A Fast Hough Transform for Segment Detection

Nicolás Guil, Julio Villalba, and Emilio L. Zapata

Abstract— **In this paper, we will describe a new algorithm for the fast Hough transform (FHT) that satisfactorily solves the problems other fast algorithms propose in the literature—erroneous solutions, point redundance, scaling, and detection of straight lines of different sizes—and needs less storage space. By using the information generated by the algorithm for the detection of straight lines, we manage to detect the segments of the image without appreciable computational overhead. We will also discuss the performance and the parallelization of the algorithm we present and will show its efficiency with some examples.**

I. Introduction

IMAGE recognition is one of the most frequent and necessary tasks in the area of computer vision. The algorithms derived from the formulation of the Hough transform (HT) [1] have been shown as efficient tools for achieving this objective. The abundant literature about the HT for straight line recognition can be classified into two large groups depending on the parametrization used for expressing the lines.

On one hand, the most abundant group corresponds to those using the parameters ρ and θ, where ρ is the distance from the line to the origin and θ is the angle from a vector which is normal to the line to the abscissas axis. In this group, we can include the work about the combinatorial Hough transform (CHT) [2] and the piecewise-linear Hough function (PLHT) [3]. Another group uses the parameters m and c to express the lines, where m is the slope and c is the point of intersection with the ordinate axis. Some articles of this second group are the binary Hough transform (BHT) [4], the randomized Hough transform (RHT) [5], and the fast Hough transform (FHT) [6].

The main difference of the first group with respect to the second is that the range of the parameters in the transformed space is limited, whereas those of the second group have the problem of infinite slopes. However, an aspect that is common to all the algorithms developed is that they require the application to the image of an edge detector, such as Sobel's operator [7], before starting with the recognition process.

Our solution for the problem of recognition, developed in this work, uses the second parametrization. For this reason, we are going to mention some of the ideas of the algorithms corresponding to this parametrization. The BHT, introduced by [4], minimizes the computation, formulating expressions so that they are reduced to simple addition and shift operations. However, in order to perform the polling and solve the problem of infinite slopes, they need four parameter spaces where they can accumulate the votes.

With the RHT, Xu *et al.* [5] try to reduce the computation by randomly performing the coupling of the points of the image space and finding, for each pair, the values of m and c. These values vote in an accumulator. When the count reaches a given threshold value, we assume that the corresponding cell in the parameter space indicates the solution. Problems can arise if the threshold value is low and there is much noise in the image, and erroneous solutions might be generated. If we raise the threshold value, on the other hand, the amount of computation increases.

Using the FHT, which is introduced in [6], the detection is carried out through a hierarchized approximation to the solution. The way in which this process is carried out permits dispensing with the vote accumulation in the parameter space. However, this algorithm also presents some problems, such as point redundance, scaling, erroneous solutions, and detection of segments of different sizes.

In this paper, we will present a new algorithm for the calculation of the FHT that satisfactorily solves the problems found in the algorithms of [6] and those mentioned here. In addition, we will show how to detect segments using the information of the FHT algorithm. As an objective for the implementation of the algorithm we present, we optimize two general aspects of the HT: i) complexity and ii) memory requirements.

The rest of this work has been organized as follows: In Section II, we briefly describe the algorithm for the FHT introduced in [6] and the problems it presents. In Section III, we develop the solutions and the algorithm we propose, with a particularization of the method for segment detection. In Section IV, we evaluate the behavior of the algorithm we propose and show an example of the behavior of the algorithm in different situations for the detection of a real image.

II. Fast Hough Transform

The FHT algorithm recognizes straight lines of an image space by performing a hierarchized approximation to the solution [6]. For this, each point $P = (x_j, y_j)$ of an image space (in which there are only boundary points) generates a straight line in a parameter space according to equation

$$c = -x_j * m + y_j \tag{1}$$

where x_j plays the role of the slope, and y_j takes the value of the point of intersection with the ordinate axis. This way, a straight line in the image space will generate a bundle of straight lines in the parameter space that will intersect in a

Manuscript received August 12, 1992; revised February 8, 1995. This work was supported by the Ministry of Education and Science (CICYT) of Spain under project TIC-92-0942-C03-01. The associate editor coordinating the review of this paper and approving it for publication was Prof. Robert M. Haralick.

The authors are with the Department Arquitectura de Computadores, University of Málaga, Plaza El Ejido, Málaga, Spain.

IEEE Log Number 9414598.

that cross the quadrant, and the presence bit associated with those lines.

In Fig. 1(c), we can observe the path followed by our algorithm in order to find the solution to the line of Fig. 1(a). Taking into account the previous considerations, the algorithm for the calculation of the solution for the points belonging to $C(C = C_\alpha \cup C_\beta)$ will look like the following:

```
FOREACH C_i in C_α, C_β {
  level = 0;
  FORALL edge_point j ∈ C_i{
    level.disn_j = (A_2j + 0.5 * A_1j + 0.5 * A_0j)/N;
    if (|level.disn_j| <DISN) {
      level.presence_j =TRUE;
      level_vote++;
    }
    else level.presence_j = FALSE;
  }
  level++;
  level.son = −1;
  WHILE (level) {
    level.vote = 0;
    level.son++;
    if (level.son < 4) {
      FORALL edge_point j ∈ C_i {
        if ((level−1).presence_j ) {
        level.disn_j = (2*((level − 1).disn_j)
          + A_1j * b [level.son][0]
          + A_0j * b[level.son][1]);
          if (|level.disn_j| < disN ) {
            level.presence_j = TRUE;
            level.vote++;
          }
          else level.presence_j = FALSE;
        }
        else level.presence_j = FALSE;
      }
      if (level.vote > THRESHOLD) {
        if (level == log_2 N + 1) {
          solution();
          remove_points();
        }
        else {
          level++;
          level.son = −1;
        }
      }
    }
    else level−−;
  }
}
```

Level indicates the level of the quadrant being processed. There will be a structure associated with each level where the information about the distance and the presence bit will be stored. DISN is the normalized distance from a vertex of a subquadrant to its center. THRESHOLD is the threshold value which indicates that a quadrant is divided again because enough lines cross it. $N \cdot N$ is the size of the image. A_{ij} are the coefficients of point j that define the line associated with this point (i can take the values 0, 1, and 2). There is a first FOREACH loop independently executed by the algorithm for the coefficients of the image points in the sets C_α and C_β. The first FORALL is in charge of the initial processing of the whole parameter space. The second FORALL processes the next subquadrants derived from the previous one. A new division of a quadrant will be performed when the number of lines that cross it is larger than the threshold (level.vote > THRESHOLD). When the processing of a given $C_i(i = \alpha, \beta)$ set is finished, all the offspring subquadrants of those parents that surpass the threshold values will have been evaluated (level = 0).

The routine remove_points() is the one in charge of eliminating the points of the set C_i, which collaborate in a solution line. These points will not have to be calculated in another node. The construction of this routine will be as follows:

```
FORALL edge_point j ∈ C_i
  if (level.presence_j)
    FOR cont_level = 0 until cont_level = log_2 N
      cont_level.presence_j = 0.
```

The routine solution() is in charge of obtaining the results of the application of the algorithm, that is, the slope and the point of intersection with the ordinate axis of C_α (1) and the inverse of the slope and the point of intersection with the abscissas axis of C_β (5). Even though, with this result, we obtain the lines of the image space, by means of some additional operations, we can detect segments.

B. Segment Detection

During the process of producing a solution in the FHT, we do not leave out the points of the image that have collaborated to that solution. This facilitates the detection of segments in the image. Thus, once the solution has been obtained, by means of the presence vector (bits in TRUE), we identify the points that participate in that result. In reality, the data we obtain directly are the coefficients A_{ij} of the points that belong to that result, but the points x_j and y_j can be easily found from them by means of (2) and (6) scaled.

The detection of the edge points of the image applying Sobel's operator was carried out by reading the image points from left to right and from bottom to top. This way, the points (really their coefficients) belonging to a solution line are ordered so that the extremes of the segment can be easily identified by finding the two points of the solution line that are further away from each other. In Fig. 2, we can observe that by performing the scan indicated above on the line with positive slope, the point (x_a, y_a) will be the first one detected and the point (x_b, y_b) the last one detected. Thus, in order to delimit the segment, we need only to find the two points of the solution whose first and last bits are one, respectively, in the presence vector. This way, the function solution() of the algorithm of the previous section will look like the following:

```
j = 0;
WHILE edge_point j ∉ straigh_line_solution
  inc j;
final_point_detection(A_ij);
j = total_edge_point;
```

slopes are in the interval $[-1, 0, 1]$, they cannot be finely detected.

4) *Detection of Lines of Different Sizes:* The lines detected will be those that have a number of points larger than the threshold value T. Due to the fact that through the solution in the parameter space we have a line for each point of the line in the image space, this line must have a number of points larger than the threshold quantity T. This may complicate the detection of lines with sizes smaller than T. On the other hand, if we lower the value of T excessively, we will increase the number of computations and we could obtain unreal solutions as a consequence of the collaboration of lines from different bundles.

III. FHT Algorithm

The problems appearing in the calculation of the FHT that we have mentioned in the previous section can be solved in a satisfactory way. We will now detail these solutions.

1) *Erroneous Solutions:* For solving this problem, we eliminate lines from the image space as we obtain solutions. This way, we restrict the number of lines in the parameter space, accelerating the calculations and reducing the probability of finding false solutions. In the algorithm of the FHT, the elimination of lines from the parameter space is very easy if we act on the presence vector. This vector will indicate if a line, of that space, crosses a given quadrant. Thus, setting to zero the bits that identify the lines generated by the points of the solution line of the image space, we will eliminate this line from further calculations.
2) *Redundance of Points:* This problem is originated by the need of changing the parametrization of the lines generated by the points of the image in order to detect infinite slopes. This change consists of expressing the equation of the line using as parameters the slope m and the point it crosses the abscissas axis so that

$$y = m * (x - x_0). \tag{4}$$

If we now make $t = 1/m$, we obtain $x = y \cdot t + x_0$. Thus, in the parameter space, each point $P_j = (x_j, y_j)$ of the image is transformed into a line

$$x_0 = -x_j * t + y_j. \tag{5}$$

The coefficients of the points to which we apply this parametrization will take the following values:

$$A_{0j} = \frac{1}{\sqrt{1 + y_j^2}} \quad A_{1j} = y_j * A_{0j} \quad A_{2j} = -x_j * A_{0j} \tag{6}$$

which, as can be observed, maintain the same expression as (2), with the difference that values x_j and y_j exchange the locations in which they appear.

Evidently, this parametrization must be applied to points belonging to lines with large slopes (in the algorithm developed in this paper, slopes larger than one). On the other hand, (2) needs only to be applied to points belonging to lines of smaller slopes (less than one). The points belonging to one group or the other may be found during the edge detection stage, before the application of the FHT algorithm, using information from the gradient vector of Sobel's operator applied to the points. This way, in a given parameter space, only the points that contribute to a solution in this space will be processed, saving computational time.

3) *Scaling:* By scaling the slopes of the lines in the parameter space, we can make the detection of lines with slopes of less than one possible. This is achieved by scaling the variable x_j of (1)

$$c = \frac{-x_j}{S} * m + y_j \tag{7}$$

where S is the scale factor. This scale factor is going to modify the values of the coefficients of (2) so that the new coefficients can be found by substituting in (2) the value of x_j by x_j/S. In the same way, the new coefficients derived from (6) will be found by substituting the value of y_j by y_j/S. Taking into account that S is a factor that is going to be dividing, in order to accelerate the algorithm, it is convenient that it be a power of 2 (typically, if the image is $N \cdot N, S = N$).

4) *Detection of Different Sized Lines:* This problem can be solved by applying the algorithm several times, starting with a high threshold value. This value will be progressively reduced, and as all the points of the solution lines are eliminated from further treatments, the lines of smaller sizes can be detected.

A. Design of the Algorithm

We now present an FHT algorithm that uses all the solutions mentioned above. This algorithm operates more than two sets of points created during the stage of application of Sobel's operator to the image. In this stage, the coefficients of the edge points that have a gradient vector with an absolute value angle between $\pi/4$ and $3\pi/4$ (they belong to lines with slopes between -1 and 1) will be calculated using (2) with scaling and will be stored in a set of coefficients called C_α. The coefficients of the rest of the points will be calculated using (6) with scaling and will be stored in C_β.

The functioning of the algorithm is similar to an inverted tree. The nodes of the tree are the quadrants to be processed and the branches are the subquadrants obtained from the parent quadrant if there are a number of lines larger than the threshold crossing it. We will use a series of structures where the values obtained in processing each level of the tree will be stored. In order to minimize the amount of memory needed without significant computational penalizations, the algorithm will implement only one structure for each level. When a quadrant is divided, the offspring subquadrants will be processed sequentially (from the first to the fourth quadrant). If one of the subquadrants must be divided, the process will be applied again in the new level over the new subquadrants. The structure must contain information about the distance to the center of the quadrant of that level, the number of lines

solution point. This solution point will indicate the slope, m, and the point of intersection with the ordinate axis, c, of the straight line in the image space, that is, the solution we sought. In Fig. 1(a), we show an example of a line made up by four points in the image space. These points will generate four lines in the parameter space following (1).

The FHT algorithm performs the identification of the solution points in the parameter space using a method for focalizing on them. In order to do it, it dimensions a square parameter space that is divided into four quadrants. A given quadrant is divided again if a large enough number of lines (larger than a threshold value) crosses it. The recursive application of this process will approach a solution. The way in which we verify whether a line crosses a given quadrant is by calculating its distance to the center of the quadrant. In order to do this, we need to know only this center and the coefficients giving the straight line. These coefficients are defined for a point $P = (x_j, y_j)$ as

$$A_{0j} = \frac{1}{\sqrt{1 + x_j^2}} \quad A_{1j} = x_j * A_{0j} \quad A_{2j} = -y_j * A_{0j} \quad (2)$$

and the distance can be calculated as

$$d_j = A_{2j} + A_{1j} * C_m + A_{0j} * C_c \quad (3)$$

where (C_m, C_c) are the coordinates of the center of the level lth quadrant.

In order to minimize the number of calculations, we define a presence vector that will indicate whether a line of the parameter space crosses a given quadrant. If the line does not cross the quadrant (corresponding bit of the presence vector is zero), it will not cross the subquadrants generated from it either, and it will not be necessary to calculate the distance of the line to these subquadrants. Thus, by using the value of the presence vector of the parent quadrant, we can eliminate computations. In addition, the method followed in [6] results in that once we have the values for the distance to the center of the first quadrant, the calculation of this distance for the subquadrants is reduced to simple addition and shift operations.

The level approximation to the correct solution will permit achieving an advantageous computation time with respect to the traditional method; we do not perform calculations when they are not useful. In addition, by using adequate algorithms, such as the one developed in this paper, we minimize the amount of memory necessary for storing the information. Nevertheless, in order to optimize the results, we have solved some problems originated by the type of parametrization used and by the detection method followed. These problems, which were not solved in [6], are the following.

1) *Erroneous Solutions:* As we have seen, a straight line in the image will produce a bundle of lines in the parameter space. The point where the lines of this bundle intersect will be the solution sought. However, incorrect solutions due to the combination of lines from different bundles that converge in a quantity larger than the threshold in a point of the parameter space can be generated. This type of event may produce, in the most extreme case, incorrect solutions or may more often "distract" the execution of the algorithm, reducing speed.
2) *Redundance of Points:* With the parametrization of the line used, slope, and point where it intersects the origin, it is impossible to detect infinite slopes. In order to solve this problem, a new parameter space is defined. In this additional space, the lines generated by the image points are represented with inverse slope. This way, there are two parameter spaces in which all the lines generated by the points of the image must be included. A repetition of these points in the two spaces is produced and results in a larger number of computations.
3) *Scaling:* Another problem that arises in the calculation of the FHT is that slopes in the image space that are smaller than one cannot be clearly detected. The parameter space initially defined has a size of $2N \cdot 2N$ ($N \cdot N$ size of the image space), which permits the detection of slopes in the range $[-N, -N+1, \ldots, -1, 0, 1, \ldots, N]$. If the

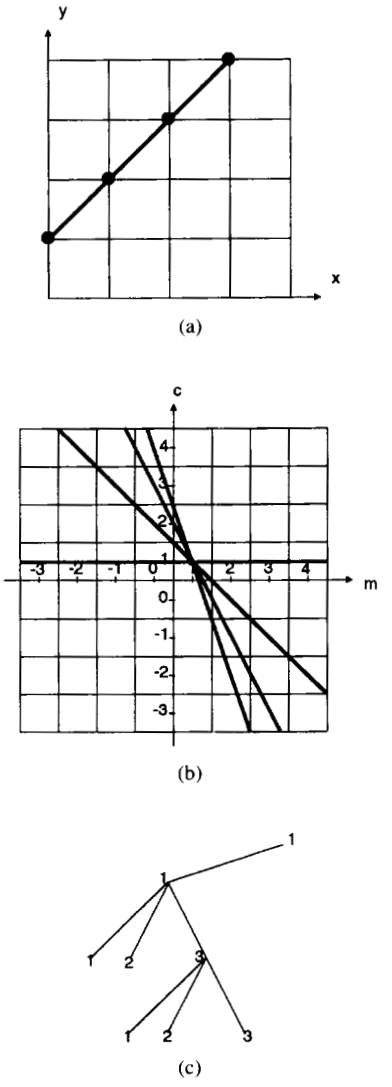


Fig. 1(a). 4×4 image space in which we show a straight line with $m = 1$ and $c = 1$; (b) parameter space and line bundle associated with the solution; (c) tree structure generated by the algorithm for finding the solution. Threshold value = 4.

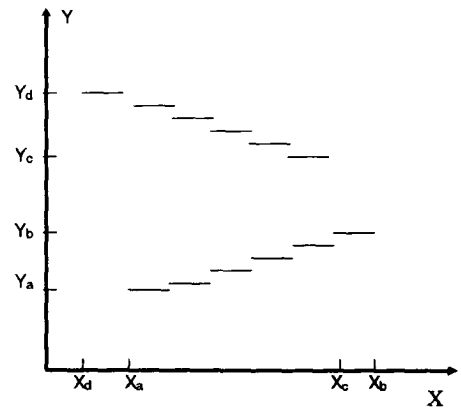


Fig. 2. Detection of segment extremes for lines with slopes smaller than one (absolute value).

WHILE edge_point $j \notin$ straight_line_solution
dec j;
final_point_detection (A_{ij}).

This way of detecting the extreme points of the segment will not work adequately in the case of negative slopes smaller than one, as the points of this solution do not have the order we mentioned. This is also shown in Fig. 2. In the figure, it can be observed that the first point of the scan of the image will not be (x_c, y_c), and the last point of the scan will not be (x_d, y_d). In this case, it will be necessary to sample all the points participating in the solution and determine whether they are extreme values.

We might find another problem if some of the solution lines found are made up of several segments. In this case, we will have to find the extremes of all the segments. The problem can be solved if we perform a sequential scan of the points belonging to the solution line (using A_{ij}). If the distance between two points that occupy consecutive positions is larger than a given value, we will consider that each of these points belongs to a different segment.

IV. Evaluation

The evaluation of the algorithm will take into account the most important aspects related with the detection problem: reliability, memory required, and complexity. In the previous section, we pointed out the reliability of the algorithm we propose in solving the problems associated with infinite slopes, scaling, erroneous solutions, etc. We will now perform a comparison of the topics of memory, complexity, and parallelism with the FHT algorithms of [6] and the HT algorithm of [1].

1) *Memory Requirements:* As the amount of memory needed by the different algorithms is well specified in their implementation, it is easy to perform a comparative analysis. Assuming P edge points in the original image, our algorithm will need the following memory space:

$$M = (\log_2 N + 1) * P \text{ words} + (\log_2 N + 1) * P \text{ bits} \quad (8)$$

because there will be a maximum of $\log_2 N + 1$ levels. If we compare this value with the one that would be needed with the traditional HT, typically N^2, we see that our algorithm for the FHT needs less storage space as long as the following condition is met

$$P < \frac{N^2}{2 * (\log_2 N + 1)}. \quad (9)$$

In [6], the best FHT algorithm proposed, from the viewpoint of memory savings, is the one that uses as strategy for following the tree that of "Depth-First," which needs four times more memory space than the one developed in this paper.

2) *Complexity:* The computation time with the HT algorithm [1] is given by the polling in the parameter space, whose complexity is of the order of $P \cdot N$, and by the time necessary to examine the maxima in this parameter space of size N^2. The algorithm we propose needs to perform $3 \cdot P$ operations in order to calculate the coefficients, apart from those necessary for the calculation of the distances (simple addition and shift operations). Although it is difficult to evaluate the amount of computations necessary in order to find the solutions, it is possible to show for a simple case in which there is only one solution the number of nodes that should be evaluated together with the number of necessary operations.

In Fig. 3(a), we show the case in which we have a solution that is located in the most favorable situation for the algorithm to find it in the minimum time. If we assume that the solution line is made up by P points, we will perform $3 \cdot P$ coefficient computations plus $(\log_2 N + 2) \cdot P$ distance calculations. In Fig. 3(b), we have assumed that the solution is in the farthest node. In this case, the number of distance computations is larger $(4 \cdot (\log_2 N + 2) \cdot P)$ as four quadrants are explored each level of the tree. The best FHT algorithm presented in [6] would have the same order of complexity as the worst case of our algorithm in this example. To carry out a rigorous study of the execution times when there are multiple solutions is very complex because of the iterative nature of the algorithm to the different values the threshold can take and to the information of the presence vector.

3) *Parallelism:* An important aspect in the evaluation of the algorithm is the study of its possible parallelization. We know that the traditional algorithm for the HT has a strong implicit parallelism that can be conveniently used, and this has been pointed out in several papers, such as in [8]–[11]. The parallelism in the algorithm we propose can be analyzed by examining the different loops.

- In the FOREACH loop, we can process the two sets of points into which the set C is divided in parallel as the computation for each set is independent.
- In the first FORALL loop, where we compute the distances from the lines to the center of the parameter space, an equal division of the load can be performed for each set so that each processor will calculate an equivalent number of distances.

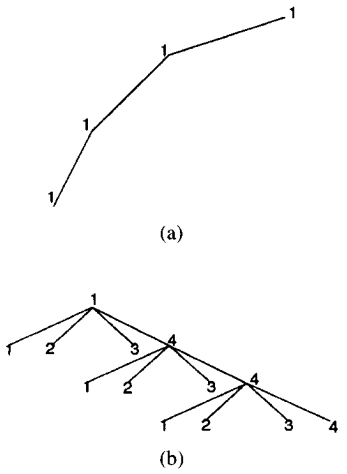


Fig. 3(a). Number of nodes (quadrants) computed in the parameter space for a solution in the optimal case; (b) nodes created in the parameter space for a more unfavorable case, in which the solution is in the farthest node.

This is achieved by assigning each processor an identical amount of point coefficients. It is clear that this division is independent from the memory model we choose (distributed or shared).

- In the second FORALL loop, where the distances to the center of all the subquadrants participating are computed, it is important to take into account the memory model used in order to perform an adequate assignment of the data.

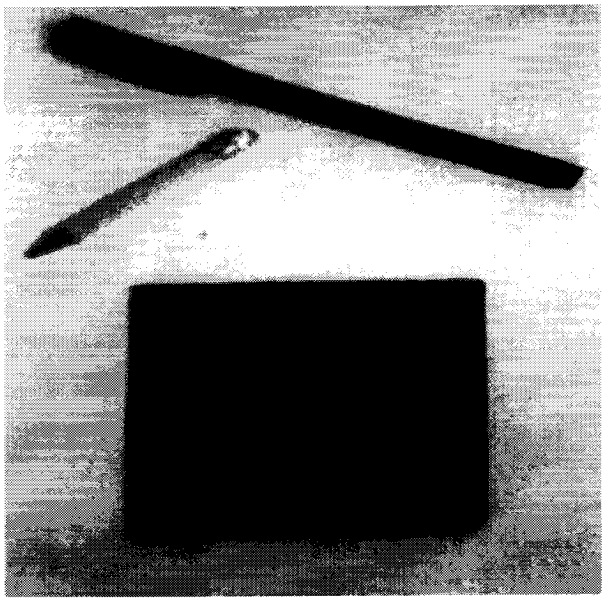
With distributed memory, the parallelism can be used by dividing the parameter space into squares. Each one of these squares is assigned to a processor that is in charge of going through it according to the method mentioned in the previous section. This way, the references to higher subquadrants for the calculation of the distances are local and will not imply interprocessor communications. A problem presented by this division is the load balance as it is possible, depending on the distribution of the solutions, that some processors are idle while others have to continue calculating. An attempt to balance the load would imply communications that would degrade the performance of the algorithm.

With shared memory, we will not have the problem of interprocessor communications, but bottlenecks in the memory system may appear. During the first iteration, there will be no problems as each processor can have the information about the distances to the center of the parameter space stored in a different memory module. However, as we go through the tree, some processors will remain idle. For this processor to be able to continue performing calculations, data from other quadrants, which will be located in other memory modules, will be needed. This will produce conflicts in the simultaneous access by several processors to the same memory module.

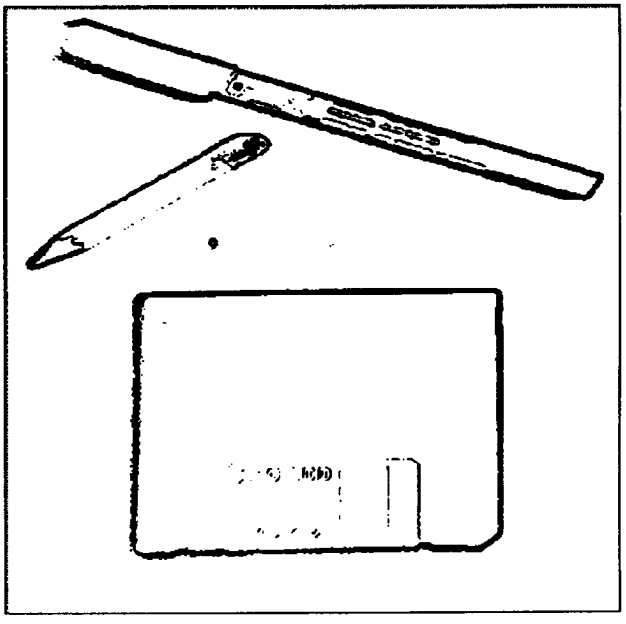
A. Experimental Results

We have carried out different experiments with images generated in the bitmap format. These are 512×512 pixel images, and each pixel contains information on the gray level of that point (in the examples, the pixel can take the binary values 0 or 1). Those images are subsequently treated by means of a Sobel operator, which generates two sets of points. Because of the error in the calculation of the gradient using the Sobel operator, the points with a gradient vector angle close to the value used to discriminate the membership to two sets will be assigned to both. The algorithm we propose acts over the set of edge points found.

In order to illustrate all the possibilites, we have used a real image with three figures formed by segments of different sizes, infinite and negative slopes, positive and negative slopes, and slopes smaller than one. In Fig. 4(a), we show the original image, and in Fig. 4(b), we show the result of the edge

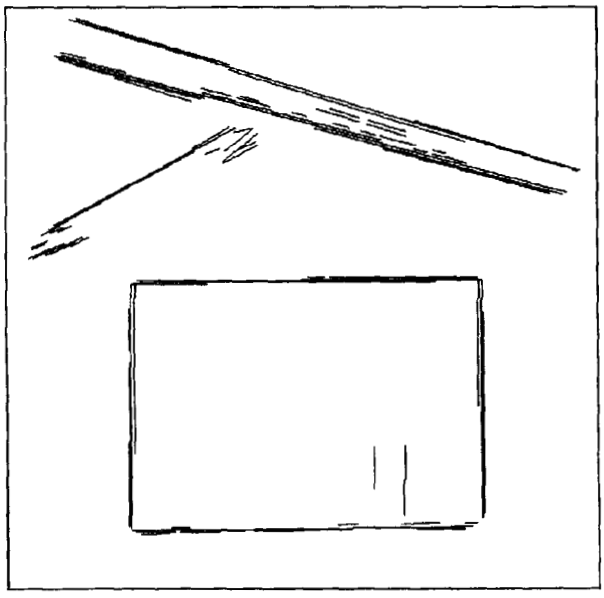


(a)

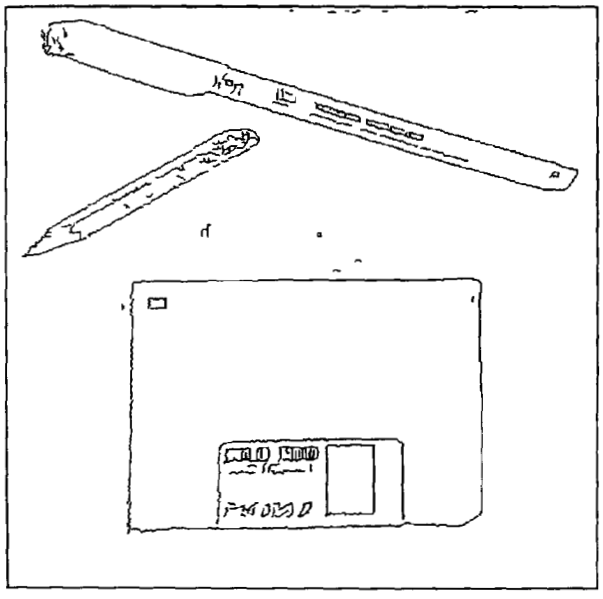


(b)

Fig. 4. (a) Real image with three objects; (b) edge detection applying Sobel's operator.



(c)

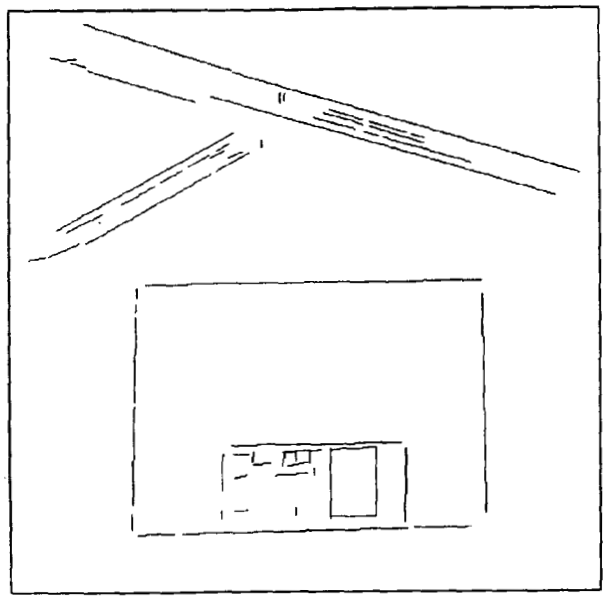


(d)

Fig. 4. (c) Segment recognition applying the algorithm we propose; (d) edge detection applying a recursive symmetric exponential filter.

detection process. We have applied the Sobel operator and found some problems in the detection of the edges. In fact, noise points appear, and several edges are lost. In order to test the robustness of the algorithm, we have not tried to improve the detected edge.

The result of the detection is shown in Fig. 4(c). The algorithm performs several iterations with different threshold values (between 1200 and 40), which allow the detection of segments of different sizes. The maximum gap between points that belong to the same segment is 10. As a solution is found, the points corresponding to this solution are eliminated by acting on the presence vector. It can be observed that as a consequence of the thick edge lines of Fig. 4(b), several nearby



(e)

Fig. 4(e). Segment recognition after applying exponential filter.

segments are detected per each edge line. This problem can be solved by applying a thinning procedure to the edge points. On the other hand, the small segments are detected, but the bottom segment of the little pen is lost since the edge line does not contain enough points.

We have also applied a recursive symmetric exponential filter in order to produce better edge detection. In Fig. 4(d), we display the detected segments. In this case, the algorithm finds the segments with more accuracy. Furthermore, the execution time for the latter edge image is shorter than for the former because the number of edge points is smaller in Fig. 4(e).

V. Final Remarks

A new FHT algorithm for the detection of straight lines has been presented. This algorithm solves all the problems of the previous one using a minimum amount of memory and saving computations. It also permits an easy implementation of segment detection without subsequent processing of the image with the solution lines and without an appreciable computational overhead. In the same way, we show that this technique is robust with respect to noise.

We have also shown the possibility of parallelization of the algorithm for multiprocessors with shared and distributed memory, taking into account the data flux and the distribution of the load among the processors.

On the other hand, the algorithm can be applied to the detection of other shapes, such as circles or ellipses, if we uncouple the parameter to be obtained and substitute the votation process in the parameter space by the focusing algorithm [12].

Therefore, for the circle we split the detection process into two stages. The first stage will detect the center coordinates, and the second stage will find the radius. For the first stage, some algorithms prolong the gradient vector for each edge point generating a beam of lines that intersect at the center of

the circle. Thus, we can use the first part of the algorithm for segments to detect the center of the circles.

In the ellipses, we apply a similar process. It is possible to split the ellipse detection algorithm into three stages. The first stage, which finds the center of the ellipse, can be implemented with information from the gradient vector, but the computational requirements are very high. By doing several transformations, the algorithm will generate a beam of lines, which again will cross the center of the figure. We can also use the algorithm for segments in order to detect the center of the ellipse and, this way, improve the computational requirements.

We have also applied the focusing algorithm for the generalized Hough transform [13]. In this case, we have previously uncoupled the parameters to be calculated (displacement, orientation, and scaling), expressing the information for the different stages in an invariant manner. Using the new information, the stage that calculates the displacement will produce a beam of lines. Then, we have used the focusing algorithm to find the displacement value.

REFERENCES

[1] R. D. Duda and P. E. Hart, "Use of the Hough transform to detect lines and curves in pictures," *Commun. ACM*, vol. 15, pp. 11–15, 1972.

[2] D. Ben-Tzvi and M. B. Sandler, "A combinatorial Hough Transform," *Patt. Recogn. Lett.*, vol. 11, pp. 167–174, 1990.

[3] H. Hoshimizu and M. Numada, "On a fast Hough transform method PLHT based on piecewise-linear Hough function," *Syst. Comput.*, Japan (USA), vol. 21, no. 5, pp. 62–73, 1990.

[4] L. Da Fontoura Costa and M. B. Sandler, "A binary Hough transform and its efficient implementation in a systolic array architecture," *Patt. Recogn. Lett.*, vol. 10, pp. 329–334, 1989.

[5] L. Xu, E. Oja, and P. Kultaken, "A new curve detection method: Randomized Hough transform (RHT)," *Patt. Recogn. Lett.*, vol. 11, pp. 331–338, 1990.

[6] H. Li, M. A. Lavin, and R. J. Le Master, "Fast Hough Transform: A hierachical approach," *CVGIP*, vol. 36, pp. 139–161, 1986.

[7] E. R. Davies, "Circularity: A new principle underlying the design of accurate edge orientation operators," *Image Vision Comput.*, vol. 2, no. 3, 1984.

[8] D. Ben-Tzvi, A. Naovi, and M. Sandler, "Synchronous multiprocessor implementation of the Hough transform," *CVGIP*, vol. 52, pp. 437–446, 1990.

[9] M. G. Albanesi, "Time complexity evaluation of algorithms for the Hough transform on mesh connected computers," in *Proc. IEEE CompEuro'91*, 1991, pp. 253–257.

[10] A. N. Choudhary and R. Ponnusamy, "Implementation and evaluation of Hough transform algorithms on a shared-memory multiprocessor," *J. Parallel Dist. Comput.*, vol. 12, pp. 178–188, 1991.

[11] R. E. Cypher, J. L. Sanz, and L. Snyder, "The Hough transform has $O(N)$ complexity on $N \times N$ mesh connected computers," *SIAM J. Comput.*, vol. 19, no. 5, pp. 805–820, 1990.

[12] N. Guil and E. L. Zapata, "Fast circle and ellipse Hough transform," in *Proc. VI Symp. AERFAI*, Spain, to be published.

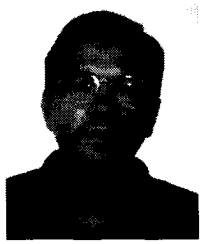
[13] N. Guil and E. L. Zapata, "Fast generalized Hough transform," in *Euro. Robotics Intell. Syst. Conf.*, Málaga, 1994, vol. 1, pp. 498–510.

Nicolás Guil received the B.S. degree in physics from the University of Sevilla, Spain, in 1986.

He worked in the I&D Department of a computer enterprise from 1986 to 1990. During 1990–1994, he was an Associate Teacher in the Department of Computer Architecture at the University of Málaga in Spain. His research interests include image processing and parallel computation.

Julio Villalba received the B.S. degree in physics from the University of Granada, Spain, in 1986.

During 1986–1991, he worked in the I + D Department of Fujitsu-Spain and was an Associate Teacher at the University of Málaga. Since 1992 he has been Full Teacher in the Department of Computer Architecture at the University of Málaga. His research interests include image processing and computer arithmetic.



Emilio L. Zapata received the B.S. degree in physics from the University of Granada, Spain, in 1978 and the Ph.D. degree in physics from the University of Santiago de Compostela in 1983.

During 1978–1981, he was an Assistant Professor at the University of Granada, and during 1982–1991, he successively was Assistant, Associate, and Full Professor at the University of Santiago de Compostela. Currently, he is a Professor with the Department of Computer Architecture at the University of Málaga. His research interests are in the area of parallel computer architecture, parallel algorithms, numerical algorithms for dense and sparse matrices, and VLSI digital signal processing.

Dr. Zapata has published more than 35 papers in refereed international journals in his areas of interest and about 50 refereed international conference proceedings.