Pergamon

Pattern Recognition, Vol. 30, No. 10, pp. 1729–1744, 1997

0031-3203/97 $17.00+.00

PII: S0031-3203(96)00191-4

LOWER ORDER CIRCLE AND ELLIPSE HOUGH TRANSFORM†

N. GUIL and E. L. ZAPATA*

Department of Computer Architecture, Campus de Teatinos, University of Málaga, P.O. Box 4114, 29080-Málaga, Spain

(*Received* 11 *September* 1995; *in revised form* 1 *November* 1996)

Abstract—In this work we present two new algorithms for the detection of circles and ellipses which use the FHT algorithm as a basis: Fast Circle Hough Transform (FCHT) and Fast Ellipse Hough Transform (FEHT). The first stage of these two algorithms, devoted to obtaining the centers of the figures, is computationally the most costly. With the objective of improving the execution times of this stage it has been implemented using a new focusing algorithm instead of the typical polling process in a parameter space. This new algorithm uses a new strategy that manages to reduce the execution times, especially in the case where multiple figures appear in the image, or when they are of very different sizes. We also perform a labeling of the image points that permits discriminating which of these belong to each figure, saving computations in subsequent stages. © 1997 Pattern Recognition Society. Published by Elsevier Science Ltd.

Hough transform Circle and ellipse detection Focusing algorithm
Decoupling of parameters Irregular computation

1. INTRODUCTION

Object recognition is one of the most frequent and necessary tasks in the realm of computer vision. The algorithms derived from the formulation of the Hough transform (HT)[1] have been shown as effective tools in order to achieve this objective. There are three types of figures to which special attention has been paid and for which specific algorithms that improve the detection process have been developed: lines, circles and ellipses.

The computational complexity of the algorithms used for the detection of circles and ellipses is going to increase in relation to the line detection algorithms, due to the large number of parameters involved in the description of these figures. In the case of the circle it is necessary to carry out votes in a three-dimensional space (center coordinates and radius) and in order to detect an ellipse, the polling is performed in five-dimensional space (center coordinates, orientation and semiaxis). The need for reducing the computational and memory requirements has led to the consideration of techniques different from those which make use of the specific geometric properties of the objects in order to improve the performance of the algorithms.

For circle detection, Qin-Zhong,[2] folds the image in each line perpendicular to the abscissas axis and performs an AND operation over pixels that coincide after folding. A threshold value is used for filtering the solution. Davies[3] uses an edge operator of size 2×2 in order to find the horizontal and vertical chord mid-points of the circles. Cao *et al.*[4] find the parameters of the circle for three points chosen from the image. Taking new point triads, a polling is carried out in a solution space. Nagata *et al.*[5] use recursive least squares estimations in order to detect ellipses.

A more efficient technique for the detection of circles and ellipses consists of decoupling or decomposing the process in order to obtain one of the parameters during the detection of the object using the information from the gradient vectors associated with the image points and which have already been calculated during the edge detection stage. This technique is used for detecting circles,[6–9] and is applied to ellipses.[10–16]

On the other hand, there are focusing algorithms which permit finding the maxima in the parameter space with a smaller computational and memory cost than the classical polling-search processes in the complete parameter space.[17–19] Among these focusing algorithms the FHT[17] stands out due to its regularity and ease of computation. However, it presents some drawbacks derived from the focusing strategy followed, which become more important when the image includes several figures of different sizes.

In this article we will present a new focusing algorithm that satisfactorily solves the problems encountered in the algorithms of reference (17). We will combine this technique with the decomposition of the parameter space in order to create two algorithms which detect circles and ellipses and which we will call Fast Circle Hough Transform (FCHT) and Fast Ellipse Hough Transform (FEHT), respectively. The aspects that are going to be improved with this combination are the following: less computation for the calculation of the centers of the figure, elimination of solutions as they are found, so that they do not interfere with the computations being carried out, and labeling of the points of the different figures, so

* Author to whom correspondence should be addressed. E-mail: ezapata@ac.uma.es.
† This work was supported by the EC under project BRPR-CT96-0170.

that subsequent stages only receive points implicated in the generation of each center.

We have organized the rest of the work as follows: In Section 2 we deduce a formula for the computational complexity of the focusing algorithm and we describe a new policy for the focusing algorithm showing how some of the problems of previous algorithms are solved. In Section 3 we present the FCHT algorithm and how to carry out the labeling of the points and the elimination of solutions in order to save computations. In Section 4 we introduce the FEHT algorithm. In this algorithm, in addition to using the improvements applied to the circle algorithm, we perform a new uncoupling of the parameter space. In Section 5 we evaluate the behavior of the different algorithms and we compare them with solutions found in the literature.

2. FOCUSING ALGORITHM

In this section we will present a new focusing algorithm derived from the FHT,[17] that employs a new focusing strategy thanks to which we will save computations. We will first present the advantages of this algorithm with respect to the most important ones found in the literature and we will deduce a new expression that bounds its computational complexity. We will later justify the use of a new focusing strategy.

2.1. *Focusing algorithms*

The computational requirements of the Hough Transform in its classical formulation are very important. Focusing algorithms for seeking the areas of the parameter space that store the maximum values in order to reduce the calculations have been employed. The justification for the use of these algorithms is given by the fact that the polling of the edge points of the different figures is going to concentrate in an area of the parameter space, so that it is not necessary to examine it all in order to obtain the solution.

Focusing algorithms perform a hierarchical approximation to the solution, starting from a coarse description of the parameter space. From this description, they examine the areas where solutions may be found in more detail. There are some algorithms which in each level of approximation to the solution calculate the complete information of the parameter space in that level, and using those values make a decision. This is the case of the AHT[18] or the MHT.[19] Others only calculate a part of the parameter space in that level and decide, using this information, whether it is appropriate to continue investigating that subspace or maybe search in some other.[17]

The main advantage of the FHT algorithm over the AHT and MHT algorithms is its regularity. In the FHT, by means of a focusing method, a hierarchical process will detect the solution, located in the intersection of the hypersurfaces. In this work, the algorithm is going to be applied to two-dimensional parameter spaces and, consequently, the solutions will be located in the intersection of a beam of lines. The focusing towards the solution is carried out by means of a process of division and insertion in the parameter space. In order to do this, a square parameter space is generated and divided into four quadrants. A given quadrant is divided again if an appreciable number of lines (larger than a threshold value) traverse it. The recursive application of this process will allow to find the solution.

In order to decide if a line traverses a given square, we calculate its distance to the center of the quadrant. If the distance is less than the radius of the circumference that circumscribes the quadrant, we consider that the line traverses it. In order to carry out this calculation we only need to know the center of the quadrant, its side and the coefficients that define the line. These coefficients, as we will later see, can be obtained from the coordinates of each point of the image and their associated gradient vectors. The expression which gives the distance of a line whose equation is $A_2y + A_1x + A_0 = 0$ to the center of a quadrant $c = (c_x, c_y)$ is

$$d_c = A_2c_y + A_1c_x + A_0, \tag{1}$$

where the values of the coefficients have been normalized in order to make $(A_2)^2 + (A_1)^2 = 1$, thus avoiding division operations in the calculation of the distances.

The formulation of the FHT algorithm permits that once the values of the distance to the center of the first quadrant have been obtained, their calculation for the subquadrants derived from it is reduced to simple addition and shift operations. On the other hand, and with the aim of reducing the amount of calculation, we define a presence vector which indicates which lines of the parameter space traverse a given quadrant (one presence vector bit for each line).

2.2. *Computational complexity*

The computations of the focusing algorithm are going to depend mainly on three factors, such as the number of edge points: the place where the solutions are located, the use of the presence vectors and the threshold value, T. In order to obtain a computational complexity expression we assume that there is only one solution in the parameter space, made up of a beam of P lines that are uniformly distributed in this space. In the worst case this solution will be located in the center of the image, and will consequently result in large number of computations in the different quadrants.

As the quadrants are subdivided, the number of lines contained in them decreases, except in the quadrant that contains the solution. To calculate the magnitude of this decrement for each quadrant, it is necessary to find the rates between the solid angle of the quadrant, Ω, and the solid angles of the quadrants derived from it, ω_i ($i \in \{0, 1, 2, 3\}$). In Fig. 1(a) we indicate what these solid angles are for the edge point A. The value of the rates depends on the position of the quadrant that is going to be split, but we have tested that in the majority of the cases, this value is around 0.5. So, we can generate a tree for the computation of the solution as shown in Fig. 1(b) taking into account that child subquadrants will have the

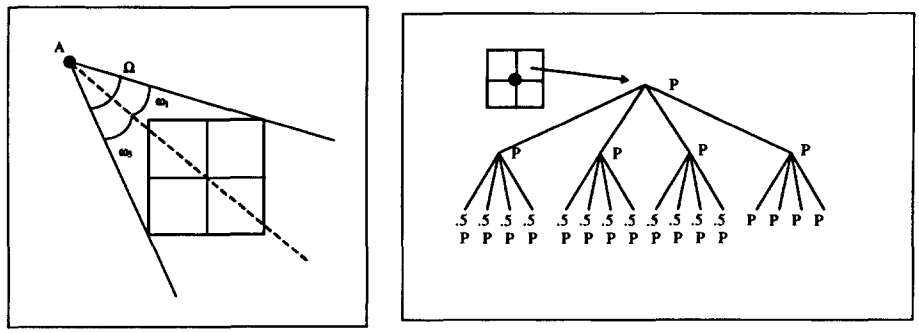

Fig. 1. (a) Solid angle for the parent quadrant (Ω) and the child subquadrants (ω_1 and ω_3). For simplicity, we only show the solid angles of subquadrants 1 and 3. (b) Several levels for the quadrant 1 of the image. The computation is indicated for each node. In the first and second level all the lines have to be computed. In the third level, child nodes only will compute the half of the lines (aprox.) except the children of the fourth subquadrant which will evaluate all the lines again.

middle of the lines rather than their parent quadrant. This way, for an image of size $N \times N$, the total number of computations is given by expression:

$$C = C_c + C_v, \tag{2}$$

where

$$C_c = P[16\log_2 N - 11], \quad C_v = P\left(24 \sum_{j=0}^{\log_2 N-3} \sum_{i=0}^{j} 2^j\right). \tag{3}$$

The first term, C_c, indicates the number of computations carried out for all the quadrants that contain the solution. The expression for C_c is deduced taking into account that the number of tree levels is $\log_2 N+1$ and the number of computations for the first, second and the rest of the levels are P, $4P$ and $16P$, respectively.

The expression for C_v is calculated from the quadrants which contain fewer than P lines (see these nodes which produce $0.5P$ computations in the third level of the tree). This result has been obtained under the assumption that the non-containing solution quadrants generate half of computations of their parents.

In order to obtain this expression we have not taken into account the limitation in the division of the quadrants introduced by the threshold T. If we choose $T=P/2^t$, the expression for C_c remains constant though C_v modifies its value according to

$$C_v = P \sum_{j=0}^{\log_2 N-3} \sum_{i=0}^{\min(j,t-1)} 2^i. \tag{4}$$

So the final expression for the computational complexity is

$$C = P[16\log_2 N - 11] + [1 - \delta(t)]24[2^t(\log_2 N - t) - \log_2 N], \tag{5}$$

where $\delta(t)$ is 1 when $t=0$ and 1 for any other value of t.

By choosing an adequate value for T, the number of computations can be significantly improved. A lower bound for this number will occur when $t=0$ making $C = P(16\log_2 N - 11)$. However, in the case where we have several figures in the image, this minimum computation will not be achievable. Thus, if for instance we have four figures, which generate $P/4$ lines each, it will be necessary to use a threshold value with $t=2$ in order to be able to detect the solutions, which, according to equation (4), will generate more computations.

In order to test the correctness of the above expression for the computational complexity, we have calculated the number of computations generated from a beam of P lines which cross the center of the image. These real values have been represented in Fig. 2. On the other hand, we have also indicated the computations using the theoretical expression (5). From Fig. 2, it follows that equation (5) is an upper limit for the computational complexity of the focusing algorithm.

In general, and following equation (5), the computational complexity of the focusing algorithm is going to be $O(P(\log_2 NP/T))$. For large values of P/T, the complexity will be given by $O(P^2/T)$. On the other hand, the classical polling process has a complexity of $O(PN)$ and the memory search $O(N^2)$. Therefore, we can guarantee that the focusing algorithm is computationally faster than the classical one whenever $P/T < N$. It is important to note that in order to reduce the computational complexity expressions, we have assumed that all the nodes with a value higher than the threshold must be examined. Generally, this will not be necessary as the solution may be found before having to evaluate all the nodes. In addition, as we will see in the next sections, the lines are eliminated

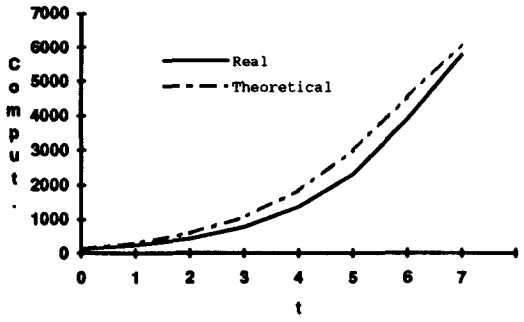

Fig. 2. Real and theoretical computaton for $N=512$. The ordinate axis indicates the number of distance calculations in P units.

as the solutions are found, consequently, the number of computations improves for this reason. equation (4) is thus an upper bound to the computational complexity of the focusing algorithms.

2.3. Focusing strategy

Li *et al.*[17] propose several strategies for deciding how to divide the quadrants, and thus, how to investigate the quaternary tree thus formed. However, we have found that for the case in which different figures are present in the images producing several solutions, which consequently superimpose their votes, some problems arise. The *Breath-First* strategy, tends to open the tree, unfolding all the nodes of each level. This will require maintaining many active nodes and thus the use of large amounts of memory. In addition, as a criterion for ordering the nodes that must be divided does not exist, it is possible for the focusing towards the solution to be slow. The *Depth-First* strategy will tend towards opening new levels. However, no criterion is used in order to choose, among all the possible candidates, the one that will most probably lead to the solution. One possible improvement we have evaluated is to choose out of the four candidates, the one containing the largest number of lines. However, this may also be too constraining when we have several solutions in the parameter space.

One example of the constraint induced by this last criterion can be seen in Fig. 3. We assume that each one of the two solutions is generated by a line beam, formed by P_1 and P_2 lines, respectively, and the lines are uniformly distributed with respect to their cross points. We present the first two levels of the tree generated by each one of the solutions by itself, indicating in each node the number of lines going through them, as a fraction of the total number of lines. If we use a Depth First strategy with ordering, the focusing continues in all the levels of the correct path. However, superposing the two trees in order to allow the interaction among both solutions and taking the same values for P_1 and P_2, we see that the focusing momentarily takes a wrong path (left branch). This will cause a larger number of computations in the algorithm.

An added problem arises in the case where the figures of the image generate solutions with very different votes, whether as a consequence of its smaller size or that part of them are hidden. In this case, to choose a threshold value presents a problem; because if it is too small, many computations will be performed and if it is too large the solutions with less votes will not be detected. In reference (20) the problem is solved by executing the algorithm several times, starting with a large threshold value which is decreased in successive executions. The solutions extracted in one of the stages are eliminated so that they cannot interfere with those of the next stage.

With the aim of solving the problems we have described, we have developed a new focusing algorithm in which the threshold value remains constant with a small value (less than the solution with the least number of votes) and which uses a *Best-Breath* and *Depth* strategy. Starting from the first level of the tree, all the nodes are calculated, but we are only going to study those whose value is larger than or equal to the maximum value multiplied by a given factor, we will call the discrimination factor. This will allow us to simultaneously expand more nodes so that in case we found a problem such as the one shown in Fig. 3, it could be later solved in lower levels. Thus, applying our algorithm to Fig. 3, we make the nodes that really contain the two solutions continue progressing.

In order to avoid studying many nodes of a level, in the case where all of them surpass the criterion given by the factor, we will limit the maximum number of nodes that can be simultaneously explored (active nodes). In this case, we will simultaneously examine the ones with the largest numbers of votes. This will also permit limiting the memory needs of the algorithm. This way, and taking into account that in the case of having P lines, we need P storage positions per level, the memory needs for the algorithm will be of the order of $O(PK \log_2 N)$, where K is the number of active nodes per level.

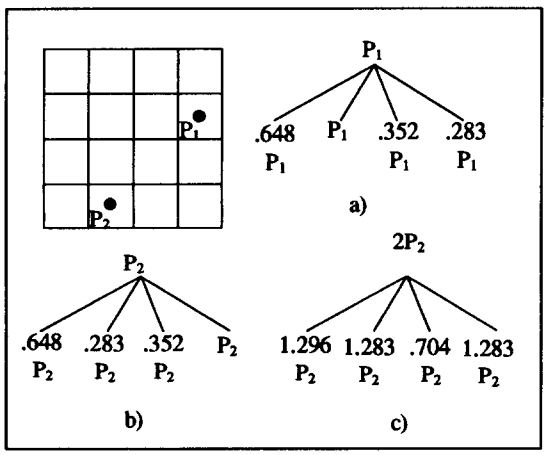

Fig. 3. Trees generated for the different solutions: (a) only P_1, (b) only P_2 and (c) P_1 and P_2 combined (we assume P_1=P_2).

3. CIRCLE DETECTION

The geometrical place occupied by the points, $P = (x_p, y_p)$, that make up the circumference of a circle is given by:

$$(x_p - x_0)^2 + (y_p - y_0)^2 = r^2, \tag{6}$$

where x_0 and y_0 are the coordinates of the center and r is the radius of the circle. Applying the traditional HT we need a three-dimensional parameter space in order to accumulate the votations. This method is costly, both computationally and with respect to space requirements. For this reason other techniques which improve both aspects have been used.

Davies[21] and Chan *et al.*[6] find the parameters of the circle employing two stages. In the first one they calculate the center and in the second the radius. In another more recent work by Chan and Siu,[7] the votes for the

center and the radius are carried out in parallel using a double decomposition of the parameters in order to find the pairs $x_0 - y_0$ and $x_0 - r$ simultaneously.

In these algorithms, the detection of the parameters of the circle is made more difficult by the following problems: generation of erroneous solutions and difficulty of detection of small circles. The first problem arises as a consequence of the fact that the intersection of lines produced by points from different circles may surpass the threshold value we are considering and, consequently, generate an erroneous solution. The second problem is due to the difficulty in the detection of circles of small sizes. This is caused by the effect of the masking in the polling produced by the circles of larger circumference. In fact, the votes of these larger circles appear dispersed around the solution and some of them can hide the center of a smaller circle.

In addition to solving these problems, the algorithm for the detection of circles we propose in this work, Fast Circle Hough Transform (FCHT), performs the detection of the circle in two stages, improving the computational and storage aspects. In the first stage the center of the circle is detected and the points are labeled using the FHT algorithm. In the second stage the radius is obtained with the help of this labeling. We will now indicate how the different stages work.

3.1. Detection of the center

In order to find the center of the circle it is necessary to previously detect the gradient vectors associated with the points located on its circumference. As can be observed in Fig. 4, the intersection point of the line beam obtained when prolonging these vectors will be the center we seek. The equation of one of these lines containing a point of the image $P = (x_p, y_p)$ is

$$y = \frac{\nabla_y^p}{\nabla_x^p}(x - x_p) + y_p, \tag{7}$$

where ∇_x^p and ∇_y^p are the components of the gradient vector associated with each point of the image. The normalized coefficients for each point P corresponding to equation (1) are

$$A_0^p = \frac{y_p - \Psi_p \cdot x_p}{\sqrt{1+\Psi_p^2}}, \quad A_1^p = \frac{\Psi_p}{\sqrt{1+\Psi_p^2}}, \quad A_2^p = -\frac{1}{\sqrt{1+\Psi_p^2}}, \tag{8}$$

where $\Psi_p = \nabla_y^p / \nabla_x^p$. Applying the FHT algorithm to the set of lines we will be able to determine where these lines intersect by more than T (threshold value). These places correspond to the centers of the circles of the image.

In the case where there are multiple circles in the image, we find the problem of generation of erroneous solutions as a consequence of the fact that points belonging to different circles may collaborate in the apparition of an untrue solution. In order to avoid, as much as possible, the generation of erroneous solutions, we will use a technique that is analogous to the one employed in the case of lines.[20] That is, we will consider information from the presence vector and we will eliminate the lines that have generated a center.

3.2. Detection of the radius

Once the center of a circle is detected we can calculate its radius using the values for the distance between the center of the circle and each one of the points of the image. The distance values vote over a one-dimensional space. The position of the accumulator that at the end of the process has the largest number of votes will indicate the value of the radius. However, in this process all the points of the image are involved, and they can be many in the case of images with multiple circles. The ideal situation is only to involve those points that have helped in the production of the center. As shown below, this problem is solved by taking the presence vector into account.

In an image with P points, the presence vector generated during the production of a center c of one of the circles, $V_c = \{v, \ldots, v\}$, contains the information of the image points that have collaborated towards obtaining this solution. Thus point i of the image will be used in the radius detection stage if its bit v is set to 1. These points, except some generated by noise or by interference with another solution, are the ones that make up the circle and so the polling process will be restricted to them. Using this information, we will calculate, for each point separates it from the center of the circle. With this value it votes in the one-dimensional space so that at the end of the process there will be a peak that identifies the solution. In the case of concentric circles, several peaks will appear, corresponding to the different radii. We will now detail the working of the algorithm.

```
void FCHT( ) {
  FORALL edge_points j
    Calculate A_0^j, A_1^j, A_2^j;

  WHILE (find_center_with_FHT) {
    FORALL v=1 {
      Calculate d = ||i - c||;
      Radius_acc[d]++;

    }
    Find_Max(Radius_acc);
  }
}
```

The coefficients of the lines associated with each point j, according to equation (7), are stored in A_0^j, A_1^j and A_2^j. Variable c contains the value of each center detected. The value of the presence vector associated with each point i of the image when a center c is found is v and d is the distance between the calculated center and each one of the points. The find_center_with_FHT function seeks the next center according to the algorithm we propose and returns its coordinates. It returns a value of FALSE if there are no more centers in the image. Radius_acc is the accumulator space where the radius is voted.

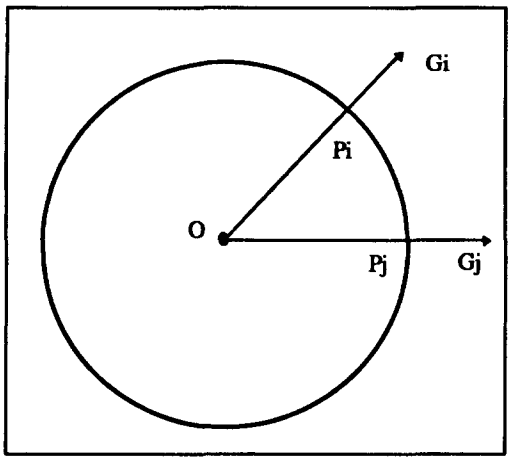

Fig. 4. Generation of the line beam from the gradient vectors.

In Fig. 5 we show the way the algorithm works. Figure 5(a), of 512 × 512 pixels, is the original figure consisting of four circles of different sizes, two of them are concentric and the other two partially overlapping. An edge detector has been applied to the original image, in our case a Sobel detector, together with a Laplacian operator[9] in order to reduce the number of edge points. The Sobel operator has a good behavior in the circular edge detection [as it has been indicated by Davies[4]] producing a rms error of 0.73 in the angle calculation. In Fig. 5(b) we present the result of applying these operators. The result of the detection is shown in Fig. 5(c).

The discrimination factor used has been 0.8, so that all the nodes (quadrants) calculated in a level that had 80% of the votes of the node with the largest votation are divided. For the memory needs not to grow out of control, we have limited the number of active nodes (in memory) per level. In the graph of Fig. 6(a) we can observe the

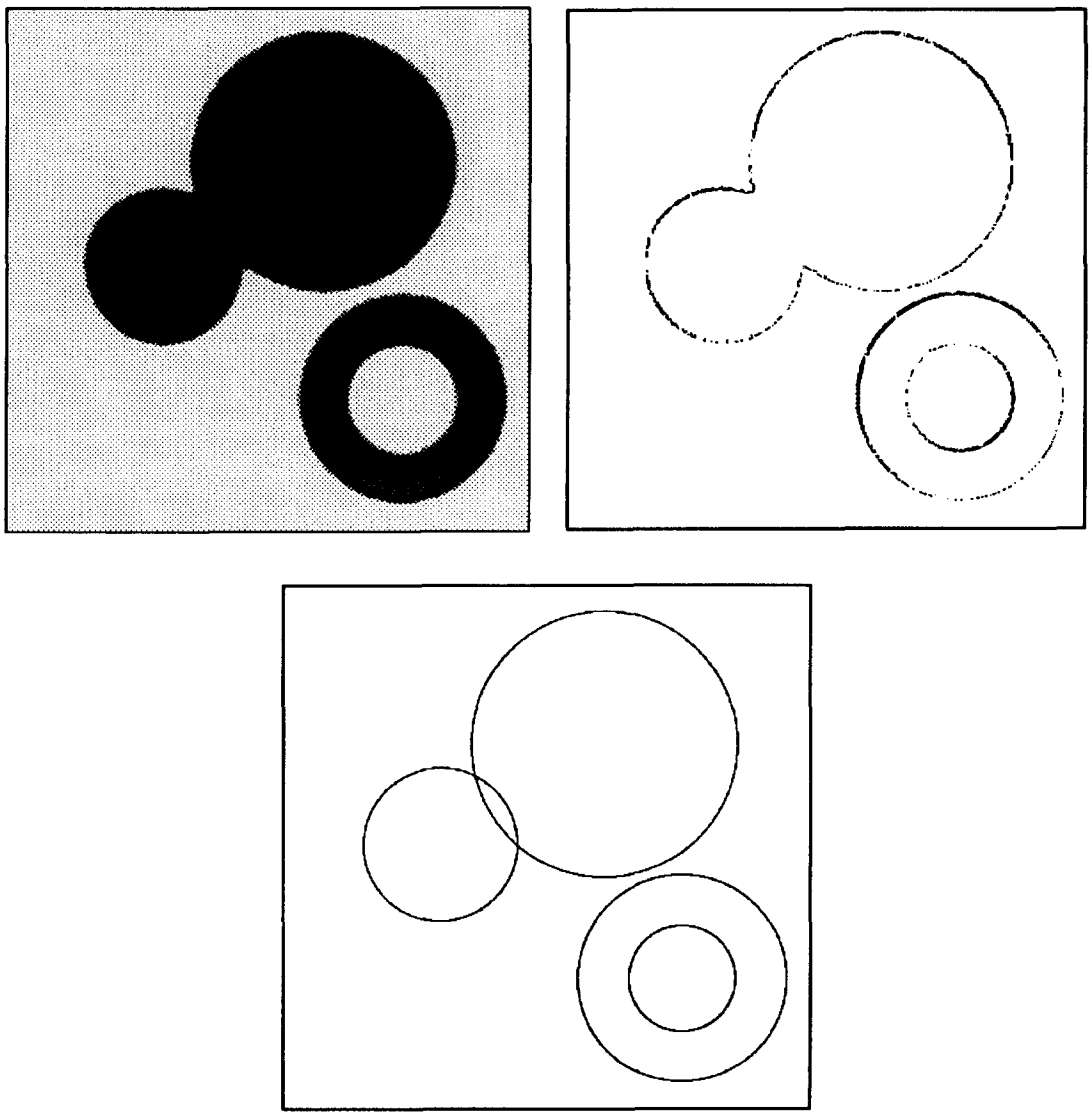

Fig. 5. (a) Image with four circles. (b) Edge of the image after applying the Sobel operator. (c) Result of the detection.

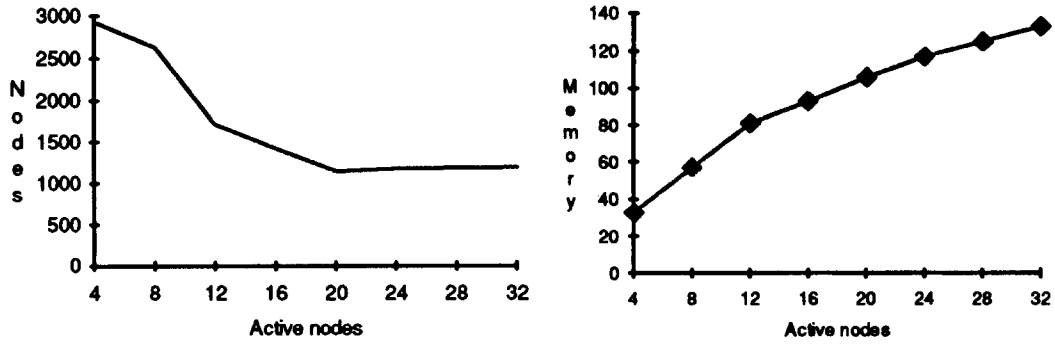

Fig. 6. (a) Total number of nodes computed as a function of the maximum number of active nodes per level. (b) Memory employed, in nodes, as a function of the maximum number of active nodes per level.

Table 1. Computational times and speedup for Fig. 5(a)

Active nodes	Time (s)	Speedup %
4	2.3	0
8	2.0	13
12	1.8	21
16	1.5	34

Table 2. Computational times and speedup for the Fig. 7(a)

Active nodes	Time (s)	Speedup %
4	33.8	0
8	31.5	7
12	26.5	21
16	22.1	33

number of nodes examined until the solutions are found as a function of the maximum number of active nodes per level. It can be observed that as this limit grows, the computations are reduced until they reach a level in which they are basically stable or even worsen slightly. This is due to the fact that once the optimal value for the number of active nodes has been reached, the opening of additional nodes will only cause the calculation of new nodes but will not accelerate the search for the solution. In Fig. 6(b) we indicate the memory needs, in number of nodes, as a function of the active nodes per level. It follows that the required memory for the algorithm increases as the maximum number of active nodes per level is higher because more nodes can be simultaneously expanded in each level.

In Table 1, we show the computational time of the algorithm in a SUN workstation with a superSPARC processor. In the case of four active nodes per level, we are actually using a depth-first policy like the classical FHT algorithm. As the allowed number of active nodes per level is increased, the computational time improves. The best speedup is obtained for 16 active nodes per level.

The image in Fig. 7(a) have been used to test the FCHT algorithm behavior in a noisy environment. Figure 7(b) shows the result of applying a Sobel's operator to the original image. We can see that a high number of noise points appear and, futhermore, several edge points of the figure have disappeared. Under these conditions, the detected circles are shown in Fig. 7(c). As it can be observed, all the circles, except the little concentric one in the uppermost figure have been detected. Other circles, next to the correct ones, have been extracted from the image as a consequence of the erroneous center generation due to the noise points. In this situation, the radius detection stage can be used to discriminate these erroneous circles using several rules. Thus, when the radius is calculated, the edge points which distance from the center coincides with the calculated radius are identified. Next, the distribution around the center is obtained. If the circle arc (or circle arc pieces) that form the points is large enough, the circle is extracted. This technique avoids to extract, like the correct circle, circlewise point clusters as indicated in Fig. 7(c).

The execution times for circle detection are shown in Table 2 as a function of the maximum number of active nodes per level. When this number is increased, the execution time improves. However, the improvement is lesser than the previous example due to the computation required for the noise points.

4. ELLIPSE DETECTION

The points that make up an ellipse satisfy the following equation:

$$\frac{[(x_p - x_0)\cos\phi + (y_p - y_0)\sin\phi]^2}{a^2} + \frac{[(x_p - x_0)\sin\phi - (y_p - y_0)\cos\phi]^2}{b^2} = 1, \quad (9)$$

where x_0 and y_0 are the coordinates of the center, ϕ is the orientation angle and a and b are the semiaxis. Using the traditional formulation of the FHT it is necessary to create a five-dimensional parameter space in order to perform the polling. This method requires large amounts of storage and computation. However, using information from the gradient vector it is possible to decompose the calculation of the parameters into several stages. In the literature we can find different ways of performing this decomposition.

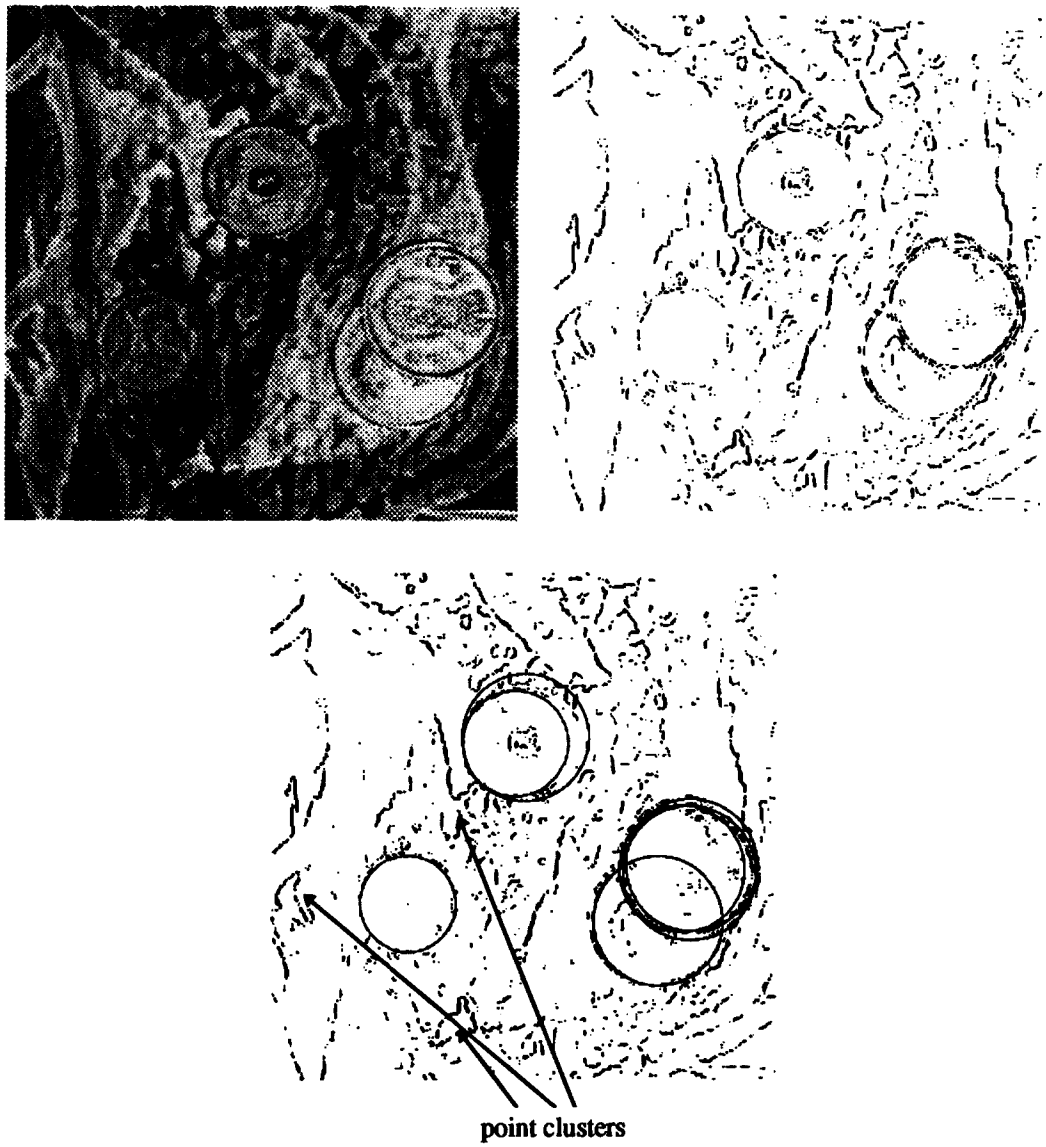

Fig. 7. (a) Original image. (b) Edge points after applying a Sobel's operator. (c) Result of the detection. The detected circles have been superimposed on the edge points in order to show the fitting.

Tsuji and Matsumoto[10] first detect the center of the possible ellipses and discard candidates using geometrical properties. After this and in order to obtain the five parameters of the ellipse, they apply the least squares method to the points that make up each of the candidates. Tsukune and Goto[11] perform the detection in three stages using information from the gradient vectors of the points of the image edge. Ho *et al.*[16] use two stages. In the first one, two symmetrical axes (vertical and horizontal) are extracted finding the midpoint of edge points scanned downwards and rightwards. The cross-point of these two axes will indicate the center of the figure. The rest of the ellipse parameters are calculated employing a standard voting process on an accumulator space. All these works present problems in the case of partially hidden ellipses and, therefore, we will not consider them in Section 5.2.

Yuen *et al.*[12] solve the problem of detecting partially hidden ellipses by means of an algorithm that works in two stages. In the first stage they detect the centers and in the second they apply the Adaptive Hough Transform[18] in order to find the three remaining parameters. A higher uncoupling degree is achieved by Muammar and Nixon[14] who use three stages. Finally, Yoo and Sethi[15] find the possible parameters for each pair of points and after this, by means of a three stage process they extract the solutions.

The algorithm for ellipse detection, Fast Ellipse Hough Transform (FEHT), developed in this work uses three stages: (a) detection of the center by means of a focusing algorithm that finds the point where a beam of lines intersect; (b) production of the angle ϕ and of the ratio a^2/b^2 using information from previous stages and from the gradient vectors of the figures;

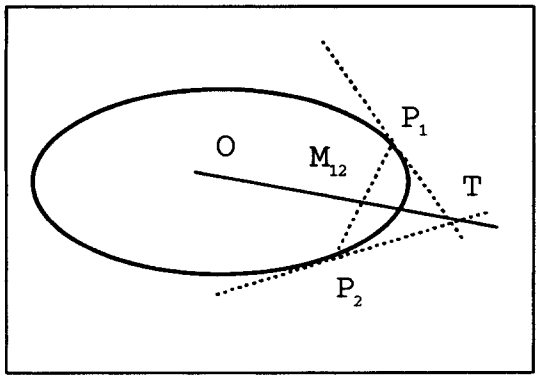

Fig. 8. Generation of the line beam for the ellipse.

(c) calculation of the semiaxis a and b from the equation of the ellipse.

4.1. Detection of the center

Let $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ be two points of an ellipse whose slopes, of value ξ_1 and ξ_2, respectively, intersect in point T_{12} as shown in Fig. 8. The line that connects this point T_{12} with the center point $M = [(x_1 + x_2)/2, (y_1 + y_2)/2]$ goes through the center of the ellipse.[10] Thus, pairing points of the ellipse with unparallel slopes we create a beam of lines that intersect in the center of that ellipse. Applying the FHT algorithm to the line beam we will find the center of the ellipse. In this stage, each line of the beam is defined by the normalized coefficients corresponding to equation (1):

$$A_0^{P_1P_2} = \frac{t_1 m_2 - t_2 m_1}{\Gamma_{12}}, \quad A_1^{P_1P_2} = \frac{t_2 - m_2}{\Gamma_{12}}, \quad A_2^{P_1P_2} = \frac{t_1 - m_1}{\Gamma_{12}}, \tag{10}$$

where

$$t_1 = \frac{y_1 - y_2 - x_1\xi_1 + x_2\xi_2}{\xi_2 - \xi_1}, \quad t_2 = \frac{\xi_2\xi_1(x_2 - x_1) - y_2\xi_1 + y_1\xi_2}{\xi_2 - \xi_1}, \quad m_1 = \frac{x_1 + x_2}{2}, \quad m_2 = \frac{y_1 + y_2}{2}, \quad \Gamma_{12} = \sqrt{(t_2 - m_2)^2 + (t_1 - m_1)^2}. \tag{11}$$

The slopes of the points in the edge of the image are obtained from their gradient vectors. This way, if θ is the angle of the gradient vector for point P, the angle of the tangent to that point will be $\xi_p = \theta_p - \pi/2$.

Because of the way in which the lines are generated, their number will be of the order of the square of the number of points in the image. The large number of lines that can be considered leads us to establishing certain constraints on the possible pairings of the points, both for the case of a single ellipse in the image and for the case of multiple ellipses. If the image is made up of a single ellipse, the problems will be caused by the points with almost parallel slopes. In this pairings, small deviations in the calculation of the gradient may cause large errors in the calculation of the intersection point of the tangents. The problem can be eliminated if one of the conditions for associating two points P_i and P_j is that $|\theta_i - \theta_j| > \alpha_t$ where α_t is a threshold angle.

In the case of multiple ellipses, the pairings of points belonging to different ellipses will lead to the creation of erroneous lines. In order to reduce the number of erroneous pairings we will add three new constraints

1. The distance between points that are going to be paired must be less than a threshold value d_t, this is, $|\mathbf{OP}_i - \mathbf{OP}_j| < d_t$. The value of d_t must be adequately chosen. Values that are too high may cause undesired pairings and values that are too small can restrict the number of correct pairings within an ellipse. A good criterion could be based on *a priori* acknowledge of the ellipse sizes. In our experiments we have allowed the pairing of points separated less than 1/8 of the image size. This criterion works well if the ellipse sizes are unknown or ellipses with very different sizes appear in the image.
2. The vector products of the gradient vectors of each point times their corresponding tangent vectors must have opposite signs in the pair of points we want to associate. This can be indicated as $\text{sign}(\nabla_{Pi} \times t_i) \neq \text{sign}(\nabla_{Pj} \times t_j)$, where ∇_{Pi} is the gradient vector associated with point i, and $\mathbf{t}_i$ is the tangent vector obtained from normalizing vector $\mathbf{P}_i\mathbf{T}_{ij}$. This property is verified for all the points belonging to an ellipse, as shown in Fig. 9 for the pairing of P_1 and P_2. However, it prevents pairing some of the points that belong to different ellipses, as can be appreciated in the same figure for P_3 and P_4.
3. So as to avoid the pairing of points that belong to different concentric ellipses, we will examine the convexity and concavity of these points. As can be observed in Fig. 10, the points of the external ellipse are concave to the illuminated area, whereas those in the internal one are convex. By avoiding the combination of these points, we will reduce the number of erroneous pairings.

The last two restriction rules work appropriately when the background is uniformly lower or higher than the

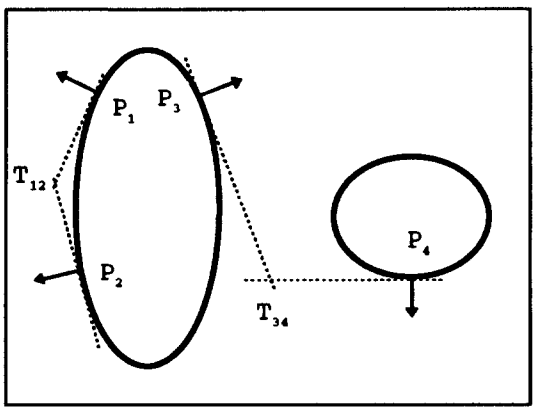

Fig. 9. The pairing between P_1 and P_2 is allowed whereas the one between P_3 and P_4 is not.

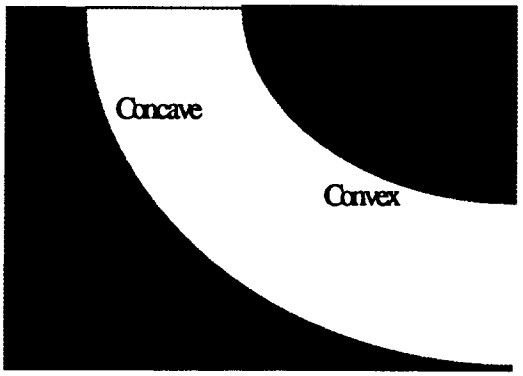

Fig. 10. Concavity and convexity of the points with respect to the shadowed area.

object. In other cases some wrong pairings can appear, affecting to the correct line generation and complicating the center detection process.

There are other methods based on windows that try to reduce the redundant information generated during the parametric transformation. So, the DGHT,[22] using geometric properties of the figures (circles and ellipses), makes a prior segmentation of them. After this process, the image is divided into several windows and only the feature points placed into the same window are going to be combined to calculate the parameters of the figures. The size of the windows is variable in order to adapt it to the different figures in the image. In the MWPT,[23] the size of the window is fixed but it is possible to combine the feature points from different windows to obtain the solutions. In the algorithm presented here, the above-mentioned constraints guarantee accurate center detection.

In order to create the pairs, during the edge detection stage we make a table (point table) where all image edge points are present. For each point P_i of the table, starting from the beginning, we seek possible associations with points P_j that occupy lower positions in the table taking into account the constraints we listed before. With all the pairs we create a new table (pairs table) where we store the value of the coefficients of the line generated and two indices i and j for each pair of points. The indices locate the value of P_i and P_j in the point table. This information is used in the next stage.

4.2. Detection of orientation

Using the partial derivatives of the equation of an unrotated ellipse, the points P of the contour satisfy expression:

$$\frac{a^2}{b^2} = \frac{(x_p - x_0)}{(y_p - y_0)} \tan\theta_p. \tag{12}$$

In the case where the ellipse is rotated an angle ϕ this expression becomes:

$$\frac{a^2}{b^2} = \frac{(x_p - x_0)\cos\phi + (y_p - y_0)\sin\phi}{(x_p - x_0)\sin\phi - (y_p - y_0)\cos\phi} \tan(\theta_p - \phi). \tag{13}$$

In this equation we know the values of the coordinates of the center of the ellipse from the application of the previous stage. Thus, we can use a two-dimensional space where each point of the ellipse votes for different orientation angles. The maxima obtained after performing the process for all the points will indicate the possible orientation values and the semiaxis ratios of ellipses in the image.

The polling process is not carried out for all the points of the image. It is restricted only to those points that have participated in finding the center. In other algorithms[14,15] finding the points that have participated in solutions requires backmapping processes and hence CPU time. An alternative solution consists in labeling the image points as the solutions for the center coordinates are obtained. This way, the points that have participated in obtaining each one of the solutions are marked in a different way. If we take into account that in the FEHT algorithm once a center has been found we can go on to the stage of obtaining the remaining parameters without detecting the remaining centers, we can see that one bit suffices for labeling each point. The presence vector allows for an easy implementation of this task, carrying out the labeling during the process of obtaining each center. This way, if the presence vector of a line is active, it indicates that this line goes through the center, or, in other words, that the points that have generated the line belong to the ellipse (except for erroneous pairings).

As we have already seen, each item in the coefficient table is the result of an association of points of the image. Once the center search process has ended, the pairings that have produced lines that go through the center will have an active presence bit. The points to which equation (12) is applied will be found using index i of the items from the coefficient table with an active presence bit in the point table. Due to the fact that several contiguous equal indices may appear as a consequence of different pairs containing the same point, we will use a variable for carrying out the polling with each point only once.

If there are concentric ellipses with different orientation angles in the image, several maxima corresponding to different values of orientations and semiaxis ratios will appear in the voting space.

4.3. Detection of the semiaxis

For all the points of the image that have collaborated in a maximum of the previous stage, we perform a rotation of value ϕ and substitute the value of the semiaxis ratio, we will call it h, in the equation of the ellipse so that the values of the semiaxis are given by:

$$b = \sqrt{h^{-1}(x_p - x_0)^2 - (y_p - y_0)^2},$$
$$a = \sqrt{(x_p - x_0)^2 + h(y_p - y_0)^2}. \tag{14}$$

It is necessary to point out that as a consequence of having obtained the semiaxis ratio in the previous stage we can reduce the complexity of the current one. This way, each point can carry out the voting process as

opposed to other papers,[14] where they need to pair the points again in order to vote the semiaxis.

After the polling process, the maxima found in the space will coincide with the values of the semiaxis of the detected ellipse. In the case where there are concentric ellipses in the image, there will be one maximum per ellipse. We will now present the algorithm for the calculation of the parameters of the ellipse:

```
void FEHT( ) {
  FORALL edge_points j
    FOR edge_points FROM i = j + 1 to P.
      IF (restrictions(i,j)==TRUE)
        Calculate A_0^ji, A_1^ji, A_2^ji;

  WHILE (find_center_with_FHT) {
    FORALL v^i==1
      IF (firstime(j)==TRUE)
        Vote H_φh(j);

    find_max H_φh;
    FORALL v^i==1
      IF (firstime(j)==TRUE) {
        Rotate (j,φ)
        Vote H_ab(j);
      }
    find_max H_ab;
  }
}
```

In the algorithm, P is the number of points of the image edge, restriction(i,j) is a function that returns a true value if the pair of points i and j verify the constraints. First_time(j) returns zero if it is the first occurrence of point j and rotate(j,ϕ) rotates point j an angle. $H_{\phi h}$ and H_{ab} are accumulator spaces where the votations are stored for the corresponding parameters.

We will now show the behavior of the algorithm over a 512×512 pixel image consisting of four ellipses in Fig. 11(a). In Fig. 11(b) we see the edge of the figures after applying the Sobel operator and a pseudorandom function for the elimination of points from this edge. The percentage of eliminated points was 50%. In Fig. 11(c) we see the result of the detection. As in the example for the circles, the discrimination factor was 0.8. In Fig. 12 we present the number of nodes computed as a function of the maximum number of nodes allowed per level. In this case, it can be observed that the minimum value for the computations becomes minimum when the number of active nodes per level is limited to 8. In the Table 3 we show the computational time in a SUN workstation.

Again we have used a noisy image in order to test the FEHT algorithm robustness. The original image is shown in Fig. 13(a). In Fig. 13(b) the edge points have been extracted applying a Sobel's and Laplacians's operator.

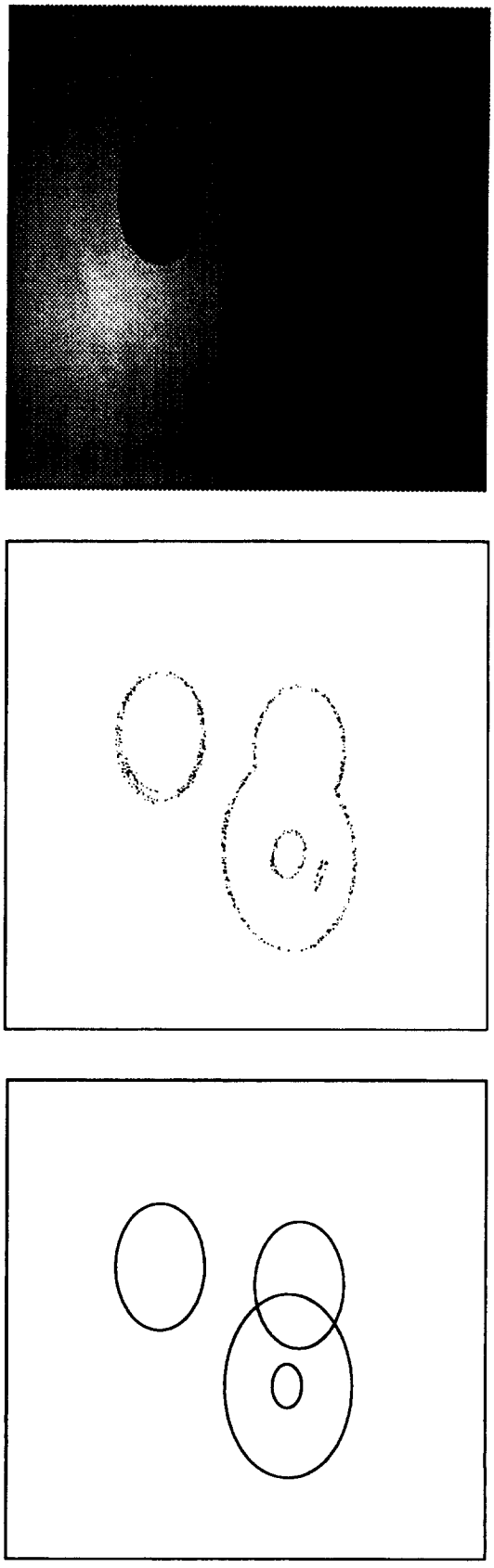

Fig. 11. (a) Image with ellipses. (b) Edge of the image after applying the Sobel operator and eliminating 50% of the edge points. (c) Results of the detection of ellipses.

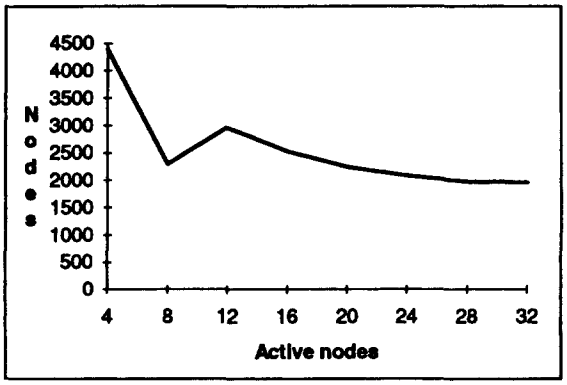

Fig. 12. Number of nodes computed as a function of the maximum number of nodes per level.

Table 3. Execution times and speedup for Fig. 11(a)

Active nodes	Time (s)	Speedup %
4	100	0
8	71	29
12	79	21
16	75	25

Table 4. Execution times and speedup for the ellipse detection of Fig. 12(a)

Active nodes	Time (s)	Speedup %
4	148	0
8	137	7
12	119	19
16	111	25

The threshold for these operators has been maintained with a high value in order to filter as many noise points as possible. Due to this, many ellipse edge points have been lost. The result of the detection process is shown in Fig. 13(c). We can see that all the ellipses have been detected, even the concentric ones, but several additional erroneous ellipses also appear due to the low threshold, during the center detection, that we have applied so as to detect all the possible solutions. In any case, these erroneous ellipses can be eliminated using a similar method to the circle algorithm. In the Table 4 the computational times are shown. We can see that as the number of active nodes goes up, the computational time goes down. The best execution time is achieved for 16 active nodes per level.

5. EVALUATION

We will next establish the computational complexity and the memory requirements of the FCHT and FEHT algorithms, as well as their comparison to other relevant algorithms.

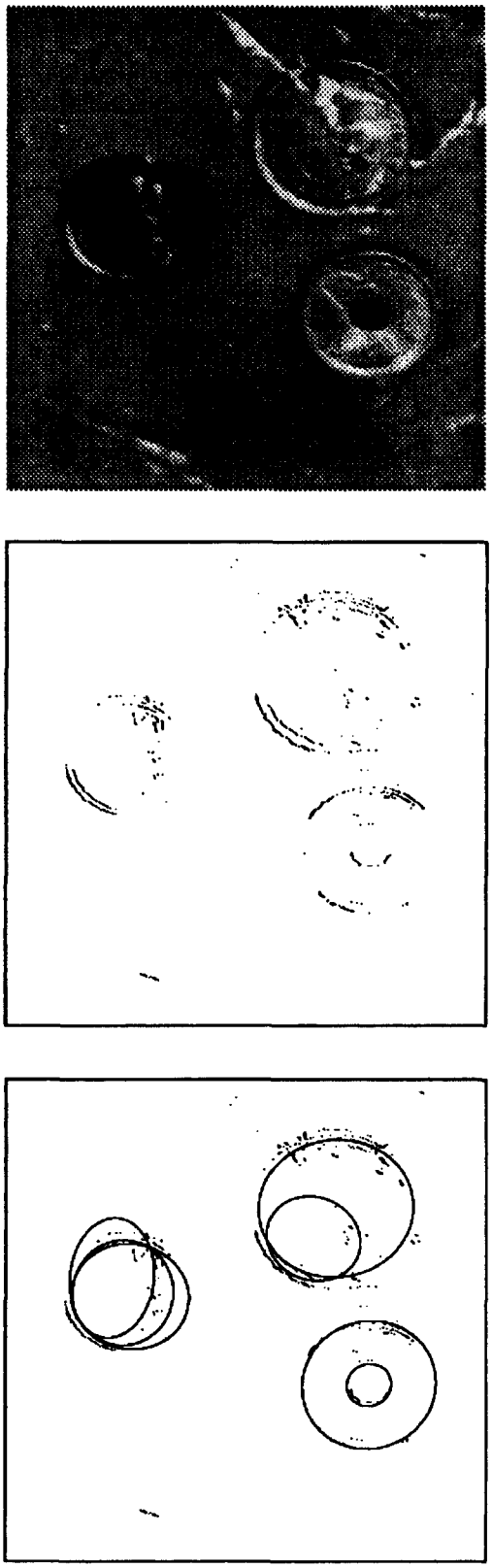

Fig. 13. (a) Original image. (b) Edge points after applying Sobel's operator. (c) Result of the ellipse detection.

Table 5. Complexity of some Hough algorithms to detect circles

Algorithm	Computation		Memory
Conker[9]	$P \cdot W^{2*}$		N^2
Davies[21]	Center	Radius	N^2
	$N \cdot P$	$f \cdot P$	N^2
Chan y Siu[7]	y–x Pairs	x–r Pairs	N^2
	$N \cdot P$	$N \cdot P$	
FCHT	Center	Radius	$P \log_2 N$
	P_2/T	P	$P \log_2 N$

* $W \times W$ is the window size used to pairing points.

5.1. Circles

Conker[9] solves the circle equation for pairs of points from the edge of the image using the gradient vector and votes in two parameter spaces, each one of size $N \times N$. One of the spaces will store the votes for the centers and the other the votes for the radii. In order to reduce the number of pairings, the equations are only applied to one of the possible pairs contained in a window of $W \times W$ size. The value of W may be 2 if the edges have completely extracted, but if the image has noise it is necessary to take larger window sizes. In Table 5 we display the computational complexity of these processes and the memory required.

Davies[21] uses a stage for calculating the center voting in an $N \times N$ space and another stage for calculating the radii voting in a one-dimensional space. In the first stage the votes occur along the direction of the gradient vector associated to each point. This way, each point of the image will produce N votations (each votation needs one multiplication and one addition). However, as there is no point labelling in the algorithm, the second stage must be applied to all the points of the image. In the case where there are f centers in the image (originated by f circles), $f \times P$ possible values of the radius are computed. The number of operations required for the computation of the algorithm is specified in Table 5.

Chan and Siu[7] use two parameter spaces associated with the pairs y–x and x–r so that the polling in both can be carried out in parallel. In each one of the spaces the polling is carried out along a line whose direction is calculated from the gradient vector associated with each point. In a later case, we relate the solutions found in both spaces. In Table 5 we summarize the cost of the algorithm. On the other hand, the algorithms we have pointed out must carry out a search of maxima in the parameter space once the solution has been found, and even when the polling is too sparse a convolution is previously carried out. The computational cost of these tasks has not been included as it depends on the location of the solutions in the parameter space, but in the worst case it will be proportional to the size of the parameter space.

In the FCHT algorithm, during the detection of the center, a line is generated for each image edge point. Assuming there are P points in the edge, the computational complexity and the memory needed will be of the order of $P \times \log_2 N$. Once the centers have been detected, the radii must be calculated. For this we will label the points in order to require less computations in this stage. The labeling is specially effective in the case of multiple circles in the image. If we consider an image I with f figures, image I consists of the union of all the points belonging to all the figures $I = F_1 \cap F_2 \cdots F_f$, where F_i are the points of the image which make up figure i (including those generated by noise). The computational complexity in the calculation of the radius for every circle will be given by the addition of the calculations of the distances between the points of the image whose presence bit is one and the center of the circle. This way, in our case, this amount will be proportional to $P_1 + P_2 + \cdots + P_f$, where P_i is the number of points that make up figure i. The storage space requirements for the implementation of this stage will be given by the resolution we want to use for the determination of the radius. If this resolution coincides with that of the image, the space required is N.

If we compare the computation of the FCHT algorithm with that of references (21,7), we will see that the complexity of our algorithm is less if $P/T < N$. The algorithm[9] does not present dependence with respect to N, but carries out complex operations for each votation, and for this reason also performs worse than the algorithm we present. Regarding memory requirements, we also see that the FCHT algorithm depends on N logarithmically, whereas the others present a quadratic dependence.

5.2. Ellipses

The algorithms in the references we are going to use for the comparison also permit finding partially hidden ellipses. Yuen *et al.*[12] calculate the parameters of the ellipse in two stages. In the first they vote in an accumulator space of size $N \times N$. Once the maxima have been found, they apply the Adaptive Hough Transform,[18] after a backmapping process, in order to detect the remaining parameters. The complexity of this stage, as pointed out in Table 6, will be given by the parameter range reduction factor, γ, in each iteration of the algorithm, the size of the parameter space, β, the size of the image N and the number of points P.

Muammar and Nixon[14] use three stages for the detection of ellipses, but they need to implement a backmapping process in order to perform the labeling. This process has a computational complexity which is equivalent to the detection of centers. The detection of the orientation angle is carried out pairing points located at equal distances from the center. This method can be affected if we randomly restrict the number of points of the edge. On the other hand, the detection of the semiaxis is carried out pairing points for the figures and solving the equations of the ellipses. The complexities of the different stages of this algorithm are shown in Table 6.

Finally, Yoo and Sethi[15] first calculate the parameters of the general equation of a conical figure and from this information they obtain, in different stages, the center,

Table 6. Complexity of some Hough algorithms to detect ellipses

Algorithm	Computation				Memory
Yuen[12]	Center $N \cdot \sum_{i=1}^{f} \frac{P_i(P_i+1)}{2}$	Backmapping $\sum_{i=1}^{f} \frac{P_i(P_i+1)}{2}$	Semiaxis and orientation $d \cdot (\beta+1)\frac{\log_2 N}{\log_2 \gamma} \cdot P$		N^2
Maummar[14]	Center $N \cdot \sum_{i=1}^{f} \frac{P_i(P_i+1)}{2}$	Backmapping $\sum_{i=1}^{f} \frac{P_i(P_i+1)}{2}$	Orientation $\sum_{i=1}^{f} \frac{P_i^2-4P_i}{8}$	Semiaxis $\sum_{i=1}^{f} \frac{P_i(P_i+1)}{2}$	N^2
Yoo[15]	Conical par. and center $N \cdot \sum_{i=1}^{f} \frac{P_i(P_i+1)}{2}$	Semiaxis $f \cdot \sum_{i=1}^{f} \frac{P_i(P_i+1)}{2}$	Orientation $\sum_{i=1}^{f} P_i$		$^*K_P \cdot \log_2 N \cdot \sum_{i=1}^{f} P_i^2$
FEHT	Center $2^t \cdot \sum_{i=1}^{f} \frac{P_i(P_i+1)}{2}$	Orientation $\sum_{i=1}^{f} P_i$	Semiaxis $\sum_{i=1}^{f} P_i$		$^*K_P \cdot \sum_{i=1}^{f} P_i^2$

* K_P is the elimination factor value for the edge points.

semiaxis and orientation angle. In order to calculate the parameters of the general equation, points of the ellipse are paired and the shoulder point of the pole point found for each pair is sought. The complexity of the search process is N, as the image must be explored in one direction until the shoulder point is found. This algorithm, as can be observed in Table 6, also uses backmapping, during the stage of orientation detection, in order to limit the number of points that pass from one stage to the next. It is also necessary to point out that the evaluation of these last three algorithms has not included the times required for finding the maxima produced by the center votation in the accumulator spaces.

Assuming P edge points, the method used in the FEHT algorithm for the detection of the center of the ith ellipse generates, in the worst case, $P_i \times (P_i + 1)/2$ lines for figure F_i with a computational complexity of $2^t \times P_i \times (P_i + 1)/2$, where 2^t is the rate between the number of generated lines and the threshold T. All the pairings between points are not really going to occur due to the constraints that have been imposed, but even so, the number of computations can be very high. One way of reducing the computation is to eliminate points from the edge using a pseudorandom function in order to make the number of pairings of the order of P.[15] Despite the point elimination, the algorithm has behaved in a very robust manner.

During the calculation of the orientation and the semiaxis ratio, each point of the ellipse votes in an angle range between 0° and 180°. The complexity of this process will be given by the number of points that make up the ellipse, that is P_i. In addition we will use a two-dimensional space where the votations will be carried out. In the detection of the semiaxis, each point of the figure votes on a value for semiaxis a and one for semiaxis b. The complexity of this stage will be of the order to the number of points of the figure in the image, P_i. For this stage we need a two-dimensional space whose size will depend on the resolution with which we want to find the semiaxis and which will be given by a maximum of N^2. In Table 6 we indicate the order of the computational complexity of each stage of the FEHT algorithm and of the memory.

Comparing the behavior of the FEHT algorithm with the rest we can observe that the order of the complexity in the detection of the center of the algorithm presented will be less than the complexities of references (12,14,15) if $2^t < N$. In the detection of the axes, the advantages of the FEHT can also be clearly observed as it presents a complexity of the order of the number of image edge points whereas in references (14,15) we find a quadratic dependence. In the case of the detection of the orientation, our algorithm presents the same complexity as in reference (8) which is lower than that of reference (15). However, the dominant term with respect to the amount of memory used is less in reference (12,14).

With respect to the memory, the algorithms of Yuen and Muammar use a bidimensional parameter space to vote. Yoo stores the five ellipse parameters for each candidate edge point pair in a table. The FEHT needs, at least, $\sum P_i^2$ memory positions for each level. The memory requirements for these two algorithms can be improved using the elimination of points. Of course, if we apply elimination the points, the computational complexity of all the algorithms in the Table 6 will improve. This is the reason because we have only indicated the improvement in the needed memory of FEHT and Yoo algorithms.

5.3. *Considerations about the parallelism*

Taking into account that the traditional algorithm for the HT has a strong implicit parallelism as shown in references (24–29) and the computation time required for the Hough algorithms, we will make some considerations about the parallelism of the FCHT and FEHT algorithms.

With distributed memory, we can benefit from the parallelism by dividing the parameter space into squares. Each one of these squares is assigned to a processor that is in charge of processing it according to the method we discussed in the previous section. This way, the references to higher subquadrants for the calculation of distances are local and will not imply communications between processors. A problem this distribution presents is the load balance, as it is possible, depending on the solutions, for some processors to remain idle while others

have to continue performing calculations. An attempt to balance the load would imply communications which would degrade the performance of the algorithm. We thus need a scheme for the initial distribution of the load that is more complex than the aforementioned one and which permits the progression of the algorithm with the minimum number of load rebalancing operations. Taking into account the independence of the communications between points belonging to the same figure (except for the total vote over a subquadrant) we can choose a scatter (sparse cyclic) distribution of the points among the processors.[30] It will eventually be necessary to rebalance the load, as shown in reference (31), when the new distribution compensates the overhead generated by rebalancing operation. We have studied different alternatives for dynamic balancing using distributed and centralized rebalancing mechanisms.[32] Both mechanisms have a good behavior when the number of processors is low (less than 32), but it gets worse when this number is increased owing to the rebalancing mechanism overhead.

The best results have been obtained using a static balancing where all the processors collaborate to calculate the voting in each tree node. In this situation, speed-ups around 50 have been achieved using 64 processors (in the parallel computer AP-1000 of Fujitsu).

In the work carried out up to date for the calculation of circles and ellipses performing the polling in the parameter space, a force sequentially between the different stages is introduced. Thus, the stage for the detection of the center of the ellipse or the circle must completely end (generate all the possible centers) before the rest of the stages can progress. However, using the FCHT and FEHT algorithms it is possible to obtain several solutions in parallel, as they do not require the complete end of a stage for the next stage to progress. This way, if we had different hardware for processing each stage, the production of solutions could be pipelined, accelerating the execution of the algorithms. This idea has been applied to develop a new parallel pipelined Hough transform that parallelizes the complete process detection including the edge detection.[33] In order to obtain a scalable algorithm the information has been redistributed in each stage with the aim to achieve a good balancing

6. CONCLUSIONS

In this work we have presented two algorithms for the detection of circles and ellipses based on a new FHT algorithm. The new FHT algorithm solves the problems of the previous focusing algorithms using a new focusing policy and extending the use of the presence vectors. The FCHT and FEHT algorithms consider a decomposition of the parameter space that permits calculating the values of the figures in the image in a stagewise manner. The new focusing algorithm is applied in order to find the centers and new techniques are used for detecting the orientation and the value of the two semiaxis in the ellipses. The focusing process permits the progression of subsequent stages of the algorithms without the first one having completely ended. In addition, the labeling of the points through presence bits results in significant computational savings.

Acknowledgements—Special thanks are due to the reviewer who contributed to many helpful comments.

REFERENCES

1. R. D. Duda and P. E. Hart, Use of the Hough transform to detect lines and curves in pictures, *Comm. ACM* **15**, 11–15 (1972).
2. Qin-Zhong Ye, A preprocessing method for Hough transform to detect circles, *Proc. IEEE Conf. Comput. Vis. Pattern Recognition*, pp. 651–653 (1986).
3. E. R. Davies, Circularity: A new principle underlying the design of accurate edge orientation operators, *Image Vision Comput.* **2**(3), 134–142 (1984).
4. X. Cao and F. Deravi, An efficient method for multiple-circle detection, *Proc. IEEE Int. Conf. on Computer Vision*, pp. 744–774 (1990).
5. T. Nagata, H. Tamura and K. Ishibashi, Detection of an ellipse by use of a recursive least-square estimator, *J. Robotic Systems* **2**(2), 163–177 (1985).
6. R. Chan and W. C. Siu, A new approach for efficient Hough transform for circles, *Proc. IEEE Pacific Rim Conf. on Comunications, Computers and Signal Processing*, pp. 99–102 (1989).
7. R. Chan and W. C. Siu, New parallel Hough transform for circles, *IEE Proc.-E* **138**(5), 335–344 (1991).
8. E. R. Davies, A skimming technique for fast accurate edge detection, *Signal Process.* **26**, 1–16 (1992).
9. R. S. Conker, A dual plane variation of the Hough transform for detecting non-concentric circles of different radii, *Comput. Vision Graphics Image Process.* **43**, 115–132 (1988).
10. S. Tsuji and F. Matsumoto, Detection of ellipses by a modified Hough transform, *IEEE Trans. Comput.* **C-27**(8), 777–781 (1978).
11. H. Tsukune and K. Goto, Extracting elliptical figures from an edge vector field, *Proc. IEEE Computer Vision and Pattern Recognition Conf.* WA, U.S.A., pp. 138–141 (1983).
12. H. K. Yuen, J. Illindworth and J. Kittler, Detecting partially occluded ellipses using the Hough transform, *Image Vision Comput.* **7**(1), 31–37 (1989).
13. C. L. Huang, Elliptical feature extraction via an improved Hough transform, *Pattern Recognition Lett.* **10**, 93–100 (1989).
14. H. K. Muammar and M. Nixon, Tristage transform for multiple ellipse extraction, *IEE Proc.-E* **138**(1), 27–35 (1991).
15. J. H. Yoo and I. K. Sethi, An ellipse detection method from the polar and pole definition of conics, *Pattern Recognition* **26**(2), 307–315 (1993).
16. C. Ho and L. Chen, A fast ellipse/circle detector using geometric symmetry, *Pattern Recognition* **28**(1), 117–124 (1995).
17. H. Li, M. A. Lavin and R. J. Le Master, Fast Hough transform: A hierarchical approach, *CVGIP* **36**, 139–161 (1986).
18. J. Illingworth and J. Kittler, The adaptive Hough transform, *IEEE Trans. PAMI* **9**(5), 690–698 (1987).
19. M. Atiquzzaman, Multiresolution Hough transform: An efficient method of detecting patterns in images, *IEEE Trans. PAMI* **14**(11), 1090–1095 (1992).
20. N. Guil, J. Villalba and E.L. Zapata, Fast segment Hough transform, *IEEE Trans. Image Process* **4**(11), 1541–1548 (1995).

21. E. R. Davies, A modified Hough scheme for general circle location, *Pattern Recognition Lett.* **7**, 37–43 (1988).
22. V. F. Leavers, The dynamic generalized Hough transform: Its relationship to the probabilistic Hough transform and an application to the concurrent detection of circles and ellipses, *CVGIP: Image Understanding* **56**(3), 381–398 (1992).
23. A. Califano and R. M. Bolle, The multiple window parameter transform, *IEEE Trans. PAMI* **14**(12), 1157–1170 (1992).
24. D. Ben-Tzvi, A. Naovi and M. Sandler, Synchronous multiprocessor implementation of the Hough transform, *CVGIP* **52**, 437–446 (1990).
25. M. Grazia Albanesi, Time complexity evaluation of algorithms for the Hough transform on mesh connected computers, *IEEE CompEuro'91*, 253–257 (1991).
26. A. N. Choudhary and R. Ponnusamy, Implementation and evaluation of Hough transform algorithms on a shared-memory multiprocessor, *J. Parallel Distributed Comput.* **12**, 178–188 (1991).
27. 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.* **19**(5), 805–820 (1990).
28. M. Ferreti, The generalized Hough transform on mesh-connected computers, *J. Parallel Distributed Comput.* **19**, 51–57 (1993).
29. S. Kumar, N. Ranganathan and D. Goldgof, Parallel algorithms for circle detection in images, *Pattern Recognition* **27**(8), 1019–1028 (1994).
30. L. F. Romero and E. L. Zapata, Data distribution for sparse matrix vector multiplication, *Parallel Comput.* **21**(4), 583–605 (1995).
31. D. Gerogiannis and S. C. Orphanoudakis, Load balancing requirements in parallel implementation of image feature extraction tasks, *IEEE Trans. Parallel Distributed Systems* **9**(4), 994–1013 (1993).
32. N. Guil and E. L. Zapata, Parallelization of irregular algorithms for shape detection, *Int. Conf. on Image Processing* **2**, 129–132 (1996).
33. N. Guil and E. L. Zapata, A parallel Pipelined Hough Transform, *Euro-par'96 Conf.*, **2**, 131–138 (1996).

About the Author — NICOLAS GUIL received the B.S. degree in Physics from the University of Sevilla, Spain, in 1986, and the Ph.D. degree in Computer Science from the University of Malaga in 1995. He worked in the R&D Department of a computer enterprise from 1986 to 1990. During 1990–1996 he has been an Assistant Professor in the Department of Computer Architecture at the University of Malaga in Spain. His research interests include image processing and parallel computation.

About the Author — 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 Processor at the University of Santiago de Compostela. Currently, he is a Professor with the Department of Computer Architecture in the University of Malaga. His research interests are in the area of parallel computation architectures, parallel algorithms, numerical algorithms for dense and sparse matrices, and VLSI digital signal processing. Dr Zapata has published more that 40 papers in refereed international journals in his area of interest and about 60 refereed international conference proceedings.