Real-Time Line Detection Through an Improved Hough Transform Voting Scheme

Leandro A. F. Fernandes, Manuel M. Oliveira

Universidade Federal do Rio Grande do Sul
Instituto de Informática - PPGC - CP 15064
91501-970 - Porto Alegre - RS - BRAZIL
Tel.: +55 (51) 3316-6161
Fax: +55 (51) 3316-7308

Abstract

The Hough transform is a popular tool for line detection due to its robustness to noise and missing data. However, the computational cost associated to its voting scheme has prevented software implementations to achieve real-time performance, except for very small images. Many dedicated hardware designs have been proposed, but such architectures restrict the image sizes they can handle. We present an improved voting scheme for the Hough transform that allows a software implementation to achieve real-time performance even on relatively large images. Our approach operates on clusters of approximately collinear pixels. For each cluster, votes are cast using an oriented elliptical-Gaussian kernel that models the uncertainty associated with the best-fitting line with respect to the corresponding cluster. The proposed approach not only significantly improves the performance of the voting scheme, but also produces a much cleaner voting map and makes the transform more robust to the detection of spurious lines.

Key words: Hough transformation, real-time line detection, pattern recognition, collinear points, image processing

Email addresses: laffernandes@inf.ufrgs.br (Leandro A. F. Fernandes), oliveira@inf.ufrgs.br (Manuel M. Oliveira).

URLs: http://www.inf.ufrgs.br/~laffernandes (Leandro A. F. Fernandes), http://www.inf.ufrgs.br/~oliveira (Manuel M. Oliveira).

1 Introduction

Automatic detection of lines in images is a classic problem in computer vision and image processing. It is also relevant to computer graphics in applications such as image-based modeling [1,2] and user-interfaces [3]. In vision and image processing, line detection is a fundamental primitive in a wide range of applications including camera calibration [4], autonomous robot navigation [5], industrial inspection [6,7], object recognition [8] and remote sensing [9]. The Hough transform (HT) [10,11] is an efficient tool for detecting straight lines in images, even in the presence of noise and missing data, being a popular choice for the task. By mapping each feature pixel to a set of lines (in a parameter space) potentially passing through that pixel, the problem of identifying line patterns in images can be converted into the simpler problem of identifying peaks in a vote map representing the discretized parameter space. Although conceptually simple and despite the efforts of many researchers in using hierarchical approaches [12,13], affine-transformation-based approaches [14] and FFT architectures [15], real-time performance has only been achieved with the use of custom-designed hardware [16–18]. Moreover, the peak-detection procedure may identify spurious lines that result from vote accumulation from non-collinear pixels. This situation is illustrated in Fig. 1 (c), where redundant lines have been detected, while some well-defined ones have been missed.

This paper presents an efficient voting scheme for line detection using the Hough transform that allows a software implementation of the algorithm to perform in real time on a personal computer. The achieved frame rates are significantly higher than previously known software implementations and comparable or superior to the ones reported by most hardware implementations for images of the same size. The proposed approach is also very robust to the detection of spurious lines. Fig. 1 (d) shows the result obtained by our algorithm applied to the input image shown in Fig. 1 (a). These results are clearly better than the ones obtained with the use of the state-of-the-art Hough transform technique shown in Fig. 1 (c). For instance, note the fine lines on the bottom-left corner and on the upper-left portion of the image in (d). Those results are achieved by avoiding the brute-force approach of one pixel voting for all potential lines. Instead, we identify clusters of approximately collinear pixel segments and, for each cluster, we use an oriented elliptical-Gaussian kernel to cast votes for only a few lines in parameter space. Each Gaussian kernel models the uncertainty associated with the best-fitting line for its corresponding cluster. The kernel dimensions (footprint) increase and its height decreases as the pixels get more dispersed around the best-fitting line, distributing only a few votes in many cells of the parameter space. For a well aligned group (cluster) of pixels, the votes are concentrated in a small region of the voting map. Fig. 2 (b) and (d) show the parameter spaces associated with the set of pixels shown in Fig. 2 (a). Since the influence of the kernels is restricted

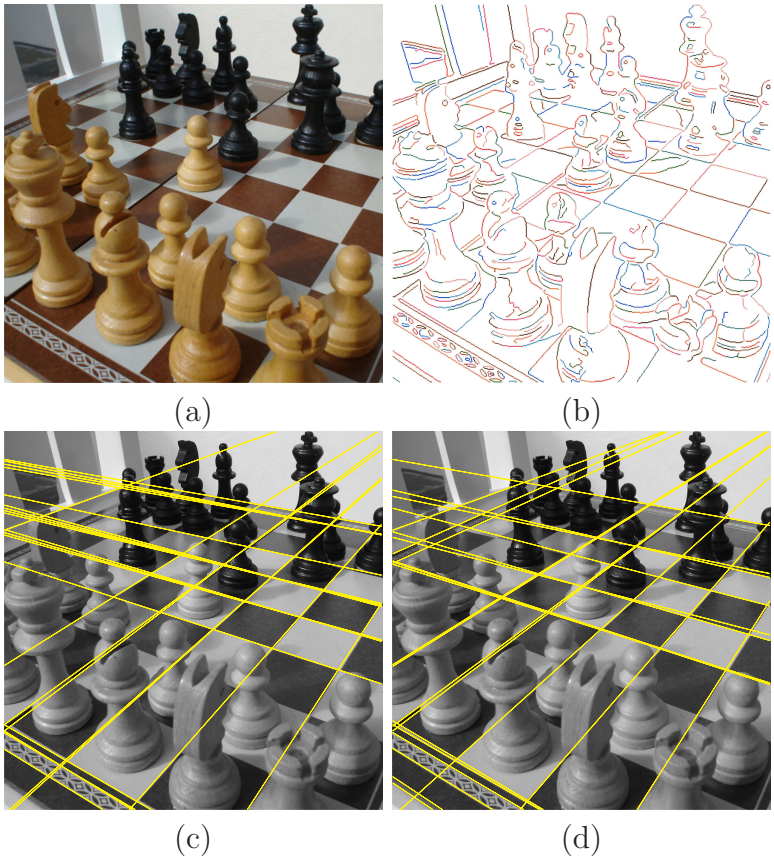

Fig. 1. Line detection using the Hough transform (HT). (a) Input image with 512×512 pixels. Note the folding checkerboard (see the dark folding line on the left of the clear king). Since the checkerboard halves are not properly aligned (level), its rows define pairs of slightly misaligned lines that cross at the folding line. (b) Groups of pixels used for line detection, identified using a Canny edge detector [19]. Note that the folding line produces two parallel segments. (c) Result produced by the gradient-based HT at 9.8 fps. Note the presence of several concurrent lines (not only two) along the rows, while some important ones are missing. (d) Result produced by the proposed approach at 52.63 fps. Note the fine lines detected at the bottom-left corner and at the upper-left portion of the image. Images (c) and (d) show the 25 most-relevant detected lines.

to smaller portions of the parameter space, the detection of spurious lines is avoided.

The central contribution of this paper is an efficient voting procedure for detection of lines in images using the Hough transform. This approach allows a software implementation to perform in real time, while being robust to the detection of spurious lines.

The remaining of the paper is organized as follows: Section 2 discusses some related work. Section 3 describes the proposed approach, whose results, advantages and limitations are discussed in Section 4. Section 5 summarizes the paper and points some directions for future exploration.

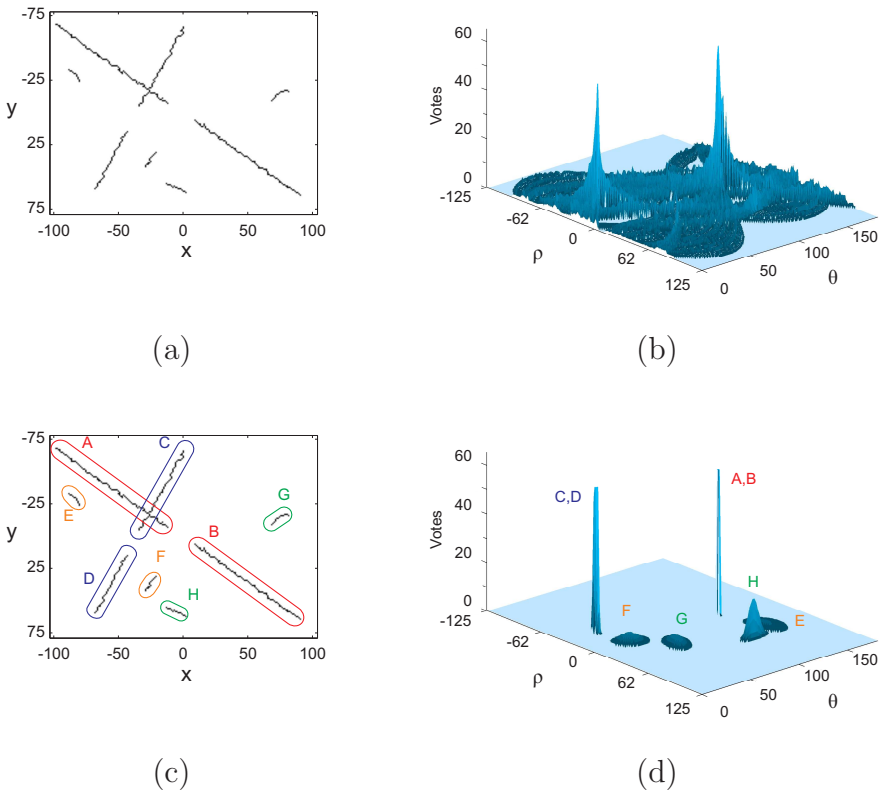

Fig. 2. The conventional versus the proposed voting procedure. (a) A simple image containing approximately straight line segments. (b) A 3D visualization of the voting map produced by the conventional voting scheme. (c) Clustering of pixels into segments. (d) A 3D visualization of the voting map produced by the proposed technique, which is much cleaner than the one shown in (b). The letters indicate the segments that voted for each of the peaks. To improve the visualization, the height of the vertical axis was truncated to 60 votes. The peaks A-B and C-D received 638 and 380 votes, respectively.

2 Related Work

Hough [10] exploited the point-line duality to identify the supporting lines of sets of collinear pixels in images. In his approach, pixels are mapped to lines in a discretized 2D parameter space using a slope-intercept parameterization. Each cell of the parameter space accumulates the number of lines rasterized over it. At the end, the cells with the largest accumulated numbers (votes) represent the lines that best fit the set on input pixels. Unfortunately, the use of slope-intercept requires a very large parameter space and cannot represent vertical lines. Chung et al. [14] used an affine transformation to improve memory utilization and accelerate Hough's algorithm. However, the results are not real-time and, like the original one, the algorithm cannot handle (almost) vertical lines.

Algorithm 1 Conventional voting process of the Hough transform

Require: I {*Binary image*}
Require: δ {*Discretization step for the parameter space*}
$Votes \leftarrow 0$ {*Initialization of the voting matrix*}
for each feature pixel $I(x, y)$ **do**
  for $0° \leq \theta < 180°$, using a δ discretization step **do**
    $\rho \leftarrow x\cos(\theta) + y\sin(\theta)$
    $Votes\,(\rho, \theta) \leftarrow Votes\,(\rho, \theta) + 1$
  end for
end for

Duda and Hart [11] replaced the slope-intercept with an angle-radius parameterization based on the normal equation of the line (Eq. 1), naturally handling vertical lines and significantly reducing the memory requirements of the algorithm.

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

Here, x and y are the coordinates of the pixel in the image, ρ is the distance from the origin of the image's coordinate system to the line, and θ is the angle between the image's x-axis and the normal to the line. By restricting θ to $[0°, 180°)$ and ρ to $[-R, R]$, where $R = \sqrt{w^2 + h^2}/2$, and w and h are the width and height of the image, respectively, all possible lines in the image have a unique representation in the parameter space. In this case, each pixel is mapped to a sinusoidal line. Assuming that the origin of the image coordinate system is at the center of the image, with the x-axis growing to the right and the y-axis growing down (Fig. 4, left), Algorithm 1 summarizes the voting process for Duda's and Hart's Hough transform. Fig. 2 (b) shows a 3D visualization of the voting map for the simple example shown in Fig. 2 (a). The two peaks represent the two main lines approximated by larger pixel segments in the image. One should notice that the discretization of the image and of the parameter space cause the sinusoidal lines to 'intersect' at many cells, creating secondary peaks. This effect is further enhanced by the fact that the pixels in each segment are not exactly collinear. As a result, the peak detection step of the algorithm may retrieve some spurious lines, as shown in Fig. 1 (c). In the proposed approach, the per-pixel brute-force voting scheme described in Algorithm 1 is replaced by a per-segment voting strategy, where the votes for each segment are concentrated on an elliptical region (in parameter space) defined by the quality of a line fitting to the set of pixels in the segment. This situation is illustrated in Fig. 2 (c) and (d). Notice that by restricting the votes to smaller regions of the parameter space, we avoid the detection of spurious lines.

Ballard [20] generalized the Hough transform to detect arbitrary shapes and used the orientation of the edges into account to improve accuracy and reduce

false positives. For straight lines, the use of the normal parameterization with the gradient information (orientation) to limit the range of θ over which the associated ρ values should be calculated [21] has become the universally used version of the Hough transform. Although in spirit this approach is the closest to ours, it still quite different and casts too many votes.

Many researchers have proposed variations and extensions to the HT. Illingworth and Kittler [22] and Leavers [23] provide in-depth discussions of many of these algorithms. All these variations, however, remained computationally intensive and unable to execute in real time on current general-purpose processors. This has inspired the design of dozens of specialized architectures for HT [17,18], as well as the use, for this purpose, of graphics hardware [8], specialized image-processing hardware [24], and FFT architectures [15]. Albanesi et al. [17] presents a comprehensive survey of specialized architectures for real-time Hough transform. As they point out, ranking these architectures is a difficult task because the various proposals use different kinds of parameterizations, different quantizations of the parameter space, and even different image sizes. Out of thirty four analyzed architectures, only three were capable of handling images with dimensions of 1024×1024 or bigger [25,16] and only a few can handle 512×512 images.

The proposed optimization for the Hough transform voting scheme allows a software implementation running on a PC to exhibit real-time performance, comparable or superior to the ones obtained by most specialized hardware for HT.

3 The Kernel-Based Voting Algorithm

As mentioned in the previous section, the biggest source of inefficiency and false positives in the conventional Hough transform is its brute force voting scheme. Assuming the existence of an ideal transformation that maps pixels to a continuous parameter space, one would notice, in the voting map, a set of main peaks surrounded by smaller ones. The main peaks would correspond to lines that better represent the image lines, whereas the secondary peaks would result from the uncertainty due to the discretization of the image space and to the fact that the feature pixels might not be exactly collinear. This suggests a simple and natural way of casting votes: (i) starting from a binary image, cluster approximately collinear feature pixels, (ii) for each cluster, find its best fitting line and model the uncertainty of the fitting, and (iii) vote for the main lines using elliptical Gaussian kernels computed from the lines associated uncertainties. A flowchart illustrating the proposed line detection pipeline is shown in Fig. 3. Next, we present a detailed explanation of each stage of this process.

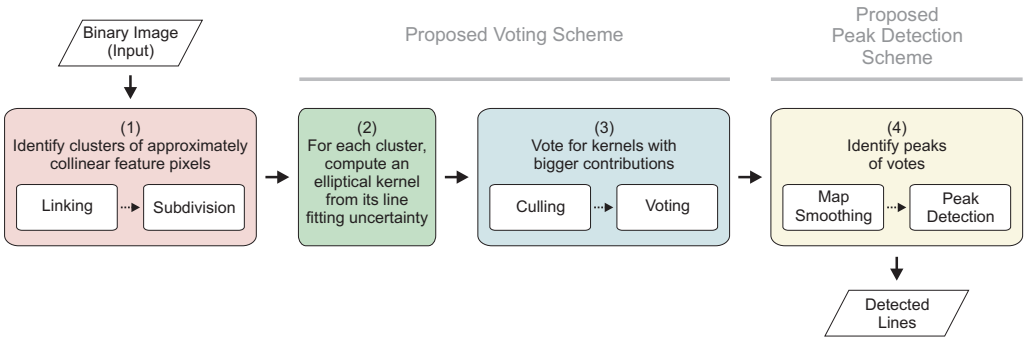

Fig. 3. Flowchart for proposed line detection pipeline. First, feature pixels from an input binary image are grouped into clusters of approximately collinear pixels (1). Then the best-fitting line for each of the clusters is found and its uncertainty used to define elliptical kernels in the parameters space (2). A culling strategy is used to avoid casting votes for kernels whose contributions are negligible (3). Finally, the resulting voting map is smoothed before peak detection identifies the most significant straight lines (4).

In order to identify clusters of approximately collinear feature pixels in the binary image (which has been obtained using a Canny edge detector [19] plus thresholding and thinning), we link strings of neighbor edge pixels [26] and then subdivide these strings into sets of most perceptually significant straight line segments [27]. We use the linking procedure implemented by the VISTA image processing library [26], which consists of starting with a seed pixel and creating a chain of connected feature pixels. A pseudo code for this algorithm is presented in Appendix 1. The subdivision of the resulting chains is performed using the technique described in [27], which consists of recursively splitting a string at its most distant pixel from the line segment defined by the string end points. A string split is performed whenever the new sub-strings are better fit by the segments defined by their end points than is the parent string itself, given the constraints that each resulting sub-string might have at least so many pixels (an informed parameter). Thus, each group of almost collinear feature pixels is composed by the pixels in the sub-string between the end points of its defining segment. Although both the linking and subdivision techniques used here ([26] and [27], respectively) are well known, to the best of our knowledge, their combination has not been used previously. Clusters of feature pixels are illustrated in Fig. 2 (c), where each letter is associated with one recovered segment. Notice, however, that different segments have different numbers of pixels, as well as have different degrees of dispersion around their corresponding best-fitting lines. The smaller the dispersion, the more concentrated the votes should be on the voting map. This property is modeled using elliptical-Gaussian kernels, where the dimensions and the height of the kernels are computed based on the uncertainties of the corresponding segments. Therefore, the footprint of a kernel associated with a more collinear cluster of pixels will be smaller than for a more disperse one, thus concentrating more votes on a smaller portion of the voting map. The details about how to

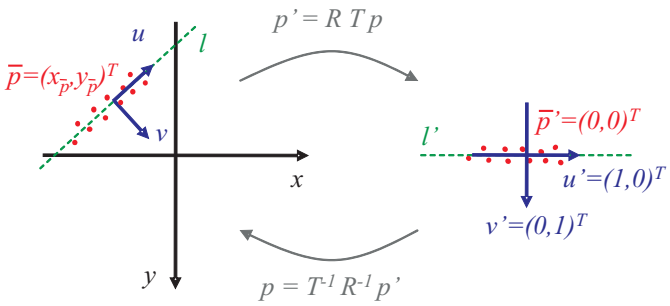

Fig. 4. Coordinate system used to avoid vertical lines during line fitting using linear regression. Original cluster and its eigenvectors u and v (left). The new coordinate system is defined by $\bar{p}$, the centroid of the pixels (new origin), and by the rotated eigenvectors u' and v' (right). The best-fitting line is guaranteed to be horizontal.

compute these kernels are presented next.

3.1 Computing the Elliptical Kernels

Given a cluster S of approximately collinear pixels in an image, its votes are cast around the cell (in parameter space) that corresponds to the cluster's best-fit line l. l is computed from $\bar{p} = (x_{\bar{p}}, y_{\bar{p}})^T$, the centroid of the pixel distribution, and $u = (x_u, y_u)^T$, its eigenvector with the largest eigenvalue. This situation is illustrated on the left part of Fig. 4. $v = (x_v, y_v)^T$, the second eigenvector, is perpendicular to l, allowing one to write:

$$Ax + By + C = x_v x + y_v y - (x_v x_{\bar{p}} + y_v y_{\bar{p}}) = 0 \tag{2}$$

Comparing Eq. (1) and (2), one obtains the values for ρ and θ:

$$\begin{aligned} \rho &= -C & &= x_v x_{\bar{p}} + y_v y_{\bar{p}} \\ \theta &= \cos^{-1}(A) & &= \cos^{-1}(x_v) \end{aligned} \tag{3}$$

Since $\theta \in [0°, 180°)$, $sin(\theta) \geq 0$. Also from Eq. (1) and (2), one has $sin(\theta) = y_v$. Therefore, we need to enforce that $y_v \geq 0$ so that the coordinate system and the limits of the parameter space match the ones defined by [11].

The footprint size used for casting votes for S around (ρ, θ) is given by the uncertainty associated to the fitting of l to S. This uncertainty is represented by the variances σ_ρ^2 and σ_θ^2, which are used to define the shape and extension of the corresponding elliptical-Gaussian kernel G. The orientation of the kernel is given by the covariance $\sigma_{\rho\theta}$. Clusters formed by fairly collinear pixels will present small variances, thus concentrating their votes in small regions of the parameter space, such as clusters A, B, C and D in Fig. 2 (d). Clusters

presenting bigger variances will spread their votes over larger regions of the parameter space, such as clusters E, F and G.

One way to compute the variances and covariance for ρ and θ is to rewrite Eq. (1) as

$$y = mx + b = -\frac{\cos(\theta)}{\sin(\theta)}x + \frac{\rho}{\sin(\theta)} \tag{4}$$

and use linear regression to estimate the coefficients m and b, as well as their variances σ_m^2 and σ_b^2, and covariance σ_{mb}, from which σ_ρ^2, σ_θ^2 and $\sigma_{\rho\theta}$ can be computed. However, Eq. (4) is not defined when $\sin(\theta) = 0$. Moreover, the quality of the fitting by linear regression decreases as the lines approach the vertical direction [28]. We avoid these problems by mapping each cluster to a new coordinate system, whose origin is defined by $\bar{p}$ and whose horizontal axis is given by the rotated eigenvector u'. This transformation is depicted in Fig. 4 and guarantees that the best-fitting line is always horizontal.

Eq. (5) and (6) show how to map pixels and lines, respectively, to the new coordinate system. Note that the coefficients of the line equations need to be transformed using the inverse transpose of the operation applied to the pixels. This situation is analogous to the transformations of vertices and normal vectors performed by the graphics pipeline (the vector (A, B) is perpendicular to l).

$$p_i' = \begin{pmatrix} x_{p_i'} \\ y_{p_i'} \\ 1 \end{pmatrix} = RTp_i \tag{5}$$

$$l' = \begin{pmatrix} A' \\ B' \\ C' \end{pmatrix} = \left((RT)^{-1}\right)^T l = R\left(T^{-1}\right)^T l \tag{6}$$

Here, p_i are the pixel coordinates represented in homogeneous coordinates, and $l = (A, B, C)^T$ (Eq. 2). R and T are the following rotation and translation matrices:

$$R = \begin{pmatrix} x_u & y_u & 0 \\ x_v & y_v & 0 \\ 0 & 0 & 1 \end{pmatrix}, T = \begin{pmatrix} 1 & 0 & -x_{\bar{p}} \\ 0 & 1 & -y_{\bar{p}} \\ 0 & 0 & 1 \end{pmatrix} \tag{7}$$

Applying linear regression to the set of n transformed pixels from cluster S, one can estimate m', b' and their associated variances and covariance [28]:

$$\sigma_{m'}^2 = \frac{S_\sigma}{\Delta}, \sigma_{b'}^2 = \frac{S_{x^2}}{\Delta}, \sigma_{m'b'} = \frac{S_x}{\Delta} \tag{8}$$

where

$$\begin{aligned} &S_\sigma = \textstyle\sum_{i=0}^{n-1} \frac{1}{\sigma_i^2},\ S_{x^2} = \sum_{i=0}^{n-1} \frac{\left(x_{p_i'}\right)^2}{\sigma_i^2}, \\ &S_x = \textstyle\sum_{i=0}^{n-1} \frac{x_{p_i'}}{\sigma_i^2},\ \Delta = S_\sigma S_{x^2} - S_x^2 \end{aligned} \tag{9}$$

σ_i is the uncertainty associated to the v' coordinate of the transformed pixel p_i' (Fig. 4, right). Assuming that this uncertainty is ± 1 pixel (due to image discretization), then $\sigma_i^2 = 1$. Also, since $\bar{p'} = (0,0)^T$ (Fig. 4, right), $S_x = 0$, and Eq. (8) is simplifies to

$$\sigma_{m'}^2 = \frac{1}{\sum_{i=0}^{n-1} \left(x_{p_i'}\right)^2}, \sigma_{b'}^2 = \frac{1}{n}, \sigma_{m'b'} = 0 \tag{10}$$

Note that Eq. (10) only uses the $x_{p_i'}$ coordinate of p_i'. Therefore, Eq. (5) only needs to be solved for $x_{p_i'}$. By inverting Eq. (6) and expressing A', B' and C' in terms of m' and b', one gets:

$$\begin{aligned} l = \begin{pmatrix} A \\ B \\ C \end{pmatrix} &= \left(R\left(T^{-1}\right)^T\right)^{-1} l' = T^T R^T \begin{pmatrix} -m' \\ 1 \\ -b' \end{pmatrix} \\ &= \begin{pmatrix} x_v - x_u m' \\ y_v - y_u m' \\ \left(x_u x_{\bar{p}} + y_u y_{\bar{p}}\right) m' - x_v x_{\bar{p}} - y_v y_{\bar{p}} - b' \end{pmatrix} \end{aligned} \tag{11}$$

From Eq. (11), ρ and θ can be expressed in terms of m', b', $\bar{p}$, and the eigenvectors u and v:

$$\begin{aligned} \rho = -C \quad &= x_v x_{\bar{p}} + y_v y_{\bar{p}} + b' - \left(x_u x_{\bar{p}} + y_u y_{\bar{p}}\right) m' \\ \theta = \cos^{-1}(A) &= \cos^{-1}(x_v - x_u m') \end{aligned} \tag{12}$$

Finally, the variances and covariance for ρ and θ can be estimated from the variances and covariance of m' and b' using a first order uncertainty propagation analysis [29]:

$$\begin{pmatrix} \sigma_\rho^2 & \sigma_{\rho\theta} \\ \sigma_{\rho\theta} & \sigma_\theta^2 \end{pmatrix} = \nabla f \begin{pmatrix} \sigma_{m'}^2 & \sigma_{m'b'} \\ \sigma_{m'b'} & \sigma_{b'}^2 \end{pmatrix} \nabla f^T \tag{13}$$

where ∇f is the Jacobian of Eq. (12), computed as shown in Eq. (14). Given the Jacobian, one can evaluate Eq. (13) taking into account the fact that both m' and b' are equal to zero (*i.e.*, the transformed line is horizontal and passes through the origin of the new coordinate system, as shown in Fig. 4, right).

$$\nabla f = \begin{pmatrix} \frac{\partial \rho}{\partial m'} & \frac{\partial \rho}{\partial b'} \\ \frac{\partial \theta}{\partial m'} & \frac{\partial \theta}{\partial b'} \end{pmatrix} = \begin{pmatrix} (-x_u x_{\bar{p}} - y_u y_{\bar{p}}) & 1 \\ (x_u / \sqrt{1 - x_v^2}) & 0 \end{pmatrix} \tag{14}$$

Angles should be treated consistently throughout the computations. Note that virtually all programming languages represent angles using radians, while the (ρ, θ) parameterization uses degrees. Thus, one should convert angles to degrees after they are computed using Eq. (3) and take this transformation into consideration when evaluating the partial derivative $\frac{\partial \theta}{\partial m'}$, which is used in the computation of the Jacobian matrix ∇f (13).

3.2 Voting Using a Gaussian Distribution

Once one has computed the variances and covariance associated with ρ and θ, the votes are cast using a bi-variated Gaussian distribution given by [30]:

$$G_k(\rho_j, \theta_j) = \frac{1}{2\pi\sigma_\rho\sigma_\theta\sqrt{1 - r^2}} e^{\left(\frac{-z}{2(1-r^2)}\right)} \tag{15}$$

where

$$z = \frac{(\rho_j - \rho)^2}{\sigma_\rho^2} - \frac{2r(\rho_j - \rho)(\theta_j - \theta)}{\sigma_\rho\sigma_\theta} + \frac{(\theta_j - \theta)^2}{\sigma_\theta^2}$$
$$r = \frac{\sigma_{\rho\theta}}{\sigma_\rho\sigma_\theta}$$

Here, k is the k-th cluster of feature pixels, ρ_j and θ_j are the position in the parameter space (voting map) for which we would like compute the number

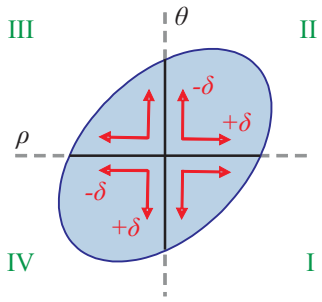

Fig. 5. The kernel footprint is divided into quadrants for the voting process (lines 17-20 of Algorithm 3). The arrows indicate the order in which each quadrant is evaluated.

of votes, ρ and θ are the parameters of the line equation (3) of the best-fitting line for cluster k, and σ_ρ^2, σ_θ^2 and $\sigma_{\rho\theta}$ are the variances and covariance of ρ and θ (Eq. 13).

In practice, Eq. (15) only needs to be evaluated for a small neighborhood of ρ_j and θ_j values around (ρ, θ) in the parameter space. Outside this neighborhood, the value returned by Eq. (15) will be negligible and can be disregarded. The threshold g_{min} used for this is computed as:

$$g_{min} = \min\left(G_k\left(\rho + \rho_w\sqrt{\lambda_w}, \theta + \theta_w\sqrt{\lambda_w}\right)\right) \tag{16}$$

where $w = (\rho_w, \theta_w)^T$ is the eigenvector associated to λ_w, the smaller eingevalue of the covariance matrix shown on the left side of Eq. (13). λ_w represents the variance of the distribution along the w direction. Intuitively, we evaluate Eq. (15), starting at (ρ, θ) and moving outwards, until the returned values are smaller than g_{min}, the value produced at the border of the ellipse representing the footprint of the kernel computed in Section 3.1. Fig. 5 illustrates the inside-going-out order used to evaluate Eq. (15) during the voting procedure.

The complete process of computing the parameters used for the elliptical-Gaussian kernels is presented in Algorithm 2. Special attention should be given to lines 21 and 22, where the variances σ_ρ^2 and σ_θ^2, used to compute the kernel dimensions in Algorithm 4, are scaled by 2^2. Such a scaling enforces the use of two standard deviations (*i.e.*, $2\sigma_\rho$ and $2\sigma_\theta$, respectively) when computing the kernel's footprint. This gives a 95.4% assurance that the selected region of the parameter space that receives votes covers the *true* line represented by the cluster of feature pixels in the image. If the pixels are exactly collinear, such variances will be zero. This can happen in the case of horizontal and vertical lines represented by rows and columns of pixels, respectively. In this case, a variance value of $2^20.1$ (*i.e.*, a small variance of 0.1 scaled by 2^2) is assigned to $\sigma_{\theta_k}^2$ in order to avoid division by zero in Eq. (15).

Algorithm 2 Computation of the Gaussian kernel parameters

Require: S {*Groups of pixels*}
1: **for** each group of pixels S_k **do**
2: $p \leftarrow$ pixels in S_k {*Pixels are 2D column vectors*}
3: $n \leftarrow$ number of pixels in S_k
4: {*Alternative reference system definition*}
5: $\bar{p} \leftarrow$ mean point of p {*Also a column vector*}
6: $V, \Lambda \leftarrow eigen((p - \bar{p})\,(p - \bar{p})^T)$ {*Eigen-decomposition*}
7: $u \leftarrow$ eigenvector in V for the biggest eigenvalue in Λ
8: $v \leftarrow$ eigenvector in V for the smaller eigenvalue in Λ
9: {$y_v \geq 0$ *condition verification*}
10: **if** $y_v < 0$ **then**
11: $v \leftarrow -v$
12: **end if**
13: {*Normal equation parameters computation (3)*}
14: $\rho_k \leftarrow \langle v, \bar{p} \rangle$
15: $\theta_k \leftarrow \cos^{-1}(x_v)$
16: {$\sigma^2_{m'}$ *and* $\sigma^2_{b'}$ *computation, substituting Eq. (5) in (10)*}
17: $\sigma^2_{m'} \leftarrow 1/\sum \langle u, p_i - \bar{p} \rangle^2$
18: $\sigma^2_{b'} \leftarrow 1/n$
19: {*Uncertainty from* $\sigma^2_{m'}$ *and* $\sigma^2_{b'}$ *to* $\sigma^2_{\rho_k}$, $\sigma^2_{\theta_k}$ *and* $\sigma_{\rho_k\theta_k}$}
20: $M = \nabla f \begin{pmatrix} \sigma^2_{m'} & 0 \\ 0 & \sigma^2_{b'} \end{pmatrix} \nabla f^T$ {∇f *matrix in Eq. (14)*}
21: $\sigma^2_{\rho_k} \leftarrow 2^2 M_{(1,1)}$
22: $\sigma^2_{\theta_k} \leftarrow 2^2 M_{(2,2)}$ if $M_{(2,2)} \neq 0$, or $2^2 0.1$ if $M_{(2,2)} = 0$
23: $\sigma_{\rho_k\theta_k} \leftarrow M_{(1,2)}$
24: **end for**

3.3 The Voting Procedure

Once the parameters of the Gaussian kernel have been computed, the voting process is performed according to Algorithms 3 and 4. Here, δ is the discretization step used for both dimensions of the parameter space, defined so that $\rho \in [-R, R]$ and $\theta \in [0°, 180° - \delta]$. Algorithm 3 performs the voting process by splitting the Gaussian kernel into quadrants, as illustrated in Fig. 5. The ellipse in Fig. 5 represents the kernel footprint (in parameter space), with its center located at the best-fitting line for the cluster of pixels. The arrows indicate the way the voting proceeds inside quadrants I to IV, corresponding to the four calls of the function $Vote()$ in lines 17-20 of Algorithm 3.

During the voting procedure, special attention should be given to the cases where the kernel footprint crosses the limits of the parameter space. In case ρ_j is outside $[-R, R]$, one can safely discard the vote, as it corresponds to a line outside of the limits of the image. However, if $\theta_j \notin [0°, 180°)$ the voting

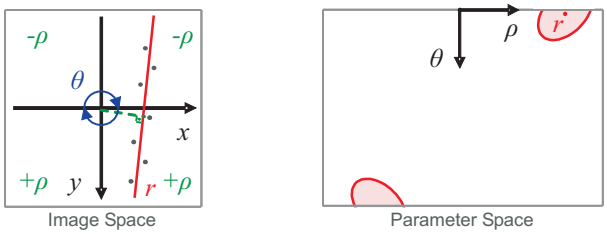

Fig. 6. Special care must be taken when the θ_j parameter is close to 0° or to 180°. In this situation, the voting process for line continues in the diagonally opposing quadrant, at the $-\rho_j$ position.

should proceed in the diagonally opposing quadrant in the parameter space. This special situation happens when the fitted line is approximately vertical, causing ρ_j to assume both positive and negative values. As ρ_j switches signs, the value of θ_j needs to be supplemented (*i.e.*, $180° - \theta_j$). This situation is illustrated in Fig. 6, where the cluster of pixels shown on the left casts votes using the wrapped-around ellipse on the right. In Algorithm 4, the cases where the kernel crosses the parameter space limits are handled by lines 23 (for ρ_j) and 7-19 (for θ_j).

Voting accumulation can be performed using integers. This is achieved scaling the result of Eq. (15) by g_s (lines 22 and 26 of Algorithm 4). g_s is computed in line 15 of Algorithm 3 as the reciprocal of g_{min} (16). Such a scaling guarantees that the each cell in the parameter space covered by the Gaussian kernel will receive an integer number of votes. The use of an integer voting map optimizes the task of peak detection.

Another important optimization in Algorithm 3 is a culling operation that helps to avoid casting votes for kernels whose contributions are negligible. In practice, such kernels do not affect the line detection results, but since they usually occur in large numbers, they tend to waste a lot of processing

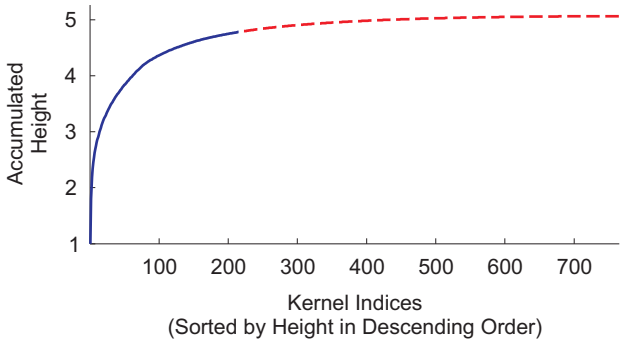

Fig. 7. Cumulative histogram for the normalized heights of the kernels associated to the clusters of pixels found in Fig. 1 (a). Using $h_{min} = 0.002$, only 216 (Fig. 8, right) out of the original 765 clusters (Fig. 8, left) pass the culling test. These clusters correspond to the continuous portion of the graph.

Algorithm 3 Proposed voting process. The *Vote()* procedure is in Algorithm 4.

Require: S {*Kernels parameters computed by Algorithm 2*}
Require: h_{min} {*Minimum height for a kernel* $[0, 1]$}
Require: δ {*Discretization step for the parameter space*}
1: {*Discard groups with very short kernels*}
2: $h_{max} \leftarrow$ *height* of the tallest kernel in S
3: $S' \leftarrow$ groups of pixels in S with $(height/h_{max}) \geq h_{min}$
4: {*Find the* g_{min} *threshold.* G_k *function in Eq. (15)*}
5: $g_{min} \leftarrow 0$
6: **for** each group of pixels S'_k **do**
7: $M \leftarrow \begin{pmatrix} \sigma^2_{\rho_k} & \sigma_{\rho_k\theta_k} \\ \sigma_{\rho_k\theta_k} & \sigma^2_{\theta_k} \end{pmatrix}$
8: $V, \Lambda \leftarrow eigen(M)$ {*Eigen-decomposition of* M}
9: $w \leftarrow$ eigenvector in V for the smaller eigenvalue in Λ
10: $\lambda_w \leftarrow$ smaller eigenvalue in Λ
11: $g_{min} \leftarrow \min\left(g_{min}, G_k\left(\rho_k + \rho_w\sqrt{\lambda_w}, \theta_k + \theta_w\sqrt{\lambda_w}\right)\right)$
12: **end for**
13: {*Vote for each selected kernel*}
14: $Votes \leftarrow 0$ {*Initialization of the voting matrix*}
15: $g_s \leftarrow \max(1/g_{min}, 1)$ {*Scale factor for integer votes*}
16: **for** each S'_k group of pixels **do**
17: $Votes \leftarrow Vote(Votes, \rho_k, \theta_k, \delta, \delta, g_s, S'_k)$
18: $Votes \leftarrow Vote(Votes, \rho_k, \theta_k - \delta, \delta, -\delta, g_s, S'_k)$
19: $Votes \leftarrow Vote(Votes, \rho_k - \delta, \theta_k - \delta, -\delta, -\delta, g_s, S'_k)$
20: $Votes \leftarrow Vote(Votes, \rho_k - \delta, \theta_k, -\delta, \delta, g_s, S'_k)$
21: **end for**

time, becoming a bottleneck for the voting and peak detection steps. Such kernels usually come from groups of widely spaced feature pixels. The inherent uncertainty assigned to such groups of pixels results in short kernels (low relevance) with large footprints (responsible for the extra processing). For instance, for the example shown in Fig. 1 (a), the processing time using the culling operation was 9 ms for voting and 10 ms for peaks detection (see Table 2). On the other hand, without culling, these times were 314 ms ($\sim 34\times$) and 447 ms ($\sim 44\times$), respectively. These times were measured on a 2.21 GHz PC.

Culling is achieved by taking the height of the tallest kernel and using it to normalize the heights of all Gaussian kernels (defined by Eq. (15)) to the $[0, 1]$ range. Only the kernels whose normalized heights are bigger than a relative threshold h_{min} pass the culling operation (lines 2-3 in Algorithm 3). Thus, the culling process is adaptive as the actual impact of h_{min} varies from image to image, as a function of its tallest kernel. For the examples shown in the paper, we used a threshold $h_{min} = 0.002$, which is very conservative. Still, the

performance impact of the culling procedure is huge. Alternatively, one could perform culling using the following procedure: (i) normalize the kernel heights and sort them in descending order; (ii) compute the cumulative histogram of these heights (Fig. 7); (iii) choose a cutoff point based on the derivative of the cumulative histogram (*e.g.*, when the derivative approaches zero); (iv) perform culling by only voting for kernels to the left of the cutoff point. This alternative culling strategy produces similar results to the ones obtained with our approach, but ours has the advantages that there is no need to sort the kernels (the tallest kernel is found in $O(n)$ time by simply comparing and saving the biggest while the kernels are created), there is no need to compute the histogram, nor to analyze its derivatives.

Fig. 7 shows the cumulative histogram for the normalized heights of the kernels associated to the clusters of pixels found in Fig. 1 (a). Using $h_{min} = 0.002$, only 216 (Fig. 8, right) out of the original 765 clusters (Fig. 8, left) pass the culling test. These 216 clusters correspond to the continuous portion of the graph shown in Fig. 7. We have empirically chosen the h_{min} value and a conservative choice, like the one used here, is better than a more restrictive parameterization because linear features may be comprised by many small disjoint (even noisy) clusters. However, as shown in Fig. 8 (right), the use of such a conservative estimate allows for non-very-much-collinear clusters to be included in the voting process. On the other hand, the proposed voting algorithm naturally handles these situations, as such clusters tend to produce wide short Gaussian kernels, thus spreading a few votes over many cells. As a result, these kernels do not determine peak formation.

3.4 Peak Detection

To identify peaks of votes in the parameter space, we have developed a sweep-plane approach that creates a sorted list with all detected vote peaks, according to their importance (number of votes).

Given a voting map, first we create a list with all cells that receive at least one vote. Then, this list is sorted in descending order according to the result of the convolution of the cells with a 3×3 Gaussian kernel. This filtering operation smooths the voting map, helping to consolidate adjacent peaks as single image lines. Fig. 9 compares the detection of the 25 most relevant lines in Fig. 1 (a). Fig. 9 (left) shows the obtained result with the use of convolution, whereas for Fig. 9 (right) no convolution has been applied. Note the recovered line corresponding to the limit of the checkerboard on the top of Fig. 9 (left). Two segments define such a line, each one visible at one side of the dark king (Fig. 8, right). As the two segments fall on neighbor cells in the discretized parameter space (*i.e.*, voting map), the convolution combines the two peaks

Algorithm 4 Function $Vote\,()$. It complements the proposed voting process (Algorithm 3)

Require: $Votes$, ρ_{start}, θ_{start}, δ_ρ, δ_θ, g_s, S'_k $\{Parameters\}$
$\rho_k, \theta_k, \sigma^2_{\rho_k}, \sigma^2_{\theta_k}, \sigma_{\rho_k\theta_k} \leftarrow$ Parameters computer for S'_k
$R \leftarrow \sqrt{w^2 + h^2}/2$
$\{$*Loop for the θ coordinates of the parameter space*$\}$
$\theta_j \leftarrow \theta_{start}$
repeat
  $\{$*Test if the kernel exceeds the parameter space limits*$\}$
  if $(\theta_j < 0°)$ **or** $(\theta_j \geq 180°)$ **then**
    $\delta_\rho \leftarrow -\delta_\rho$
    $\rho_k \leftarrow -\rho_k$
    $\sigma_{\rho_k\theta_k} \leftarrow -\sigma_{\rho_k\theta_k}$
    $\rho_{start} \leftarrow -\rho_{start}$
    if $\theta_j < 0°$ **then**
      $\theta_k \leftarrow 180° - \delta_\theta + \theta_k$
      $\theta_j \leftarrow 180° - \delta_\theta$
    else
      $\theta_k \leftarrow \theta_k - 180° + \delta_\theta$
      $\theta_j \leftarrow 0°$
    end if
  end if
  $\{$*Loop for the ρ coordinates of the parameter space*$\}$
  $\rho_j \leftarrow \rho_{start}$
  $g \leftarrow round(g_s G_k\,(\rho_j, \theta_j))$ $\{G_k$ *from Eq. (15)*$\}$
  while $(g > 0)$ **and** $(-R \leq \rho_j \leq R)$ **do**
    $Votes(\rho_j, \theta_j) \leftarrow Votes(\rho_j, \theta_j) + g$
    $\rho_j \leftarrow \rho_j + \delta_\rho$
    $g \leftarrow round(g_s G_k\,(\rho_j, \theta_j))$
  end while
  $\theta_j \leftarrow \theta_j + \delta_\theta$
until no votes have been included by the internal loop

into a larger one, resulting in a significant line. For the result shown in Fig. 9 (right), since no convolution has been applied, the two smaller peaks were both missed and replaced by the edges on the wall and on the chess horse, which are less relevant. The convolution operation is also applicable to the conventional voting scheme and was used to generate the results used for comparison with the proposed approach.

After the sorting step, we use a sweep plane that visits each cell of the list. By treating the parameter space as a height-field image, the sweeping plane gradually moves from each peak toward the zero height. For each visited cell, we check if any of its 8 neighbors has already been visited. If so, the current cell should be a smaller peak next to a taller one. In such a case, we mark the

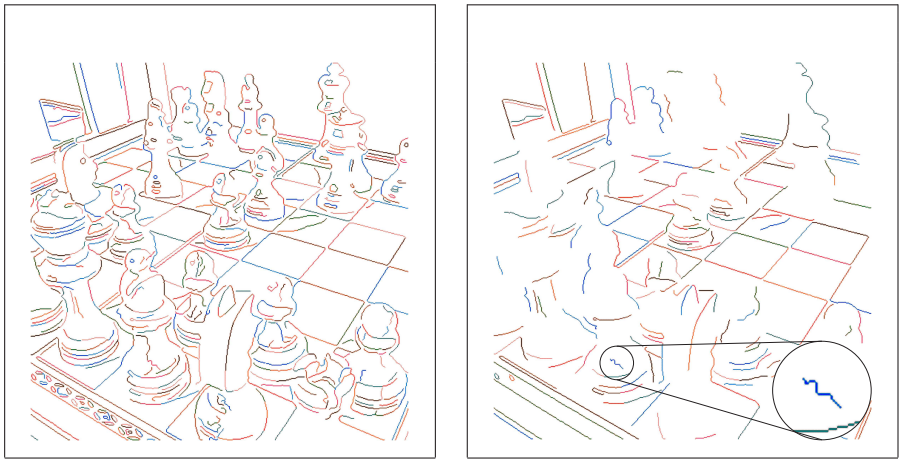

Fig. 8. Comparison between the complete collection of clusters (left) found in Fig. 1 (a) and the subset of clusters used by the proposed voting approach (right). Note that many non-collinear clusters, like the highlighted one, remain after the culling procedure.

current cell as visited and move the sweeping plane to the next cell in the list. Otherwise, we add the current cell to the list of detected peaks, mark it as visited and then proceed with the sweeping plane scan to the next cell in the list. The resulting group of detected peaks contains the most significant lines identified in the image, already sorted by number of votes.

4 Results

In order to compare the performances of the gradient-based (*i.e.*, the universally used version of the Hough transform) and the proposed Hough transform voting schemes, both techniques were implemented using C++ and tested over a set of images of various resolutions, some of which are shown in Figs. 1, 10 and 11. Edge extraction and thinning were performed with standard image processing techniques using Matlab 7.0. The resulting feature pixels were used as input for the gradient-based Hough transform (GHT) algorithm, which has been optimized with the use of lookup tables. It was also assumed that each edge pixel votes for a range of 20 angular values, computed from its gradient information. The C++ code was compiled using Visual Studio 2003 as dynamic link libraries (DLLs) so that they could be called from Matlab. The C++ code for the proposed approach (KHT) also handles the linking of adjacent edge-pixel chains (Appendix 1) and their subdivision into the most perceptually-significant straight line segments [27]. After the voting process, the peaks in the parameter space are identified using the technique described in Section 3.4. The reported experiments were performed on an AMD Athlon 64 3700+ (2.21 GHz) with 2 GB of RAM. However, due to some constraints imposed by the available Matlab version, the code was compiled for 32 bits

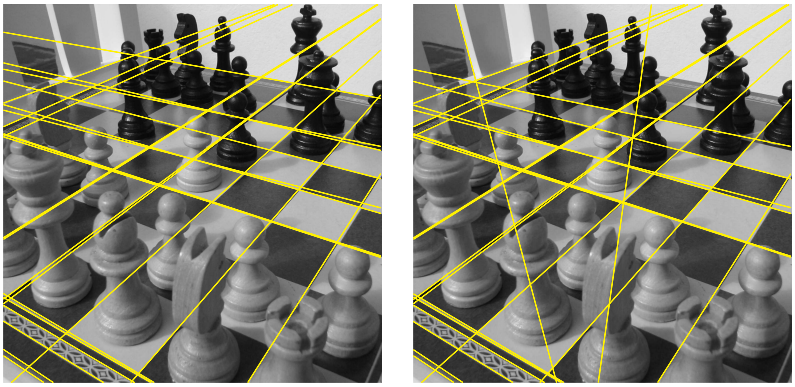

Fig. 9. The 25 most relevant detected lines for the image shown in Fig. 1 (a) obtained using our kernel-based voting and peak detection algorithms. On the left, a 3x3 Gaussian kernel was convolved with the voting map before performing peak detection. On the right, no convolution was applied prior to peak detection.

only, not being able to exploit the full potential of the hardware.

Tables 1 and 2 summarize the results obtained on seven groups of images (Figs. 1, 10 and 11) resampled at various resolutions. For each image, the intermediate-resolution version was obtained by cropping the full-resolution one, and the lower resolution was obtained by resampling the cropped version. The complexity of these images are indicated by their number and distribution of the feature pixels, which are depicted in Fig. 1 (b) and in the second row of both Figs. 10 and 11, respectively. The images shown in Fig. 10 (left) and Fig. 11 were chosen because they represent challenging examples. For these, the results produced by the conventional voting scheme (Fig. 1 (c) and third row of both Figs. 10 and 11) lend to many variants (alias) of the same lines, leaving many other important ones unreported. This situation becomes evident on the right side of the *Road* where multiple concurrent lines were extracted, leaving important lines undetected on the painted pavement. The same problem is observed on the bottom of the *Church* and *Building* images in Fig. 11. Note that Figs. 10 (center) and (right) contain much cleaner sets of straight line segments, which tends to simplify the line detection task. These images were used to demonstrate that the conventional brute-force voting approach tends to create spurious peaks and identify several variations of the same line, even when applied to simple configurations. As a result, it failed to report some important lines as significant ones. For instance, note the undetected diagonal lines on the four corners of the *Wall* image (Fig. 10). In comparison, the proposed approach (Fig. 1 (d) and fourth row of both Figs. 10 and 11) was capable of identifying all important lines in all seven images. Since applications that use Hough transform usually have to specify the number of lines to be extracted, robustness against the detection of false positives is of paramount importance.

In Table 1, the third column (Edge Pixels) shows the number of extracted

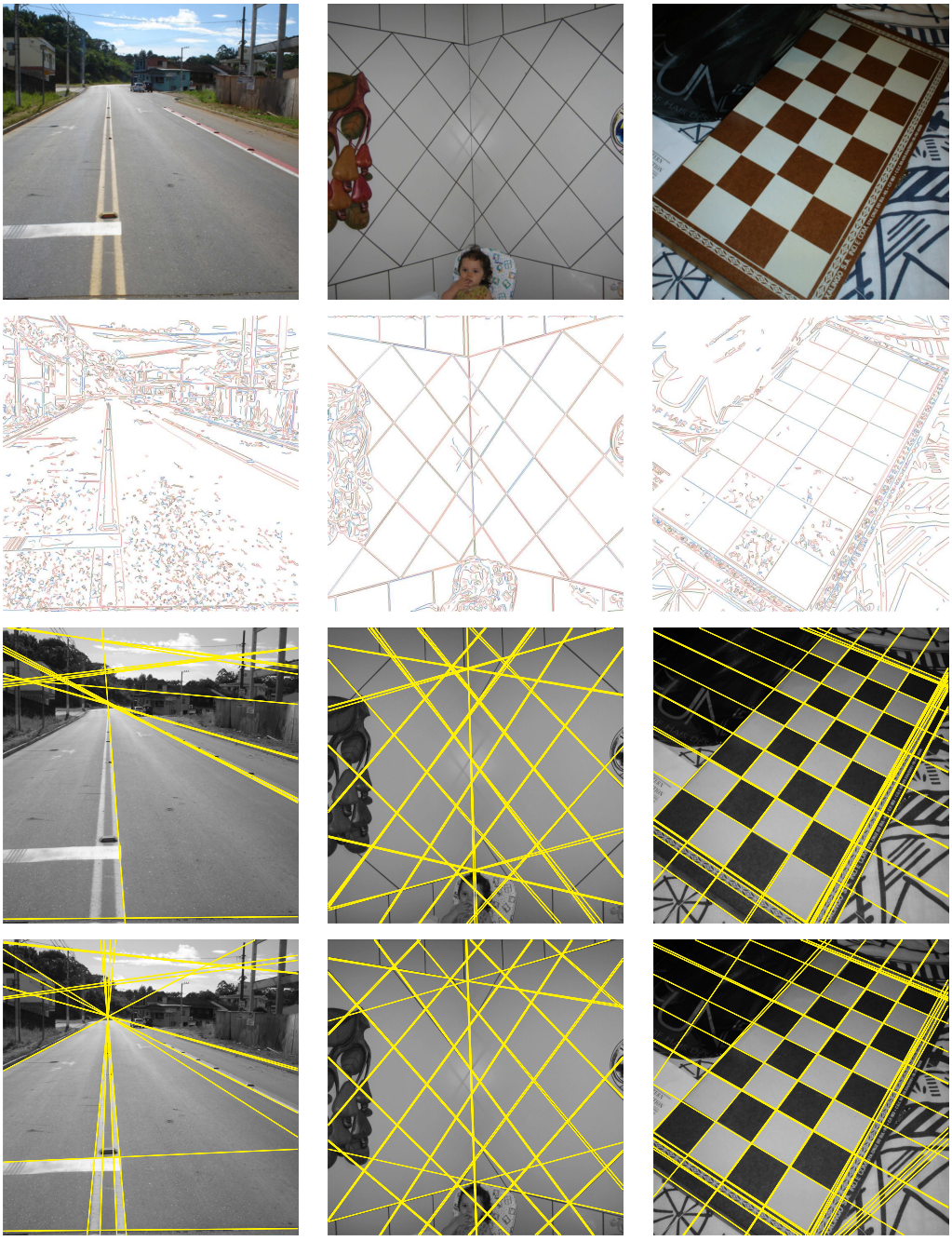

Fig. 10. Comparison of the line-detection results produced by the gradient-based (GHT) and the proposed (KHT) Hough transform voting and line detection schemes. Set of test images (top). From left to right, these images correspond to the *Road*, *Wall* and *Board* datasets referenced in Tables 1 and 2 and exhibit the 15, 36 and 38 most important lines detected by each algorithm, respectively. Note that in all examples the GHT did not detect important lines, which were detected by the KHT (bottom).

feature pixels in the images. The fourth and fifth columns (Edge Groups) show the numbers of identified clusters and the numbers of clusters that passed the

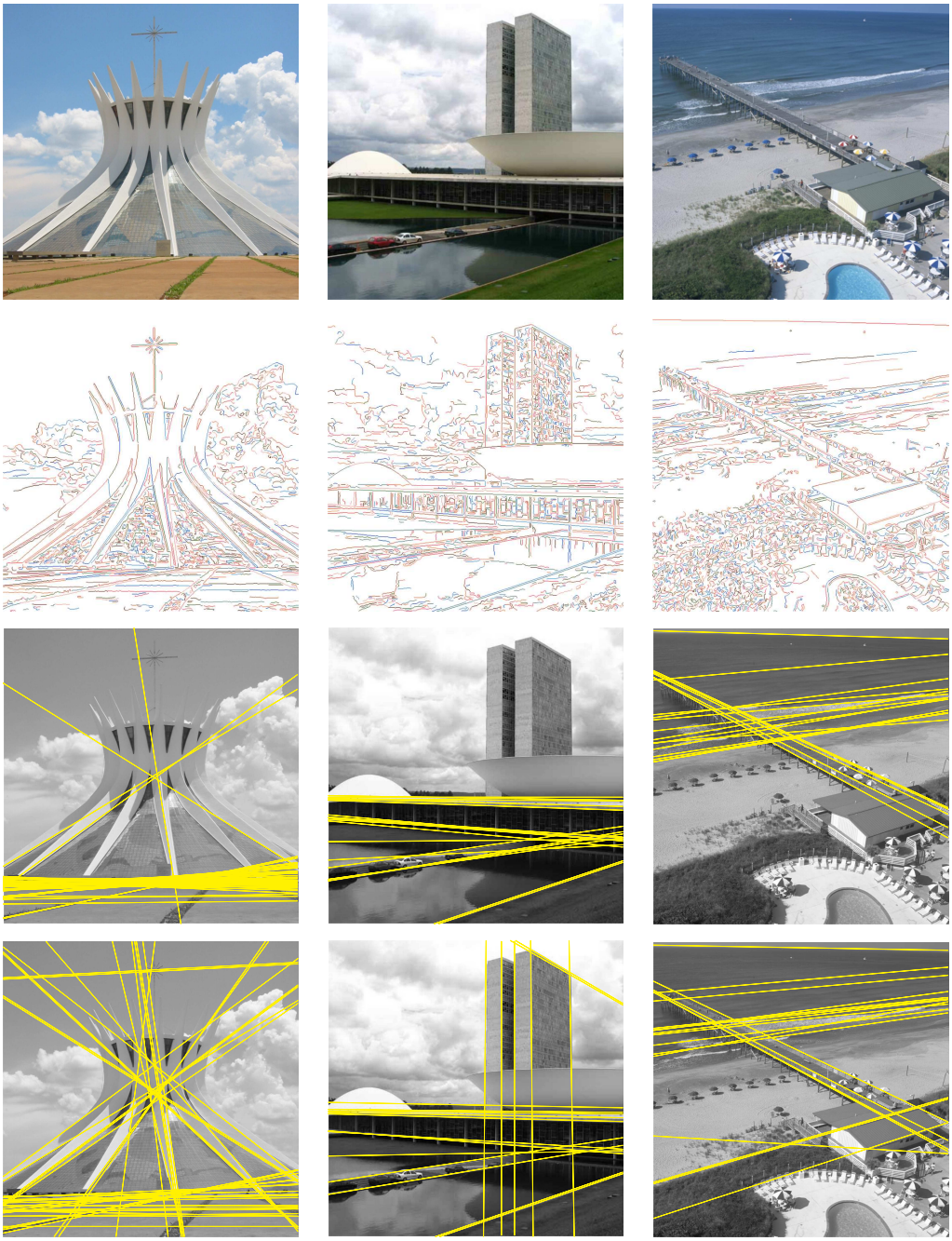

Fig. 11. Comparison of the line-detection results produced by the gradient-based (GHT) and the proposed (KHT) Hough transform voting and line detection schemes. The first row shows the original images, whose feature pixels, which were detected using a Canny edge detector, are shown on the second row. From left to right, these images correspond to the *Church*, *Building* and *Beach* datasets referenced in Tables 1 and 2 and exhibit the 40, 19 and 19 most important lines detected by each algorithm, respectively. Rows three and four show the most relevant lines detected by GHT and KHT, respectively. Note the important lines undetected by the GHT.

culling optimization described in Section 3.3. Columns six and seven of Table 1 (Used Pixels) show the actual numbers of feature pixels processed by the KHT approach and their percentage with respect to the entire image. In Table 2, the 'GHT Voting' and 'KHT voting' columns show the time (in milliseconds) involved in the execution of the GHT and KHT, respectively. One should note that the time for KHT includes the processing required to link and subdivide clusters of neighbor pixels, which is not necessary for the GHT. The last four columns of Table 2 show the number of detected peaks by both approaches and the corresponding detection times. These times were computed by averaging the results of 50 executions of the reported tasks. A detailed account for the times involved in each step of the proposed voting strategy are presented in Table 3 for various resolutions of the images in Fig. 1, 10 and 11.

Table 2 shows that the proposed approach can achieve up to 200 fps for images with 512×512 (*e.g.*, *Board* and *Building*) without performing peak detection, and up to 125 fps with peak detection. For the same *Board* and *Building* images, respectively, the gradient-based Hough transform executes at 58.82 fps and 45.45 fps without peak detection, and 10.75 fps and 9.71 fps, with peak detection. When considering only the computation of the Gaussian kernel parameters and voting, KHT can process images with 1600×1200 pixels (*e.g.*, *Wall*) at 100 fps, and with 512×512 pixels at 200 fps. Including the time required to link and subdivide cluster of pixels (but without peak detection), KHT can still process a 1600×1200 image at 34.48 fps. These results are superior to the ones reported by [17] for 31 of the 34 analyzed architectures for real-time Hough transform, and comparable to the other three.

Fig. 12 provides a graphical comparison between the performances of the gradient-based Hough transform and our approach. The horizontal axis enumerates the set of input images of various resolutions and the vertical axis shows the execution times in milliseconds. Note that KHT is always much faster than GHT (KHT's entire process takes less than GHT's voting time) and processes almost all images under the mark of 33 ms (30 Hz, represented by the dotted line). The only exceptions were high resolution versions of *Road*, *Wall* and *Beach*, which took 34, 36 and 36 ms, respectively.

A careful inspection of Table 2 reveals that although the total time (*i.e.*, voting time plus peak detection time) for KHT correlates with the size of the images, the peak detection time for the 512×512 version of the *Chess* image took longer (extra 4ms) than for the higher resolution versions of the same image. This can be explained considering that many small groups of non-collinear feature pixels are found in the *Chess* image (Fig. 8, left). As the image was filtered down, the relative uncertainty in the resulting feature pixels increased due to the filtering itself and to the decrease in resolution. As a result, the peak of the highest Gaussian used for normalizing the heights of all kernels became smaller, thus allowing a larger number of low-height kernels to pass

Table 1
Resolution and numbers of edge pixels, edge groups, and feature pixels used in the test images shown in Fig. 1, 10 and 11.

Image		Edge Pixels	Edge Groups		Used Pixels	
Name	Resolution	Count	Count	Used	Count	Rate
Chess	1280×960	$70,238$	$2,756$	110	$12,717$	1.03%
	960×960	$52,547$	$2,035$	172	$12,136$	1.32%
	512×512	$18,653$	765	216	$8,704$	3.32%
Road	1280×960	$101,756$	$4,069$	170	$16,099$	1.31%
	960×960	$75,281$	$2,981$	151	$13,814$	1.50%
	512×512	$19,460$	698	127	$7,677$	2.93%
Wall	1600×1200	$72,517$	$2,050$	414	$36,679$	1.91%
	960×960	$38,687$	$1,026$	267	$22,604$	2.45%
	512×512	$16,076$	438	230	$11,870$	4.53%
Board	1280×960	$66,747$	$2,292$	162	$19,023$	1.55%
	960×960	$44,810$	$1,467$	142	$16,150$	1.75%
	512×512	$15,775$	453	99	$7,435$	2.84%
Church	768×1024	$67,140$	$2,532$	305	$20,472$	2.60%
	768×768	$45,372$	$1,529$	242	$17,813$	3.02%
	512×512	$18,934$	644	166	$9,811$	3.74%
Building	1024×768	$64,453$	$2,379$	57	$10,826$	1.38%
	768×768	$51,181$	$1,917$	57	$9,792$	1.66%
	512×512	$21,568$	748	63	$6,729$	2.57%
Beach	869×1167	$105,589$	$4,564$	210	$14,785$	1.46%
	800×800	$63,164$	$2,629$	150	$10,904$	1.70%
	512×512	$24,880$	$1,019$	42	$3,873$	1.48%

the h_{min} culling test. Note that the number of detected peaks (column 7 in Table 2) is bigger for the lower-resolution version of the *Chess* image than for its higher resolution versions. A similar phenomenon can be observed for other images listed in the same table, but at reduced scale.

For GHT the number of accesses to the voting map grows with the number of feature points, the image dimensions, the discretization step δ, and the range r of selected θ values over which the associated ρ values should be calculated. Such number of access can be expressed as $q' = rn/\delta$, where n is the total

Table 2
Comparison of the performance of the gradient-based (GHT) and proposed (KHT) algorithms for the steps of voting and peak detection on the images shown in Figs. 1, 10 and 11. For KHT, voting time includes clustering approximately collinear feature pixels into segments. Time is expressed in milliseconds (ms).

Image		GHT Voting	KHT Voting	GHT Peaks		KHT Peaks	
Name	Resolution	ms	ms	Count	ms	Count	ms
Chess	1280×960	83	21	42,060	223	98	6
	960×960	60	17	35,779	179	184	6
	512×512	20	9	18,992	82	370	10
Road	1280×960	123	27	47,414	257	192	7
	960×960	87	20	40,513	217	166	6
	512×512	20	6	22,814	90	200	5
Wall	1600×1200	81	29	48,648	238	647	17
	960×960	42	14	34,570	144	295	9
	512×512	17	8	17,213	65	265	9
Board	1280×960	78	21	44,438	223	133	6
	960×960	51	15	39,280	173	111	5
	512×512	17	5	21,605	76	73	3
Church	768×1024	73	21	33,088	175	469	11
	768×768	49	14	28,714	138	364	8
	512×512	20	7	17,843	77	250	6
Building	1024×768	70	17	35,736	181	47	4
	768×768	54	13	29,720	146	45	4
	512×512	22	5	18,510	81	52	3
Beach	869×1167	120	28	37,879	222	276	8
	800×800	70	17	30,976	166	178	6
	512×512	26	6	19,276	92	36	3

number of pixels in the image and $0° \leq r < 180°$. For KHT, the number of accesses depends on the number m of clusters that cast votes and the areas of their kernel footprints. Given an average footprint area s (expressed in cells of the voting map), the number of accesses is given by $q'' = sm/\delta$. Thus, KHT has a smaller number of accesses whenever $sm < rn$, which tends to always happen, since $s \ll n$.

Table 3
Processing time for all steps of KHT's voting procedure.

Image		Link	Subdivide	Kernel	Vote
Name	Resolution	ms	ms	ms	ms
Chess	1280×960	9	7	5	< 1
	960×960	7	5	4	1
	512×512	2	2	1	4
Road	1280×960	11	7	8	1
	960×960	8	5	6	1
	512×512	2	1	1	2
Wall	1600×1200	12	7	4	6
	960×960	6	4	2	2
	512×512	2	1	1	4
Board	1280×960	9	7	4	1
	960×960	6	5	3	1
	512×512	2	2	1	< 1
Church	768×1024	7	5	5	4
	768×768	5	4	3	2
	512×512	2	2	1	2
Building	1024×768	7	5	5	< 1
	768×768	5	4	4	< 1
	512×512	2	2	1	< 1
Beach	869×1167	11	7	8	2
	800×800	6	5	5	1
	512×512	2	2	2	< 1

Although no pre-processing was applied to the images used in our experiments, we noticed that the presence of severe radial and tangential distortions [2] may affect the quality of the results produced by KHT. This can be explained as the fact that straight lines might become curved toward the borders of the image, resulting in possibly large sets of small line segments, which will tend to spread their votes across the parameter space. This problem also affects other variants of the Hough transform, and can be easily avoided by pre-warping the images to remove these distortions. Such a warping can be effectively implemented for any given lens-camera setup with the use of lookup tables.

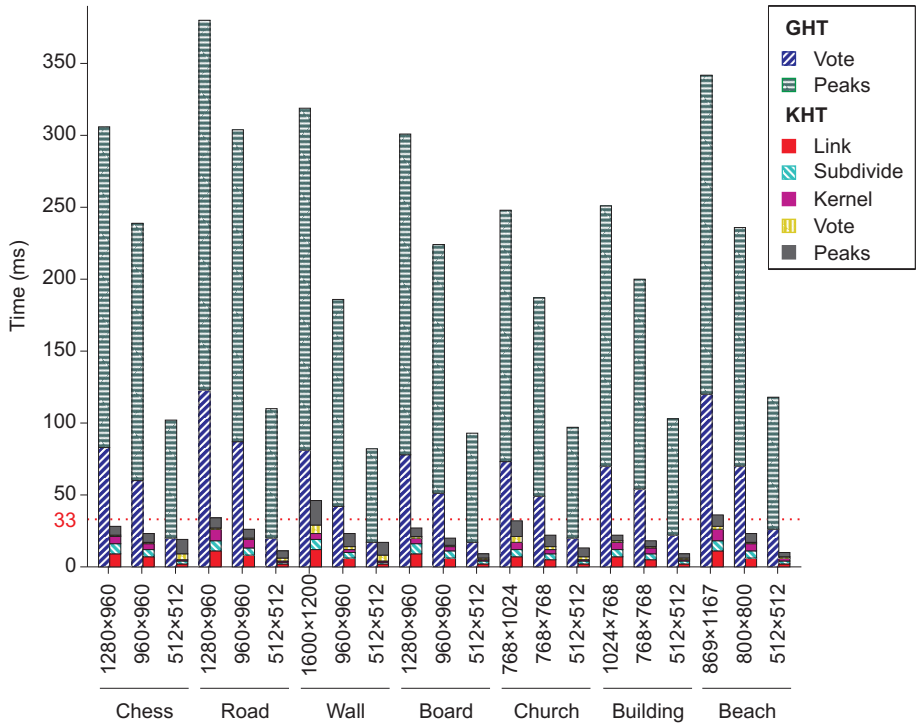

Fig. 12. Comparison between the processing times of the gradient-based Hough transform (GHT) and of the proposed approach (KHT). For each input image (horizontal axis), the left bar represents GHT and the right one represents KHT. The height of the bar is given by the sum of the time spent by each step of the corresponding approach. The dotted line defines the mark of 33 ms, approximately 30 Hz.

A limitation of the proposed technique is the lack of detection of thin dotted lines, where the grouping algorithm is not capable of identifying strings of neighboring pixels. Although in practice we have not experienced this kind of problem, it may happen and can be avoided by changing the strategy adopted for linking such pixels, or by filtering the image down before applying the algorithm. In our current implementation of the described algorithm, we have chosen to group approximately collinear feature pixels using the linking procedure from [26] and the subdivision scheme from [27] because they are fast and handle the general case. However, other approaches for chaining and subdivision can be used for achieving specific results, such as handling the dotted line example just mentioned.

4.1 Memory Utilization

The amount of memory required by the GHT consists of the voting map and lookup tables used to optimize the transform implementation. In the case of the KHT, no lookup tables are necessary, but it requires 6 floats for each elliptical-Gaussian kernel: ρ and θ (the 'center' of the ellipse), σ_ρ^2, σ_θ^2 and

$\sigma_{\rho\theta}$, and the height of the kernel (used to optimize the culling of the small kernels). Thus, assuming an image with 1600×1200 pixels and $\delta = 0.5$, the voting map for both approaches would require 10.99 MB. The lookup tables for the GHT would take 2.81 KB, whereas the extra information required by the KHT would need 23.44 KB. This minor increase on memory needs of the KHT is more than compensated by the significant speedups obtained by the method.

5 Conclusions and Future Work

We have presented an improved voting scheme for the Hough transform that allows a software implementation to achieve real-time performance even for relatively large images. The approach clusters approximately collinear edge pixels and, for each cluster, casts votes for a reduced set of lines in the parameter space, based on the quality of the fitting of a straight line to the pixels of the cluster. The voting process is weighted using oriented elliptical-Gaussian kernels, significantly accelerating the process and producing a much cleaner voting map. As a result, our approach is extremely robust to the detection of spurious lines, very common in the traditional approach.

The KHT is not only significantly faster than previous known software implementations of the Hough transform, but its performance also compares favorably to the reported results of most specialized hardware architectures for HT.

We also presented a simple sweep-plane-based approach for identify peaks of votes in the parameter space. We believe that these ideas may lend to optimizations on several computer vision, image processing and computer graphics applications that require real-time detection of lines. We are exploring ways to extend our kernel-based voting scheme to detect arbitrary curves and objects of known geometry.

Acknowledgements

This work was partially sponsored by CNPq - Brazil (477344/2003-8) and Petrobras (502009/2003-9). The authors would like to thank Microsoft Brazil for additional support, Prof. Roberto da Silva (PPGC-UFRGS) for some discussions, and the anonymous reviewers for their comments and insightful suggestions.

References

[1] P. E. Debevec, C. J. Taylor, J. Malik, Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach, in: Proceedings of the 23th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH-96), Addison Wesley, 1996, pp. 11–20.

[2] R. I. Hartley, A. Zisserman, Multiple view geometry in computer vision, Cambridge University Press, Cambridge, UK, 2000.

[3] A. Ferreira, M. J. Fonseca, J. A. Jorge, M. Ramalho, Mixing images and sketches for retrieving vector drawings, in: Eurographics Multimedia Workshop, Eurographics Association, China, 2004, pp. 69–75.

[4] B. Caprile, V. Torre, Using vanishing points for camera calibration, International Journal of Computer Vision (IJCV) 4 (2) (1990) 127–139.

[5] M. Straforini, C. Coelho, M. Campani, Extraction of vanishing points from images of indoor and outdoor scenes, Image and Vision Computing 11 (2) (1993) 91–99.

[6] L. F. Lew Yan Voon, P. Bolland, O. Laligant, P. Gorria, B. Gremillet, L. Pillet, Gradient-based discrete Hough transform for the detection and localization of defects in nondestructive inspection, in: Proceedings of the V Machine Vision Applications in Industrial Inspection, Vol. 3029, SPIE, San Jose, CA, 1997, pp. 140–146.

[7] H.-J. Lee, H.-J. Ahn, J.-H. Song, R.-H. Park, Hough transform for line and plane detection based on the conjugate formulation, in: Proceedings of the IX Machine Vision Applications in Industrial Inspection, Vol. 4301, SPIE, San Jose, CA, 2001, pp. 244–252.

[8] R. Strzodka, I. Ihrke, M. Magnor, A graphics hardware implementation of the generalized Hough transform for fast object recognition, scale, and 3D pose detection, in: Proceedings of the International Conference on Image Analysis and Processing, 2003, pp. 188–193.

[9] A. Karnieli, A. Meisels, L. Fisher, , Y. Arkin, Automatic extraction and evaluation of geological linear features from digital remote sensing data using a Hough transform, Photogrammetric Engineering & Remote Sensing 62 (5) (1996) 525–531.

[10] P. V. C. Hough, Methods and means for recognizing complex patterns, U.S. Patent 3.069.654 (1962).

[11] R. O. Duda, P. E. Hart, Use of the Hough transformation to detect lines and curves in pictures, Communications of the ACM 15 (1) (1972) 11–15.

[12] J. Princen, J. Illingworth, J. Kittler, A hierarchical approach to line extraction based on the Hough transform, Graphical Model and Image Processing 52 (1) (1990) 57–77.

[13] S. B. Yacoub, J. Jolion, Hierarchical line extraction, IEE Vision, Image and Signal Processing 142 (1) (1995) 7–14.

[14] K.-L. Chung, T.-C. Chen, W.-M. Yan, New memory- and computation-efficient Hough transform for detecting lines, Pattern Recognition 37 (2004) 953–963.

[15] C. G. Ho, R. C. D. Young, C. D. Bradfield, C. R. Chatwin, A fast Hough transform for the parametrisation of straight lines using Fourier methods, Real-Time Imaging 6 (2) (2000) 113–127.

[16] L. Lin, V. K. Jain, Parallel architectures for computing the Hough transform and CT image reconstruction, in: Int. Conf. On Application Specific Array Processors, 1994, pp. 152–163.

[17] M. G. Albanesi, M. Ferretti, D. Rizzo, Benchmarking Hough transform architectures for real-time, Real-Time Imaging 6 (2) (2000) 155–172.

[18] K. Mayasandra, S. Salehi, W. Wang, H. M. Ladak, A distributed arithmetic hardware architecture for real-time Hough-transform-based segmentation, Canadian Journal of Electrical and Computer Engineering 30 (4) (2005) 201–205.

[19] J. Canny, A computational approach to edge detection, IEEE Transactions on Pattern Analysis and Machine Intelligence 8 (6) (1986) 679–698.

[20] D. H. Ballard, Generalizing the Hough transform to detect arbitrary shapes, Pattern Recognition 13 (2) (1981) 111–122.

[21] V. F. Leavers, M. B. Sandler, An efficient Radon transform, in: Proceedings of the 4th International Conference on Pattern Recognition (ICPR-88), Springer, Cambridge, UK, 1988, pp. 380–389.

[22] J. Illingworth, J. V. Kittler, A survey of the Hough transform, Graphical Model and Image Processing 44 (1) (1988) 87–116.

[23] V. F. Leavers, Survey: which Hough transform?, Graphical Model and Image Processing 58 (2) (1993) 250–264.

[24] X. Lin, K. Otobe, Hough transform algorithm for real-time pattern recognition using an artificial retina camera, Optics Express 8 (9) (2001) 503–508.

[25] L. F. Costa, M. Sandler, A binary Hough transform and its efficient implementation in a systolic array architecture, Pattern Recognition Letters 10 (1989) 329–334.

[26] A. R. Pope, D. G. Lowe, Vista: a software environment for computer vision research, in: Proceedings of Computer Vision and Pattern Recognition (CVPR'94), IEEE Computer Society, Seattle, WA, 1994, pp. 768–772.

[27] D. G. Lowe, Three-dimensional object recognition from single two-dimensional images, Artificial Intelligence 31 (1987) 355–395, section 4.6.

[28] N. R. Draper, H. Smith, Applied regression analysis, John Wiley & Sons, New York, 1966.

[29] L. G. Parratt, Probability and experimental errors in science, John Wiley and Sons Inc., New York, 1961.

[30] E. W. Weisstein, Bivariate normal distribution, http://mathworld.wolfram.com/BivariateNormalDistribution.html, mathWorld, Last Update: Jan. 2005 (2005).

Appendix 1: Linking procedure

The linking procedure (Algorithm 5) creates a string of neighboring edge pixels. It takes as input a binary image and a reference feature pixel from which one would like to retrieve a pixel chain. Note that this algorithm destroys the original binary image (lines 6 and 14). The pseudo code shown in Algorithm 5 was inspired on an implementation available in the VISTA image-processing library [26].

Algorithm 5 Linking of neighboring edge pixels into strings. The pseudo code for the function $Next\,()$ is presented in Algorithm 6.

Require: I {*Binary image*}
Require: p_{ref} {*A feature pixel*}
1: $S \leftarrow \oslash$ {*Create an empty string of pixels*}
2: {*Find and add feature pixels to the end of the string*}
3: $p \leftarrow p_{ref}$
4: **repeat**
5: $S \leftarrow S + p$
6: $I\,(x_p, y_p) \leftarrow 0$
7: $p \leftarrow Next\,(p, I)$
8: **until** p be an invalid pixel position
9: {*Find and add feature pixels to the begin of the string*}
10: $p \leftarrow Next\,(p_{ref}, I)$
11: **if** p is a valid pixel position **then**
12: **repeat**
13: $S \leftarrow p + S$
14: $I\,(x_p, y_p) \leftarrow 0$
15: $p \leftarrow Next\,(p, I)$
16: **until** p be an invalid pixel position
17: **end if**

Algorithm 6 Function $Next\,()$. It complements the linking procedure (Algorithm 5).

Require: I {*Binary image*}
Require: p_{seed} {*Current feature pixel*}
1: **for** each neighbor pixel of p_{seed} **do**
2: $p \leftarrow$ current neighbor pixel of p_{seed}
3: **if** p is a feature pixel **then**
4: Break the loop and return p {*The next pixel was found*}
5: **end if**
6: **end for**
7: Return an invalid pixel position {*The next pixel was not found*}