}^n}\right) \quad (9.c)$$

$$\sigma^2(\eta^{(j)}) \le \frac{n_t + n_t^2}{p_{\max}^{(j)\,2}} \ll (n_t + n_t^2) \frac{N^{2n}}{\prod_{l=0}^{n-1} (n_{\min} - l)^2}$$

$$\sim O\left(\frac{n_t^2 N^{2n}}{n_{\min}^{2n}}\right). \quad (9.d)$$

(2) *Estimating* $E(n_{ep}^{(j)})$. After a candidate c_a is found in period (j, r), if it is a true curve, then epoch(j) finishes and $n_{ep}^{(j)}$ is just r; otherwise period $(j, r + 1)$ will continue. The key to estimate $E(n_{ep}^{(j)})$ is the probability $p_t^{(j)} = P(c_a$ is a true curve when c_a is a candidate found during epoch$(j))$, since the probability $P(n_{ep}^{(j)} = r)$ is given by $p_t^{(j)} (1 - p_t^{(j)})^{r-1}$ and then we can have

$$E(n_{ep}^{(j)}) = \sum_r r P(n_{ep}^{(j)} = r) = \sum p_t^{(j)} (1 - p_t^{(j)})^{r-1}$$

$$= 1/p_t^{(j)}. \quad (10.a)$$

Assuming that period(j, r) is completed at its kth GBT trial, then the probability for a particular curve c_i being the candidate c_a is $C_{k-1}^{n_t-1} p_i^{(j)n_t} (1 - p_i^{(j)})^{k-n_t}$. Thus, $p_t^{(j)}$ could be roughly approximated by

$$p_t^{(j)} = \frac{\sum_{i=j}^{m_t} C_{k-1}^{n_t-1} p_i^{(j)n_t} (1 - p_i^{(j)})^{k-n_t}}{\sum_{i=j}^{m_t+m_{ps}} C_{k-1}^{n_t-1} p_i^{(j)n_t} (1 - p_i^{(j)})^{k-n_t}} = \frac{1}{1 + L},$$

$$L = \frac{\sum_{i=m_t+1}^{m_t+m_{ps}} p_i^{(j)n_t} (1 - p_i^{(j)})^{k-n_t}}{\sum_{i=j}^{m_t} p_i^{(j)n_t} (1 - p_i^{(j)})^{k-n_t}}. \quad (10.b)$$

Let $p_{mp}^{(j)} = \text{Max}\{p_i^{(j)}, i = m_t + 1, \ldots, m_t\}$, $p_{\min}^{(j)} = \text{Min}\{p_i^{(j)}, i = j, \ldots, m_t\}$. Since for $x > 0$, $x^{n_t}(1 - x)^{k-n_t}$ is monotonically increasing with x and $p_{mp}^{(j)}$ is usually very small, we have

$$L < \frac{\sum_{i=m_t+1}^{m_t+m_{ps}} p_{mp}^{(j)n_t} (1 - p_{mp}^{(j)})^{k-n_t}}{\sum_{i=j}^{m_t} p_{\min}^{(j)n_t} (1 - p_{\min}^{(j)})^{k-n_t}}$$

$$= \frac{m_{ps}(1/p_{\min}^{(j)} - 1)^{n_t}}{(m_t - j + 1)(1/p_{mp}^{(j)} - 1)^{n_t}} \left(1 - \frac{p_{\min}^{(j)} - p_{mp}^{(j)}}{1 - p_{mp}^{(j)}}\right)^{-k}. \quad (10.c)$$

Moreover, $(1 - (p_{\min}^{(j)} - p_{mp}^{(j)})/(1 - p_{mp}^{(j)}))^{-k} \approx (1 - p_{\min}^{(j)})^{-k} \approx (1 + kp_{\min}^{(j)})$ due to $p_{\min}^{(j)}$ usually being also quite small according to Eq. (5.b), where $kp_{\min}^{(j)}$ is just the average score obtained by the shortest true curve. Thus, $kp_{\min}^{(j)} \le n_t$. Then it follows from Eq. (5.b) that

$$L < \frac{m_{ps} p_{mp}^{(j)n_t}}{(m_t - j + 1) p_{\min}^{(j)n_t}} (1 + n_t)$$

$$= (1 + n_t) \frac{m_{ps}}{(m_t - j + 1)} \left(\frac{\prod_{l=0}^{n-1} (n_{\text{pmax}} - l)}{\prod_{l=0}^{n-1} (n_{\min} - l)}\right)^{n_t}, \quad (11.a)$$

where $n_{\min}$ is the length of the shortest true curve, i.e., $n_{\min} = \text{Min}\{n_i, i = 1, \ldots, m_t\}$ and n_{pmax} is the length of the longest pseudo curve, i.e., $n_{\text{pmax}} = \text{Max}\{n_i, i = m_t + 1, \ldots, m_t + m_{ps}\}$.

Consequently from Eqs. (10.a) and (10.b), we approximately have

$$E(n_{ep}^{(j)}) = 1 + L < 1 + \frac{(1 + n_t) m_{ps}}{(m_t - j + 1)} \left(\frac{\prod_{l=0}^{n-1} (n_{\text{pmax}} - l)}{\prod_{l=0}^{n-1} (n_{\min} - l)}\right)^{n_t} \approx 1 + \frac{(1 + n_t) m_{ps}}{(m_t - j + 1)} \rho_{t-ps}^{-nn_t}, \quad (11.b)$$

where $\rho_{t-ps} = n_{\min}/n_{\text{pmax}}$ reflects the ratio of the length of a true curve to the length of a pseudo curve.

(3) *Estimating the computing time complexity* C_t. Finally, by jointly putting Eqs. (9.c) and (11.b) into Eq. (7), we have

$$C_t < t_{rm} \sum_{j=1}^{m_t} \frac{n_t N^n}{n_{\min}^n} + t_{ch} \sum_{j=1}^{m_t} \left(N - \sum_{i=1}^{j-1} n_i\right) \left[1 + \frac{(1 + n_t) m_{ps}}{(m_t - j + 1) \rho_{t-ps}^{nn_t}}\right] < t_{rm} \frac{m_t n_t N^n}{n_{\min}^n} + t_{ch} m_t N + t_{ch} (\ln m_t) N m_{ps} \rho_{t-ps}^{-nn_t} \sim O\left(\frac{m_t n_t N^n}{n_{\min}^n}\right); \quad (12)$$

i.e., the first term dominates the complexity because obviously it dominates the second term and we also consider that the third term is lower than the order of the first term. To make it more clear, divide the third term into two

factors $[t_{ch}(\ln m_t)m_{ps}\rho_{t-ps}^{-n}][N\rho_{t-ps}^{-n(n_t-1)}]$. For the first factor, m_{ps} is bounded by $(C_N^n - \sum_{i=1}^{m_t} C_{n_i}^n)$ which is of the order $O(N^n)$, and ρ_{t-ps} is about the same order as $n_{\min}$ since the length of a true curve is usually much larger than that of a pseudo curve; thus the first factor is about the same order as the first term in Eq. (12). For the remaining factor, when $n_t \geq 2$, usually N is much smaller than $(\rho_{t-ps}^n)^{(n_t-1)}$ since $n \geq 2$ for any curve and ρ_{t-ps} is about the same order as $n_{\min}$. In other words, the second factor could only reduce the order of the first factor. Therefore, it concludes the result given by Eq. (12).

The interesting point is that the above three-step analysis has not only produced an estimate on the time complexity C_t for RHT.bsa, but it also indicated some hints for appropriately selecting parameters n_t and $ku_{\max}$.

From Eq. (12), one can see that a large n_t can reduce the time complexity of the third term such that the whole complexity is controlled by the first term. But on the other hand, a large n_t will also proportionally increase the complexity of the first term, which suggests that n_t could not be too large. According to the above analysis, to let the first term of Eq. (12) dominate, we can select n_t such that $N < (\rho_{t-ps}^n)^{(n_t-1)}$. Then it follows that

$$n_t > 1 + \frac{\ln N}{n(\ln n_{\min} - \ln n_{\text{pmax}})}. \tag{13.a}$$

Furthermore, we know that the average length of epoch(j) is smaller than that given by Eq. (11.b) and this length reaches its maximum for detecting the last true curve (i.e., $j = m_t$). According to this, we could estimate $ku_{\max}$ such that

$$ku_{\max} > (3 \sim 10)\left\{1 + \frac{(1+n_t)m_{ps}}{(m_t - j + 1)}\left(\frac{\prod_{l=0}^{n-1}(n_{\text{pmax}} - l)}{\prod_{l=0}^{n-1}(n_{\min} - l)}\right)^{n_t}\right\}. \tag{13.b}$$

In summary, we can conclude:

For implementing the algorithm RHT.bsa, *the storage complexity is of the order* $O(N_a^n)$, *and it follows from Eq.* (12) *that the computing time complexity is lower than an upper bound which is approximately of the order* $O(m_t n_t N^n / n_{\min}^n)$, *and again it is considerably smaller than the time complexity of the conventional* HT. *In addition, parameters* n_t *and* $ku_{\max}$ *can be selected according to Eqs.* (13.a) *and* (13.b), *respectively.*

In Appendix B of this paper, quantitative analyses have also been made on the other three algorithms given in Appendix A, i.e., RHT.srl, RHT.sql, and RHT.sqh. Roughly speaking, the time complexity of each of these algorithms is bounded by $O(m_t(n_t N^n / n_{\min}^n)^2)$ which is similar to that of RHT.b or RHT.bsa. The storage complexity for RHT.srl and RHT.sql is bounded by $O(n_t N^n / n_{\min}^n)$ which is usually considerably smaller than $O(N_a^n)$—the storage complexity of RHT.b or RHT.bsa. The storage complexity for RHT.sqh is the order of $O(N_h)$, which, as shown in the example given in Section 4.2, is also considerably smaller than $O(N_a^n)$.

6. SUMMARY

The earlier version of the randomized Hough transform (RHT) was a heuristic curve detection approach based on some experimental results [2]. It was proposed by the present authors, inspired by our efforts of using learning techniques of neural networks for curve detection [1, 20]. In this paper, the basic ideas of RHT have been further developed into a more systematic and theoretically supported new method for curve detection. The fundamental framework and the main components of this method have been elaborated. The deep mechanisms behind the advantages of RHT have been exposed by both theoretical analysis and experimental demonstrations. The main differences between RHT and some related techniques have also been elucidated. In comparison with the conventional Hough transform (HT), it has been shown that the RHT has advantages of fast speed, small storage, infinite parameter space, and high resolution. These advantages are due to the consistent combinations of *random sampling, converging mapping, score* (or *evidence*) *accumulation, stepwise implementation* and the appropriate selection of *accumulator structure*, especially the first two of which play key roles responsible for the effectiveness and novelty of RHT.

This paper has also proposed several improved algorithms for implementing RHT for the curve detection problems in noisy images. They were tested by experiments on images with strong random noise, quantization errors as well as non-random interference pixels. The results of these experiments as well as an application example in document analysis have shown that the advantages of RHT are quite robust under these situations. Based on modeling the algorithms by the generalized Bernoulli process, they were analyzed mathematically to estimate their computational complexities and to decide some important parameters for their implementations. It was shown that RHT can have the time and/or storage complexity considerably smaller than the time and storage complexity of HT.

Some further extensions of RHT are possible. Here, we briefly mention two interesting ones. One is to consider the gradient information at each pixel, when using gray level images. This may further speed up the computations. However, how to handle the influence of noise in estimating gradients has to be carefully studied. The other exten-

sion is to generalize the basic ideas of RHT to generalized Hough transform (GHT) to obtain the GRHT method for effectively detecting arbitrary shapes. For example, using the transform formulae proposed by [3],

$$a_1 = x_j - (y_r - w_i)a_4 \sin(a_3) + (x_r - v_i)a_4 \cos(a_3)$$
$$a_2 = y_j - (x_r - v_i)a_4 \sin(a_3) + (y_r - w_i)a_4 \cos(a_3).$$

we can randomly pick a pixel (x_j, y_j) from the binary picture and use each point (w_i, v_i) of a given arbitrary shape to accumulate the array defined on the (a_1, a_2, a_3, a_4) parameter space. Then we find local maxima to get the translation, scaling, and rotation parameters of the detected shapes. This and some improved versions of the GRHT will be reported elsewhere.

APPENDIX A: ALGORITHMS RHT.srl, RHT.sql, AND RHT.sqh

(1) ALGORITHM RHT.srl. The main difference between RHT.srl and RHT.bsa lies in that in RHT.srl a storage P of the simple line list replaces the role of the accumulation array in RHT.bsa. A candidate $\mathbf{a_f}$ is stored in P as an element $p \in P$, p = [**param**(p), score(p)] with **param**(p) = $\mathbf{a_f}$ being directly real values. Here, ".srl" is the combination of the first characters in words "stepwise," "real values," and "line list."

Initialization. Given k_{max}, n_t, ku_{max}. Let $I = 0$, $k = 0$, $ku = 0$, *stop* := *false*, *periodend* = *false*, *epochend* = *false*; and P = *null*.

10 **repeat until** *stop* = *true*
12 **repeat until** *epochend* = *true* **or** *stop* = *true*
15 **repeat until** *periodend* = *true*
20 randomly pick a n-tuple from D;
30 do converging mapping to get its solutions $\mathbf{a_i}$, $i = 1, \ldots, r$
($r = 1$ if only one solution; $r = 0$ if no solution or infinite many solutions.)
40 **for** i=1 **to** r **do**
50 **if** there is an $p \in P$ such that its **param**(p) = $\mathbf{a_i}$
55 **then**
57 score(p) = score(p) + 1;
59 **if** score(p) = n_t **then**
61 let $\mathbf{a_f} = \mathbf{a_i}$ and *periodend* = *true*;
62 remove p from P;
63 **endif**
65 **else**
67 a new element [$\mathbf{a_i}$, 1] is added into P;
69 **endelse**
70 **endfor**
72 $k = k + 1$;
74 **if** $k \geq k_{max}$ **then**
76 find among P a p such that its score(p) is the largest one;
78 let $\mathbf{a_f}$ = **param**(p), *periodend* = *true*
79 **endif**
80 **endrepeat** { an accumulation period is finished }.
84 check whether $\mathbf{a_f}$ represents a true curve by *Definition 1*(b); if yes, *epochend* = *true*;
90 if *epochend* = *false* then $ku = ku + 1$
91 if $ku \geq ku_{max}$ then *stop* = *true*;
93 *periodend* = *false* and $k = 0$;
94 **endrepeat** { an epoch is completed };
95 $I = I + 1$; if $I = m_t$ then *stop* = *true*;
(where m_t is the total number of true curves in an image.)
96 **if** *stop* = *false* **then**
97 let $\mathbf{a_f}$ as a solution; remove from D all the pixels falling in the given δ-band of a_f;
100 let $ku = 0$, *epochend* = *false*, and P = *null*;
102 **endif**
110 **endrepeat** { all the true curves have been detected }.

(2) ALGORITHM RHT.sql. RHT.sql is obtained by adding the following extra line into RHT.srl between lines 40 and 50:

45 let $\mathbf{a}_i$ = round($\mathbf{a}_i/\Delta$) * Δ;

Where Δ is a given quantization rate. That is, instead of directly storing its real values, we store the quantized values of $\mathbf{a}_i$ in P as **param**(p). Here, ".sql" is the combination of the first characters in words "stepwise," "quantized values," and "line list."

The key advantage of this modification is that RHT.sql not only retains all the advantages of RHT.srl, but it also avoids its disadvantage of being only suitable for low-noise images. Like RHT.bsa, where $\mathbf{a}_i$ is stored implicitly after being quantized into the coordinates of accumulation array, RHT.sql also stores $\mathbf{a}_i$ in a quantized form. Thus, as RHT.bsa, RHT.sql is also suitable for images with strong noise of various types. As a result, RHT.sql combines the advantages of both RHT.srl and RHT.bsa.

(3) ALGORITHM RHT.sqh. RHT.sqh has the same pseudo program codes as RHT.sql given above except that in RHT.sqh the structure of P is a mixture of two hash tables and one linear list (see Fig. 8) instead of only one simple linear list as used in RHT.sql. Here, ".sqh" is the combination of the first characters in words "stepwise," "quantized values," and "hash table."

The size of both the two hash tables we used here is N_h. The size of the linear list is dynamic. It starts from zero and increases as some elements are stored in the list. When *line 50* is implemented, first a key derived from the quantized $\mathbf{a}_i$ is used to access a unit of Hash Table 1. If the unit is unoccupied, then store this $\mathbf{a}_i$ in it; if the unit is occupied by a p with **param**(p) = $\mathbf{a}_i$, then let score(p) = score(p) + 1; otherwise, another key is derived from the first key and the quantized $\mathbf{a}_i$ and then used to access a unit of Hash Table 2 in the same way as accessing a unit of Hash Table 1. If there is still no unit for accommodating $\mathbf{a}_i$, then one further accesses the linear list from its first element in the same way as used in RHT.sql. There are several ways to design hashing functions for keys, see [12] for details.

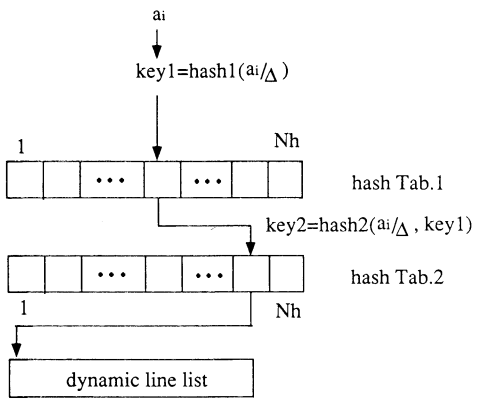

FIG. 8. The structure of P for algorithm RHT.sqh. It consists of two hash tables of size N_h and a dynamic linear list. First, Table 1 is accessed by key1. If it does not succeed, Table 2 is accessed by key2. If it still does not succeed, the dynamic linear list is accessed by enumeration.

APPENDIX B: QUANTITATIVE ANALYSIS ON ALGORITHMS RHT.srl, RHT.sql, AND RHT.sqh

(1) ALGORITHM RHT.srl. As indicated in Section 4.1, the key difference from RHT.srl to RHT.bsa lies in that the accumulation array is replaced by a dynamic linear list P. Due to this difference, both the storage and time complexities of RHT.srl are also different from those of RHT.bsa.

First, the storage complexity is no longer N_a^n, but the length ξ_L of the dynamic list. ξ_L is a random variable, hence we use $E(\xi_L)$ for describing the storage complexity in the average sense. In the implementation of RHT.srl, we have $\xi_L \le \eta^{(j)}$ (as in Section 5.3, it is the length of epoch(j)) for each epoch(j), since in the worst case each trial of epoch(j) produces a different point in the parameter space and thus needs an element of P for storing it. Consequently, we have $E(\xi_L) \le E(\eta^{(j)})$ for epoch(j). Moreover, because the list is reset to null after each epoch finishes, for the whole implementing process it follows from Eq. (9.c) that

$$E(\xi_L) \le \mathrm{Max}_j E(\eta^{(j)}) \ll \frac{n_t N^n}{\prod_{l=0}^{n-1}(n_{\min} - l)} \sim O\left(\frac{n_t N^n}{n_{\min}^n}\right). \quad \text{(B1)}$$

In addition, for an IMBP model, the maximum number of the different points mapped in the parameter space are $m_t + m_{ps}$, which also gives an upper bound for ξ_L. By combining the bound into Eq. (B1), we further have

$$E(\xi_L) \ll \mathrm{Min}\left\{m_t + m_{ps}, \frac{n_t N^n}{\prod_{l=0}^{n-1}(n_{\min} - l)}\right\} \sim \mathrm{Min}\left\{O(m_t + m_{ps}), O\left(\frac{n_t N^n}{n_{\min}^n}\right)\right\}. \quad \text{(B2)}$$

Second, we also need to modify the time complexity given in Eq. (7) into

$$C_t = \sum_{j=1}^{m_t} E(\eta^{(j)})(t_{rm} + t_{ser}) + \sum_{j=1}^{m_t} T_v^{(j)}, \quad \text{(B3)}$$

where t_{ser} is the average time for searching the dynamic list P at each trial. When P is searched by the simple

enumeration, t_{ser} is about the same order of $E(\xi_L)$, i.e., $t_{ser} \sim O(E(\xi_L))t_{com}$ with t_{com} being the computing time for comparing two given parameter points. As a result, from Eqs. (7), (12), and (B2), we see that the time complexity of RHT.srl will increase by a multiplying factor $O(E(\xi_L))$, and in the worst case the complexity will come up to

$$C_t \sim O\left(m_t\left(\frac{n_t N^n}{n^n_{\min}}\right)^2\right). \tag{B4}$$

(2) ALGORITHM RHT.sql. RHT.sql is basically the same as RHT.srl except that each parameter point is stored after quantization to achieve tolerance for noise. The storage and time complexities of RHT.sql are the same as those of RHT.srl given by Eqs. (B2) and (B4).

(3) ALGORITHM RHT.sqh. The use of hash tables for replacing the linear list as P is the key feature of RHT.sqh. For an appropriately designed hashing function, the average search time t_{ser} used in hash tables is a constant and independent of the size of hash tables (e.g., see [18, 19]). Thus the disadvantage of RHT.srl and RHT.sql caused by t_{ser} has been avoided, and the computing time complexity is no longer given by Eq. (B4), but again given by Eq. (12); i.e., it is the same as that of RHT.bsa. The storage complexity of RHT.sqh is also different from that given by Eq. (B2). When the length N_h of hash tables in Fig. 6 is appropriately selected, its storage complexity is approximately $2N_h$; i.e., the order of $O(N_h)$, which, as shown in the example given in Section 4.2, is usually considerably smaller than N^n_a.

We can observe that RHT.sqh has combined the advantage of RHT.bsa on the time complexity and the advantage of RHT.srl and RHT.sql on the storage complexity. This favorable feature makes RHT.sqh preferable in applications having constraints on both time and storage complexities.

ACKNOWLEDGMENTS

The authors thank Mr. Pekka Kultanen, Dept. Information Technology, Lappeenranta University of Technology, for discussions on the early versions of the RHT method, and Mr. Xinming Yu, Dept. Computer Science, Concordia University, for suggesting the possibility of using a hash table to replace the linear list as the storage structure P. The authors also thank the reviewers' comments which inspired many improvements on the earlier manuscript of this paper.

REFERENCES

1. L. Xu and E. Oja, *Extended Self-Organizing Map for Curve Detection,* Research Report No. 16, Dept. of Information Technology, Lappeenranta University of Technology, Sept. 1989.
2. L. Xu, E. Oja, and P. Kultanen, A new curve detection methods: Randomized Hough transform (RHT), *Pattern Recognit. Lett.* **11**(5), 1990, 331–338.
3. J. Illingworth and J. Kittler, A survey of the Hough transform, *Comput. Vision Graphics Image Process.* **43,** 1988, 221–238.
4. J. Illingworth and J. Kittler, The adaptive Hough transform, *IEEE Trans. Pattern Anal. Mach. Intell.* **9,** 1987, 690–698.
5. T. Risse, Hough transformation for line recognition: Complexity of evidence accumulation and cluster detection, *Comput. Vision Graphics Image Process.* **46,** 1989, 327–345.
6. P. V. C. Hough, *Method and Means for Recognizing Complex Patterns,* U.S. Patent 3069654, Dec. 18, 1962.
7. A. Rosenfeld, *Picture Processing by Computer,* Academic Press, New York, 1969.
8. R. O. Duda and P. E. Hart, Use of the Hough transform to detect lines and curves in pictures, *Comm. Assoc. Comput. Mach.* **15**(1), 1972, 11–15.
9. C. D. Kimme, D. H. Ballard, and J. Sklansky, Finding circles by an array of accumulators, *Comm. Assoc. Comput. Mach.* **18**(2), 1975, 120–122.
10. P. M. Merlin and D. J. Farber, A parallel mechanism for detecting curves in pictures, *IEEE Trans. Comput.* **C-24,** 1975, 96–98.
11. D. H. Ballard, Generalizing the Hough transform to detect arbitrary shapes, *Pattern Recognit.* **13**(2), 1981, 111–122.
12. L. Xu *et al., Improved RHT Algorithms for Detecting Curves in Noisy Images,* Technical Report, Dept. of Computer Science, Concordia University, March 1991.
13. S. Tsuji and F. Matsumoto, Detection of elliptic and linear edges by searching two parameter spaces, in *Proceedings of 5th IJCAI, Vol. 2, 1977,* pp. 700–705.
14. M. A. Fischler and R. C. Bolles, Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography, *Comm. Assoc. Comput. Mach.* **24,** 1981, 381–395.
15. M. A. Fischler and O. Firschein, Parallel guessing: A strategy for high-speed computation, *Pattern Recognit.* **20,** 1987, 257–263.
16. S. N. Srihari, Document image understanding, in *Proceedings, IEEE Comp. Soc. Fall Joint Computer Conf., 1986,* pp. 87–96.
17. S. C. Hinds *et al.,* A document skew detection method using run-length encoding and the Hough transform, in *Proceedings, 10th ICPR, Vol. 1, 1990,* pp. 464–468.
18. E. Horowitz and S. Sahni, *Fundamentals of Data Structures,* Pitman, New York/London 1976.
19. D. E. Knuth, *The Art of Computer Programming, Vol. 3, Sorting and Searching,* Addison-Wesley, Reading, MA, 1973.
20. L. Xu, E. Oja, and C. Y. Suen, Modified Hebbian learning for curve and surface fitting, *Neural Networks* **5,** 1992, 393–409.
21. M. Dwass, *Probability and Statistics,* Chap. 6, Benjamin, New York, 1970.
22. W. E. L. Grimson and D. P. Huttenlocher, On the sensitivity of the Hough transform for object recognition, *IEEE Trans. Pattern Recognit. Mach. Intell.* **PAMI-12,** 1990, 255–274.
23. W. E. L. Grimson, *Object Recognition by Computer: The Role of Geometric Constraints,* Chap. 11, MIT Press, Cambridge, MA, 1990.