A Fast Rule-Based Parameter Free Digital Hough Transform

B.M.A. Genswein and Y.H. Yang*
The Scene Analysis and Modelling Group
Computer Vision and Graphics Laboratory
Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan
Canada S7N 5A9
Email Address for Y.H. Yang: yang@cs.usask.ca

March 24, 1998

Abstract

This paper introduces a new digital Hough transform, DHT, that pre-computes digital line information (rules) and uses this information to detect line segments in the image. Pre-computing line information removes the need for run-time line calculations and the associated parameters. The proposed approach does not depend on the parameterization of a straight line and is formulated based on the discrete domain. This new DHT will be compared with existing Hough techniques to demonstrate the large reduction in computation time achieved by this new approach, while not sacrificing accuracy.

Keywords: Hough transform, line detection.

1 Introduction

The Hough transform, HT, is a powerful tool for extraction of parameterized shapes in binary digital images that is robust in the presence of noise and partial occlusion. The HT was first introduced by Paul Hough [3] in 1962 as a patent for an electronic device that was used to detect the tracks of high speed particles through the viewing field of a bubble chamber.

The standard HT, SHT, uses the normal representation of a line to map a single point (x, y) in the image to many points in parameter space (ρ, θ) using the following equation:

$$\rho = x\cos\theta + y\sin\theta, \qquad -\pi/2 \leq \theta \leq \pi/2. \quad (1)$$

Lines are identified by the intersections of these sinusoids. Intersections are represented by peaks (maxima) in parameter space (ρ, θ) which are detected through an exhaustive search. This mapping requires a large amount of computation and storage.

Implementing the SHT involves quantizing the continuous parameter domain into an appropriate discrete parameter domain. Finer quantization yields a higher resolution for the detected lines but a larger parameter space (multidimensional array) and larger computational requirements.

There have been many new Hough transforms, HTs, proposed to improve the SHT. This paper introduces a new digital HT, DHT, that uses only discrete information to detect lines in digital images. Also included in this paper is a brief description of other HT's and a comparison of all mentioned Hough algorithms.

Section 2 discusses the other HT's included in this paper. Section 3 provides a description of the proposed DHT. Section 4 displays and discusses the results of the tests performed in this study and Section 5 presents the conclusions made based on the test results.

2 Previous Work

This section briefly discusses some HT's previously proposed as improvements to the SHT. This section will be divided into two sections, non-probabilistic HT's and probabilistic HT's.

*Author to whom correspondence should be addressed.
The authors would like to acknowledge Natural Sciences and Engineering Research Council of Canada for financial support through grant number OGP0000370.

2.1 Non-Probabilistic Hough transforms

This section investigates four other HTs that have been proposed as improvements to the SHT. These other HTs are the Adaptive HT, AHT, Combinatorial HT, CHT, Curve Fitting HT, CFHT, and the Windowing approach to detecting Line Segments using the HT, WLSHT.

The AHT [4] uses a coarse to fine accumulation and search strategy to identify significant peaks in parameter space. This approach is a flexible iterative approach in that a small accumulator array is used in the first iteration and the information from the first iteration is used intelligently to redefine the range of parameters for the next iteration.

Once the array has been accumulated it is binarized by using a threshold, typically 0.9 of the highest count. A connected components algorithm is used to label every connected subset of 1's. Each unique label is processed in successive iterations at a finer resolution determined by the location of labels along the parameter axis.

The CHT [2] calculates the intersections of the sinusoids of (1) for every two-point combination in the image. Solving (1) for the joint equations using points $(x_1, y_1), (x_2, y_2)$ yields: $\theta' = \arctan\left[\frac{(x_1 - x_2)}{(y_2 - y_1)}\right]$ and substituting back into (1) with either (x_1, y_1) or (x_2, y_2) gives: $\rho' = x_1 \cos\theta' + y_1 \sin\theta'$. The parameter space is now defined as (ρ', θ'). Solving (2.1) and (2.1) for each pair of points provides a vote to the accumulator array (ρ', θ'). For m points in an image there are $C_m^2 = \frac{m!}{2!(m-2)!}$ possible combinations of points for solving (2.1) and (2.1). If m is large enough the computation required may exceed that of the SHT. This problem is addressed by segmenting the image and then each segment is handled separately, with all segments contributing to the same (ρ', θ').

The CFHT [8] attempts to fit a segment of a curve to be detected to a small neighbourhood of edge points. If the fitting error is less than a given threshold T_E, then the parameters from the curve fitting are used to map the curve segment into one point in parameter space. A parameter list is used instead of a fully discretized parameter space (multidimensional array). Each entry in the parameter list stores the values of parameters and the accumulated vote of points or segments lying on each curve to be detected.

The WLSHT [10] uses (2.1) and (2.1) to compute intersections of pairs of points. The WLSHT uses densely overlapping windows, small windows, a parameter list, and tracks collinear non-overlapping line segments.

In the WLSHT, there is a window centred at each pixel. The window is small enough so that there are very few possible line segments in the window. Using the centre pixel with every other pixel in the window (2.1) and (2.1) are computed and the solutions are stored in a local list. After all pixels have been processed the most likely line segment in the local list is chosen and stored in the global list. The window is then centred on the next pixel and the same processing iteration occurs. Both lists use parameters (ρ', θ') and are indexed by ρ'. These lists keep track of the start and end points of each line segment and when two line segments are found to be collinear and non-overlapping they are combined into one. The meaning of collinear and non-overlapping can change to allow for slightly different orientations and locations.

2.2 Probabilistic Hough transforms

In this section a new family of Hough techniques called probabilistic HTs is investigated. These include three different techniques, the Dynamic Combinatorial HT, DCHT, Probabilistic HT, ProbHT, and the Randomized HT, RHT. In addition, three extensions of the RHT that are proposed to improve its performance are discussed. These are the Dynamic RHT, DRHT, Random Window RHT, RWRHT, and the Window RHT, WRHT.

The DCHT [1] is proposed to reduce the computational overhead of the CHT. In the CHT, if there are m collinear points, then m-1 two-point line segments all contribute to the same (ρ', θ') entry in parameter space. The DCHT puts all points in an image segment into a list, θ' is calculated from (2.1) for every two-point line segment involving the first point and the remaining points in the list are accumulated into a θ' histogram. If m of the points in the list are collinear with the first point then there is a peak with value m at the θ' value in the histogram.

The ProbHT [7] employs only m edge points ($m < M$) selected at random, from an image with M edge points, which are used as input to the HT. Since the parameter space (ρ, θ) is divided into $N_\rho \times N_\theta$ cells to represent the accumulation array, the number of operations is proportional to $m \cdot N_\theta$. Significant computational saving can be achieved if m can be made much smaller than M.

The limit as to how small m can be made is the requirement that the features in the image are able to be detected as significant peaks in parameter space. Thus, the optimal choice for the size of m is prob-

lem dependent.

The RHT [9] creates a set D of all edge pixels, then at each iteration two pixels $d_1 = (x_1, y_1)$, $d_2 = (x_2, y_2)$, $d_1 \neq d_2$ are taken from set D such that all points have an equal probability of being selected. Then (3) and (4) are solved to obtain parameter point $p_i = [a_1(i), a_2(i)]$ where $a_1 = \rho', a_2 = \theta'$, and put this point into parameter set P. It can be seen that collinear points on a line will accumulate several p_i points at point (a_1, a_2).

The parameter data set P is a list that is searched each time point p_i is obtained. If there is an element in P with the same parameter pair as p_i then the score of that element is increased by one. If no element is found then a new cell is created with the parameters of p_i and a score of one, and is inserted into P as a new element. When an element has a score larger than a threshold, n_t, it is considered an accumulated cell and all points d_i in D lying on the line with the parameters given by p_i are removed.

The RHT can also be used to detect curves linear with respect to the parameters (α_i)

$$\alpha_1 z_1 + \alpha_2 z_2 + \ldots + \alpha_n z_n + z_0 = 0 \qquad (2)$$

where $z_i, i = 0, \ldots, n$ depend only on (x, y) and are constants. For these curves, n pixels are randomly picked and the n parameters can be solved from the n joint linear equations. The RHT can not be used directly for curves expressed by equations which are nonlinear with respect to the parameters.

The following three Hough techniques are extensions proposed to improve the performance of the RHT.

The DRHT [6] is an iterative process of two RHTs.

1. First the RHT is executed until a global maximum in parameter space is detected (i.e. until a curve is found).
2. Second, the set of points found in the first iteration is used to guide scanning along the curve with a width of a few pixels to collect points that are near the curve.
   (a) This scan accounts for the possible errors in quantization of the curve.
   (b) In the second iteration the resolution and threshold are higher than in the first iteration.
3. After both iterations are performed, all points in the image that were found are removed, the accumulator is zeroed, and the first iterative process is performed again (Step 1).

The RWRHT [6] randomly selects a window location for an $m \times m$ window, in a binary image, in which the RHT procedure is performed. The size m is also randomized ($m_{min} \leq m \leq m_{max}$). Random sampling is repeated R times, where R could be a function of m. This window sampling is repeated until a predefined threshold t is reached. Lines are detected one by one until the pre-specified maximum number of lines is reached (l_{max}) or some other heuristic criterion is reached. When a line is found, and it is verified to be a true line, its pixels are removed from the binary image.

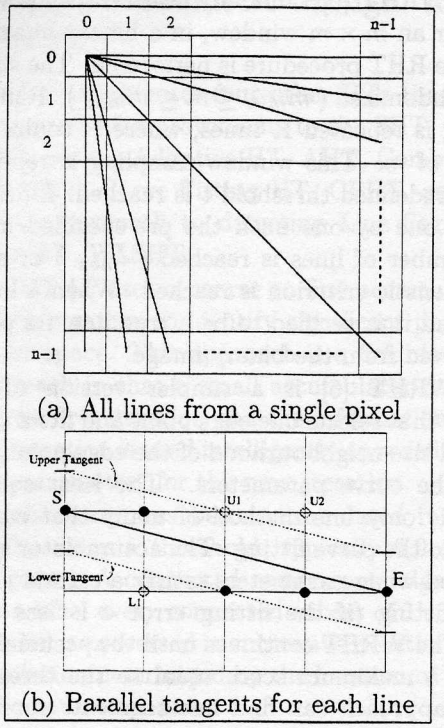
The WRHT [6] is a simpler version of the RWRHT that selects one edge point and fits a curve to a fixed size neighbourhood of the edge point, and defines the curve parameters. The least squares method is only one method of many that can be used to do the curve fitting. The accumulator space is updated when parameters satisfy a certain goodness of fitting (if the fitting error e is less than e_{max}). The WRHT continues until the accumulator contains a maximum score equal to the threshold t. This approach can find line segments curve by curve.

3 Proposed Digital Hough Transform, DHT

The proposed HT is called a digital HT because it detects lines in an image entirely in the discrete domain. The DHT uses pre-computed *digital* lines to guide line detection and thus does not depend on any parameterization of a straight line. All pre-computed digital lines are converted to a rule-based approach that employs a refined set intersection as the only run-time calculation, thus execution is very fast. The proposed DHT is comprised of two distinct steps. The first step generates all digital lines in an $n \times n$ neighborhood. The second step uses the lines from the first step to detect lines in images.

3.1 Step 1: Pre-compute Digital Line Information

In this step, all possible digital lines, with a thickness of 1 pixel, are computed in an $n \times n$ neighborhood. Starting from pixel $(0, 0)$ to pixel $(0, (n-1))$, then $(0, 0)$ to $(1, (n-1))$, and so on, as shown in Figure 1(a). Then pixel $(0, 1)$ to $(1, (n-1))$, and so on until all pixels that lie on the perimeter of the $n \times n$ neighborhood are used as the start and end points for the pre-computed line information. To account for all possible lines it is necessary to include the



(a) All lines from a single pixel

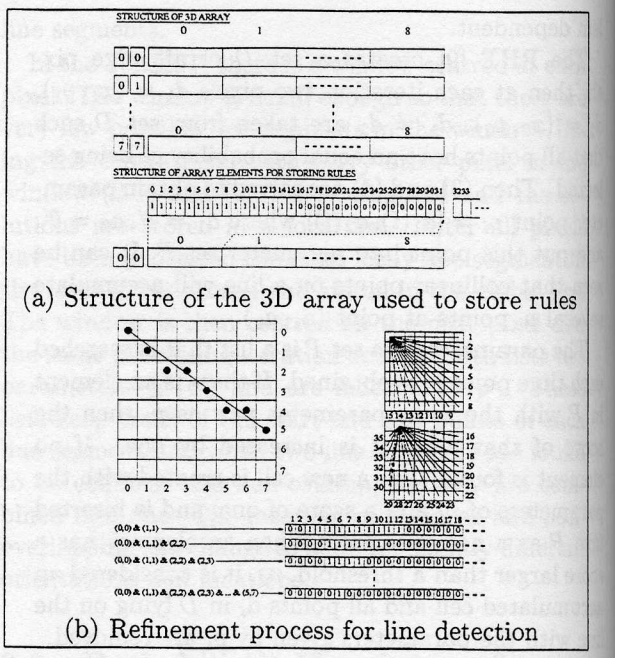
(b) Parallel tangents for each line

Figure 1: Line generation in an $n \times n$ neighborhood.

tangents of each line, as shown in Figure 1(b). Tangents of a line start at ± 0.5 of the start and end points of the desired line. For example, the Upper Tangent to the line shown in Figure 1(b) starts at $S - 0.5$ and ends at $S + 0.5$. Since a digital line is a discrete approximation of a continuous line, it is possible for a line to begin anywhere between where the Upper and Lower Tangents begin and end where the Upper and Lower Tangents end and have a line with the same start and end points as the desired line (S and E) but with different points in between (e.g. $U1, U2, L1$). By including points $U1, U2, L1$, as seen in Figure 1(b), all possible points for lines with the discrete end points S and E are captured.

A *rule* is simply a number associated with a particular generated line. For each line that is generated, a *rule number* is associated with that line. For example, the first line generated from $(0,0)$ to $(0,(n-1))$ will be rule 1, $(0,0)$ to $(1,(n-1))$ will be rule 2, and so on for all possible lines.

Each pixel that is part of line 1 will have rule 1 as part of its set of rules, and the same for each pixel in all lines. For example, if the neighborhood is 8×8, we can see from Figure 1(a) that the set of rules for pixel $(0,0)$ would be $1, 2, \ldots, 15$. Each pixel's set of rules provides all lines that a pixel is part of.



(a) Structure of the 3D array used to store rules

(b) Refinement process for line detection

Figure 2: Storage and use of rules for line detection.

3.2 Step 2: the Digital Hough transform, DHT

An initialization step for the algorithm is to load all pre-computed rules into a 3D array, as shown in Figure 2(a). Using an array of type *int* which represents an integer as 32 bits (4 bytes) allows 32 rules to be packed into each array element, this can be seen from the *Structure of Array Elements* in Figure 2(a). The position of a bit denotes the corresponding rule. If the i^{th} bit is set, it implies the i^{th} rule is in the set. Rules 0-32 are packed into array[i][j][0], 33-64 are packed into array[i][j][1], and so on for all rules where (i,j) represent the pixel coordinates in the $n \times n$ neighborhood. The size of the 3D array is determined by the size of the $n \times n$ neighborhood by:

$A = L/R$ where,

A is the number of array elements needed to store all rules for each pixel

L is the number of lines generated (number of rules) in the $n \times n$ neighborhood

R is the number of rules that can be packed into one array element

For example, if an 8×8 neighborhood is used, shown in Figure 2(a),

$L = 270$

$R = 32$ (bits per integer)

$A = 270 \div 32 = 8.4375$ therefore the possible array elements are $1, 2, \ldots, 9$.

The 3D array is then $8 \times 8 \times A$, thus there are $64 \times 9 = 576$ elements in the array. Array[0][0][$1, 2, \ldots, 9$] is used to store rules for pixel (0,0), array[6][2][$1, 2, \ldots, 9$] is for pixel $(6, 2)$ and its rules.

The DHT uses the pre-computed line information in the following algorithm:

1. Scan the image inside the $n \times n$ neighborhood until an edge pixel, P^0, is encountered, with N^0 being the 8-neighborhood of P^0.

   if P^1 is an edge pixel, $P^1 \in N^0$ and $P_R^0 \cap P_R^1 \neq \emptyset$, then

   $L_R^1 = P_R^0 \cap P_R^1$, where P_R^i is the set of rules for pixel P^i and L_R^i is the set of rules for the line in the i^{th} iteration.

2. Continue adding pixels that are 8-neighbors and that belong to the line

   $i = 1$

   while(**begin** P^{i+1} is an edge pixel, $P^{i+1} \in N^i$ and $P_R^i \cap P_R^{i+1} \neq \emptyset$)

   $L_R^{i+1} = L_R^i \cap (P_R^i \cap P_R^{i+1})$, where this refinement process is the accumulation process as shown in Figure 2(b).

   $i = i + 1$

   end

   When an edge pixel has no 8-neighbors where the set intersection is not the empty set, or the edge of the $n \times n$ window is reached, the line segment is added to the list of detected line segments. The line segment is first compared with all existing segments in the list to see if it has the same orientation (collinear) and if the end points of the two segments are within a certain distance from each other (overlapping). If it is collinear and overlapping with a segment in the list, the two segments are combined into one segment; if not it is added to the end of the list as a new segment.

3. Process all edge pixels, P^i, in the $n \times n$ neighborhood as given in the two steps above.

4. Process all $n \times n$ neighborhoods in the image as given in the three steps above.

This constant refinement of the lines set of rules using the set intersection is a fast means of creating sets of pixel that belong to the same line without the need for other run-time calculations or parameters.

4 Experimental Results

Many tests were conducted. Because of the page limitations, only the results of two sets of data are reported in this paper. Figures 3 contains 7 lines with random noise. Figure 4 contains a complex image with many small artifacts and parallel lines very close together. The images in Figures 3 & 4 are 256×256. All HTs were implemented in C. The source code for the WLSHT was obtained from an ftp site provided in [10]. All other HTs were obtained from a Web site provided in [5]. The source code for all HTs was modified to include consistent timing across all techniques and to handle different image sizes. The results of this testing are shown in Table 1. All testing was performed on a Sun UltraSPARC 1 with 128Mb of memory running Solaris 2.5.1.

All HTs are evaluated on their accuracy (detecting the correct number of lines in noisy images), computation time (time to process the input image), and visual evaluation (does the output of each HT resemble the input image). Computation time and accuracy measurements are given in Table 1.

The proposed DHT is shown to be the fastest of all the HT's by quite a large margin and the accuracy can be seen to be among the best of the HT's tested.

In Figure 3 it can be seen that the SHT, CHT, and DHT produce results that are quite accurate, although the SHT introduces a false line. Again, it can be seen that that DHT is substantially faster then either of the other methods that produce acceptable results. In fact, the DHT is 10 times faster than the nearest HT with similar results.

In Figure 4 it can be seen that a number of HT's produce an acceptable representation of the input image and handle noisy edges quite well. Determining an acceptable representation of the input image is subjective criteria, but it can easily be seen that the CFHT introduces a large number of artifacts while the AHT, PROBHT, and WRHT, do not produce results that represent the input image. Although, the WLSHT appears to do a reasonable job when its parameters are set at 3,1 we can see from Table 1 that it detects an unreasonably large number of lines which demonstrates its inaccuracy.

The test results show that the proposed DHT is one of the most accurate HT's with all types of im-

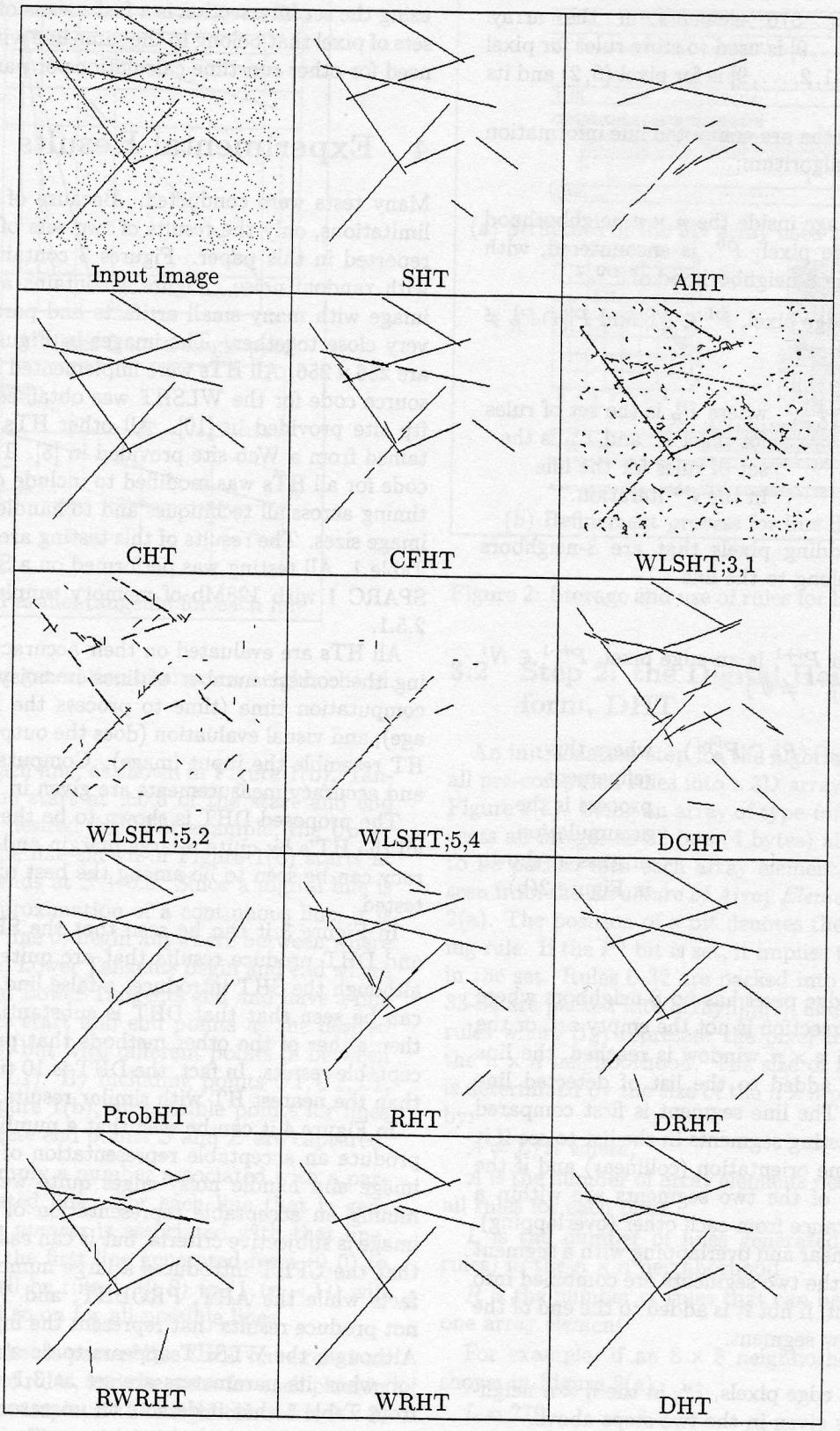


Figure 3: The results of applying each Hough technique to the input image is shown.

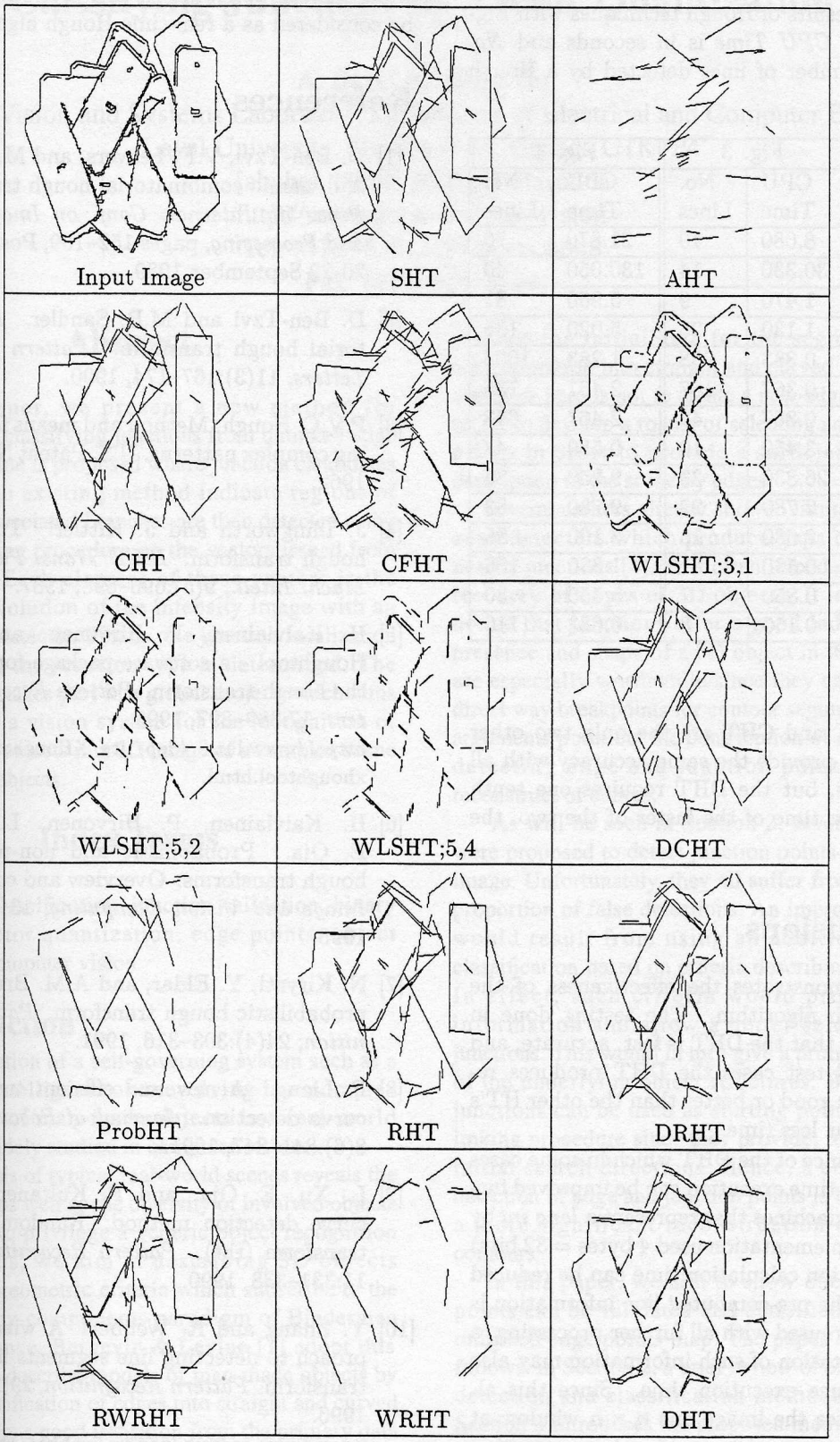


Figure 4: The input image is a 256 × 256 binary image with many small artifacts, close parallel lines, and noisy edges. The results of applying each Hough technique to the input image is shown.

Table 1: Test results of Hough techniques with Figures 3 and 4. *CPU Time* is in seconds and *No. Lines* is the number of lines detected by a Hough technique.

	Fig. 3		Fig. 4	
	CPU Time	No. Lines	CPU Time	No. Lines
SHT	8.080	10	21.870	70
AHT	30.330	14	130.050	40
CHT	1.470	9	5.950	81
CFHT	1.130	25	5.020	135
WLSHT;3,1	0.383	958	1.283	1005
WLSHT;5,2	0.367	167	2.467	736
WLSHT;5,4	0.367	42	2.467	642
ProbHT	3.450	10	9.550	43
DCHT	26.830	33	9.250	83
RHT	2.780	22	2.780	83
DRHT	2.680	16	3.100	85
RWRHT	0.530	31	1.380	106
WRHT	0.850	7	5.450	30
DHT	0.150	8	0.683	110

ages. The SHT and CHT are the only two other techniques that provide the same accuracy with all types of images, but the DHT requires one tenth the computation time of the faster of the two, the CHT.

5 Conclusions

This paper demonstrates the effectiveness of the proposed Hough algorithm. The testing done in this paper show that the DHT is fast, accurate, and efficient. In all test cases the DHT produces results that are as good or better than the other HT's tested, but in far less time.

The performance of the DHT which in some cases approaches real-time execution can be improved further. By using machines that represent a *long int* by 8 bytes (this implementation used 4 bytes = 32 bits) the set intersection calculation time can be reduced further. Since the pre-computed line information is static and thus reused with all further processing, a VLSI implementation of such information may also be used to reduce execution time. Since this algorithm processes the image one $n \times n$ window at a time, parallel processing could reduce execution time substantially.

Considering the results from the testing conducted in this paper and the further performance enhancement possibilities, this proposed DHT can be considered as a real-time Hough algorithm.

References

[1] D. Ben-Tzvi, V.F. Leavers, and M.B. Sandler. A dynamic combinatorial hough transform. In *Proc. 5th Internat. Conf. on Image Analysis and Processing*, pages 152–159, Positano, Italy, 20-22 September 1989.

[2] D. Ben-Tzvi and M.B. Sandler. A combinatorial hough transform. *Pattern Recognition Letters*, 11(3):167–174, 1990.

[3] P.V.C. Hough. Method and means for recognizing complex patterns. U.S. Patent No. 3069654, 1962.

[4] J. Illingworth and J. Kittler. The adaptive hough transform. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(5):690–698, 1987.

[5] H. Kalviainen, P. Hirvonen, and E. Oja. Houghtool - a software package for the use of the hough transform. *Pattern Recognition Letters*, 17:889–897, 1996.
http://www.lut.fi./dep/tite/XHoughtool/xhoughtool.html.

[6] H. Kalviainen, P. Hirvonen, L. Xu, and E. Oja. Probabilistic and non-probabilistic hough transforms: Overview and comparisons. *Image and Vision Computing*, 13(4):239–252, 1995.

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

[8] P. Liang. A new and efficient transform for curve detection. *Journal of Robotic Systems*, 8(6):841–847, 1991.

[9] L. Xu, E. Oja, and P. Kultanen. A new curve detection method: Randomized hough transform (rht). *Pattern Recognition Letters*, 11:331–338, 1990.

[10] Y. Zhang and R. Webber. A windowing approach to detecting line segments using hough transform. *Pattern Recognition*, 29(2):255–265, 1996.
ftp.csd.uwo.ca/pub/papers/webber/Yuefeng/HoughLineSegmentCode.c.