MVA '96 IAPR Workshop on Machine Vision Applications, November. 12-14, 1996, Tokyo, Japan

Architectures for the Hough Transform: a Survey

Marco Ferretti
Dipartimento di Informatica e Sistemistica
University of Pavia, Italy
E-mail:FERRETTI@IPVMV1.UNIPV.IT

Maria Grazia Albanesi
Dipartimento di Informatica e Sistemistica
University of Pavia, Italy
E-mail: ALBANESI@IPVMV1.UNIPV.IT

Abstract

This survey reviews the implementations of the Hough transform on parallel systems and on special purpose devices. The Hough transform continuously receives much attention because of its usefulness both as a tool in industrial applications and as a step in building perceptual representations for computer vision tasks. Its computational complexity has motivated many efforts towards fast implementations: all parallel systems, built or just conceived in the last twenty years, have been used as a test bed for parallelization strategies. Furthermore, practical, dedicated real time solutions have emerged to meet the needs of on-line inspection. This paper covers SIMD, MIMD and special purpose, dedicated implementations. The analysis of asynthotic computational complexity is paired with more practical considerations on the feasibility of each solution.

1 Introduction

The Hough transform (HT) is a well established method to detect shapes in images. It was introduced originally for locating lines [1] and has been later extended to other analytic curves (circles, ellipses); the so called generalized Hough transform (GHT) [2,3] handles the detection of shapes specified with a template of boundary points. The method has received much attention and has been used extensively in many application environment, notably in inspection tasks.

The purpose of the present work is to review the implementations of the Hough transform on general purpose parallel systems and on dedicated ones: the main interest here is on the architecture of the systems on which the transform has been implemented. The suitability of the method for parallelism and, vice versa, the suitability of existing and/or proposed systems for efficiently embedding it are the guide-lines of this review. In an effort to improve the efficiency of the method, a number of variations to the two basic types of the HT have appeared. The reader is referred to the review papers [4,5], which cover such extensions.

The motivation for this paper rests upon the number of implementations that have been given to speed up the computations required by this task; the Hough transform has been implemented on virtually every existing parallel system and many dedicated solutions have already been fabricated. The main contributions are here organized and grouped according to a categorization of the method that highlights its suitability for parallelization: the topology of the system and the match between the computation and the architecture are used to classify the implementations.

2 The Hough Transform

The Hough transform is a structural method to describe shapes. In this transform, a shape is characterized by a set of parameters that specify its pose: location, orientation, scaling factor and so on. As an instance, a line is specified by two parameters, a circle by three. The structural characterization can be explicit, when the shape has an analytic description (HT for lines, circles, ellipses); or implicit, when the shape is known only as a set of boundary points that make up a template of all possible instances (GHT).

The transform requires that the image be pre-processed to identify "feature points", that is those locations that carry the salient information associated to the outline of the shape: these are obviously the boundary points. The coordinates of each feature point are used along with the structural description to identify the loci of the parameters compatible with each feature point. These loci are a sub-set of the "parameter space", the multi-dimensional hyper-space where all possible shape instances are mapped to different hyper-points. The transformation from the feature point space to the parameter space is also known as "voting" process: each feature point votes for a set of points (one of the loci) in the parameter space. The parameter space in turn is frequently referred to as "accumulator" space: when a quantization is introduced in the parameters, the space is divided into "bins" and the voting process accumulates the votes in the bins that make up the discrete approximation of the loci identified by feature points.

Of the various shapes for which an analytic structural description is possible, lines are by large the most studied and the most used in practical applications. Not surprisingly, almost all parallel implementations of the HT address the line detection problem. The parametrizations for lines used in parallel implementations are: slope and ordinate intercept (m,c); distance from the origin and slope angle of the normal (ρ,θ); slope angle and ordinate intercept (ϕ,c) and intersection with image borders (s1,s2). In the last one, s1,s2 (s1<s2) are the distances of the intersection points

of a line from the lower left corner of the image measured along the border in counter-clock-wise direction. The corresponding line equations for the other are:

$$(m,c) \qquad c = mx + y \qquad (1)$$

$$(\rho,\theta) \qquad \rho = x\cos\theta + y\sin\theta \qquad (2)$$

$$(\phi, c) \qquad c = x\,\mathrm{tg}\phi + y \qquad (3)$$

For the purpose of this work, it suffices to note that the first one has to cope with an unbounded range of slope values. Obviously, the parameter space is bi-dimensional: the quantization of each parameter is usually uniform.

The generalized version of the transform (GHT) describes a shape by means of the points lying on its boundary and uses consistently the directional information of the gradient along the shape's contour. The structural description consists of a model built by choosing a reference point P_r and a set of boundary points P_b. The model is organized in a table, the so-called *reference table*, which stores the displacement vectors P_b-P_r as a function of the direction $\Phi(P_b)$ of the gradient at P_b along the boundary of the shape. The number N_v of points selected to build the template of the shape is the cardinality of the reference table. The process of votes accumulation obeys the *voting rule*; each feature point in the image space contributes as many votes as the entries in the reference table. The locations of the parameter space where such votes are to be accumulated are identified by composing the displacement vectors stored in the table with the position vector of the feature point. The parameter space must be expanded to four dimensions if the shape has to be detected with its orientaion and with its scale. This technique is very flexible, but somewhat less robust to noise than the analytic versions because of its dependance on the edge gradient direction. From the point of view of parallel implementation, the true difficulty lies in the irregular pattern of accumulator bins addressed with the voting rule at each feature point. Only few architectural solutions have been proposed for this formulation of the transform; they are covered in section 7.

The Hough transform can be characterized as the composition of three transformations: the first from the image space I to the feature point space $\mathcal{F}$, the second from the feature point space $\mathcal{F}$ to the parameter point space $\mathcal{P}$, the third from $\mathcal{P}$ onto itself, for the detection of the shape instances. Let us make the following assumptions: the image is a bi-dimensional array of dimension NxN; the feature points are N_f, the number of votes generated by each feature point is N_v; the parameters describing the shape are p and each is quantized into α values. Then, the time complexity of a straightforward implementation of the transform on the Von-Neuman machine depends on three factors: the detection of feature points, requiring $O(N^2)$ elementary operations; the voting process, requiring $O(N_f\, N_v)$; and the analysis of the parameter space, requiring $O(\alpha^p)$. Moreover, the space complexity is dominated by the cost of the explicit representation of the whole parameter space with the same level of quantization for all parameters.

The reduction of the complexity of the method has been pursued basically in two ways: by paralleling the voting process and the subsequent analysis of the parameter space; and/or by reducing the storage requirement. The former approach is more directly influenced by the architecture of the system. The space saving approach has given rise to various improved algorithms, based on a dynamic quantization of the parameter space, on an iterative focusing procedure for the voting process and on associative votes accumulation in small, fixed size accumulator arrays.

3 Architectures for HT: a Taxonomy

The proposals to implement the "standard" HT on machines other than the Von-Neuman system can be split into two disjoint sets: proposals that map the algorithm onto existing general purpose parallel systems, and proposals which envision the design and construction of a special purpose, dedicated "Hough engine".

The implementations on a general purpose parallel system can be analyzed using two criteria: the parallelization strategy and the embedding of the phases of the algorithm on the hardware available. The asymptotic computational complexity of the proposed algorithms gives a hint of the theoretical speed-up achievable; usually, little consideration is given to practical feasibility.

The analysis of dedicated solutions is based on efficiency: short execution time and low cost. The proposed systems are reviewed with a peculiar emphasis on the structure of the I/O sub-system, on the number of the components and their technology. Parallelism is exploited in the time domain, leading in most cases to pipelined systems.

The SIMD case

The transformations making up the HT perform the same type of operation on all the elements of the respective co-domain. Taken in the order $I \rightarrow \mathcal{F} \rightarrow \mathcal{P} \rightarrow \mathcal{P}$ they are a pipeline of tasks, each consisting of a single stream of elementary operations carried out on a number of data initially large and then smaller and smaller. This calls for systems that efficiently support the SIMD parallel modality. As a consequence of this peculiarity of the HT, the majority of the parallel implementations produced rely on SIMD architectures; there is no intrinsic parallelism in tasks to justify a MIMD architecture and the few MIMD systems on which the HT has been implemented are used in a sort of SIMD emulation. Section 4 covers SIMD implementations.

A taxonomy of SIMD implementations can be based on the topology of the system: the basic classes are linear array, mesh (fixed topology, augmented, reconfigurable), tree, pyramid and hypercube. The point is: how does the topology support the transformations? is a single system suitable for the whole process? in case a single system is used anyway, which is the appropriate mapping between the system topology and the data structures involved?

Following [6], four approaches can be identified, namely: *image space parallelization*, *feature point space parallelization*, *parameter space parallelization* and mixed *feature point and parameter space parallelization*.

In *image space parallelism* processing elements (PEs) map directly the image space and "the image array is traversed by a set of counters, which follow the curves to be detected". Such counters accumulate votes by counting the number of feature points encountered.

Feature point space parallelism is defined as a one-to-one association among PEs and feature points; each point in parameter space is examined serially, broadcast to the PEs and the resulting count of votes collected. This parallelization requires a SIMD structure with inter PEs data communication for the accumulation phase. The best topologies are those that support efficiently associative operations.

Parameter space parallelism is the counterpart to image space parallelism, since the primary association is among PEs and the bins of the accumulator space; in this approach, the equation describing the curve is solved simultaneously on all PEs with different values of the parameters for each feature point: no communication among PEs is necessary and the parallel system maps a sub-set (2 in the mesh) of the p dimensions of the parameter space. If the feature points are themselves mapped onto the same system, interaction among the PEs is required to extract and broadcast them serially to the whole set of PEs.

Mixed feature point and parameter space parallelism partitions the PEs into groups, each allocated to a single feature point, while the PEs within a group are associated to the different values of one of the p parameters; each PE determines the pth parameter using the feature point and the value of the parameter to which it is statically allocated and the set of p-1 remaining parameters resulting from the scanning of the parameter space.

So far, a single system is assumed for the whole process. The mapping of the three spaces is forced to be on a single topology and it is not guaranteed that the communication primitives available always match the computation. Reconfigurable systems have an advantage here.

As to image space parallelism, the topologies which more naturally match the data structures of the three spaces are those which embed directly a mesh, namely the mesh and the pyramid. The hypercube offers mesh embeddings of different efficiency. The image space is a bi-dimensional array and can be set in correspondence with the mesh array either by distributing the processors on the image or by partitioning the image into sub-images and allocating each processor one such sub-image. Actually, most implementations assume that the mesh array is large enough to store the whole image. Feature points are easily mapped on the mesh and retain their location. The parameter space can be mapped completely when p=2, provided that the quantization levels are at most as many as the PEs of a row/column; no attempt has been made to use this topology in the case p>3 with an explicit embedding of the whole parameter space.

The pyramid can be seen as a stack of decreasing edge-length arrays and has at least the same capabilities of the mesh, as far as mapping is concerned. Moreover, it can be used profitably for a pipelined implementation of the transform, when a stream of images has to be processed in a sequel.

In image space parallelism the transformation $\mathcal{I}\rightarrow\mathcal{F}$ is trivial, while the $\mathcal{F}\rightarrow\mathcal{P}$ one is the real issue. Even in the simple case p=2, this transformation involves global data communication, because each feature point contributes in the most general situation to many bins of the parameter space, not necessarily contiguous. Essentially, the data movement techniques used distinguish one implementation from the other. From this point of view, the $\mathcal{F}\rightarrow\mathcal{P}$ transformation can be seen as a *histogramming* process; the parameter space is the p-dimensional histogram of a population (feature points) spread out in a bi-dimensional space. When the histogram is itself bi-dimensional, as in the line detection problem, the histogramming process can be decomposed into row-column histogramming by using a *recursive doubling* technique. Alternatively, the votes produced by feature points can be organized in “runs” of contiguous packets addressed to the same bin with a *sorting* step; the length of each such packet is the count which has to be routed to the proper destination PE. The line detection problem is somewhat peculiar: feature points tend to retain contiguity in space and this opens up two more ways of collecting votes. The former addresses the HT as an instance of the Radon transform and uses projections in the image space: logically, the image is *rotated* to align feature points lying along a given direction with a column (row) and then a column/(row) histogram is performed. The latter directly exploits contiguity by injecting from the borders of the array counters that travel along the chosen direction and count the feature points aligned according to that direction: a *ray tracing* technique.

The increased topology of the pyramid offers more paths to the movements of votes: assuming that both the image space and parameter space are mapped on the base mesh, the hierarchical interconnections of the remaining levels suggest a divide-and-conquer approach. The votes are combined within blocks of larger dimensions by merging smaller blocks at higher levels; the resulting votes are distributed back onto the base (the parameter space) by data movements peculiar of the pyramid topology.

In feature point space parallelism, the topology of the system is used to support associative operations. Actually, the tree has been proposed, but the pyramid is apt as well. One can even conceive the use of a mesh for the generation of votes and of a special purpose network dedicated to the counting phase, such as the tree of adders that have been proposed to augment meshes. Reconfigurable meshes are clearly superior in this approach, since they can be tuned to the histogramming process better than fixed topologies.

Parameter space and mixed feature point and parameter space parallelizations are by large the least used techniques; this depends on the cumbersome usage of the image (feature point) space they introduce. Since the image is usually the input data of the transform, mapping the parallel structure on the destination space (parameter) requires extensive reorganization of the data structure storing the image; it can either be allocated to an external controller, accessed serially and broadcast to the parallel system, or stored on the parallel system itself and routed on a pixel per pixel basis to the PEs acting for the bins of the accumulator. In either case, the parallel system topology has almost no influence. The mesh does not offer any peculiar advantage, since neighboring PEs do not interact; the tree has a smaller diameter and this can be used if image/feature points cannot be broadcast and must use the interconnections available to reach the destination PE.

There is one more possibility to cast the HT on a parallel system which is not captured by the classification just reviewed. In terms of the characterization of the transform as a sequence of steps, the HT can be described as a direct mapping from the feature point space to the set of detected instance: $\mathcal{F} \rightarrow \mathcal{S}$, with no explicit use of the parameter space. This direct transformation is best applied when a-priori knowledge is available on the cardinality of the transformed set, either because only a few instances of the sought for shape are known to be in the scene, or because only a few, the most probable ones, are actually needed. The transformation operates a dramatic reduction in the cardinality of the input set; it groups local evidence and filters out unreliable groupings. The topology that best matches this process is the pyramid, since it offers a set of meshes (local interconnections) of decreasing resolution (smaller cardinality).

The MIMD case

As already anticipated, the HT is an inherently SIMD computation in all its phases, since the co-domains of transformations are treated uniformly with the same sequence of instructions. The MIMD paradigm cannot be exploited in its widest scope and offers only increased flexibility in data sharing. MIMD systems are built out of coarse grain processors, in contrast to SIMD ones, where the PE is usually rather simple; thus, each processor in a MIMD system can address a very large memory space. The number of available processors is usually much smaller than in the SIMD case and this leads naturally to the problem of memory distribution and allocation.

In the $\mathcal{F} \rightarrow \mathcal{P}$ transformation, the address space available for the memory can either be shared or partitioned: to avoid excessive contention, a complete mapping of both spaces on a single shared resource is avoided. The alternatives therefore are on the allocation of the feature and parameter spaces to a set of disjoint memory modules. Three modes are possible: *input partitioning*, *output partitioning* and *mixed input and output partitioning* [7]. The first solution is the one that causes the heaviest form of contention, since the shared resource (parameter space) is the one that undergoes the updating operations (vote collection). It is viable if the numbers of votes generated by each of the N_f feature point is very small, possibly 1. Output partitioning allows for more alternatives to the problem of contention in feature point space access: data replication and pointer replication differently trade memory for speed. The third strategy is possible but requires extensive data transmission among the processors; indeed, if a PE is assigned a sub-set of the parameter space, it must anyway analyze the whole feature point space and only a segment of it is readily available in its private memory.

The only way to apply a true MIMD strategy consists of allocating different tasks to different processors. In the HT case, one such possibility is offered by the two $I \rightarrow \mathcal{F}$ $\mathcal{F} \rightarrow \mathcal{P}$ steps. Careful load balancing and inter-processor communication are required

Section 5 reports on the implementations falling within such classes.

4 General Purpose SIMD Systems

In the sequel, the implementations of the HT on general purpose SIMD systems are briefly described on the basic topologies of the systems.

Linear array. The image space is fed to the array in rows in [8], with the N PEs concurrently computing the HT according to a modified (ρ,θ) equation. In $O(N\, N_\theta)$ time a single ρ histogram is computed. An external controller completes the transform in $O(N_\theta\, N_\rho)$ time. Two more implementations are discussed in [9].

Basic mesh array. In this model, we assume a network of NxN PEs, with fixed local interconnections. The diameter of the mesh and the global data movements of the histogram computation of the HT limit the time complexity to $\Omega(N)$. Unless otherwise stated, the (ρ,θ) equation is used; also, the image space is mapped on the mesh on a pixel-per-PE basis.

The first published implementation [10] achieved $O(N N_\theta)$ complexity by serially computing the N_θ histograms with a recursive doubling technique. A slight improvement is obtained in [11] by using the directional information extracted from the gradient at the feature points and voting for a subset N_s of the N_θ values of the quantized θ at each feature point. The complete histogram is computed in the mesh accumulating runs of identical (ρ,θ) values along the rows, counting and finally routing them to their final destination. The time complexity is $O(N\, N_s)$. Linear time algorithms have been obtained by using the ray tracing technique [12,13]. Both proposals achieve $O(N+N_\theta)$ time complexity. By skipping the $I \rightarrow \mathcal{F}$ transformation and by loading an N_fxN_f mesh with feature point coordinates, $O(N_f+N_\theta)$ complexity is achieved in [14].

Augmented mesh. A possible way to improve the routing capability of the basic mesh consists of introducing local interconnection autonomy. In [15] this capability is used to create groups of PEs that store co-linear feature points. The parameter space is not mapped explicitly and the algorithm detects line segments, their length and orientation directly in the image space. The mesh is divided into q vertical slices; within these slices, N_θ orientations are tested. The procedure is bound by $O(N_\theta(\log(N/q))$.

With a different approach, a more efficient topology for histogram computation can be added to the basic mesh. A tree was used in [13], but without improvements with respect to the ray tracing technique. In [16] the mesh is augmented with a set of trees, one for each row. The mesh stores N_f feature points; (ρ,θ) are quantized so that $N_\theta = N_\rho$. By pipelining along the trees the computation of the histogram, a time complexity of $O(N_\theta + \log(N_f/N_\theta))$ is obtained in a rectangular mesh of $N_\theta \times (N_f/N_\theta)$ PEs, where $N_\theta = N_f^{1/2}$. A local memory of size $O(N_\theta)$ is required. In the same paper, a 3D arrangements of N_f PEs is shown capable of computing the HT in $O(N_\theta + (N_f/N_\theta)^{1/2})$ time, with the same local memory requirement and N_f/N_θ parallel inputs.

An arrangement of $N \times N$ memories and of q PEs make up the Mesh Connected Module architecture [17]. This architectures partitions the memories and the PEs into a mesh of $k \times k$ basic modules; the m PEs in a module have row and column busses to access the sub-mesh of $m \times m$ memories. As a result, $q = k^2 m$. The HT is obtained by mapping the image space in the $N \times N$ memory; the N_θ histograms are computed in $O(N/m + N_\theta m)$ time using efficient sorting and histogramming primitives.

Reconfigurable mesh. This model exploits the local configuration capability of the PEs to set-up row and column busses of various lengths at run-time. Over such busses, non conflicting, unit-time broadcast operations are assumed possible.

The first proposal to implement the HT on this architecture [18] exploit the reconfiguration capability to embed the four parallelization strategies outlined in section 3 [6]. More algorithms have followed. In [16], a mesh with N_f PEs arranged in a rectangular grid of $N_\rho x(N_f/N_\rho)$ elements uses $O(N_\theta)$ local storage to accumulate partial histograms along the columns, with near-neighbor connectivity. Broadcast busses along the rows build up the final HT in $O(N_\rho \log N_f / \log N_\rho)$ time. As with the basic mesh algorithm, $N_r = N_\theta = N_f^{1/2}$. The ray tracing technique is adapted to the reconfigurable mesh in [19]. The $N \times N$ mesh stores the image and the quantization of the parameter space is such that $N = N_\rho$, $N_\theta \ll N$. The ray tracing algorithm is first executed on sub-meshes, then the whole HT histogram is computed using broadcast within the sub-meshes and among sub-meshes, with an overall complexity $O(N_\theta \log(N/N_\theta))$. A similar result is obtained in [20].

The ultimate goal of a parallel implementation is the so-called "*constant time*" execution. To achieve such a result, the architecture must be increased well beyond the reconfigurable mesh, trading more space (PEs) for execution speed. Various configurations of arrays have been used: 1D, 3D and even 4D. Furthermore, increased interconnection autonomy is added to the reconfiguration busses, which are sometimes controlled at the bit level.

The first constant time algorithm [21] uses $O(N_f^3 N_\theta)$ PEs with a parameter space quantization $N_\theta = N_\rho$. The topology is very rich: it consists of a stack of N_θ 2D meshes of $N_f \times N_f$ PEs, having a lower triangular mesh on each column; furthermore, the first rows of each mesh make up another $N_\theta \times N_f$ mesh, thus building a "4D" structure. The algorithm broadcasts in constant time on the $N_\theta \times N_f$ mesh all possible combinations of the N_f feature points with the N_θ orientation; the mesh computes all ρ values, which are then broadcast within the $N_f \times N_f$ meshes for constant time accumulation through the lower triangular meshes. With the same assumptions on parameter space quantization and on feature point usage, an asynthotically better solution [22] uses $N_\theta\ N_f\ (\log N_f)^2$ PEs. These constant time algorithms use the busses to propagate the required multi-bit values. The m bits of the busses have been used differently in two more architectures [23,24]. The interconnections between adjacent PEs can be controlled by proper switches at the bit level. The original algorithm [23] represents numbers in a base-m number system and shows how to compute prefix sums of a sequence of N bits in $O(\log_m N)$ with N PEs with enhanced bit-level crossover capability. Assuming a bus bandwidth $m = N^{1/c}$, prefix sums are computed in constant time. On the basis of this result, histograms take constant time as well. The HT is implemented on the basis of these constant time primitive operations. The first algorithm [23] uses a 3D arrangement of $N_\theta \times N_\theta \times N_f$ PEs ($N_\theta = N_r$), the second [24] improves this result by using only $N_\theta(N_f + 1)$ PEs in a 1D topology.

Tree and pyramid. In this paragraph, the analysis covers a few algorithms based on various types of hierarchical topologies: binary tree [25], homogeneous quad pyramid [26-30], augmented heterogeneous tree [31] and heterogeneous pyramid [32]. Most implementations share the use of the hierarchy to decrease the time complexity of the histogramming phase, but are otherwise quite different in the HT formulation. Also, some use a M-SIMD structure to exploit in different ways the levels of the hierarchy. A comparison of asymptotic time complexity is therefore impossible.

The tree was use in the VON-NON system [25] to compute the HT according to the (m,c) parameter space quantization. The system is capable of reconfiguring its PEs in a single linear arrangement, in a single binary tree or in a set of sub-trees. This last configuration is used to map each feature point N_f at the root of a sub-tree. Each sub-tree has N_m PEs. The (m,c) couples are computed concurrently in the sub-trees in constant time. A double histogramming phase (one for each parameter space dimension) in the set of sub-trees builds the final HT histogram in $O(N_m + N_c + h)$, where h is the height of the overall VON-NON tree and is $O(\log N_m N_c)$.

Quad pyramid algorithms differ in the usage of the parameter space, which is seldom computed completely. The hierarchy is often used to select only a subset of the (ρ,θ) bins, namely the most voted ones.

The complete parameter space is constructed in [26]. Both image and parameter space ($N = N_\theta = N_\rho$) are mapped on the base layer of the pyramid: for each θ a ray tracing voting procedures is executed on pixels within sub-images. Partial results are transmitted to the middle layer of size $N^{1/2} \times N^{1/2}$ and merged with a divide-and-conquer technique; finally, merged results are transmitted downwards to build the final HT space. The overall complexity is $O(N_\theta N^{1/2})$. Two other algorithms are reported in [29], with a somewhat improved $O(N_\theta \log N)$ complexity.

The approach followed in [27-28,31-32] avoids building all (ρ,θ) bins of the Hough space; with an "election strategy", only strong evidence of line segments in the image space are hierarchically merged in the upper layers of the pyramid. The edge orientation θ_f of each feature points stored in base of the pyramid is locally paired with the corresponding ρ_f value computed with respect to image space coordinates. In [27], such couples are merged at successive layers: clusters of couples from the four children PEs are merged through a similarity function that takes into account both direction and orientation, and only the strongest m couples are retained and transmitted upwards. Taking into account local sorting at each bit-serial PE, the procedure costs $O(m \log N (\log m)^2)$. In a slightly different way, algorithm [28] extracts the m strongest bins by performing the voting process in an intermediate layer l, rather than in the basis. PEs in this layer receive from the base all feature points (with ρ_f, θ_f and gradient magnitude information) detected in the corresponding $2^l \times 2^l$ sub-image of the base. Only the strongest magnitude edge pixels are used for voting; then, a ranking and selection procedure similar to that of [27] is performed in the remaining layers of the pyramid. Both algorithms have the disadvantage of merging collinear segments located far apart in the image. To overcome this problem, the algorithm proposed in [31] uses a finer merging procedure, based on the length and on the location of line segments. To speed up the more complex merging, a dynamic quad-tree like parameter space quantization is used in the layers above the base. This is possible because the underlining architecture is a M-SIMD heterogeneous structure, consisting of a linear array of simple PEs in the base, and of a tree of Transputers in the upper layers. The Warwick Pyramid [32] is a set of heterogeneous clusters arranged as a mesh; each cluster is a hierarchy of three levels, having a small SIMD mesh of 16×16 PEs at the basis, a controller at the second level, and a Transputer at the root. Each small mesh is augmented with a fast associative "counting" chip. The HT is computed within each cluster to detect the strongest m local bins (line segments), with the aid of the counting chip for fast local histogramming. Cooperation among Transputers of the clusters helps in detecting collinear segments across the meshes: a second voting process is executed. A multiresolution HT is reported in [30]. A SIMD pyramid is used to build a multi-resolution representation of the image: a Gaussian pyramid is computed first and is then binarized by the DoG approximation of the Laplacian operator (this approach requires in reality an *overlapped* pyramid, rather than a quad-connected one). The voting process proceeds bottom-down according to a coarse-to-fine strategy: the range of ρ and θ at each lower layer is restricted logarithmically while descending the pyramid. The voting phase within each layer is thus constrained to the most promising (ρ,θ) pairs computed in the layer above. Admittedly, the end result in the base layer highlights only the dominant line. The algorithms is capable of sustaining pipelined processing of images entering the pyramid at the base, with a pipelining delay between image input and HT output of $2\log N$-1 cycles; furthermore, each layer l has to store $2(L-l)$ binary images, if L are the levels of the pyramid.

Hypercube. Actual systems have been built according to the topology of the hypercube, namely the Connection Machine (models 2 and 5) and the NCUBE. An implementation of the HT on the CM-2 is reported in [33]. It applies the specialized "*scan*" primitives to efficiently collect the HT histogram. Two $O(N_\theta + \log N)$ algorithms are reported in [34] for a SIMD hypercube with $N \times N$ PEs, along with experimental result obtained on the NCUBE systems. By mapping feature points to PEs and by increasing the number of PEs in the hypercube to $O(N_f N_\theta^2)$, an $O(\log N_f + \log N_\theta)$ algorithms was obtained [35].

5 General purpose MIMD systems

Tasks partitioning and memory allocation strategies differentiate the HT algorithms on MIMD systems [7, 36-39]. Coarse-grained processors will be denoted with CP.

An implementation [7] on the OSSMA shared memory multiprocessor uses image and parameter space partitioning, but resorts to synchronization among CPs to prevent conflicts during image scanning. The $N \times N$ image is partitioned into q segments and distributed to q CPs; the HT (ρ,θ) space is likewise split into q segments, so that each CP stores $N_\rho (N_\theta / q)$ bins. Image scanning proceeds synchronously with the aid of a counter at each PE; when a feature point is detected at any CP, a signal is broadcast to all others, which can compute the x_f, y_f coordinate of the feature point by reading the local counter. Voting for all local $N_\rho (N_\theta / q)$ bins proceeds without contention.

Image input partitioning is used in an implementation on the BPP [37], a set of standard microprocessors with private and shared memory. The shared HT space is quantized according to the (s1,s2) parametrization ($0 \leq s1 \leq N$, $N \leq s2 \leq 3N$, in $N \times N$ images). Using gradient information, each feature point contributes a single (s1,s2) pair; this fact, along with the huge dimension of the parameter space, lowers contention in memory accesses (which are $O(N_f)$) and allows to split the image space among the CPs. To balance CP load, image lines are broken down into segments and statically allocated to the q CPs. A (ρ,θ) version of the HT on the same architecture has worse performance, due to the high number of votes generated $O(N_f N_\theta)$.

At the cost of larger memories, the HBA system [36] eliminates image sharing by distributing the whole (binary) image to all processors, which are connected through a common video bus. The (ρ,θ) space is partitioned. This costly arrangement avoids contention.

The approach reported in [38-39] is focused on balancing the load of features points detection and parameter space construction among the CPs. The experiments on an NCUBE 10 reported in [39] highlight the communication overhead due to the distribution of detected feature points to CPs in charge of building a segmented HT space. Almost linear speed-ups were obtained when $N_f \cong (1/4)N^2$ both in the native hypercube topology and with a mesh embedding; in either case, only one CP was assigned the task of feature point detection.

6 Dedicated architectures

This section covers several dedicated systems which target real-time execution of the HT; overall, they can be described as "on-line" implementation, since they assume serial scanning of the image (usually in row-major order) and try to complete the HT process within a frame time slot. Technology plays a major role and a few proposals are clearly out-dated. Nethertheless, they are briefly described as a reference for future, up-to-date alternatives.

Systolic structures. HT systems with a systolic architecture usually rely on image preprocessing to obtain a stream of incoming feature points. The first such proposal [40] was based on an array A1 of N_θ PEs in charge of computing ρs and on a set A2 of N_θ arrays for ρs accumulation. To detect collinear segments and to merge correctly (ρ,θ) votes, each PE of A1 transmits to its A2 array the x_f,y_f coordinates: PEs in A2 compute the distance from the first x_f,y_f of the subsequent feature points contributing to the same (ρ,θ) bin. These distances are bubble-sorted in a further unit and forwarded to a final filtering array, that detects collinear segments among the accumulated bins. A similar system [41] computes the HT for a single θ value using only simple additions. Collinear segments are traced by examining consecutive adjacent rows of the $N\times N$ bit image of feature points, which is input in parallel in a linear array of computing cells. A subsequent routing array and a further accumulator array complete the process. The area-time AT complexity of this system is $O(N^2 N_\theta N_\rho \log N)$, a factor N smaller than in [40].

A HT implementation suitable for any analytic shape is reported on the WARP systolic array [42]. PEs in WARP are much more powerful than in other systolic system, since they are equipped with a considerable local memory and can be programmed to execute various local operations. Each PE stores a section of the parameter space and computes a partial histogram for each feature point that passes through it. The array of PEs is then used to extract a subset of the most voted bins.

The systolic approach followed in [43] is based on a revised (m,c) parametrization that avoids unbounded m values by using four sub-systems and by properly exchanges the roles of x and y in the line equation. The major outcome of this new formulation is use of simple additions and shift operations. Only $4N_m$ adders are necessary to complete the HT in $O(Nf + \log N_m)$ time.

Pipeline systems. The implementations grouped in this family [44-48] differ considerably in the set-up of pipeline of processing units. At the macro level, the first stage of the pipeline takes care of the $I \rightarrow \mathcal{F}$ mapping; the potential explosion of votes generated by the second stage ($\mathcal{F} \rightarrow \mathcal{P}$) is partially solved with buffering FIFOs. Standard memory banks map the parameter space and their set-up and speed conditions the accumulating phase and final peak detection.

The proposed P^3E system [44] uses a K-stage pipeline of modules: a module is devoted to computing the HT histogram for a single θ orientation and uses pre-stored lookup tables to avoid time-consuming multiplications. The HT Real Time Processor [45] builds a three-stage pipeline with off-the-shell MSI components: a 1024 FIFO stores detected feature points during a row-major image scan. Two subsequent stages compute votes and build the histogram: even though a double buffering scheme allows an histogram to be accumulated concurrently with the generation of a new set of votes, the speed of the parameter space memories is the bottleneck in the pipeline.

A recent system [48] uses fast memories and FPGA components. An internal pipelined unit is capable of accumulating the histogram of a single ρ for 64 θ values in 5.3 ms, with a conservative clock running at 12 MHz. A 128 128 image can be read from external memory and processed in 11.17 ms, under the assumption that it contains $N_f = (1/10)\ N^2$ feature points and that ρ is quantized into 256 values. This result is obtained because the accumulating memories are as fast as the internal pipeline that generates the votes.

Other systems. Other specific HT implementations use approaches that cannot be classified in any specific way. It is worth noting that industrial chips have been marketed to support the histogramming phase of HT [49].

A complex implementation based on wafer scale integration is reported in [50]. The wafer silicon area is allocated partly to multiply-and-accumulate cells, partly to statically routed connections and is used only for the computation of the HT histogram. The system is able to execute the HT almost at frame rate. Edge pixels enter the wafer at a rate of 5 MHz; 4 ρ values are sent out every 200 ns, so that the output of the total 64 θ values takes 3.2 μsec. These data are used to address external memory, with a read-modify-write cycle of 200 ns. A 256x256 image, at a throughput rate of 3.2 μs for edge pixel, can be processed in about 200 ms. If $N_f = (1/10)\ N^2$, the time for the HT implementation is about 20 ms.

Content addressable memories [51] are used in the HiPIC system both for line [52] and for circle detection [53]. The 336-Kbit CAM chip has 4K words: each word contains 84 search bits, 72 of which are writable. The CAM supports maskable search, read and maskable write operations and generates single-hit and multi-hit flags. True real-time HT processing is achieved. In the HT algorithm, each CAM word is associated to a (ρ,θ) bin. The writable part of a each word is split into a decision field and into an accumulator field: the decision fields store all possible intersections of lines with a single image row and are updated easily at each new image line. During row major image scanning, all decision fields that match the current feature point y_f coordinate are flagged and parallel voting into the accumulator fields occurs at the end of each line. A 64×64 parameter space is mapped completly in a single CAM chip and its histogram is accumulated in 0.7 ms. Using symmetry due to line orientations and mapping four accumulators and four hit-flag fields in a single CAM word), a 256×256 image is processed against a 256×256 parameter space in 15.1 ms; four CAM chips process a 512×512 image in 19.1 ms.

Very simple and efficient HT solutions are based on simple serial counters [54] and on analogue processing [55].

7 The Generalized Hough Transform

A serial implementation of the GHT to detect shapes described by N_v points, considering N_ϕ orientations and N_s scales requires $O(N_f N_v N_\phi N_s)$ operations on a $N \times N$ image containing N_f feature points. The space required to represent the 4D parameter space is itself huge. Furthermore, the almost random direction of edge gradients in images gives rise to randomly distributed votes in the GHT space, making parallel processing on any architecture very difficult. This section analyzes the few hardware implementations of the GHT; each takes a different approach and often considers a reduced version of the transform (no orientation, no scale).

A cache-based GHT [56-58] tackles the reduction of parameters space. Although conceived for a reduced GHT, it is based on a very general principle; the parameter space is substituted with a small content addressable memory, which only stores voted bins. Once the memory fills up, a flushing operation is required: thresholding is the simplest approach, but hierarchical based strategies based on multiple caches have been suggested as well.

A special purpose chip-set for the reduced GHT has been designed to overcome the same problem [59-61] and to cope with real-time implementation. The underlining idea is that the displacement vectors stored in the reference table have a magnitude d bound by the radius of the circle enclosing the shape. Usually, such d is much smaller than the image size N. During row major scanning of the image, votes cannot be outside the $2d$ rows centered on the current one. By keeping "old strong" accumulated votes and discarding the weak ones, the memory requirement can be dramatically reduced. The chip-set consists of an edge extractor chip, a voting chip, and a systolic accumulator queue capable of on-line accumulating and sorting votes. It is possible to show that shape orientation can be detected by a single second GHT using a new reference table with a second reference point.

The same principle is applied to map the GHT on simple meshes [62]. By local shift operations in a $d \times d$ window, the GHT can be computed in $O(N_v d^2 \log N)$ time. This result does not take into account the increased flexibility of reconfigurable meshes. An investigation of the use of CM-2 system is reported in [63]. The reduced GHT is implemented according to the local shift operation approach described in [62] using a mesh embedded in the underlining hypercube, or as a generalized histogram, using the "send and accumulate" primitives of the CM-2 system. The efficiency depends heavily on the ratio of "virtual" to physical processors (VPR), since the three spaces (image, features and parameters) map differently on the hypercube. The "send-and-accumulate" algorithm proves better on the 32K processor CM-2; the larger VPR, the higher the speed-up.

A fast "linear" GHT is proposed in [64]: conflicts in the accumulation phase are reduced by replacing peaks in parameter space with linear numeric patterns. A parallel implementation on a SIMD linear array is reported. A hierarchical processing scheme and an inverse GHT are the basis of an algorithm [65] suitable for pyramid machines. A shape is described at more resolutions through scaled down reference tables: voting starts at a coarse resolution and a refinement is obtained by using a more detailed reference table on the projected accumulator space. This method is based on an inverse voting process (from $\mathcal{P}$ back to $\mathcal{F}$) and on mirrored reference tables.

8 Conclusions

Work on efficient implementations of the Hough transform continues steadily in recent years. Approaches not reported here are those based on neural networks and on optical processing. The latter dispenses with numerical problems and takes advantage of the speed-of-light processing rate. It is still hampered by the limited transduction capabilities of electro-optical devices, which allow for limited image resolutions. The decline in general purpose parallel processing systems makes the special purpose approach more promising: VLSI technology improves on and on, with a declining cost in the realization of ASIC circuits. The emergence of very fast DSPs, embedding parallel sub-units (the Texas Instruments C80 family is just an example) might offer the opportunity to capitalize on the "fancy" efforts at paralleling the HT meanwhile using standard, advanced industrial components.

References

[1] P. V. C. Hough, "Method and means for recognizing complex patterns," U.S. Patent 3069654, 1962.

[2] D. H. Ballard, "Generalizing the Hough transform to detect arbitrary shapes," *Pattern Recognit.* , vol. 13, pp. 111-122, 1981.

[3] E.R. Davies, *Machine Vision*, Academic Press, 1990.

[4] J. Illingworth and J. Kittler, "A survey of the Hough transform," *Computer Graphics Image*

Processing, vol. 44, pp. 87-116, 1988.

[5] V.F. Leavers, "Which Hough transform," *Computer Vision, Graphics Image Proc. - Image Understan.*, vol. 58, n. 2, pp. 250-264, 1993.

[6] M. Maresca, M. Lavin and H. Li, "Parallel Hough transform algorithms on Polymorphic Torus Architecture," in *Multicomputer Vision*, S. Levialdi ed., ACADEMIC PRESS, 1988.

[7] D. Ben-Tzvi, A. Naqvi and M. Sandler, "Synchronous multiprocessor implementation of the Hough transform," *Comput. Vision, Graphics and Image Proc.*, vol. 52, pp. 437-446, 1990.

[8] A.L. Fisher and P.T. Highman "Computing the Hough transform on a scan line processors (image processing)," *IEEE Pattern Anal. Mach. Intell.*, PAMI vol. 11, n. 3, pp. 262-265, 1989.

[9] Z.N. Li, F. Tong and R.G. Laughlin, "Parallel algorithms for line detection on a 1xN array processor," IEEE Computer Society Press, Los Alamitos, 1991.

[10] T. M. Silberberg, "The Hough transform on the Geometric Parallel Processor," *IEEE Workshop on CAPAIDM*, New York, Nov. 1985, pp. 387-393.

[11] A. Rosenfeld, J. Ornelas and Y. Hung, "Hough transform algorithms for mesh-connected SIMD parallel processors," *Computer Graphics Image Processing*", vol. 41, pp. 293-305, 1988.

[12] C. Guerra, S. Hambrush, "Parallel Algorithms for Line Detection on a Mesh," *IEEE Workshop on CAPAMI*, Seattle, WA, 1987, pp. 99-106.

[13] R.E. Cypher, J.L. Sanz and L. Snyder, "The Hough transform has O(N) complexity on SIMD NxN mesh array architectures", in *IEEE Workshop on CAPAMI*, Seattle, WA, 1987, pp. 115-121.

[14] C.S. Kannan and H.Y.H. Chuang, "Fast Hough transform on a mesh connected processor array," *Inform. Proc. Lett.*, vol 33, pp. 243-248, 1990.

[15] D.B. Shu, J.G. Nas, M.M. Eshaghian and K. Kim, algorithms on an orthogonally-connected memory-based architecture," in *Proc. 10th ICPR*, IEEE Computer Society Press, vol. II, pp. 350-355, 1990.

[16] Y. Pan and H.Y.H. Chuang, "Faster line detection raight-Line Detection on a Gated-Connection VLSI Network," *Proc. 10th ICPR*, IEEE Computer Society Press, vol. II, pp. 456-461, 1990.

[17] H.M. Alnuweiri and V.P. Kumar, "Optimal image algorithm on enhanced mesh connected arrays," *IEE Proc. E.*, vol. 140, n. 2, pp. 95-100, 1993.

[18] M. Maresca, H. Li and M.M. Sheng, "Parallel computation on a polymorphic torus architecture", *Machine Vision Appl.*, vol. 2, n. 4, pp. 215-230, 1989.

[19] J.F. Jenq and S. Sahni, "Reconfigurable mesh Hough transform on reconfigurable meshes," *Image Vision Comput.*, vol 11, n. 10, pp. 623-628, 1993.

[20] S. Olariu, J.L. Schwing and J. Zhang, "Computing uting constant time algorithm for computing Hough transform," *Pattern Recogn.*, vol. 26, n. 2, pp. 277-286, 1993.

[21] T. Kao, S. Horng, Y. Wang and K. Chung, "A gorithms for the Hough transform," *J. Parallel Distr. Computing*, vol. 20, pp. 69-77, 1994.

[22] Y. Pan, "A more efficient constant time algorithm for computing the Hough transform," *Parallel Proc. letters*, vol. 4, nn. 1-2, pp. 45-52, 1994.

[23] T.W. Kao and S.J. Horng, "A O(*1*) time algorithm for computing histogram and Hough transform on a cross-bridge reconfigurable array of processor," *IEEE Trans. Syst. Man Cybernet.*, vol. 25, pp. 681-687, 1995.

[24] S. Lee, S. Horng, T. Kao and H. Tsai, "Optimal computing Hough transform on a reconfigurable array of processors with wider bus networks", *Pattern Recogn.*, vol. 29, n.4, pp. 603-613, 1996.

[25] H. Ibrahim, J. Render and D. Shaw, "On the application of massively parallel SIMD tree machines to certain intermediate-level vision tasks," *Computer Vision Graphics Image Processing*, vol. 36, pp. 53-75, 1986.

[26] G. Bongiovanni, C. Guerra and S. Levialdi, "Computing the Hough transform on a pyramid architecture," *Machine Vision and Applications*, vol. 3, no. 2, pp. 117-123, 1990.

[27] S.T. Tanimoto, "From pixel to predicates in pyramid machines," in *From Pixel to Features*, J. C. Simon ed., Elsivier Science Publishers B. V. (North-Holland), 1989, pp. 383-392.

[28] J.M. Jolion and A. Rosenfeld, "A O(log *n*) pyramid Hough transform," *Pattern Recogn. Letters*, vol. 9, pp. 343-348, 1989.

[29] A. Kavianpour and N. Bagherzadeh, "Parallel Hough transform for image processing on a pyramid architecture," *Inter. Conf. on Parallel Processing*, pp. 1395-1398, 1991.

[30] M. Atiquzzaman, "Pipelined implementation of the multiresolution Hough transform in a pyramid multiprocessor," *Pattern Recogn. Letters*, vol. 15, pp. 841-851, 1994.

[31] N.L. Ze and Z. Dampo, "Fast line detection in a hybrid pyramid," *Pattern Recogn. Letters*, vol. 14, N.1, pp. 53-63, 1993.

[32] N. D. Francis, G. R. Nudd, T. J. Atherton, D. J. Kerbyson, R. A. Packwood and J. Vaudin, "Performance evaluation of the hierarchical Hough transform on an associative M-SIMD architecture," *Proc. 10th ICPR*, IEEE Computer Society Press, vol. II, pp. 509-511.

[33] J.J. Little, G. Blelloch and T. A. Cass, "Algorithmic techniques for computer vision on a fine-grained parallel machine," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-11, n. 3, pp. 244-256, 1989.

[34] S. Ranka and S. Sahni, "Computing Hough transforms on hypercube multicomputers," *Journal of Supercomputing*, vol. 4, n. 2, pp. 169-190, 1990.

[35] Y. Pan and H.Y.H. Chuang, "Parallel Hough transform algorithms on SIMD hypercube arrays," *Proc. Int. Conf. Parallel Process.*, vol III, pp. 83-86, 1990.

[36] Richard S. Wallace, Michael D. Howard, "HBA Vision Architecture: Built and Benchmarked", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. II, no. 3, pp. 227-232, March 1985.

[37] T. J. Olson, L. Bukys and C. M. Brown, "Low level image analysis on an MIMD architecture", *Proc. 1st IEEE ICCV*, London, 1987, pp. 468-475.

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

[39] F.O. Ozbek and M.D. Wagh, "A parallel Hough transform for nonuniform images," *Pattern Recogn. Letters*, vol. 15, pp. 253-259, 1994.

[40] H. Y. Chuang and C.C. Li, "A systolic array for straight line detection by modified Hough transform," *IEEE Workshop. on CAPAIDM*, New York, Nov. 1985, pp. 300-304.

[41] H.F. Li, D. Pao and R. Jayakumar, "Improvements and systolic implementation of the Hough transformation for straight line detection," *Pattern Recogn.*, vol. 22, n. 6, pp. 697-706, 1989.

[42] H.T. Kung and J.A. Webb, "Global Operations on a Systolic Array Machine," *Proc. IEEE Int. Conf. Computer Design: VLSI Comput. Process*, pp. 165-171, 1985.

[43] L. da Fontoura Costa and M. Sandler, "A binary Hough transform and its efficient implementation in a systolic array architecture," *Pattern Recognition Letters*, vol. 10, pp. 329-334, 1989.

[44] J. L. C. Sanz, "Two real-time architectures for image processing and computer vision," in *Real Time object Measurement and Classification*, A. K. Jain, Ed., Springer-Verlag, 1988, pp. 1-23.

[45] K. Hanahara, T. Maruyama and T. Uchiyama, "Real-time processor for the Hough transform," *IEEE Trans. on Pattern Anal. Machine Intell.*, vol. PAMI-10, no. 1, pp. 121-125, 1988.

[46] A.B. Martinez and V. Llario, "Real time holes location. A step forward in bin picking tasks," in *Sensor devices and Systems for Robotics*, NATO ASI Series, vol. Fç", pp. 167-185, 1989.

[47] S.B. Shukla, V. Ramakrishnan and D.P. Agrawai, "A pipelined architecture for on-line low-level vision," in *EURoMICO 90' Work. on Real Time*, Horsholm, Denmark, pp. 198-204, 1990.

[48] R. Cucchiara, G. Neri and M. Piccardi, "A real-time architecture for the Hough transform," Tech. Rep. ING/039, Ist. Ingegneria, University of Ferrara, Italy, July 1996.

[49] L64250 Histogram/Hough Transform Processor (HHP), Digital Signal Processing Databook, LSI Logic Corporation, November 1990.

[50] M. Rhodes et al., "Monolithic Hough transform processor based on Restructurable VLSI," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-10, no. 1, pp. 106-110, 1988.

[51] T. Ogura, M. Nakanishi, T. Baba, Y. Nakabayashi, R. Kasai, "A 336-Kbit content addressable memory for highly parallel image processing," Proc. CICC'96, 1996.

[52] M. Nakanishi and T. Ogura, "A real-time CAM-based Hough transform algorithm and its performance evaluation," in *Proc. 13nth ICPR*, IEEE Press, vol. B, pp. 516-521, 1996.

[53] M. Meribout, T. Ogura and M. Nakanishi, "Real-time circle extraction using CAM-based Hough transform", Proc. MVA'96, 1996.

[54] D. Ben-Tzvi and M. Sandler, "Counter-based Hough transform", private communication, submitted to Electronic Letters.

[55] D. Ben-Tzvi and M. Sandler, "Analouge implementation of the Hough transform," *IEE Proceedings-G*, vol. 138, n. 4, pp. 457-462, 1991.

[56] C.M. Brown and D.B. Sher, "Hough transformation into cache accumulators: considerations and simulations," CSD, University of Rochester, TR 114, 1982.

[57] D. Sher and A. Tevanian, "The vote tallying chip: a custom integrated circuit," CSD, University of Rochester, TR 144, 1984.

[58] C.M. Brown, "Peak finding with limited hierarchical memory," *Proc. 7th ICPR*, Montreal, Canada, July 1984, pp. 246-249.

[59] M. Albanesi and M. Ferretti, "Shape detection with limited memory," *Pattern Recognition*, vol. 24, n. 12, pp. 1153-1166, 1991.

[60] M. Albanesi, and M. Ferretti, "Systolic merging and ranking of votes for the Generalized Hough Transform," *IJPRAI*, vol. 9, n. 2, pp. 315-341, 1995.

[61] M. Albanesi, A. Antola, M. Ferretti and R. Negrini: "A Chip Set for the Generalized Hough Transform," *Journal of VLSI Signal Processing*, vol. 12, pp. 115-134, 1996.

[62] M. Ferretti, "The generalized Hough Transform on Mesh Connected Computers", *Journal of Parallel and Distributed Comput.*, vol. 19, pp.51-57, 1993.

[63] M. Albanesi, M. Ferretti, and D. Todorova, "Object Recognition on the CM2," in Progress in Image Analysis and Processing, S. Impedovo Ed., World Scientific (Singapore), pp. 713-720, 1994.

[64] Z.N. Li, B. Yao and F. Tong, "Linear generalized Hough transform and its parallelization," *Image Vision Comput.*, vol. 11, n. 1, pp. 11-24, 1993.

[65] S. Jeng and W. Tsai, "Fast generalized Hough transform," *Pattern Recogn. Letters*, vol. 11, pp. 725-733, 1990.