Pipelined Implementation of the Hough Transform in a Pyramid Multiprocessor

M. Atiquzzaman
Dept. of Computer Science
La Trobe University, Melbourne 3083, Australia.
Email: `atiq@LATCS1.lat.oz.au`

Abstract

The Hough transform is used to detect patterns in images. Efficient algorithms and their implementations in multiprocessor systems have been proposed to overcome the computational requirements. An efficient pipelined implementation of the transform in a pyramid machine has been proposed. The proposed implementation considers task scheduling to ensure uniform computational load among the processor elements in the pyramid. The Multiresolution Hough transform (MHT) has been selected for implementation in the pyramid machine.

Index terms: Pyramid architecture, Hough transform, Image processing, Multiresolution analysis, Pipelined processing.

1 Introduction

The Hough transform [1] is a powerful technique to detect patterns in images. The transform maps patterns from the image plane to a parameter space. The advantages of the transform are its robustness to noise in the image and discontinuities in the pattern to be detected. The disadvantage of the transform is its requirement for a large amount of computing power. The computational requirement of the transform has deterred its widespread use in real-time applications. *Efficient* Hough transform *algorithms* and implementations in *multiprocessor systems* are the common approaches to overcome the computational requirements. The approach of designing efficient algorithms has been used in the Multiresolution Hough transform (MHT) [2], the Adaptive Hough transform (AHT) [3], and the Fast Hough transform (FHT) [4]. Implementations of the transform in various types of multiprocessor systems have been described in the literature. Mapping of the transform to the mesh architecture has been described in [5, 6].

Mesh type architectures have been found to be very suitable for execution of image processing algorithms. Because of the large communication diameter of meshes, pyramid architectures, consisting of a stack of successively smaller-sized meshes, have been suggested. Pyramid architectures combine the features of mesh and tree type architectures, and are very suitable for implementation of vision algorithms for several reasons. Firstly, they are capable of efficiently storing several representations of an image at different spatial resolutions. Different objects in a given scene often have different resolutions at which they are most readily recognized by a computer. Secondly, a coarse-to-fine resolution search strategy can significantly reduce the processing time of an image. A considerable amount of research has been carried out in implementing computer vision algorithms on pyramid machines [7, 8, 9].

Implementations of the Hough transform in pyramid architectures have been proposed in [10, 11]. None of the implementations use all the layers of the pyramid for the voting and vote accumulation steps of the Hough transform . The voting and accumulation steps are computed in a single layer, and the other layers are used to accumulate or combine the results. Moreover, the implementations are meant for analysis of single images, and are not efficient for analysis of a continuous sequence of images. Certain tasks require a continuous sequence of images to be analyzed in real-time. For example, an industrial inspection system requires the real-time analysis of a continuous sequence of images.

The objective of this paper is to present an efficient implementation of the Hough transform on a pyramid machine. The proposed implementation is suitable for real-time

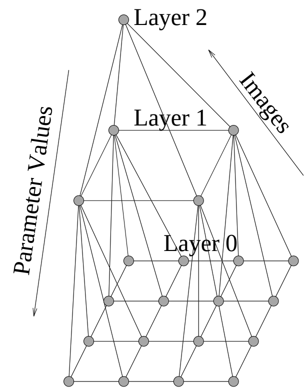

Figure 1: A typical pyramid computer for $r = 2$.

analysis of a continuous sequence of images, such as those found in typical industrial inspection systems. The efficiency of the implementation is due to the uniform distribution of voting and votes accumulation among the layers of the pyramid, resulting in a high utilization of the processors.

In the proposed implementation, a sequence of images are successively pumped into the base layer of the pyramid, and are reduced in spatial resolution as they move up the pyramid. The images are analyzed in a pipelined fashion by the PEs at the different layers of the pyramid, using accumulator arrays of different spatial resolutions. Use of multiresolution images and accumulator arrays reduce the total computational demands. Previously proposed implementations of the Hough transform are not efficient for analysis of a continuous sequence of images. The complexity of the proposed implementation is $O(\log n)$, with an image being analyzed at every pipeline cycle.

The rest of this paper is organized as follows. The pyramid architecture and the MHT are briefly described in Sections 2 and 3 respectively. The proposed implementation of the transform in a pyramid machine is presented in Sections 4 and 5, followed by conclusions in Section 7.

2 The Pyramid Machine

A pyramid machine (Figure 1) consists of a number of successively tapering layers of processors arrays, where each layer is a SIMD machine with its own control unit. For a pyramid with a base size of $\sqrt{n} \times \sqrt{n}$, there are a total of $\frac{1}{2}\log_r n$ layers. The i-th layer is labeled L_i, $0 \leq i \leq \frac{1}{2}\log_r n - 1$, where r^2 is the factor by which successively higher layers are reduced in size. The base layer is L_0 and successively higher layers are numbered in an increasing order. Except the topmost layer which has a single PE, a layer consists of a 4-connected two-dimensional array of PEs with their own local memories. A PE at layer l is also connected to r^2 children at layer $l-1$, $\frac{1}{2}\log_r n - 1 \leq l \leq 1$, and a parent at layer $l+1$, $\frac{1}{2}\log_r n - 2 \leq l \leq 0$. Figure 1 shows a pyramid with $r = 2$. The l-th layer has a size of $(\sqrt{n} \times \sqrt{n})/r^2$.

3 The Multiresolution Hough Transform

The MHT uses a set of multiresolution images and accumulator arrays which can be efficiently mapped into the variable-sized processor arrays at the different layers of the pyramid. Implementation of the transform in a pyramid machine will be described in Section 4. To familiarize the reader with the MHT, its salient features are described in this section

The MHT uses a set of variable resolution images constructed from an original high resolution gray level image, $h^0_{x,y}$. A set of L Gaussian smoothed [12, 13] reduced resolution gray level images denoted by $G = \{g^i_{x,y}, 0 \leq i \leq L-1, 0 \leq x, y \leq g^i_{\text{size}} - 1\}$ are generated from $h^0_{x,y}$, where $g^i_{\text{size}} \times g^i_{\text{size}}$ is the spatial resolution of the smoothed image after i successive reductions. Successive images are reduced in spatial resolution by a positive integer factor of σ. Therefore, $g^i_{\text{size}} = g^0_{\text{size}}/\sigma^i$. A set of multiresolution binary images, $F = \{f^i_{x,y}, 0 \le i \le L-1, 0 \le x, y \le f^i_{\text{size}} - 1\}$, corresponding to the set of images G, are then generated by applying thresholds to the images in G. Note that $f^i_{\text{size}} = g^i_{\text{size}}$. A set of L multiresolution Hough accumulator arrays will be represented by $A = \{a^i_{\rho,\theta}, 0 \le i \le L-1, 0 \le \rho, \theta \le a^i_{\text{size}} - 1\}$, where, $a^i_{\text{size}} \times a^i_{\text{size}}$ is the spatial resolution of the $i-$th accumulator array. Successive accumulator arrays also reduce by a factor of σ. Therefore, $a^i_{\text{size}} = a^0_{\text{size}}/\sigma^i$.

A straight line, represented by $\rho = x\cos\theta + y\sin\theta$, will be assumed as the pattern to be detected in the original image $h^0_{x,y}$. Hough transform is first applied to the smallest binary image $f^{L-1}_{x,y}$, and the votes are accumulated in the smallest accumulator array $a^{L-1}_{\rho,\theta}$. Full parameter ranges are used in the first iteration. Since the image and the accumulator array used in the first iteration are of low resolutions, the values of the $\rho-\theta$ parameters obtained are very rough approximations of the actual values. A second iteration of the Hough transform is applied to the image $f^{L-2}_{x,y}$ and the votes are accumulated in $a^{L-2}_{\rho,\theta}$. Prior knowledge of the approximate values of the parameters obtained from the first iteration, are used to zoom down into a narrow range of parameters in the second iteration. A smaller range of investigation of the parameter values, in addition to the larger accumulator array size, results in better approximations to the parameter values than those obtained at the first iteration. Similarly, approximate values of parameters obtained during the second iteration are used in the third iteration. In general, the approximate parameter values obtained at the $i-$th iteration are used in the $(i+1)-$th iteration for a total of L iterations.

The Hough transform is applied to $f^{L-i}_{x,y}$ and the votes are accumulated in $a^{L-i}_{\rho,\theta}$ in the i-th, $1 \le i \le L$, iteration. Estimated parameter values obtained from the i-th iteration are denoted by ρ^{L-i}_{est} and $\theta^{L-i}_{\text{est}}$.

The MHT uses an iterative coarse-to-fine search strategy. The ranges of investigation for ρ and θ are logarithmically narrowed down in successive iterations. The ranges of investigation for ρ and θ during the i-th iteration, $2 \le i \le L$, are based on $\rho^{L-i+1}_{\text{est}}$ and $\theta^{L-i+1}_{\text{est}}$ respectively.

4 Implementation in a Pyramid Machine

An efficient Hough transform algorithm has been described in the previous section. It consists of generating multiresolution binary images from a gray level base image, applying the Hough transform to the multiresolution images, and finding the peaks in the multiresolution accumulator arrays. The grey-level base image ($h^0_{x,y}$) of size $\sqrt{n} \times \sqrt{n}$ is loaded into the base layer of the pyramid, one pixel/PE. The image is successively pushed up the layers of the pyramid. Without loss of generalization, $r = 2$ will be used in the rest of this paper for ease of explanation.

In the steady state execution of the machine, image sequences enter the pyramid through the base layer and are pushed up the pyramid towards the peak. As the images travel towards the peak, they are reduced in size and converted to binary images by the Laplacian-of-Gaussian (LoG) [13] operator. The LoG operator is implemented in the pyramid by approximating it with the Difference-of-Gaussian (DoG) [12].

4.1 Generation of Multiresolution Images

Multiresolution images are generated by loading a grey-level image $h^0_{x,y}$ at the base layer of the pyramid, and applying Gaussian smoothing operators at the different layers of the pyramid as the image moves up towards the apex of the pyramid. To apply the Gaussian smoothing operator, $h^0_{x,y}$ is pushed up to layer 1 where a smoothing operator is applied to $h^0_{x,y}$, and subsequently reduced in size by a factor of two to form $g^1_{x,y}$ which is stored in the PEs at layer 1. Similarly, $g^1_{x,y}$ is pushed to layer 2 where the smoothing operator is applied to $g^1_{x,y}$ and then reduced in size to obtain $g^2_{x,y}$ at layer 2. In general, $g^l_{x,y}$ is obtained by applying the following Gaussian smoothing operator to $g^{l-1}_{x,y}$ at layer l of the pyramid.

$$g^l_{x,y} = \begin{cases} \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n) g^{l-1}_{2x+m,2y+n} & \text{for } 1 \leq l \leq L \\ f^0_{x,y} & \text{for } l = 0 \end{cases} \tag{1}$$

where, $w(m,n)$ is a 5×5 two-dimensional kernel approximating the Gaussian probability distribution.

4.2 Construction of Edge Maps

A set of multiresolution binary images, F, (containing edge map of the pattern to be detected) corresponding to the original grey-level image $h^0_{x,y}$ are generated at the different layers of the pyramid by searching for zero crossings in the multiresolution Laplacian images obtained by applying the Laplacian operator to the multiresolution Gaussian-smoothed images obtained in Section 4.1. The Laplacian operator can very conveniently be approximated in a pyramid computer by the DoG operator. The Laplacian images at layer l is obtained by subtracting an enlarged version of the Gaussian image at layer $l+1$ from the Gaussian image at layer l, as given below.

$$f^l_{x,y} = g^l_{x,y} - g^{l+1,1}_{x,y}, \quad 0 \leq l \leq L-1 \tag{2}$$

where, $g^{l+1,1}_{x,y}$ is an expanded version of $g^{l+1}_{x,y}$. It is obtained by expanding $g^{l+1}_{x,y}$ once using a 5×5 kernel as follows.

$$g^{l+1,1}_{x,y} = 4 \sum_{m=-2}^{2} \sum_{n=-2}^{2} w(m,n) g^{l+1}_{(x-m)/2,(y-n)/2}, \quad 0 \leq l \leq L-1 \tag{3}$$

The summation in equation (3) is carried out only for terms having $(x-m)/2$ and $(y-n)/2$ as integers.

Each PE at layer $l+1$ has an element of $g^{l+1}_{x,y}$ stored in its local memory. To generate $g^{l+1,1}_{x,y}$, each PE at layer $l+1$ shifts the element of $g^{l+1}_{x,y}$ (which is stored in its local memory) down to all its 25 children at layer l. The PEs at layer l can then calculate $f^l_{x,y}$ using equations (2) and (3).

Zero crossings in $f^l_{x,y}$ are determined by an 8-neighbor connectivity analysis [14]. To implement the zero crossing algorithm in a 2D mesh, each PE at layer l, $0 \leq l \leq L-1$, exchanges $f^l_{x,y}$ with its four siblings, followed by execution of the above zero crossing algorithm.

4.3 Pipelined Operation of the Pyramid

Once the multiresolution binary images have been generated and stored at the different layers of the pyramid as described in Sections 4.1 and 4.2, the Hough transform is successively applied to the binary images. The transform is first applied to the image at layer $L-1$ of the pyramid. Approximate ρ, θ values obtained by applying the Hough transform to $f^{L-1}_{x,y}$ are passed down to layer $L-2$. Parameter ranges of investigation are logarithmically reduced at the successive iterations in the different layers. The approximate parameter values obtained from layer $L-1$ are used to determine the limits of the reduced parameter range of investigation at layer $L-2$. In general, parameter values obtained at layer l, $1 \leq l \leq L-1$, are passed down to layer $l-1$. At the end of the iterations, the parameter values of the straight line under investigation are obtained from layer 0 of the pyramid. It is apparent that only one layer of the pyramid is active at any instant of time, resulting in a low processor utilization.

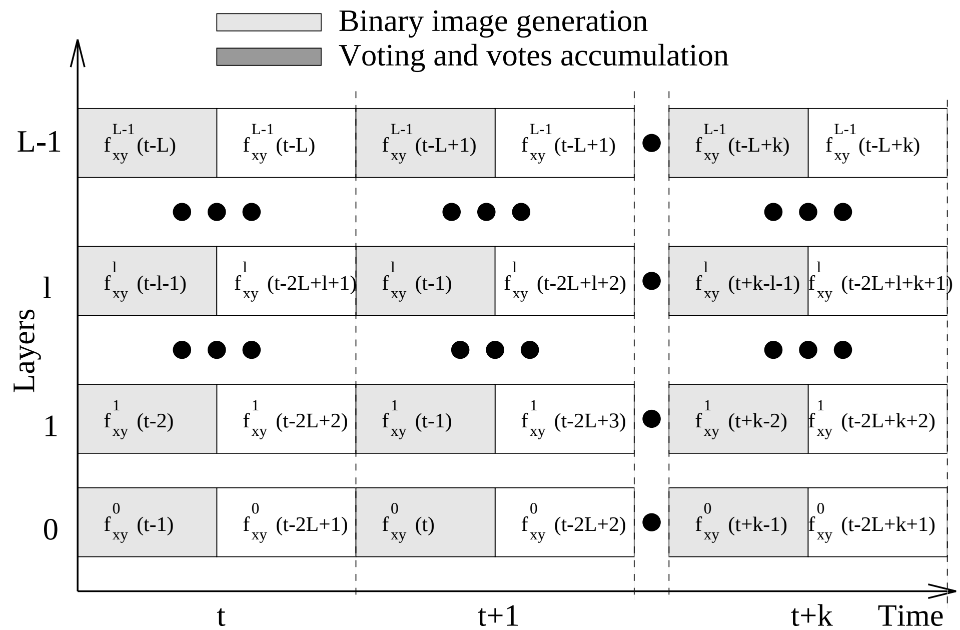

Figure 2: Gantt chart showing the activities at the layers of the pyramid during different pipeline cycles.

To increase the processor utilization, a pipelined operation of the machine is proposed below. Each layer in the pyramid acts as a stage of the pipeline. To distinguish between the images that enter the pyramid during different pipeline cycles, the image that enters the pyramid at the t-th cycle will be denoted by $h^0_{x,y}(t)$ $(= g^0_{x,y}(t))$, and its corresponding Gaussian and binary images at layer l at any instant of time by $g^l_{x,y}(t)$ and $f^l_{x,y}(t)$ respectively. In the steady state pipelined operation of the pyramid, the following tasks are carried out at every layer of the pyramid during the t-th pipeline cycle (Figure 2).

1. $h^0_{x,y}(t)$ is loaded into layer 0 of the pyramid at the beginning of the t-th cycle. While $h^0_{x,y}$ is being loaded into layer 0, the image $h^0_{x,y}(t-1)$ which was loaded into layer 0 at the beginning of the $(t-1)$-th pipeline cycle is pushed up the pyramid to layer 1. Layer 1 in turn pushes $g^1_{x,y}(t-2)$ up the pyramid into layer 2 at the beginning of the t-th cycle. In general, $g^{l-1}_{x,y}(t-l)$ is pushed from layer $l-1$ into layer l at the beginning of the t-th cycle.

2. Layer l, $1 \le l \le L$, after receiving $g^{l-1}_{x,y}(t-l)$ at the beginning of the cycle (in step 1 above) applies a Gaussian smoothing operator to generate $g^l_{x,y}(t-l)$ according to equation (1). For layer 0, $h^l_{x,y}(t)$ is taken as $g^l_{x,y}(t)$.

3. Layer l, $0 \le l \le L-1$, receives $g^{l+1}_{x,y}(t-l-1)$ from layer $l+1$, and then computes the Laplacian image $f^l_{x,y}(t-l-1)$ according to equations (2) and (3).

4. Layer l, $0 \le l \le L-1$, convert $f^l_{x,y}(t-l-1)$ to a binary image by locating the zero crossings.

5. It takes L pipeline cycles for an image to reach the top of the pyramid. Since, the analysis starts with the smallest resolution image, layer $L-1$ applies the Hough transform to $f^{L-1}_{x,y}(t-L)$ during the t-th pipeline cycle. At the same time, layer $L-2$ applies the transform to $f^{L-2}_{x,y}(t-L-1)$ using the approximate values of the parameters for $h^{L-1}_{x,y}(t-L-1)$, the values being obtained from layer $L-1$

during the $(t-1)$-th cycle. In general, layer l, $0 \leq l \leq L-1$, applies the Hough transform to $f^l_{x,y}(t-2L+l+1)$ during the t-th cycle. This step is explained in detail in Section 5.

6. Approximate values of parameters obtained by applying the Hough transform at layer l, $1 \leq l \leq L-1$, during the t-th cycle are sent to layer $l-1$ at the end of the t-th cycle. Values of parameters obtained at layer 0 during the t-th cycle are output from the machine as final parameters of the straight line to be detected. This results in a total pipeline delay of $(L-1)+(L-1)+1 = 2L-1$ cycles.

A Gantt chart showing steps 4 and 5 over a number of pipeline cycles for all the layers is given in Figure 2. The overall operation of the pyramid can be visualized as images traveling up the pyramid while the approximate parameter values are moving down the pyramid in successive pipeline cycles. The images get reduced in size as they travel up the pyramid, while the approximate parameter values get closer to the actual values as they travel down the pyramid.

5 Hough Transform in the Pyramid Layers

It has been explained in Section 4.3 (step 5) that, during a cycle, layer 0 applies the Hough transform to an image which entered the pyramid $2L-1$ cycles before. This requires layer 0 to store copies of the recent $2L-1$ images at any instant of time. Similarly, it can be shown that layer $L-1$ needs to store only one image at any instant of time. In general, layer l, $0 \leq l \leq L-1$, will need to store $2L-2l-1$ binary images in addition to an accumulator array at any instant of time. Because of the one-to-one mapping of the elements of the accumulator array to the PEs in a layer, a column of PEs at layer l stores a column of $a^l_{\rho,\theta}$. During the voting process, the control units of the layers successively broadcast sets of θ_i, $\cos\theta_i$, and $\sin\theta_i$ values to the PEs in their layers. PEs containing feature points (non-zero pixels) compute and quantize the ρ values for their feature points. The quantized ρ values in the PEs of a layer are shifted horizontally to the column of PEs (in that layer) which is responsible for accumulating the votes corresponding to the broadcasted θ_i. The PEs in the column which are responsible for accumulating the votes, shift the ρ values vertically among the different PEs in the column. During the shifting process, a PE picks up the votes corresponding to the ρ value for which it is responsible, and increments its bin counter by the number of votes it has picked up.

The task following the voting and accumulation processes is the determination of the peaks in the accumulator arrays at the different layers of the pyramid. The elements of the accumulator arrays at the different layers are successively pushed up the pyramid towards the apex. A PE computes the maximum of the four accumulator array elements received from its children. A PE at layer l successively receives from its children, votes originating at layers 0 to $l-1$. To maintain the identity of the votes, all votes are tagged with their originating layer number. The determination of the peaks of the accumulator arrays at the different layers require a maximum of $O(\log n)$ time. Since, the vote accumulation takes a constant amount of time, the overall complexity of the proposed implementation is $O(\log n)$.

6 Load Scheduling in the Pyramid

The throughput of a pipelined system depends on the time taken by the slowest stage of the pipeline. This gives rise to the issue of scheduling the computational load among the stages of the pipeline. In a balanced system, all the stages require the same amount of time. In our case, we will look at two issues in load scheduling.

- Scheduling the computational load among the layers of the pyramid.
- Scheduling the computational load among the different PEs in a layer.

In the implementation of the Hough transform described in earlier sections, the image in a layer was partitioned with one pixel per PE. If the pixel assigned to a PE is a feature point, the PE computes all the votes for that feature point. If the pixel is not a feature point, the PE remains idle during the voting process. As a result of the above image partitioning scheme, the pyramid suffers from the following load unbalances.

1. During a pipeline cycle, PEs without feature points remain idle during the voting process. This results in a low utilization of the PEs in a layer.

2. The computational load of the layers of the pyramid during a pipeline cycle are not uniform. The computational load to compute the Hough transform in layer l is proportional to a^l_{size} and the number of feature points in $f^l_{x,y}$ The increase in the number of feature points by a factor of four at successively lower layers (for $r = 2$) due to the use of images of increasing sizes, is balanced by a corresponding increase in the number of PEs at successively lower layers of the pyramid. However, because of the doubling of a^l_{size} at successively lower layers, the computational load due to voting and accumulation at successively lower layers increase by a factor of two.

The first unbalance can be overcome by having a partitioning scheme where the size of layer 0 (and other layers) of the pyramid is much smaller than the image it has to process. Each PE in a layer stores and processes a subimage. The feature points in a layer are then redistributed among the PEs of the layer, such that they have approximately the same number of feature points. The high PE utilization resulting from the redistribution strategy far offsets the small time required to redistribute the feature points among the PEs in a layer. The solution to the second problem is to keep the layers of the pyramid uniformly loaded. This can be done by keeping the accumulator array sizes constant at all the layers. However, the parameter-range reduction between successive layers is maintained logarithmic in order to achieve fast convergence. Details of this method can be found in [2].

7 Conclusions

A pipelined implementation of the Hough transform in a pyramid machine has been presented. The layers of the pyramid serve as the stages in the pipeline. Images enter the pyramid through the base layer, and then move up the machine. The images are reduced in size and converted to binary images as they travel up the pyramid.

The transform is implemented as a coarse-to-fine search of the pattern. The processing of an image starts at the top layer of the machine, and successively proceeds downwards towards the base. A continuous sequence of images, one per pipeline cycle, are pumped into the base layer of the machine. Parameters of the patterns in an image are obtained from the base layer at the end of a pipeline cycle. It should be noted that during a pipeline cycle, the layers of the pyramid are concurrently analyzing images (of different spatial resolutions) which entered the machine at different pipeline cycles. The pipeline delay between the input of an image and the output of its results is $2L - 1$ cycles. There is a bidirectional data movement in the machine – images moving up and approximate parameter values flowing down the pyramid,

The following features of the proposed implementation distinguishes it from previous implementations of the transform in pyramid machines.

- All the layers of the pyramid are involved in the computation of the Hough transform. Due to this uniform distribution of the computational load among the layers, the machine has a high processor utilization. The computational load of the processors in a layer are also equal. Previous implementations employ only one layer of the pyramid for computation of the transform, thereby resulting in an unbalanced computational load among the processors in the machine.

- Due to a pipelined operation of the machine, a continuous sequence of images can be analyzed. Typical industrial vision systems often require the analysis of a continuous sequence of images. The previous implementations analyze single images, and are not suitable for analyzing a continuous sequence of images.

The complexity of the proposed implementation is $O(\log n)$. Though the complexity is the same as the best previously published implementation [11], the throughput of the machine with the proposed implementation will be significantly higher than the implementation presented in [11] because of higher processor utilization as a result of the pipelined operation of the machine.

Synthetic and real-world images were tested on a pyramid simulator executing the implementation described in this paper. The accuracy of detection and the rate of convergence in the coarse-to-fine analysis were very high. Performance figures and the multiresolution Laplacian images at the different layers for $n = 512 \times 512$ and $L = 4$ can be found in [2]. Algorithms to detect the end points of a line [15] in a pyramid machine is underway.

References

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

[2] M. Atiquzzaman, "Multiresolution Hough transform – an efficient method of detecting pattern in images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 11, pp. 1090–1095, November 1992.

[3] J. Illingworth and J. Kittler, "Adaptive Hough transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 5, pp. 690–698, September 1987.

[4] H. Li, M.A. Lavin, and R.J. LeMaster, "Fast Hough transform," *Computer Vision, Graphics and Image Processing*, vol. 36, pp. 139–61, 1986.

[5] C. Guerra and S. Hambrusch, "Parallel algorithms for line detection on a mesh," *Journal of Parallel and Distributed Computing*, vol. 6, pp. 1–19, 1989.

[6] C.S. Kannan and H.Y.H. Chuang, "Fast Hough transform on a mesh connected processor array," *Information Processing Letters*, vol. 33, pp. 243–248, January 10, 1990.

[7] V. Cantoni and S. Levialdi, eds., *Pyramidal Systems for Computer Vision.* Berlin: Springer-Verlag, 1986.

[8] A. Rosenfeld, ed., *Multiresolution Image Processing and Analysis.* Heidelberg: Springer Verlag, 1984.

[9] A. Choudhary and S. Ranka, "Parallel processing for computer vision and image understanding," *Computer*, vol. 25, no. 2, pp. 7–10, February 1992.

[10] G. Bongiovanni, C. Guerra, and S. Levialdi, "Computing the Hough transform on a pyramid architecture," *Machine Vision and Applications*, vol. 3, pp. 117–123, 1990.

[11] J.M. Jolion and A. Rosenfeld, "An $O(\log n)$ pyramid Hough transform," *Pattern Recognition Letters*, vol. 9, pp. 343–349, June 1989.

[12] P.J. Burt and E.H. Adelson, "The Laplacian pyramid as a compact image code," *IEEE Transactions on Communications*, vol. COM-31, no. 4, pp. 532–540, April 1983.

[13] G.E. Sotak and K.L. Boyer, "The Laplacian-of-Gaussian kernel: a formal analysis and design procedure for fast, accurate convolution and full-frame output," *Computer Vision, Graphics and Image Processing*, vol. 48, pp. 147–189, 1989.

[14] M. Atiquzzaman, "Implementation of the multiresolution Hough transform on a pyramid machine," Tech. Rep. 8/92, La Trobe University, Department of Computer Science, Melbourne, Australia, June 1992.

[15] M.W. Akhtar and M. Atiquzzaman, "Determination of line length using Hough transform," *Electronics Letters*, vol. 28, no. 1, pp. 94–96, January 2, 1992.