SPECIALIZED PARALLEL STRUCTURES FOR VLSI IMPLEMENTATION OF THE HOUGH TRANSFORM FOR ARBITRARY SHAPE DETECTION

By

OK SAM CHAE

Bachelor of Science
Inha University
Incheon, Korea
1977

Master of Science
Oklahoma State University
Stillwater, Oklahoma
1982

Submitted to the Faculty of the Graduate College
of the Oklahoma State University
in partial fulfillment of the requirements
for the Degree of
DOCTOR OF PHILOSOPHY
July, 1986

SPECIALIZED PARALLEL STRUCTURES FOR VLSI IMPLEMENTATION OF THE HOUGH TRANSFORM FOR ARBITRARY SHAPE DETECTION

Thesis Approved:

Louis G. Johnson
Thesis Adviser

J P Chandler

D J Soldan

[illegible]

Norman N. Durham
Dean of the Graduate College

ACKNOWLEDGMENTS

I would like to express my sincere appreciation to my thesis adviser, Louis Johnson, for his guidance and friendship throughout my program. I also greatly appreciate the time and interest offered by my doctoral committee members, Dr. Rao Yarlagadda, Dr. John Chandler, and Dr. David Soldan.

I would like to express my special gratitude to my wife and parents for their encouragement, support, and sacrifices which have been a source of motivation.

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Motivation

With the rapid progress of computer technology, inexpensive and powerful industrial computer vision systems are coming into the market every year. Many of these vision systems are equipped with a coprocessor with multiple processors. The coprocessors in these systems can compute low level computer vision algorithms, such as Sobel edge detection, thinning, and thresholding in real time. However, the capability of the coprocessor is limited to a small number of low level processing algorithms dealing with local information.

The algorithms like edge smoothing, curve fitting, classification, and segmentation are sequential in nature and can not take advantage of the coprocessors in current computer vision systems. Algorithms of this nature are processed in the host computer. The slow speed of such algorithms in a sequential host computer is not practical for many useful applications such as bin picking, inspection, identification, etc.

A high speed host computer can increase the processing speed somewhat, but the high cost of the system will limit its use in industrial vision systems. One way of increasing the speed of the systems is by making use of multiple special purpose functional units. In general, a multiple special purpose functional unit is designed for one or a few algorithms with near 100 percent hardware efficiency; it can also compute a given algorithm at lightning speed.

Advanced VLSI technology makes simple and low cost special purpose systems possible for many computer vision algorithms. If an algorithm is general enough so that it can be used for a large number of application areas, the special purpose system for the algorithm will be very valuable for industrial computer vision systems.

One such algorithm is the Hough transform, specifically the Hough transform designed for arbitrary shape detection. The Hough transform for arbitrary shape detection is a model based algorithm which can detect the boundary of known objects in the input image. This type of model based algorithm is especially useful for industrial applications where objects to appear in the scene are known beforehand. Many other model based algorithms have been developed for industrial parts recognition [1]-[3], but they have not been popular due to their slow processing speed. Those algorithms deal with high level descriptions of boundary edges obtained through a series of operations including edge detection, edge thinning, edge segmentation, curve fitting, etc. Most of these operations are computed sequentially.

Although the Hough transform requires a large amount of computation as do other model based algorithms, it has two distinctive advantages over them. First, the Hough transform makes use of raw edge data produced by thresholding the enhanced image by an edge operator (Sobel or gradient). In the Hough transform, a shape is represented by a set of parameters. For example, a circle is represented by the center and radius. The Hough transform transforms raw edge data directly into the parameter space. The problem of finding objects is then reduced to finding maxima in the parameter space. In other words, the Hough transform produces necessary information for object detection directly from the raw edge data produced by coprocessors without going through other expensive processing steps. Second, the Hough transform

has a very suitable structure for parallel implementation. It consists of a few simple operations for each input edge pixel so that it can be implemented in a parallel machine which consists of a large number of simple processing elements.

Though the Hough transform consists of a few simple operations, it has two characteristics which make the algorithm less effective for parallel implementation. First, the Hough transform is an I/O bound problem: the number of I/O operations is larger than the number of computations in Hough transform computation. Second, the Hough transform is a global algorithm where the operands for basic operations are arbitrarily distributed in the memory. In general, the performance of a global and I/O bound algorithm in highly parallel structures is limited by the I/O bandwidth of the system. Such an algorithm is less effective for VLSI implementation where the number of I/O operations is the major factor determining the performance of the system.

The main objective of this research is investigating efficient VLSI implementation of the Hough transform, specifically the Hough transform for arbitrary shape detection. To design fast,inexpensive,and compact special purpose functional units for industrial applications (partially hidden object detection, bin picking, and robotics control), the following studies have been conducted. First, existing arbitrary shape detection algorithms will be analyzed in terms of their performance in detecting industrial parts in a controlled environment and their efficiency in a parallel machine. Based on this analysis, an algorithm will be selected for parallel implementation. Second, problems involved in the VLSI implementation of the selected Hough transform will be identified and the Hough transform will be reformulated for VLSI implementation. Third, two VLSI structures will be designed specifically for the reformulated algorithm and analyzed in terms of their performance and ease of

implementation. Functional simulation of these structures for this algorithm will also be given. Finally, an algorithm will be proposed for detecting objects in an arbitrary orientation by taking advantage of these structures. Some test results of this algorithm for industrial scenes will be discussed.

1.2 Hough Transform for Analytic Shape Detection

The Hough transform was first introduced by Hough [4] in 1962. It replaces the original problem of finding colinear points in an image space with the simpler task of finding intersecting lines in a parameter space. Hough uses the slope-intercept parameters to represent a line. Thus, the line passing through three image points in Figure 1a is represented by

$$y = ax + b \tag{1.1}$$

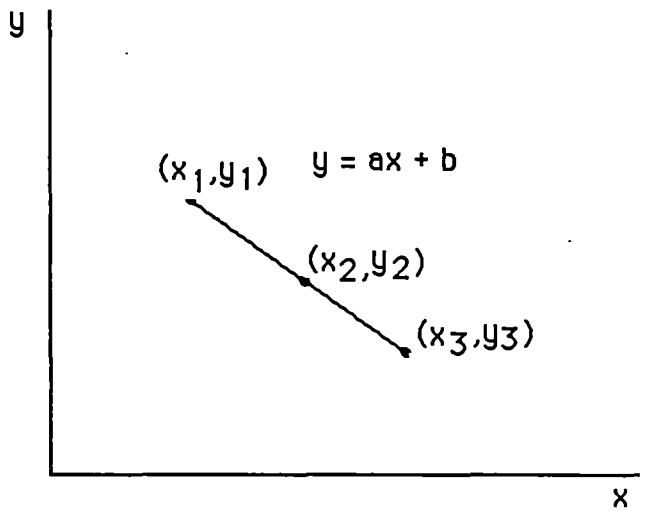
Hough suggests the following method to determine a set of parameters representing a line in the image space. First, rewrite the equation (1.1) as

$$b = -xa + y \tag{1.2}$$

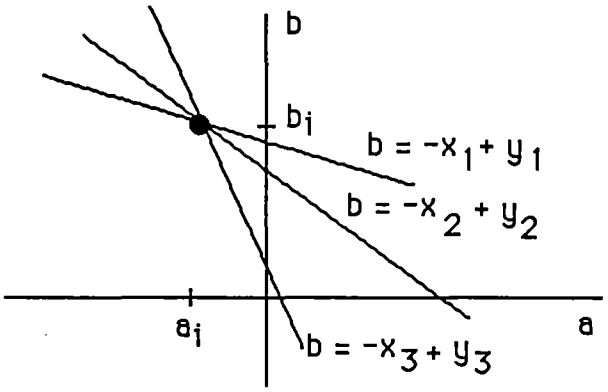
Second, draw a line in the parameter space by using the equation (1.2) for every image point on the line. This procedure is illustrated in Figure 1b for the line consisting of three image points. Third, find the point where all lines meet together in the parameter space. The location of the intersecting point in the parameter space determines the line. For example, if the peak is located at (a_i,b_i) in the parameter space, the equation characterizing the line in the image space is

$$y = a_i x + b_i$$

To find a line in an actual image, a gray-scale image is transformed into an edge representation using edge operators, such as Sobel, Hueckel, and gradient operator. The set of extracted edge points is usually referred to as "features" and an edge point is called the "feature point". All edge points in the



a) Three Edge Points in the Image Space.



b) Three Lines in the Parameter Space

Figure 1. Illustration of the Hough Transform

image space (or the picture space) are transformed into points in the parameter space according to the equation (1.2). The parameter space is represented by a two dimensional accumulator array in the digital computer. Each element of the array represents a parameter point and holds the number of lines passing through it. The parameters characterizing a line are determined by searching the maximum value in the accumulator. If the edge pixels in an image are on a line, all lines drawn for these edge pixels should intersect at one point in the parameter space. However, some lines may not due to noise in the image. Thus, the location of the accumulator cell containing the maximum value will characterize the line with the best match.

Since its introduction, the Hough transform has attracted a lot of attention from many researchers. It has been modified and extended to detect not only analytic but also nonanalytic shapes. Duda and Hart [5] modified the algorithm by Hough to eliminate the problem of sensitivity to the choice of coordinate axes in the picture plane. In the slope-intersect representation used by Hough, both the slope and intercept are unbounded. They can be arbitrarily large for the line approximately parallel to the y axis, so that the parameter space can become arbitrarily large. To avoid this problem, Duda and Hart represent a line by the angle Q of its normal and its algebraic distance P from the origin as shown in Figure 2. Thus, a line passing through an image point (x_i,y_i) is defined by a sinusoidal curve in the parameter space.

$$P=X_i \cos Q + Y_i \sin Q$$

The curves corresponding to colinear points intersect each other at one point.

In the computation of the algorithm, a feature point is transformed into the parameter space by incrementing all the accumulator cells on the curve for the feature point. If P and Q are quantized with N steps and the size of the image is N by N, the number of computations necessary for the Hough transform is

proportional to CN^3, where CN^2 is the number of edge pixels in the input image. By quantizing P and Q using a small number of steps, some speedup can be achieved, but the performance of the algorithm is sacrificed for the speedup.

The work of Duda and Hart is further extended by O'Gorman and Clows [6]. They use the direction of the gradient to determine the value of Q that is most likely to be the intersecting point in the parameter space. The direction of the local gradient is perpendicular to the picture edge. Thus, if the gradient direction of an edge point is G, the edge point (X_i,Y_i) is transformed into a point (P,G) in the Q-P plane. The relationship between P and G is given by

$$P = X_i \cos G + Y_i \sin G$$

The number of computations for this algorithm is proportional to N^2 unless only edge pixels are fed into the system in which case the amount of computations is proportional to the number of edge pixels. However, feeding only edge points and their gradient direction to the system requires a significant amount of overhead to convert an input image into a list of edge elements and is usually not justifiable.

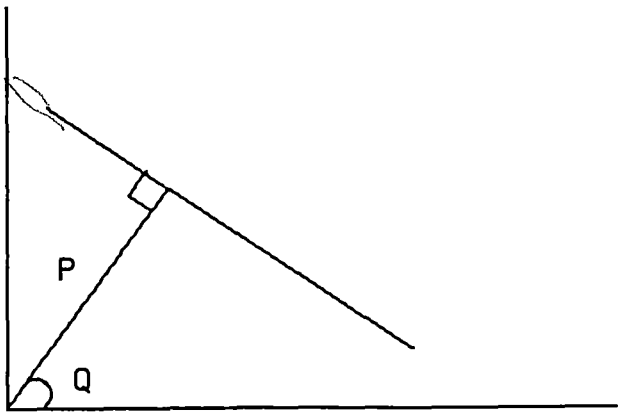


Figure 2. The Normal Parameters for a Line

Duda and Hart extend the Hough transform to circle detection by expressing a circle with the equation:

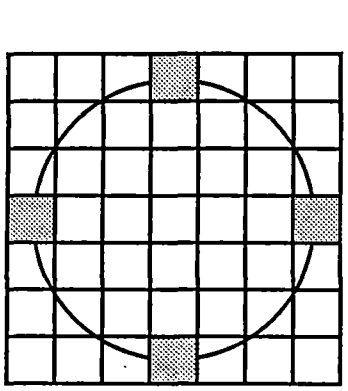
$$r^2 = (x-a)^2 + (y-b)^2$$

where r is the radius and (a,b) is the center of the circle. For a feature point (x_i,y_i), there is a set C_x of circles passing through this feature point. These circles are defined by parameters a, b, and r satisfying the following equation:

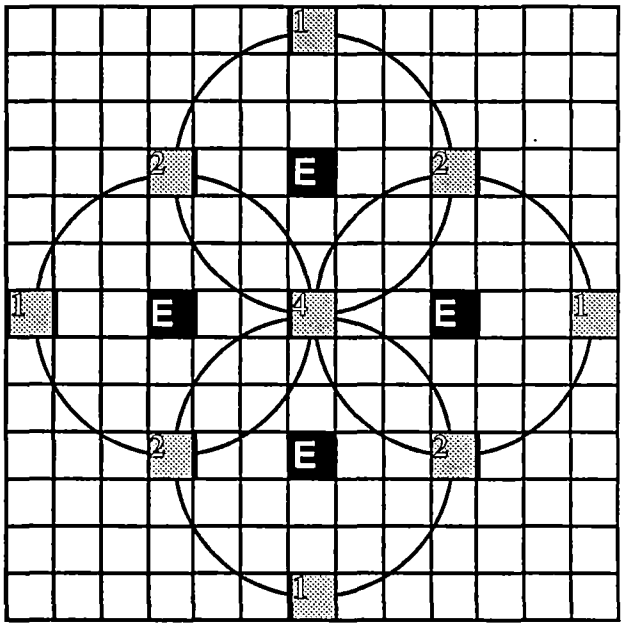
$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

Kimme and Ballard [7] extend this idea for approximate circle and circular detection. They achieved significant savings in computations by introducing two new ideas to the work of Duda and Hart. First, they generate the locus of possible centers, C_x for each feature point by computing the digitization of (rcos θ, rsin θ) for a fixed r, while q increases in increments $\Delta\theta$. Using this quantization method the number of computations for each feature point is reduced to $2\pi/\Delta\theta$. Figure 3b shows a locus of circles of the member of C_x for $r=r_0$ and $\Delta\theta=90^\circ$. To reduce the number of computations further, they use the direction of the gradient to select those portions of C_x that are likely to be circle centers. In the ideal case, the gradient at the boundary of the circle points to the center of the circle within a range of angle $\Delta\theta$. Theoretically, the number of computations can be reduced by a factor of $2\pi/\Delta\theta$. Figure 3c shows the effect of using the directions of the gradient. However, the quantization error and the error in the gradient direction will affect the result of this procedure.

The Hough transform for analytic shape detection has been studied by many other researchers including Ballard [8] and Shapiro [9]. Ballard generalizes the Hough transform for both analytic and nonanalytic shape detection as will be discussed in a later section of this thesis. Shapiro studies extensively the performance of the Hough transform for the detection of curves in noisy images. The analytic work directed toward the understanding of Hough

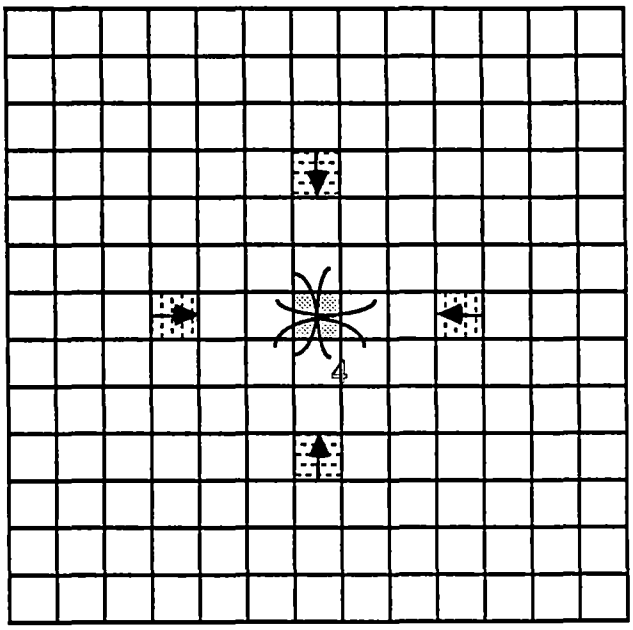


a) Model



**Dark boxes indicates location of edge points.*

b) Contents of an accumulator with no gradient direction information



c) Contents of an accumulator with gradient direction. Angle increment = 90.

Figure 3. Contents of Accumulator Array

transform has been pursued by Shapiro [10]-[12], Brown [13] [14], Thrift [15], and Stockman [16]. Shapiro and Brown study the effect of various forms of noise on the behavior of the Hough transform. Stockman gives the relationship between the Hough transform and the template matching method and shows that the Hough transform is an efficient form of the template matching method. Thrieft also shows that the Hough transform is optimal in the mean-square-error sense when the picture is the sum of the signal and statistically independent noise. To reduce the chance of finding false peaks in the parameter space, Brown [17] proposes the complementary Hough transfrom. This algorithm uses negative votes against parameters with which the feature is inconsistent, as well as positive votes for parameters with which the feature is consistent.

The Hough transform for analytic shape detection has been used for many different application areas including industrial part inspection, biomedical data processing, and target detection. Perkins and Binford [18] employ the line detection algorithm using gradient direction information to find corners on indoor scenes. Cowart [19] and Falconer [20] use the line detection algorithm to detect moving targets. Falconer computes the Hough transform of each frame of video imagery and adds successive transforms together, while Cowart computes the Hough transform of the generated image by third order frame-to-frame differencing. Since moving targets, in time, produce a cluster in the parameter space, the target can be detected by finding the cluster. Dyer [21] makes use of the Hough transform to inspect the scaling accuracy of needle-type instrument gauges. By computing the Hough transform of the difference image between the image without signal and the image with signal, the relative angular displacements of the needles are determined to measure the scaling accuracy of the gauge.

One of the popular application areas of the circular segment detection

algorithms is radiography. Kimme [7] uses a circle detection technique to detect tumors in the chest radiograghs and Wechsler [22] develops a method based on parabolic segments to detect ribs in the chest radiographs. Another application of the circle detection transform method is given by Bastian [23] for the recognition of bubble chamber photographs.

1.3 Hough Transform for Nonanalytic Shape Detection

Based on the fact that the Hough transform is an efficient form of template matching, Merlin and Farber [24] propose the nonanalytic shape detection algorithm. The algorithm consists of four steps. First, choose an arbitrary point A on the curve as a reference point. Second, rotate the given curve 180° with respect to the reference point. Third, trace the rotated curve with A on each of the edge pixels and add the value of the image point to all accumulator cells on the trace. Fourth, find the maxima in the accumulator.

This method can be interpreted as a form of the binary convolution of the shape template T(x), where edge pixels are unity and others are zero with the corresponding image E(x).

$$A(x) = T(x)*E(x)$$

The transformation of an N by N image to find an object boundary consisting of T points requires TN^2 operations. This algorithm has a structure suitable for a highly parallel computer structure. However, there are many unsolved problems when trying to implement the algorithm for industrial applications which require inexpensive and fast systems. These problems and their solutions will be the main subject of this thesis.

One drawback of the Merlin-Farber technique is that more than one instance of the desired shape may be detected in a noisy image due to coincidental pixel arrangements. We will study this problem in chapter II to

investigate the usefulness of the algorithm.

The idea of Merlin and Farber is generalized by Ballard [8] by using the gradient direction. Instead of increasing all accumulator cells on the boundary of the object with the reference point at an edge point, Ballard uses the gradient direction at an edge point to reduce the number of cells to be increased. In this algorithm the model is represented in a table which usually called an *R table*. Detailed description of both algorithms will be given in chapter II. The performance of these algorithms on finding industrial parts will be also discussed in chapter II.

The Hough transform for nonanalytic shape detection has been applied to some industrial part recognition problems. Cantoni [25] uses the generalized Hough transform similar to Ballard's algorithm to recognize some mechanical parts. He chooses several 2D parameter spaces, one for each subsection of the model being sought, instead of a 3D parameter space. Since the reference point of an arbitrary shape is unchanged regardless of its orientation, the presence and location of the shape can be determined if all edge elements are transformed into a 2D parameter space for all possible orientations. To determine the orientation of the shape, a model is divided into subsections. The transform is performed for each subsection and the orientation of the shape is determined by examining relative positions of subsections. This technique will be applied to Merlin's algorithm to determine the orientation of some industrial parts in a controlled environment. This orientation determination algorithm can take advantage of parallel architectures to be discussed in chapter IV. Turney [26] and Koch [3] use the Hough transform in a different manner to identify partially hidden industrial parts. They, first, extract the boundary segments of objects in an image and represent them in slope angle-arclength space. The model consists of many distinctive segments of the object boundary. Each

model segment is matched with the boundary segments represented in the slope angle-arclength space to determined the possible orientation of the object and then the Hough transform is used to collect the evidences of the match. Here, the Hough transform is used merely to keep track of the sequential edge segment matching operation. Similar cases can be found in other model based algorithms [1] [2].

1.4. Hardware Implementation of the Hough Transform

In the previous section, we examined many different types of Hough transforms and their applications. It is clear that the Hough transform can be used for various applications. But, one major problem of this algorithm is its large computational requirements. Although many clever ideas have been suggested for computation reduction, none of them actually solved this problem. One approach to the solution is making use of highly parallel computer structures. In fact, many researchers have mentioned the characteristics of the algorithm which can take advantage of parallel machines. Some of them suggest actual hardware implementation of the algorithm.

Stockman [16] suggests the hardwired line detection systems where an input photoelectric cell representing one input pixel is connected to all possible accumulator cells. At each accumulator cell, all the inputs are summed, compared with a given threshold, and generate a binary output indicating "found" or "not found". This system may be realizable for a small image but the complexity of the system will grow exponentially when the size of the input image is larger.

Merlin and Farber [24] indicate that their algorithm can be efficiently implemented in the parallel picture processing machine proposed by Kruse [27]. The parallel picture processing machine by Kruse consists of nine processing

elements and performs local operations of both logical and arithmetical character on three by three neighborhoods of digitized pictures. It can perform the Hough transform by adding nine image points from nine different translated images at once. However, the way of generating translated images and retrieving results from the system, which are major problems in computing the Hough transform in a parallel machine, are not addressed. Other parallel processors for the Hough transform were briefly mentioned in [28] and [29]. But no one has come up with the systems that can compute the Hough transform, specifically the nonanalytic curve detection algorithm, in real time or close to real time. In this thesis, we will develope compact, inexpensive, and fast parallel structures for the Hough transform for nonanalytic shape detection.

1.5 Considerations for VLSI Implementation of the Hough Transform

The rapid progress of VLSI technology has created an NMOS chip containing more than 10^6 transistors and the number of transistors in a chip is expected to grow up to 10^7 by late eighties [33]. One such chip may contain more functions than one of today's large minicomputers. Recently developed automated IC design tools allow computer designers to easily access this powerful technology. The VLSI electronics enables not only those involved in development of fabrication technology but also computer architects to participate in the specification and design of actual circuits. Thus, for the given algorithm, the circuits can be optimized by designing architectures suitable for actual fabrication. It means that the VLSI technology gives computer designers more freedom from the cost and size constraint for the processing logic and opens a new architectural horizon in implementing parallel algorithms directly in hardware. Now, computer researchers are using VLSI technology not only to

design building blocks of large flexible parallel computers but also to design multiple special purpose functional units. The designer can attack problems that once were computationally intractable by implementing systems in which thousands or even tens of thousands of processors cooperate to solve a single problem. Many researchers in both industry and universities are investigating special purpose high performance multiprocessors and pipelined computing devices for computationally intensive basic mathematical operations such as matrix multiplication [30], convolution [31], and discrete Fourier transforms [32]. Other important operations for which multiple special purpose functional units were investigated are low level vision algorithms including edge detection, feature extraction, filtering, etc.

The special purpose functional units are usually designed for a few algorithms. To justify their limited applicability, special purpose functional units must be designed for near 100 percent hardware efficiency, require little or no software to perform given tasks, and require relatively small design cost. Particularly, the design cost must be reduced as much as possible because the design cost is the dominating factor in the total cost of the low volume production systems. To achieve these goals, most special purpose systems are designed under the following guidelines [31], [33], [34]: (1) choose an appropriate architecture which can be decomposed into a few building blocks to be used repetitively with simple interfaces; (2) choose an algorithm that supports high degrees of concurrency; (3) employ only simple, regular communication and control to allow efficient implementation; (4) choose algorithms which can be implemented in a VLSI device with limited I/O pins.

The most difficult problem in designing a special purpose VLSI device is the I/O problems. A special purpose VLSI device can contain thousands or more processing elements (PE) and handle a large amount of computations in a

short time. With limited number of I/O pins, the VLSI device is not suitable for the problems requiring a large I/O bandwidth. For such problems, the VLSI device can not balance its computation with the I/O bandwidth. Thus, VLSI structures are suitable for implementing compute-bound algorithms rather than I/O-bound problems. Even if a given algorithm is compute-bound, the I/O problem can not be avoided in some cases. In these cases the computation must be decomposed into subcomputations. Solving a large problem by decomposing it into subcomputations may also require a substantial amount of I/O to store or retrieve intermediate results.

The Hough transform for arbitrary shape may be a good choice for parallel implementation in the sense that it consists of a few simple operations and supports a high degree of concurrency. But it is not only a global algorithm, where an element in the output image depends only on the corresponding element in the input image, but also an I/O-bound algorithm. The performance of such an algorithm in the parallel machine is limited by the number of I/O operations necessary to compute it. Thus, if we implement this type of algorithms in VLSI devices, the performance of the system will be limited by the number of I/O pins available on a chip. This may be a factor contributing to the scare amount of work on the parallel implementation of the Hough transform.

When we study the performance of an algorithm in different parallel machines, it is essential to have good performance measures for the parallel algorithm. Some excellent performance measures for parallel algorithms are discussed by Siegel [35] and and Parkinson [36]. To evaluate parallel structures discussed in the thesis, some performance measures given by Siegel are listed next.

Execution time: The execution time $T_E(N)$ is the time spent to compute an algorithm for an N by N input data set on the parallel system with E PEs and

represented by the equation: $T_E(N) = P_E(N) + O_E(N)$, where $P_E(N)$ is the time spent performing computations which are actually a part of the task, and $O_E(N)$ is the overhead, or time required to manage the parallelism. Two sources of overhead are inter-PE data transfers and masking operations to enable and disable PEs.

Speed: The speed $V_E(N) = N^2 / T_E(N)$ is the number of data points processed per unit time.

Speed up: $S_E(N) = T_1(N) / T_E(N)$, where $T_1(N)$ is the time required to compute the given algorithm on a sequential computer.

The above three performance measures: speed, speed up, and utilization, may be important measures for general purpose parallel machines. But they may be less important for the special purpose functional units designed for a single algorithm. More important measures for this type of systems will be the speed and cost of the system. As long as the system can process a given algorithm at the speed required for most applications, simplicity, cost, and size of the system will be the three major considerations to distinguish one system from another.

The execution time $T_E(N)$ only represents the time spent to compute an algorithm. In special purpose functional units, the speed of the system is dominated by the time spent to load and unload the data from a parallel machine. In the functional unit constructed by using VLSI devices which contain a large number of processing elements, the performance of the system is limited by the number of I/O pins available on a chip. In general, the I/O time, denoted by $I_E(N)$, is determined by depending on the number of I/O lines available in a given system. But, even if there are pins available for I/O operations in the system, increasing the number of I/O lines will increase the complexity and the cost of the system. Thus, the number of I/O lines for a system will affect both the

cost and the speed of the system. In this thesis, the I/O time will be expressed in terms of the number of I/O lines available for a given system. If a number of input data lines is L_{in} and a period of the clock signal running the input data bus is t_i, then the time spent to load an N by N binary edge picture into the system can be represented by

$$L(N) = t_i N^2 / L_{in}.$$

The time required to unload N^2 **c**-bit numbers from the system by using L_{out} output lines is

$$O(N) = t_o c N^2 / L_{out}$$

where t_o is the period of the clock signal running the ouput bus. The total I/O time I(N) is the sum of L(N) and O(N).

The speed versus the cost of the system is represented by the cost effectiveness: $C_E(N) = V_E(N) / c_E$, where c_E is the cost of the system with E PEs.

1.6 Convolution and Hough Transform

One of the most widely studied image processing algorithm is the convolution. As mentioned in the previous section, the Hough transform is a special form of convolution. Many structures have been proposed and actually built for 1D and 2D convolution. The most efficient VLSI architecture for convolution, as well as other parallel algorithms, is a systolic architecture. The concept of the systolic architecture was developed by Kung [31], [37] and his colleagues at Carnegie-Mellon University. Kung described the systolic structure as follows:

"A systolic system consists of a set of interconnected cells, each capable of performing some simple operations. Because simple, regular communication and control structures have substantial advantages over complicated ones in design and implementation, cells in a systolic system are typically

interconnected to form a systolic array or a systolic tree. Information in a systolic system flows between cells in a pipelined fashion, and communication with the outside world occurs only at the boundary cells. "

Next, we will discuss a 2D systolic convolution systems described by Kung [37].

For the given sequence of weights $\{w_1, w_2, \ldots, w_k\}$ and the input sequence $\{X_1, x_2, \ldots, X_n\}$, a 1D convolution can be defined by

$$\{y_i = w_1 x_i + w_2 x_{i+1} + \ldots + w_k x_{i+k-1} \mid i = 1, \ldots, n+k-1\}.$$

Various forms of systolic designs for this convolution problem were described by the Kung [31]. Figure 4 shows one of systolic arrays described by Kung and its cell definition. In this systolic array, weights are preloaded to the cells, one at each cell, and stay at the cells throughout the computation. Partial results Y_i and inputs x_i move from cell to cell in the left-to-right direction but at different speeds. To use the multiplier hardware in each cell, the x_i's move twice as fast as the Y_i's by delaying each Y_i one extra cycle at every cell it passes. In this configuration, all cells work all the time while performing a single convolution. The procedure computing the equation "$y = w_{11}x_{ij-2} + w_{12}x_{ij-1} + w_{13}x_{ij}$" is illustrated in Figure 5a through 5c. Figure 5a shows the moment to start computing Y. At this moment, the value of Y going into the leftmost cell is zero. During the first cycle $x_{ij}w_{13}$ is computed in the leftmost cell and passed to the next cell with x_{ij-1} (Figure 5a). The partial result is added to $X_{ij-1}w_{12}$ and passed to the rightmost cell with x_{ij-2} in the second cycle (Figure 5b). In the third cycle the $x_{ij-2}w_{11}$ is added to the partial result in the rightmost cell (Figure 5c).

The linear systolic convolution array in Figure 4 has been extended to implementing 2D convolutions by Kung [37]. The general layout of a kernel cell of the systolic 2D convolution array for a 3x3 window is shown Figure 6. Each row of the kernel computes the partial result for a row of the input data and the partial sums from each row are added at the row interface cell. Each row of the

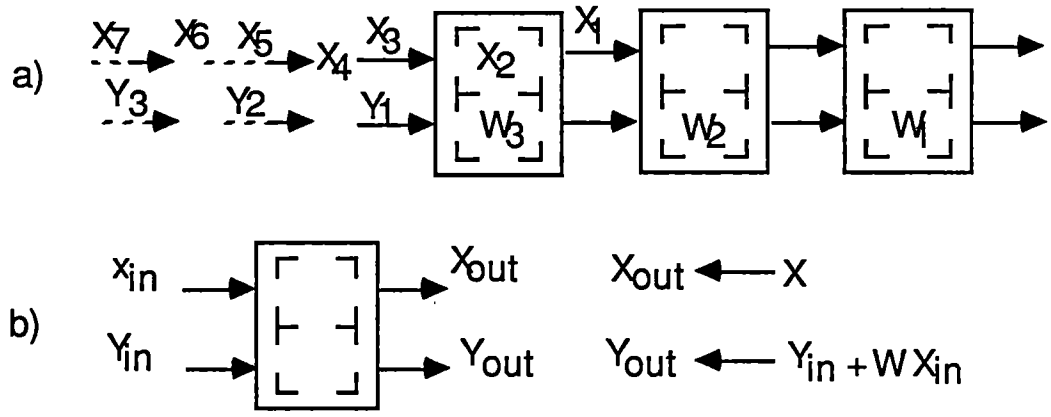


Figure 4. Systolic Convolution Array and Cell Definition

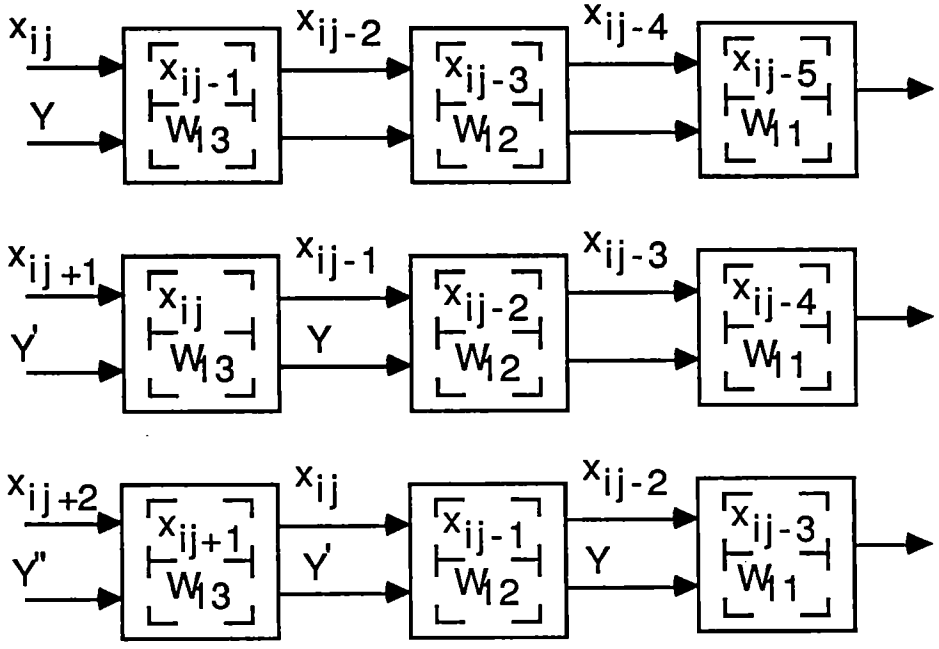
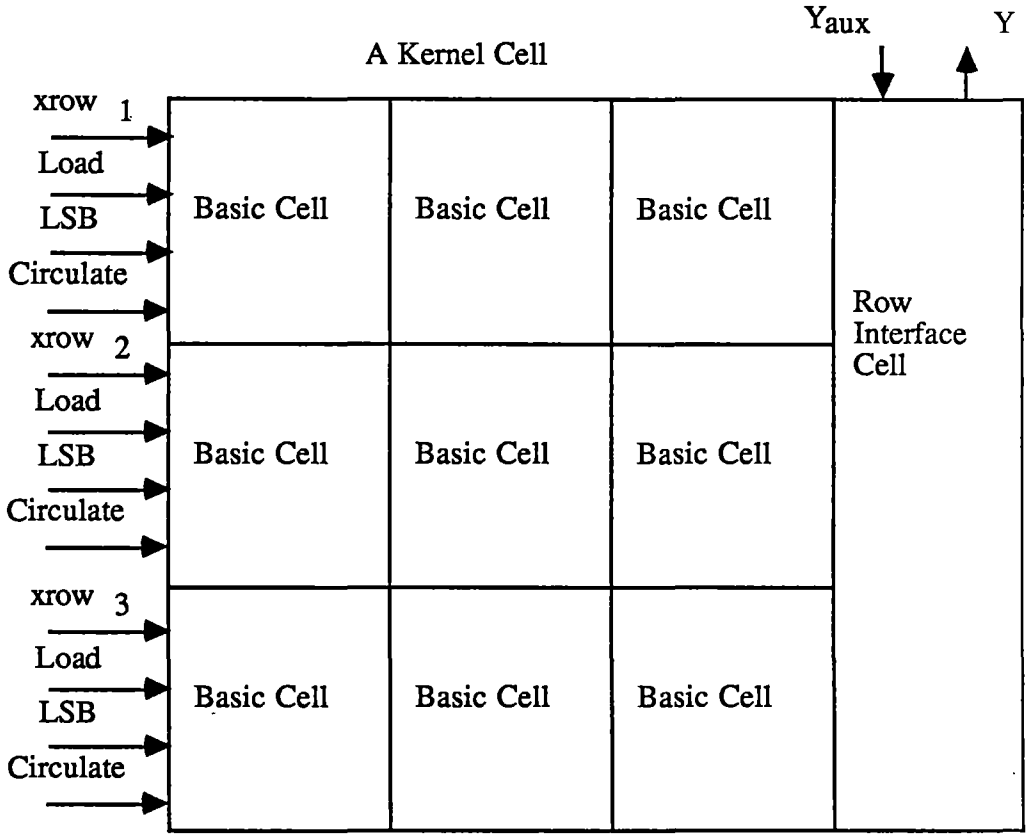


Figure 5. Procedure of Computing a Partial Sum in the First Row of the Convolution Array



$xrow_i$: Pixel value or weighting coefficient for row i.

Load : Control signal indicating weighting coefficients are being loaded.

LBS : Indicates that the Least Significant pixel Bits are being input.

Circulate : Control signal indicating the 2'nd half cycle (absence of input).

Y_{aux} : Optional input to be accumulated to the computed result.

Figure 6. Block Diagram of a 3 by 3 Convolution Array

kernel computes the partial result as shown in Figure 5. For example, to compute convolution at (i-1,j-1), three rows in the kernel cell shown in Figure 6 produce three partial sums: $y_1=w_{11}x_{i-2j-2} + w_{12}x_{i-2j-1} + w_{13}x_{i-2j}$, $y_2= w_{21}x_{i-1j-2} + w_{22}x_{i-1j-1} + w_{23}x_{i-1j}$, and $y_3= w_{31}x_{i1j-2} + w_{32}x_{i1j-1} + w_{33}x_{ij}$. These partial sums are added in the row interface cell.

In this configuration, the time spent to compute the convolution is proportional to N^2 for a small template, where N is the size of the input image, but M^2 PEs are required for an M by M template. As we mentioned earlier, the Hough transform not using gradient direction information is an efficient implementation of the convolution of an image with the shape template where edge pixels are unity and others are zero. If we compute the Hough transform by using the 2D convolution, there will be the following problems. First, the size of the template is arbitrary large, which means that M^2, the number of PEs required for the convolution, is large. Since the size of the template is close to N^2 in some cases, the size of the convolver must be N^2 to accommodate all possible shapes to appear in the scene. In addition to the processing element array, the convolver requires a summing circuit to add up the partial sum from all rows of the processing element array. If a parameter is represented by an 8 bit number, the number of adders required to construct a binary tree structured summer is (m-1). Second, the model to be detected must be loaded to the system before starting to compute the actual convolution. If the size of the convolver is smaller than the size of the template, the template must be divided into several segments and computed separately. In this case, each model segment must be loaded to the system before the actual computation starts. Third, each cycle consists of three operations: receiving data, adding two eight-bit numbers, and sending the output to the next cell. This means that the period of the clock signal will be large. Finally, a complex circuit is necessary to

feed the data into the system and to receive the data from the system. Therefore, the systolic array processor for the 2D convolution is not effective to compute an edge convolution with a large template because it does not take advantage of the data reduction achieved when representing pictures by edges. In other words, the systolic structure is expensive and not efficient for the Hough transform, which can be considered to be an edge convolution.

New specialized computer structures for the Hough transform must be designed to take advantage of the algorithm. In chapter IV, two specialized functional units for the Hough transform will be discussed. In the first unit, a processing element is connected to its four nearest neighbors. It has a mesh connected structure: more specifically a torus structure. It can compute the Hough transform in time proportional to B, where B is the length of the model boundary to be detected. But, loading and unloading data from the system is as complicated as the convolver discussed above. However, this unit is the best choice for the application where the orientation of an object is not known. The second unit has a linearly connected structure. Each processing element in this unit is connected to its right neighbor. The computation time of the Hough transform in this structure is proportional to N^2. This structure can be classified as one form of systolic array processors, where edge pixels move systolically, partial sums stay, and weights are broadcasted. It is close to the convolver discussed above but it has two significant advantages over the convolver for VLSI implementation. First, a template is represented by a one dimensional array which is very small compared with the 2D template used in the convolver. Second, weights (or template points) are broadcasted instead of loading them to the system before the actual computation starts. Third, the summing circuit is not necessary. In the convolver, the summing circuit consists of m adders when the size of the template is m by m. For an example, if the size of the template is

256x256 and a parameter is represented by an 8-bit number, then the summing circuit requires 255 8-bit adders.

CHAPTER II

NON-ANALYTIC SHAPE DETECTION

The nonanalytic shape detection algorithm can be divided into two groups. The algorithm in the first group is close to the convolution of an image with the shape template where edge pixels are unity and others are zero. It is first described by Merlin and Farber [24]. The Hough transform in the second group is suggested by Ballard [8] to improve the method proposed by Merlin and Farber. The methods in the second group make use of the gradient direction of the edge pixel to reduce the number of parameter points for a given edge pixel. From now on, we simply call the algorithms in the first group "Hough transform with No Gradient Direction information (HNGD)" and the algorithms in the second group " Hough transform With Gradient Direction information (HWGD)". Each of these algorithms has its own strong and weak points compared to the other in performance and computability. To determine an algorithm which is more suitable for VLSI implementation, both algorithms will be discussed in terms of their performance, computational requirements, and parallel implementation in this chapter.

2.1 Hough Transform with No Gradient Direction Information

Merlin and Farber [24] show how the Hough transform can be extended to nonanalytic shape detection. To find the best fit of a given shape (or curve) **S** in an edge picture **P** described by a set of points $\{ (x_i,y_i) \mid i=1,n \}$, they select first a

point on the given curve as a reference point O and rotate the curve 180 degrees with respect to the reference point. The rotated image represents the locus of possible O's for a given edge pixel. Then, the rotated curve is traced on each of the points of the picture with O placed over each edge pixel. The best fit to the given curve is the trace with O on the point where the maximum number of curves intersect. The schematical description of the above procedure is shown in Figures 7 through 9. The curve being sought and the rotated version of the curve are shown in Figure 7 and Figure 8, respectively. Figure 9 shows curves traced with O on each edge pixel. The locations of edge pixels in the picture space are also marked in Figure 9. As shown on Figure 9 all curves traced on boundary points of the given curve intersect at one point P.

To restate this algorithm in pseudocode, we define the edge picture as the matrix **P** and the accumulator representing the parameter plane where the loci of the curve will be traced as **M**. An element at the location (i,j) in the accumulator and the picture is represented by M_{ij} and P_{ij}, respectively. We need another array (or table) to describe the given curve. It is usually called *reference table* and denoted by the array **R**. The reference table **R** is a collection of coordinates of the points in the quantized plane that belong to a 180 degree rotation of the given curve **S** with the reference point O on (0,0). and represented as follows:

$$\mathbf{R}=\{(a_k,b_k) \mid k=0,T\},$$

where T is the number of points on the boundary of the given curve and (a_k,b_k) represents the location of the k'th model points.

If the size of the array **P** and **M** are N by N and M by M, respectively, then actual transformation in the digital computer is performed by using Algorithm 2.1. Algorithm 2.1 does not include the rotation operation (necessary to detect the orientation of an object) and maxima finding operation. The number of

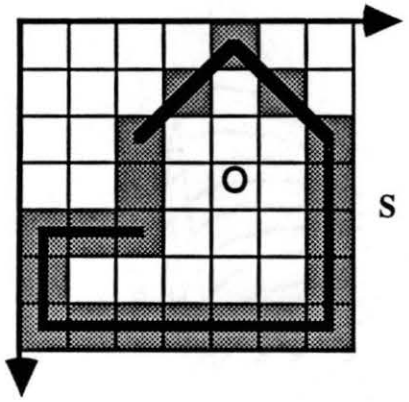


Figure 7. Given Shape S

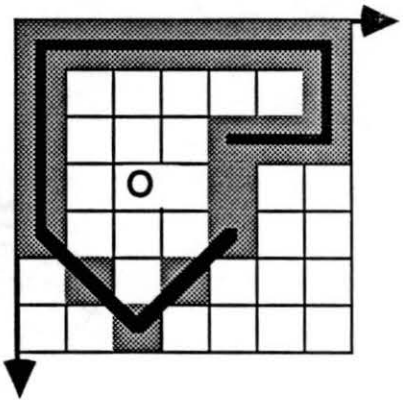
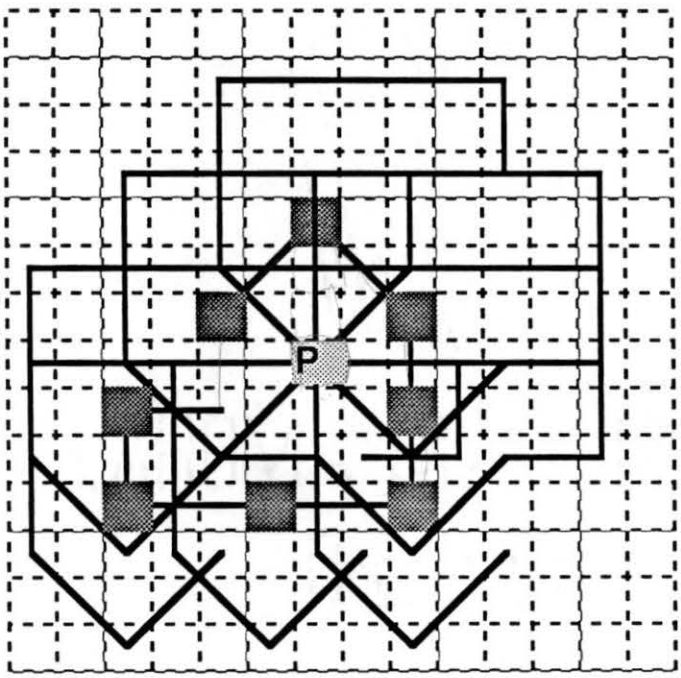


Figure 8. Rotated Model



Edge element detected from input image

p

: Parameter indicating the best match of the given model

Figure 9. Traces of the Rotated Model at All Edge Points

Algorithm 2.1

clear the accumulator ***M***

for i = 1 to N

  for j = 1 to N

    for k = 1 to T

      begin

      compute (a_k,b_k)*;*

        $x = i-a_k$;

        $y = j-b_k$;

      $M_{x,y} = M_{x,y} + P_{i,j}$;

      end;

Algorithm 2.2

for k = 1 to T

  begin

  compute (a_k,b_k)*;*

  for i = 1 to N

    for j = 1 to N

      begin

      $x = i-a_k$;

      $y = j-b_k$;

    $M_{x,y} = M_{x,y} + P_{i,j}$;

      end;

  end;

reference point computations is TN^2 times in this algorithm. To reduce the number of computations, Merlin and Farber modified this algorithm by changing the order of the operations. The algorithm 2.2 is the modified version of the Algorithm 2.1.

In the modified algorithm, the value of P_{ij} is added to $M_{x,y}$, where $x=i-a_k$ and $y=j-b_k$, and this procedure is repeated for all pair (a_k,b_k) that define the 180 degree rotation of the given curve. With this algorithm, each value for the increment index (a_k,b_k) is computed only once for all the points in **P**. Merlin and Farber mentioned without explanation that Algorithm 2.2 can be executed by translations of the **P** matrix and additions of the translated array to the **M** matrix.

2.2 Hough Transform with Gradient Direction Information.

Ballard [8] generalizes the Hough transform to detect arbitrary shapes by adding the O'Gorman-Clowes technique to the Merlin-Farber algorithms described above. This generalized Hough transform provides a mapping from the orientation of an edge-element to a set of instances of a given arbitrary shape **S**. This mapping allows all local evidence for a particular instance of **S** to contribute to global decisions about the figure. A detailed description of all the aspects of this algorithm is given by Ballard [8] and Sloan [38].

Let us consider the circular boundary detector with a fixed radius r_0 for a moment. If the gray-level of the object is lower than that of the background, the gradient direction Q of an edge pixel (x_i,y_i) points to the direction where the center of the circle is. Otherwise, it points the opposite direction. We can represent this relationship with the equation 2.1.

$$|\mathbf{r}| = r_0$$
$$\text{Angle}(\mathbf{r}) = Q(\mathbf{r}) \qquad (2.1)$$

Since the gradient direction Q is the angle of the vector pointing to the center of the circle to be detected, theoretically, the number of votes for each edge point is reduced to 1. As shown in the Figure 3c, we only increase a single point **x** + **r** for each gradient point **X** with direction Q. Using the gradient direction information we can have two major improvements over the algorithm not using it. First, the number of computations is reduced by T if the number of points used to describe the circle to be detected is T. Second, the error in choosing a circle is significantly reduced. However, the gradient direction does not always point to the center of the circle due to noise in an input image and errors in the process of computing gradient directions. The success of the Hough transform using the gradient direction depends on the reliable determination of edge element orientation [38], [39].

Now suppose we have an arbitrary shape like the one shown in Figure 10. We can extend the idea of the circle detector with fixed radius to this case by using an R table representing an arbitrary shape. The R table is constructed as follows: first, choose a reference point O at the location $\mathbf{y}=(y_x,y_y)$ for the shape; second, compute $Q(\mathbf{x})$ which is the gradient direction for the boundary point $\mathbf{x}=(x_x,x_y)$ and $\mathbf{r} = \mathbf{y} - \mathbf{x}$, and then store **r** as a function of Q. The value of the gradient direction Q is quantized so that the number of entries in the R table can be reduced. Since the value of Q for each edge pixel is arbitrary and discrete, an index Q in the R table may have many values of **r.**

The R table is used to detect instances of the shape **S** in an image in the following manner. For each edge pixel **x** in the image, compute the local gradient direction Q and increment all the corresponding points **x** + **r** in the accumulator array M, where **r** is a table entry indexed by Q. Maxima in M correspond to possible instances of the shape **S.** However, a problem arises in detecting maxima in the array M due to the effect of noise. For example, if

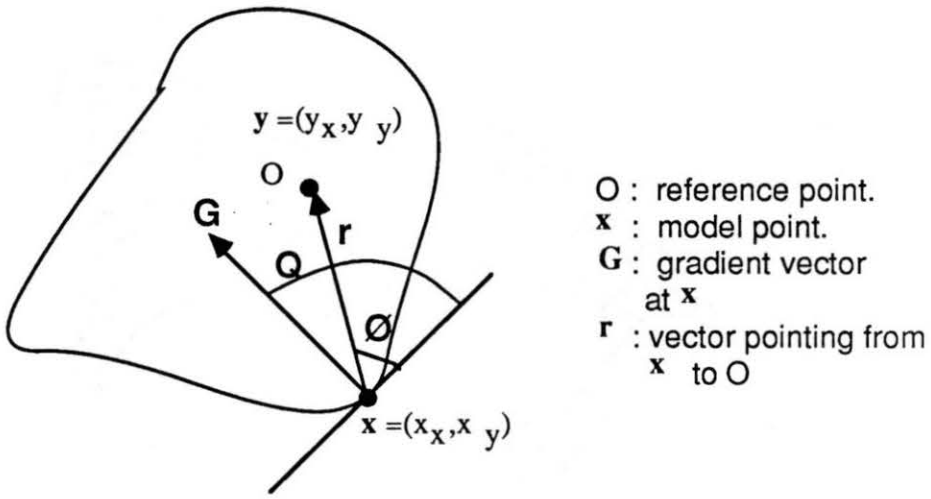


Figure 10. Relationship between the Gradient Direction and the Entry of the R Table for a Model Point $\mathbf{x}$

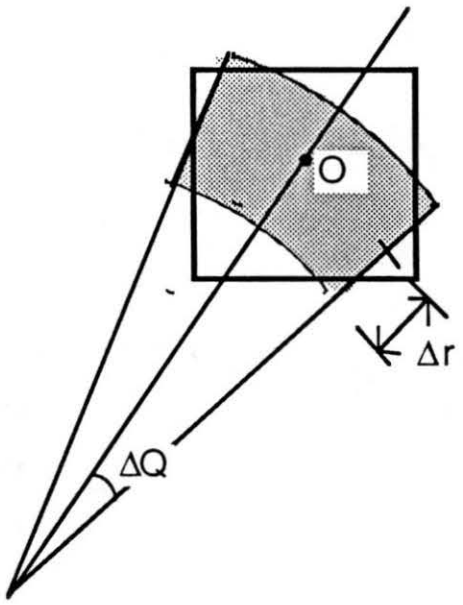


Figure 11. Effect of Error in the Gradient Direction and Radius on Circle Detection

uncertainties in the gradient direction Q and the radius r in the circle detection problem are $\pm\Delta Q$ and $\pm\Delta r$, respectively, the reference point for a given feature point is one of the points in the shaded band in Figure 11. To compensate the effect of the noise, Shapiro [11], Brown [13], and Ballard [8] adopt a rule, sometimes called the *voting rule*, specifying how a particular edge-element affects the value of **M.** Ballard and Shapiro increment all parameters which fall within the shaded band of Figure 11 to compensate for the noise in the circle finding. However, compensation of the noises for the arbitrary shape detection is not so simple because r varies arbitrary. Furthermore, uncertainties in the gradient direction Q is large for an arbitrary shape. Consequently, the issue of noise compensation of the arbitrary shape detection algorithm has been discussed very little in the literature. If we assume that the size of the image and accumulator are N by N and M by M, respectively, and the number of **r** for an index Q is **N**(Q), then the algorithm can be expressed by the pseudocode in Algorithm 2.3. If the average of **N**(Q) for all Q in the R table is ***a***, the order of the computation required for the generalized Hough transform is $(a\,CN^2+N^2)$, where CN^2 is the number of edge pixels in the input image. The value of ***a*** is close to 1 if the shape to be detected is a circle and it is close to T if the shape is a line. In general, the value of ***a*** will be larger than 1. The value of ***a*** is determined depending on the shape of the model, objects in the input image, and the number of steps used to quantize the gradient direction. Several different properties of this algorithm are discussed by Ballard [8].

2.3 Considerations for Parallel Implementation

The Hough transform is a simple and efficient form of template matching designed to detect known objects in noisy images. But its excessive computational requirement prohibits wide-spread use. Even if the most advanced

Algorithm 2.3

```
for i = 1 to N
    for j = 1 to N
      begin
     compute the magnitude and the gradient direction
        of the edge point at (i,j);
        if magnitude of the edge is larger than the given
         threshold,
            then
        for k = 1 to N(Q)
           begin
                compute the corresponding point x-rk in
            the accumulator array M and increase a
                  point at x-rk or a group of points
                  surrounding the point x-rk to compensate
                 the error due to noise.
             end;
    end;
```

sequential computer available today is used, the real time computation of the Hough transform for an image with moderate size is almost impossible. For practical industrial applications requiring real time computation, the Hough transform must take advantage of highly parallel computer structures. Specifically, it must take advantage of a fast and inexpensive VLSI architecture. To be implemented in such parallel architectures, the Hough transform must satisfy the conditions described in section 1.5. Next, we will discuss the HWGD and the HNGD from a parallel implementation point of view.

Let us consider algorithms 2.2 and 2.3. For a moment, neglect the portion of the algorithm 2.3 computing the magnitude and angle of the gradient. In the sequential computer the magnitude and angle of the gradient are, in general, computed while computing the Hough transform to save memory space and some computations. But this scheme will cost more computing time when the Hough transform is used to detect objects with arbitrary orientation and size. For our discussion we will assume that the magnitude and angle are computed and stored in the memory.

To compute the number of computations and I/O accesses, the following assumptions have been made.

(1) Each gradient magnitude is thresholded to a binary number and stored in an N by N array with one bit depth.

(2) Each gradient angle is represented by an one byte integer and stored in an N by N array.

(3) Incrementing an accumulator cell and computing an index value is performed by one and 32 bit additions, respectively.

(4) The largest possible number in the accumulator is less than 2^{16}, which means that an accumulator cell is represented by a 16-bit register.

(5) The total number of edge pixels per frame is E.

(6) The number of model boundary points is T.

Under these assumptions, the order of computations and the number of the I/O accesses are computed and listed in tables I and II. The variable C is E/N^2 and has a value between 0.01 and 0.3. Next, we will examine some problems related to actual implementation of these two algorithms based on the two tables.

2.3.1 Sequential Implementation. If we use a sequential computer to compute the Hough transform, the number of computations for the HWGD is $T/(1+4C\boldsymbol{a})$ times less than that for the HNGD. The value of $\boldsymbol{a}$ depends on the shape of the object under consideration. If the object is a rounded shape, the number of computations in the HWGD increases proportional to T/(*number of steps used to quantize the gradient angle*) so that the computation time is not strongly affected by the number of template points T. For the rounded shape, the value of $\boldsymbol{a}$ is 1. However, if the boundary of the object consists of straight lines, then the value of $\boldsymbol{a}$ is close to T and the HWGD may be slower than the HNGD. In general, the HWGD is faster than the HNGD in a sequential computer if T is large and the processor to be used has a capability of fast arithmetic computation for the gradient angle computation. If a computer is not equipped with a arithmetic processor, which is the case for many microprocessor based systems, the time required to compute indices will be a dominating factor in the total computation time for the HWGD. High precision arithmetic computations are necessary to compute the **r** values. The number of index value computations for the HNGD is proportional to T.

2.3.2 Parallel Implementation. For parallel implementation, an algorithm should have the following characteristics: 1) the number of operations per image

TABLE I

NUMBER OF COMPUTATIONS

Operation	HWGD	HNGD
Addition	$C\alpha N^2$	TN^2
Comparison	$N^2+C\alpha N^2$	0
Index Computation	$2C\alpha N^2$	2T
Total # of Operation	$B^2(1+4C\alpha)$	$T(N^2+2)$

TABLE II

NUMBER OF MEMORY ACCESSES

Data	HWGD	HNGD
R Table	$2C(1+\alpha)N^2$	2T
Accumulator	$C(1+\alpha)N^2$	TN^2
Magnitude	N^2	TN^2
Gradient Angle	CN^2	0
Total	$(1+4C+3C\alpha)N^2$	$2(1+N^2)T$

pixel must be small, 2) the computation is regular so that different parts of the image are treated in the same manner, 3) operands for each operation must be easily accessed. Keeping these in mind, two algorithms will be studied for parallel implementation.

In the HWGD, only image elements above a threshold that is edge elements, influenced the output. Thus, if an entire image is fed to the parallel machine, most of the processing elements will be idle. One way of increasing the utilization of the system is feeding only edge elements in the image. All edge elements in the image are extracted along with their gradient direction and fed to the system. Then, the utilization of the system will be increased significantly. However, extracting only edge elements from the input image and computing gradient direction for edge elements are also time consuming tasks and may not be able to take advantage of a parallel architecture. Furthermore, after the input image is reduced to a string of edge elements, it is difficult to make use of a priori information (approximate location and orientation of objects) to reduce the number of computations. Thus, reducing the input image into a set of edge elements may not be a good idea for the parallel machine unless the algorithm is used to determine the orientation of an object. To determine the orientation of an object, a 2D accumulator must be computed for every possible orientation and the parameter representing the object is searched in the 3D accumulator.

If an input image is directly fed into the parallel machine, the angle and magnitude of the gradient at each image point must be computed for the HWGD. Therefore, each Processing Element (PE) must be capable of floating point computation and table lookup for the gradient angle computation. It means that complex hardware is necessary for a PE.

The number of operations for an edge element in the HWGD depends on

the gradient direction at that pixel. The number of vectors in the entry of the R table indexed by a gradient direction is determined depending on the shape of the object, which means that the number of operations for an edge pixel may differ from that of other edge pixels. Therefore, the HWGD yields poor hardware utilization. When a parallel system consists of P processing elements, a set of data is generally divided into P segments and each segment is assigned to a separate processor. If all the segments take the same amount of time to process, the hardware utilization will be close to 100%. Otherwise, the overall processing time will be dominated by the time spent to process the largest segment.

Another difficulty in the parallel implementation of the HWGD is that operands needed for an execution cycle are arbitrarily distributed in the memory. For example, the locations of accumulator cells to be updated for a given edge pixel are determined depending on the gradient direction at the pixel, which means that accessing more than one operand per memory cycle is difficult. When the number of processing elements P increase, the number of memory accesses per unit time increase proportional to P. Since every machine has a fixed memory bandwidth, performance of the parallel machine will be saturated at a certain number of processing elements. If the number of memory accesses per data point is large, then the performance improvement by increasing the number of processing elements will be small.

In summary, the HWGD is not a good choice for the SIMD structures which leads to a simple and inexpensive parallel implementation. It can be implemented in a multiple instruction and multiple data (MIMD) structure where each PE has a large local memory and is capable of executing its own programs. But, even in the MIMD structure, the HWGD will suffer from the same problems. Furthermore, the complexity of the control and hardware increase

drastically while the efficiency of the system decreases exponentially, when the number of processing elements increases.

In the HNGD, all pixels in an image are treated in the same manner. Operations required for an image element are T one-bit additions and T accumulator array accesses. Thus, the time required to process a given image is predictable. Other advantages over the HWGD are: 1) the HNGD only performs one bit addition (or increment) and 2) the location of operands are known. In other words, the processing element for the HNGD is extremely simple and the number of I/O accesses can be reduced by accessing a large segment of operands per memory cycle. It also can take advantage of multibus systems to increase memory bandwidth. Thus, the HNGD can be implemented efficiently in the SIMD consisting of a large number of processing elements with a local memory. But, the HNGD in the form of Algorithm 2.2 is not much better than the HWGD for the SIMD structure. Algorithm 2.2 will also suffer from the I/O bound problem. A processing element must update T accumulator cells in the main memory (or the local memory in other processing elements) for an edge element. To be implemented efficiently in this structure, the algorithm in Algorithm 2.2 must be modified. In the next chapter, we will discuss modifications of the algorithm for parallel implementation.

2.4 Performance Analysis of Arbitrary Shape Detection Algorithms

In the previous section, it is shown that the HNGD by Merlin is a better candidate for VLSI implementation. However, this algorithm requires a significantly greater number of computations in a sequential computer than the HWGD by Ballard. It has been also thought to be inferior to the HWGD in performance. Consequently, the HNGD has been almost neglected by

computer vision researchers. No one has seriously studied this algorithm as a candidate for practical computer vision.

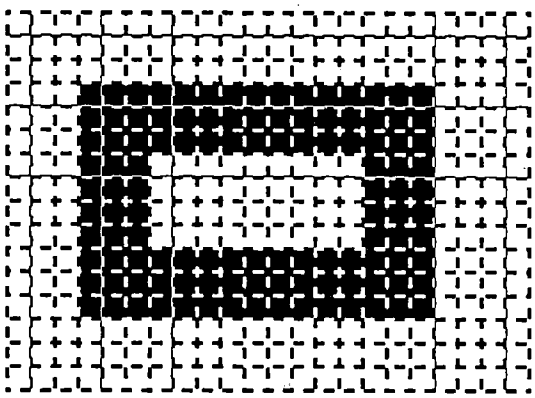
Now, it is time to reconsider the HNGD for industrial applications. With the advanced VLSI technology, it is possible to implement a special purpose functional unit for a computationally intensive algorithm, like the HNGD at low cost and with compact size. As discussed in the previous section, the HNGD has an excellent structure for VLSI implementation. Thus, the computational requirement of this algorithm should no longer a serious problem for industrial applications.

Articles concerning the performance analysis of the HNGD for industrial applications have rarely been published. On other hand, the HWGD has been studied by some researchers. But, no one has compared these two algorithms for industrial applications. In industrial applications, we have control over the work environment so that the quality of the input image for the system is somewhat adjustable and predictable. This may enhance the performance of the HNGD for industrial applications. In this section, we will analyze the performance of the HNGD and HWGD for industrial applications based on experimental results and work by Ballard [8], Shapiro [10]-[13], and Laws [39]. For this analysis, both algorithms are implemented in C on the IRI D256 computer vision system, and a set of selected images taken in a controlled environment is processed by using these algorithms. The obtained results are analyzed to give some ideas about the performance of the HNGD compared with the HWGD. Here, we are not trying to prove that the HNGD is superior to the HWGD. We want to simply show that the output of the HNGD is as good as that of the HWGD in locating industrial parts in a controlled environment.

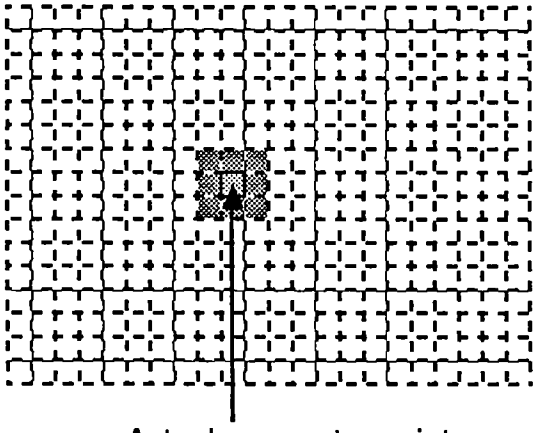
Ballard points out that the HNGD generates many false instances of the desired shape in an image with a multitude of edge pixels due to coincidental

pixel arrangements. Figure 12 shows one example. If the width of the detected boundary is 3 pixels wide, nine accumulator cells around the actual parameter for the given shape have the same values. This problem can be solved by applying an average operator on the accumulator array. For the given example, the peak can be detected by applying a 3 by 3 average operator. In practice, the thickness of the edge is decided by the threshold value used to detect edge elements from the input image. Figure 14 shows two edge pictures of a pair of pliers obtained by thresholding the Sobel image of figure 13 at a gray level of 110 and 51, respectively. Figure 14a consists of 1796 edge elements and Figure 14b consists of 3506 edge elements. These two edge pictures are processed by both HNGD and HWGD and the results are shown in Figures 15 and 16. Figure 15 shows the contents of accumulator arrays obtained from the edge picture in Figure 14a. In Figures 15 and16, part a and c show the contents of an accumulator and part b and d show the contents of a row passing through a peak in accumulator, which is called "x-profile". As we can see from Figure15 and16, decreasing the threshold value (or increasing the thickness of the edge) does not affect the performance of the HWGD, but it affects that of the HNGD. The near peak sidelobes in Figure 16 is much larger than that in Figure 15.

The peak value is also affected by the threshold value in the HNGD while it is almost independent from the threshold value in the HWGD. In the x-profiles in both figures, the peak has been found at the same location, but differences in the near peak sidelobes may play an important role in detecting parts partially occluded by other objects. The noise from other objects may shift the location of the peak in the accumulator. This indicates that the HNGD is more sensitive to the threshold value than the HWGD. Fortunately, the effect of the threshold value can be minimized in the industrial applications by carefully constructing the lighting system.



a) Thick Edge in the Picture Space



b) Maxima in the Parameter Space.

Figure 12. Effect of Multitude Edge Pixels on the Hough Transform

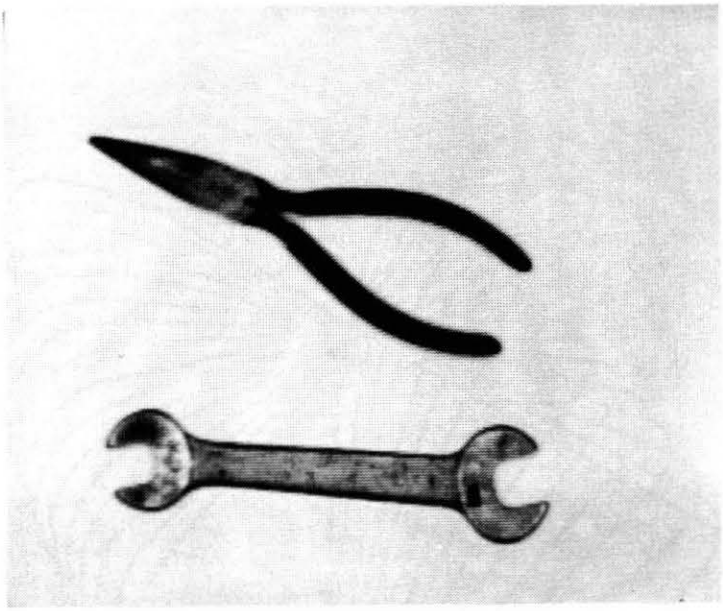


Figure 13. Digitized Picture

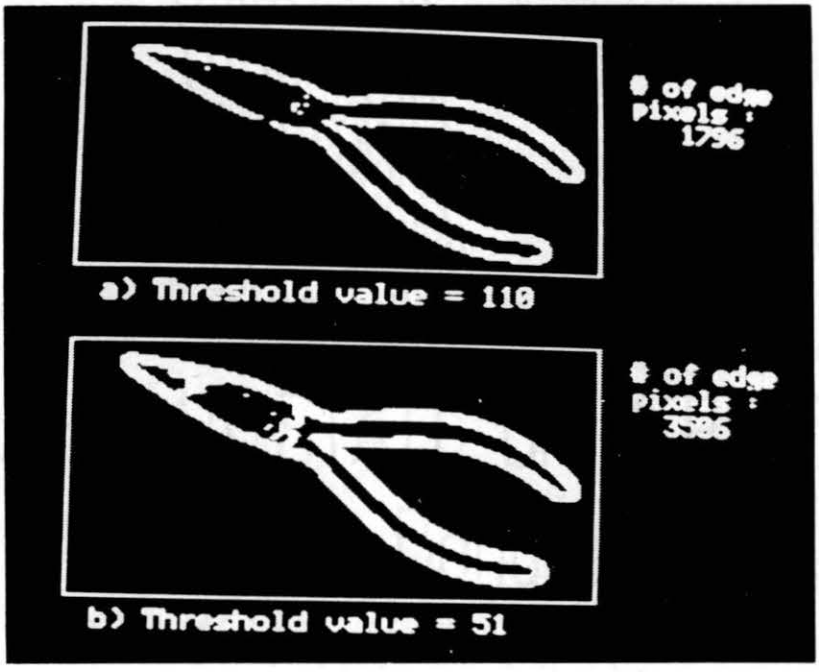


Figure 14. Edge Pictures

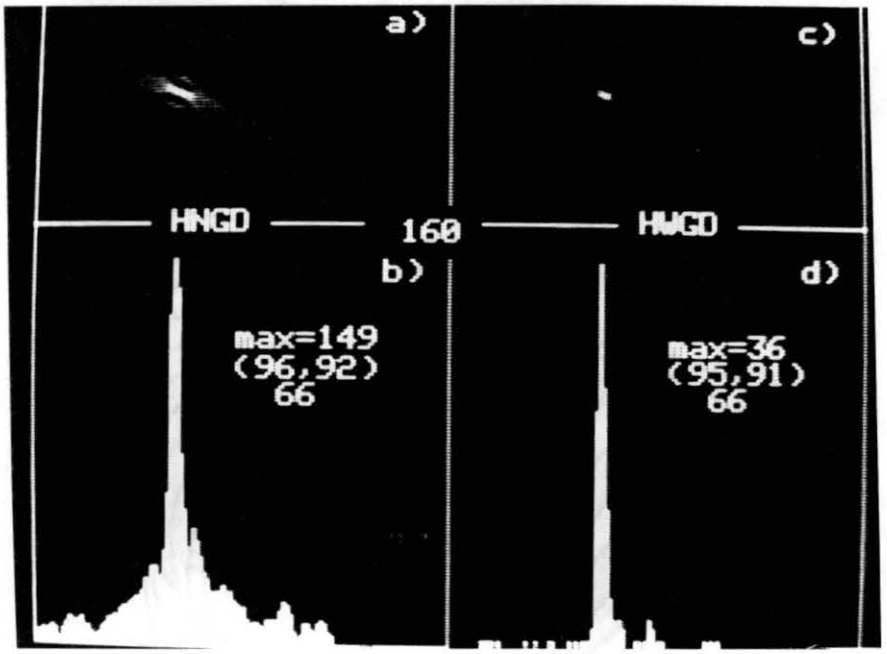


Figure 15. Contents of Accumulator Array for the Edge Picture Thresholded at 110

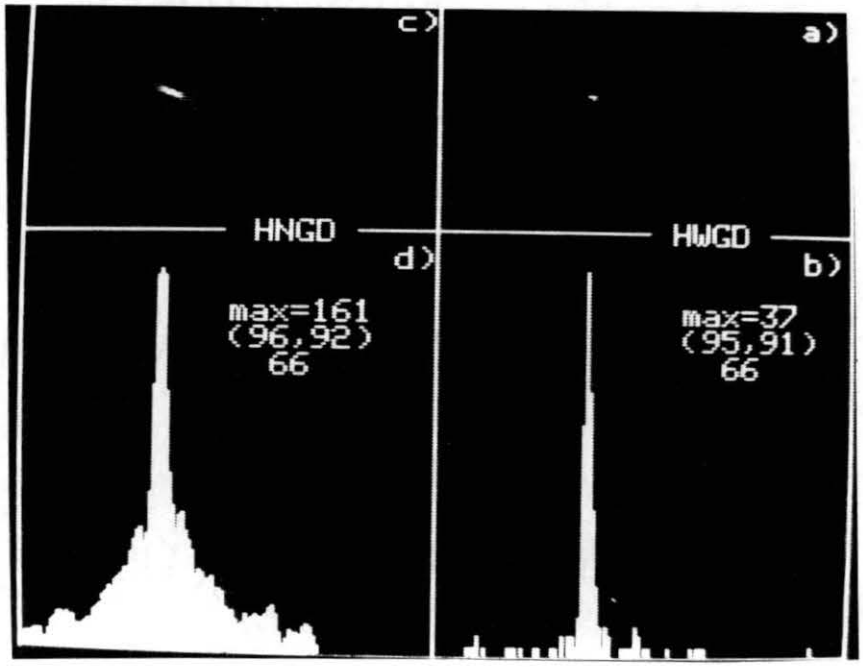


Figure 16. Contents of Accumulator Array for the Edge Picture Thresholded at 51

The HWGD reduces the chance of detecting false instances in an image with a multitude of edge pixels by using the gradient direction information at each edge pixel. It eliminates the edge pixels with gradient directions not belonging to any of the model boundary points. By using the gradient direction at an edge pixel, the number of possible parameter points for the edge pixel is, in general, much smaller than T, where T is the number of model boundary points. Thus, the peak in the accumulator array computed by the HWGD is generally sharper than the peak in the accumulator generated by the HNGD. Figures 15 and16 support this argument. Figure 16b has a much sharper peak than Figure 16d.

Theoretically, the HWGD appears to be much faster and more reliable for arbitrary shape detection than the HNGD. However, as we discussed earlier, the number of operations for the HWGD depends on the shape of the object to be detected. The HWGD is greatly affected by the error in the gradient angle. The gradient direction may contain two types of errors. One is due to the uncertainty in the gradient angle computed from the input image. The gradient angle of an edge element on an input object is not always equal to the gradient angle of the corresponding point on the model even if the orientation of the object is fixed. Specifically, if an object has gradual, uncertain boundaries or strong internal gradients, there exists a large uncertainties in the gradient angle. The amount of error in the gradient angle also depends on the edge operator used to compute it. Thus, the HWGD does not produce good results in locating such objects.

Another error in the gradient angle is due to quantization. In nonanalytic shape detection, a model is described by an R table. Since the R table can not have an infinite number of entries, the gradient angle must be quantized with a certain number of steps. If the gradient angle is equally divided into 360 steps,

the quantization error will be ± 0.5 degrees. If the Hough transform is used to determine the orientation of an object, an object is represented by three parameters (two for location and one for orientation). Thus, the orientation is quantized with a certain number of steps, and then a 3D accumulator array is constructed by computing a 2D accumulator for every possible orientation. The location and orientation of an object is determined by searching a maximum value in the 3D accumulator. Thus, the number of computations necessary for orientation determination depends on the number of steps used to quantize the orientation. If the number of steps is decreased to reduce the number of computations, the quantization error in the orientation will be increased. The quantization errors in the gradient angle and orientation are added to other errors in the gradient angle.

Let us reconsider Figure 11. This figure illustrates how the error in the gradient angle affects the circle detection algorithm. The parameter for the edge pixel with gradient direction q may fall anywhere in the shaded band when uncertainties in the gradient direction and the radius are given by Δq and Δr. In the circle detection problem, the errors in the parameter space can be reduced by increasing all accumulator cells in the shaded band [12] or smoothing out the uncompensated accumulator array [8]. But, compensating errors in the nonanalytic shape detection is much more complicated than that in the analytic shape detection. In the HWGD, a shape S is represented by an R table which is generally denoted by R(q). R(q) is a set of vectors **r** pointing to possible parameter points for the edge pixel with the gradient direction q. If the object is rotated by j and this transform is denoted by T_j, then

$$T_j[R(q)] = \mathrm{Rot}\{R[(q-j)\mathrm{mode}2\pi],j\},$$

where Rot{**a**,d} rotates the vector **a** by angle d. For instance, if the gradient angle of an edge pixel is q, the set of vectors at the table entries (or set of vectors

r) for this pixel will be indexed by the heading (q-j)mod2π. The entries will be rotated by j and used to determine possible parameter points for the edge pixel. Thus, a table index (q-j)mod2π is a function of the following three components: the gradient angle at the pixel, the current orientation, and the sum of all the errors. When an error occurs, the table entries shown for the index value computed for a specific edge pixel may contain vectors that do not point to the correct parameter space location corresponding to that pixel. Unfortunately , there is no easy way of compensating for the error in nonanalytic shape detection.

In circle detection, the distance between a reference point and a model boundary point is fixed, and the error in the gradient direction directly affects the process of selecting a possible parameter point. Therefore, the error can be compensated by increasing all accumulator cells in the square in Figure 11 or by convolving the uncompensated accumulator with a template. The size of the template will be the size of the square in Figure 11, and the contents of the template will be generated to give the best result for a given task.

In nonanalytic shape detection, the distance can be any value and the error in the gradient angle will affect ,indirectly, the process of selecting a set of possible parameters for an edge point. The gradient angle at an edge pixel is used to find a set of vectors pointing to possible parameters from the edge point. Thus, the size of the window for the compensation operation must be determined separately for all the edge points in the image. This means that the smoothing operation suggested by Ballard for analytic shape detection is less effective and that increasing all accumulator cells in the shaded band is much more complicated in nonanalytic shape detection.

The shaded band for nonanalytic shape detection is defined in the same manner as it is defined for circle detection, but the size of the shaded band is

arbitrary because the value of r is arbitrary in nonanalytic shape detection. The shaded bands defined for two different distances are shown in Figure 17. Furthermore, the number of accumulator cells in the shaded band will be much larger than that in the analytic shape detection since the quantization error in the orientation and the gradient angle is larger. Thus, the error compensation of the nonanalytic shape detection algorithm is less effective and computationally expensive compared with that of the analytic shape detection algorithm. However, if the distance between the object and the camera is fixed, which is possible in industrial applications, the error in r can be neglected. Thus, the error compensation is somewhat simplified. If the error in the computed index I is $\pm e$, the performance degradation due to the error will be minimized by increasing all accumulator cells pointed by vectors in entries in the R table indexed by heading (I-e) through (I+e).

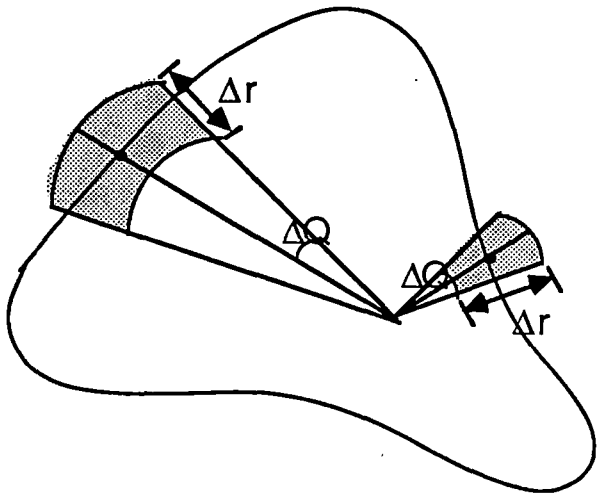


Figure 17. Effect of the Error in the HWGD

To study the effect of the error on the HWGD, an image shown in Figure 13 is processed by both HNGD and HWGD. For every angle in the range between 0 and 90, an Hough transform is computed and the peak in the accumulator array is found. The peak heights are then plotted against orientation in Figure 18. Parts a and b in Figure 18 are obtained, respectively, by the HNGD and the HWGD. The R table used to obtain the plots in Figure 18 is encoded by using 80 model points. For reliable determination of the orientation of an object, the plot must have a sharp peak with small sidelobes. Figure 18 indicates that the HWGD without the error compensation may be less effective than the HNGD for the orientation determination.

The plot in Figure 18b has a smaller peak and relatively large sidelobes. Figure 19c proves that the sidelobes in Figure 18b are due to the error in the gradient direction. Figure 19c shows the same type of plot generated by the HWGD with the error compensation discussed earlier. The peak in this plot is much sharper than the peak in Figure 18b. The model used to generate this plot consists of 160 boundary points instead of 80. Almost identical results were obtained by using the model consisting of 80 boundary points. The contents of accumulator arrays at peak orientation are shown in Figures 15 and 19. From Figure 15, it is clear that the HWGD produces a much sharper peak than the HNGD for a given orientation. However, the HWGD produces the peak at the location a few pixels off from the actual location, while the HNGD produces the peak at the actual location. The parameters of the object found by HWGD and HNGD are (95,91) and (96,92), respectively. The boundary of the objects represented by these parameters is shown in Figure 20. The boundary of the object detected by the HWGD matches less accurately the original image than that detected by the HNGD. The HWGD with the error compensation produces a

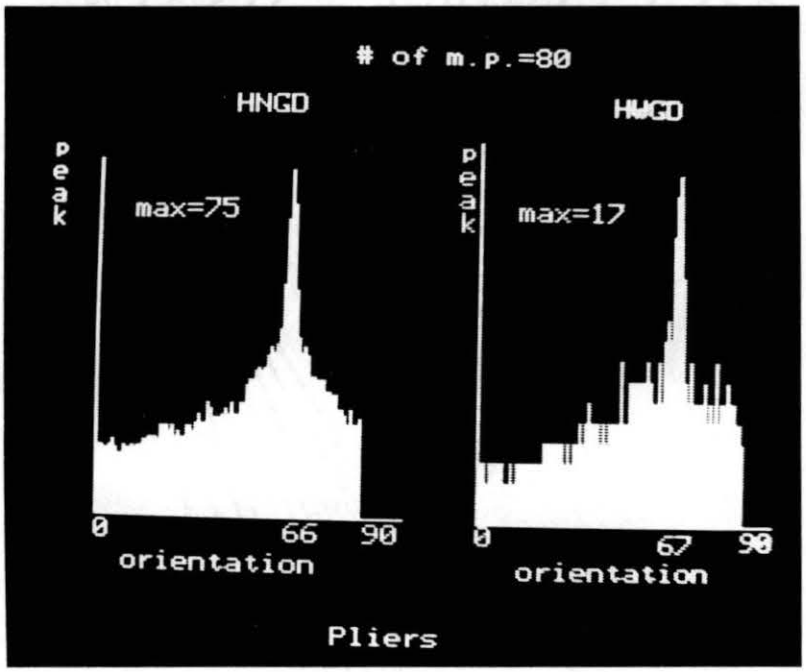


Figure 18. Peak versus Orientation

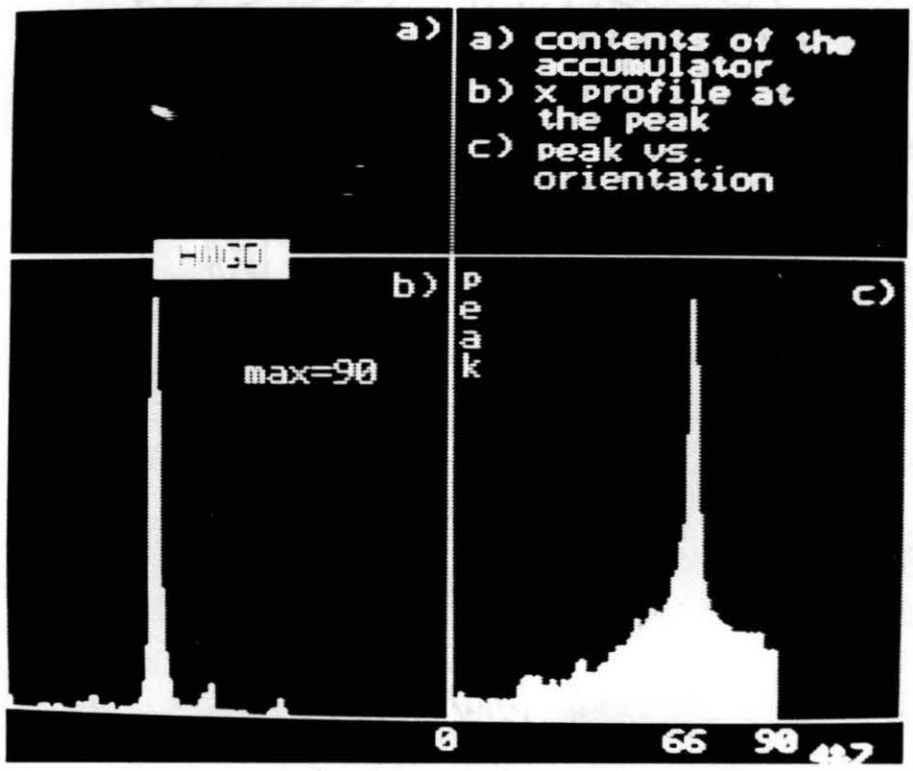


Figure 19. Results Obtained by the HWGD with Error Compensation

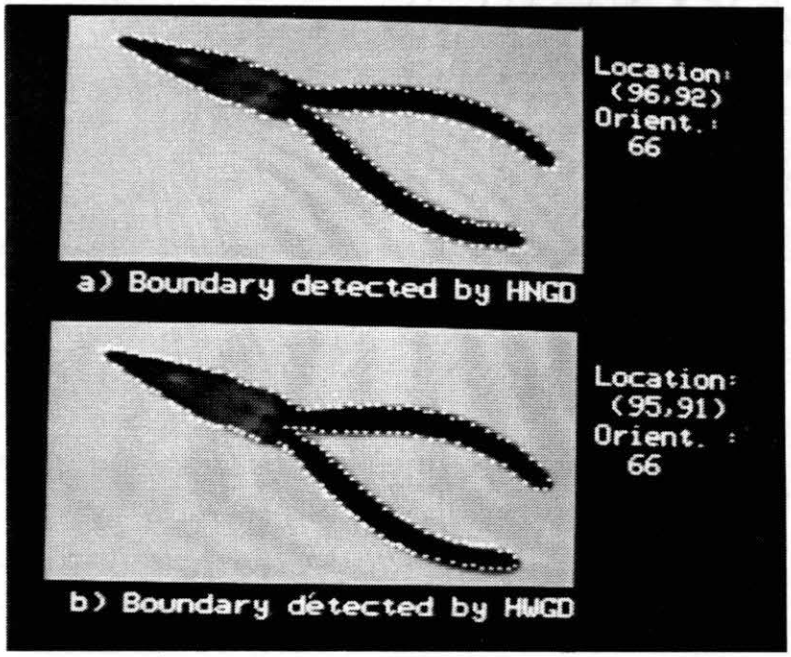


Figure 20. Plier Model Superimposed upon a Gray-scale Picture

peak at the actual location, (95,92). In addition to the pliers, both algorithms are used to detect three objects: a piece of puzzle, a wrench, and connectors. The results show that the HWGD produces a sharper peak than the HNGD but the parameter obtained by the HNGD produces a closer match than that obtained by the HWGD.

In the HWGD, the computation time is proportional to the number of model boundary points. Thus, the number of model points must be reduced as much as possible. But, reducing the number may affect the performance of the algorithm. Specifically, if there are many other objects in the image, reducing the number of model boundary points will increase the chance of finding false peaks. It is not obvious how reducing the number of model points will affect the accuracy of this algorithm. To study this problem, the parameters of a pair of pliers has been computed for the model with four different resolutions by using both HWGD and HNGD. A different number of boundary points is used to describe the model in each case: 160, 80, 40, and 20. Table III shows the result of this operation. The performance of the HWGD was less affected by the number of model boundary points than that of the HNGD. In the HNGD, the peak value was reduced as the same proportion the number of model boundary points. But, the peak value in the HWGD remained largely unaffected. The peak location was also affected by this number in the HNGD.

In the above experiments, we learned that the accuracy of the HWGD is slightly better in some points but not much better than that of the HNGD for finding industrial parts in a controlled environment. We also found that the performance of the HNGD is good enough to be used for industrial applications. In Figure 21, the model of a connector detected by the HNGD are superimposed upon the gray scale picture of the connector.

Several other images containing different objects (including puzzles,

TABLE III

EFFECT OF THE NUMBER OF MODEL BOUNDARY POINTS ON PEAK DETECTION

Num. of M.P.	Orientation		Position		Peak Value	
	HNGD	HWGD	HNGD	HWGD	HNGD	HWGD
160	66°	66°	96,92	95,91	149	36
80	66°	67°	96,92	96,94	75	17
40	66°	66°	95,91	96,94	38	11
20	66°	66°	95,92	96,94	20	9

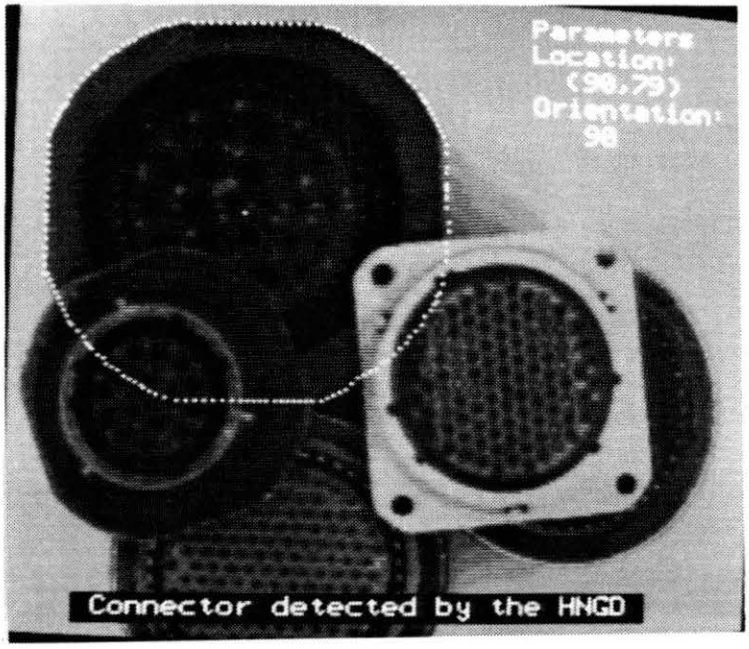


Figure 21. Connector Model Superimposed upon a Gray-scale Picture

wrenchs, and ring nuts) have been processed by using both the HWGD and the HNGD. The experimental results indicates that the HNGD can be used to detect partially occluded industrial parts as long as the objects has a small number of stable positions. The HNGD is particularly suit for flat objects. The performance of the HWGD was also analyzed by other researchers. Laws [39] tests the HWGD designed by Ballard [8] and reports some characteristics of the algorithm. In his report, Laws mentions some interesting characteristics of the algorithm. First, the algorithm works well for finding large and well defined objects. Objects with gradual, uncertain boundaries or strong internal gradients are not located accurately. Second, the quality of object detection is dependent on the reliable determination of the gradient angle. Law's experiment partially supports our findings.

CHAPTER III

REFORMULATION FOR HARDWARE IMPLEMENATION

In the previous chapter, we discussed two types of Hough transform for nonanalytic curve detection in terms of performance, computational requirements, and parallel implementation. The HWGD has less chance of finding false peaks in the accumulator if the error in the gradient angle is compensated. It also needs less computations in most cases than the HNGD unless the shape consists of long straight lines. On the other hand, the HNGD has a regular structure and requires simple processing elements. Which makes this algorithm a better choice for a compact, fast, and inexpensive VLSI implementation. As we showed in the last chapter, the HNGD can detect industrial parts in a controlled environment fairly reliably although it may be less effective than the HWGD for pictures containing large amount of noise.

In general, the HNGD requires significantly larger amount of computation than the HWGD for a given image. But it has some important characteristics for efficient parallel implementation. First, the number of operations per edge pixel is small and the same for all edge pixels no matter what the shape of the object is. Second, operands are distributed in the memory in a way that makes possible fetching more than one operand per memory cycle. Third, the operations required are simple so that they can be performed in a PE with little hardware. As a result, this algorithm can be implemented with good hardware utilization in a SIMD structure consisting of simple processing elements and interface circuits. Thus, the HNGD is the best choice for our goal of designing

inexpensive, compact, and fast specialized functional units for the Hough transform. From now on, we will only consider the HNGD for the specialized system design. In the rest of this thesis, the Hough transform simply means the Hough transform not using the gradient direction for arbitrary shape detection.

In the Hough transform in the form of Algorithm 2.2, the accumulator array must be defined in the global memory and updated by each processing element. In such a case, the efficiency of the system will decrease drastically due to increasing memory contention as the number of processing elements gets larger. To avoid this problem, the Hough transform must be reformulated for efficient parallel implementation. In this section we will reformulate the Hough transform to reveal its important properties for parallel implementation. The reformulated algorithm will be analyzed in terms of parallel implementation.

The HNGD by Merlin uses the edge picture of the model as a template. Thus, it can not be used to detect a shape with unknown orientation. The reformulated algorithm will accommodate shapes with unknown orientation. Based on the reformulated algorithm, one way of determining the orientation and position of an arbitrary shape will be suggested. This suggested algorithm is designed to take advantage of the specialized parallel structures to be discussed in the next chapter.

3.1 Reformulation

In most Hough-like transformations, each pixel is tested to see if it belongs to the boundaries of objects in the image by comparing the magnitude of the gradient of the pixel with a given threshold. If a pixel belongs to a boundary of an object, it is transformed into the parameter space by incrementing accumulator cells representing a set of possible parameters for the edge pixel. The accumu- lator cells representing possible parameters are the cells on the

trace of the model rotated by 180° with the reference point at the edge pixel location in the accumulator array. In the new approach, a set of image elements to be tested for a given accumulator cell is determined according to the location of the accumulator cell. To compute an accumulator cell M(i,j), the model is traced on the edge picture P with the reference point at the picture element P(i,j). Then, the set of picture elements on the trace of the model are tested. The value of the accumulator cell is the number of edge elements in the set. Thus, in this approach, an accumulator array can be determined cell by cell. In other words, an accumulator array can be divided into several subarrays and computed separately. If some information such as approximate location and orientation of an object are known, the object can be detected by computing only a portion of the accumulator array.

This approach has some important characteristics for efficient parallel implementation. First, a priori information is easily taken into account to reduce computations. Second, the computation can be easily decomposed into several subproblems and each subproblem can be solved independently. Since an accumulator cell is computed independently, the accumulator array can be divided into subarrays and computed by large numbers of PEs without contentions for operands. In Algorithm 2.3 by Ballard and Algorithm 2.2 by Merlin, location of accumulator cells to be updated for a given edge pixel are not predictable. A set of accumulator cells for an edge element is determined depending on the location of the pixel in the image space and an image consists of an arbitrary number of edge pixels. Thus, the task of computing those algorithms are not, in general, easily divided into subtasks which can be computed independently without contention for operands. In other words, these two algorithms are data dependent. The new approach has two other differences compared with the conventional way of computing the Hough transform

which is shown in Algorithms 2.2 and 2.3. In this approach we can quantize the parameter space with any desired resolution to reduce the number of computations. We can also adapt this algorithm for detection of objects with arbitrary orientation by representing the model with a R table similar to the one used by Ballard. In Merlin's algorithm, the size of the accumulator array and input image are identical and a model is represented by a two dimensional binary template containing a complete edge description of the object to be detected. Next, we will discuss this new approach in more detail.

3.1.1 Model Generation. The model generation of the proposed algorithm is somewhat similar to the the model generation of Ballard's algorithm. For convenience, we are assuming that a complete edge description of the object to be detected is given in the image plane **I**.

The first step of the model generation is marking a reference point in the image plane **I**. In general, any point in **I** can be used as a reference point, but the center of the object is preferable because the longer the distance between a reference point and a point on the boundary is the larger the effect of the quantization error of the orientation is. The second step is tracing each point on the object boundary with respect to the reference point O. The relative position of each boundary point to the reference point is stored in the reference table R. The i'th point on the boundary of the object is represented by a vector **r** whose magnitude and direction is r_i and q_i, respectively.. If the edge representation of an object is given by Figure 22, for example, table IV is one of possible R table for the given object.

Until now, we have considered objects with fixed orientation and scale. To detect an object of arbitrary orientation o and scale s, these two parameters must be added to the description of the object and the R table must be adjusted according to the value of o and s. As suggested by Ballard [8], simple

transformation of the R table will allow it to be used to detect scaled or rotated instances of the same object. If the object is scaled by s and rotated by o, for example, the R table for the object is

$$R = \{ (s\, r_i, (q_i + o) \bmod 2\pi) \mid i=1,T\} \qquad (3.1)$$

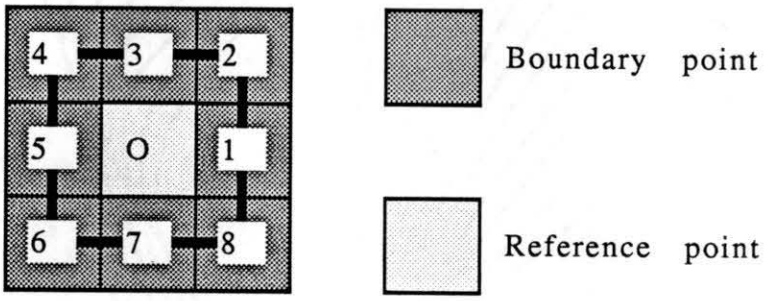


Figure 22. A Model with a Reference Point

TABLE IV

POLAR COORDINATE SYSTEM REPRESENTATION OF THE REFERENCE TABLE FOR THE MODEL IN FIGURE 22

Index	1	2	3	4	5	6	7
R(i)	(1,0)	($\sqrt{2}$,π/4)	(1,π/2)	($\sqrt{2}$,3π/4)	(1,π)	($\sqrt{2}$,5π/4)	(1,6π/4)

3.1.2 Mapping. If the image space and parameter space are digitized as N by N arrays, the model with a reference point at the image element P(i,j) is

used to determine the contents of an accumulator cell M(i,j). The points on the boundary of the model are traced with respect to the reference point in the image space and the number of votes for the accumulator cell is determined by counting the number of edge elements on the trace. Algorithm 3.1 shows this new approach for the problem of computing accumulator cells in the M by M window starting at (xstart,ystart). T is the number of boundary points used to describe the model. In this algorithm, R(i,1) and R(i,2) represent the magnitude and direction of the vector starting at the i'th model boundary point and ending at the reference point.

In Algorithm 3.1, the position of the model boundary point is computed TM^2 times for the given orientation o. To reduce the number of computations, Algorithm 3.1 is reformulated and shown in Algorithm 3.2. The reformulated algorithm computes x_k and y_k only T times. If the image space and parameter space are quantized by the same number of steps, Algorithm 3.2 is equivalent to adding a translated edge picture to the accumulator array for each model point. For instance, if the relative position of a model point is given by (x_k,y_k), then the input edge picture is translated by $(-x_k,-y_k)$ and added to the accumulator array. The parallel architectures discussed in the next chapter are designed based on this idea.

The number of operations required to compute the Algorithm 3.2 is

$$T_1(M) = TM^2. \tag{3.2}$$

If an approximate position and orientation of an object is known, it can be detected by using a small accumulator array. Thus, the computation can be significantly reduced if the approximate position of an object is known. To take maximum advantage of this algorithm, a hierarchical search method with pyramid structure is desirable. At the start, all probable objects and their approximate positions are determined by using a small number for T and a

Algorithm 3.1

```
for i = xstart, xstart+M
    for j = ystart, ystart+M
        for k = 1 to T
            begin
            x_k = R(k,1)*cos(R(k,2)+o );
            y_k = R(k,1)*sin(R(k,2)+o );
        M(i,j) = M(i,j) + P( x_k+i,y_k+j );
            end
```

Algorithm 3.2

```
for k=1,T
    begin
    x_k = R(k,1)*cos( R(k,2)+o );
    y_k = R(k,2)*sin( R(k,2)+o );
    for i = xstart, xstart+M
        for j = ystart, ystart+M
        M(i,j) = M(i,j) + M(x_k+i,y_k+j);
    end;
```

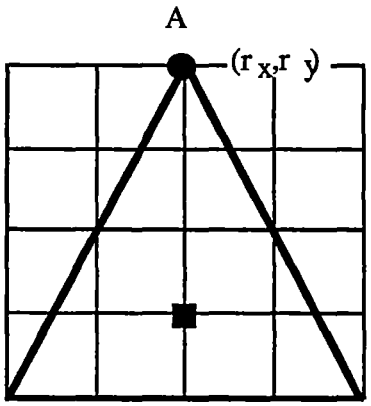
coarsely quantized parameter space. Finer descriptions and positions of objects found in the previous step are determined by computing Hough transformations with larger T and higher resolution accumulator windows defined around the approximate positions found in the previous step. The resolution of the parameter space can be adjusted either by using a lower resolution input image or by adding more than one picture element to an accumulator cell for every model boundary point. When the parameter space and the image space are quantized by the same number of steps, only one pixel on the trace of a model boundary point is added to the accumulator array. But, when the parameter space is quantized more coarsely than the image space, more than one pixel around the trace of each model point are added. For example, when the resolution of the parameter space is nine times less than the image space, all nine points in the 3 by 3 window with the center at the trace of a model point are added to accumulator.

The algorithm using the coarsely quantized parameter space is less sensitive to the orientation of the object and uses a smaller accumulator array than the algorithm using the finely quantized parameter space. Thus, it is much faster in finding the approximate position and orientation of an object.

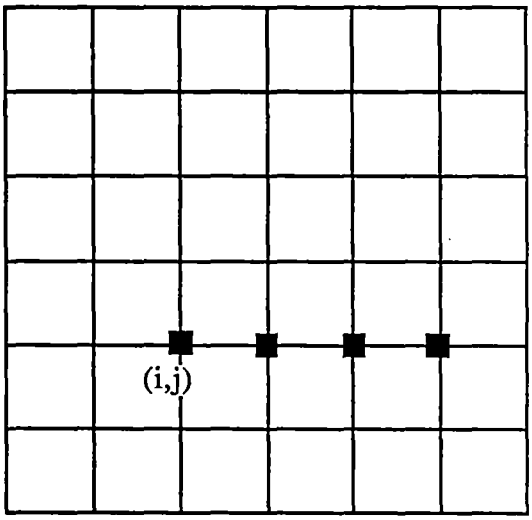
3.1.3 Special Property for Parallel Implementation. Algorithm 3.1 and Algorithm 3.2 are computationally equivalent to the Hough transform by Merlin if they are computed in a serial computer. However, the new approach will be a significant improvement over Merlin's algorithm for parallel implementation. In the next chapter, we will explore three parallel structures for the new approach using an accumulator array whose size is equal to that of the input image. Before we discuss those structures, let us study a special property of the reformulated algorithm that will allow us to use such structures.

In the reformulated algorithm, a point in the parameter space (a cell in the

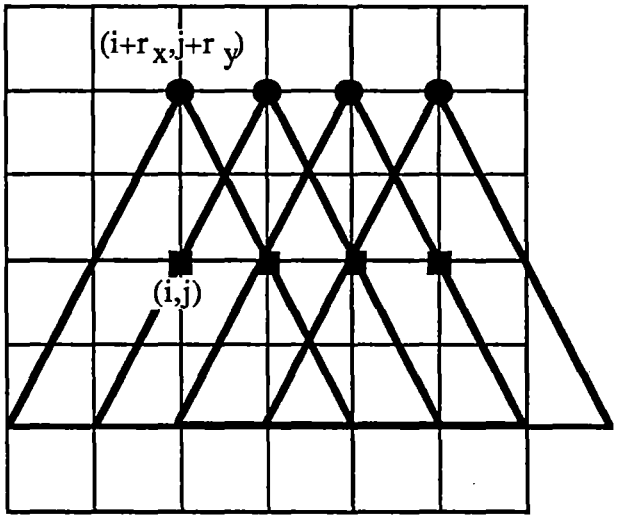
accumulator array) is used as a reference point in the image space (or edge picture). The boundary of the object is traced with respect to the reference point in the image space. (Note that the algorithms described in the chapter II trace the boundary of the 180 degree rotated object by using an edge point as the reference point of the object and all parameter points on the trace are increased by one.) The value of pixels encountered while tracing the boundary is added to the accumulator cell representing the parameter of the object traced. For example, if we consider four consecutive points starting at (i,j) in the accumulator array shown in Figure 23b and the object given in Figure 23a, the trace of the boundary point A of the objects represented by these four parameters are shown in Figure 23c. In Figure 23c, the four black rectangular boxes indicate reference points and four circular dots represent the trace of the boundary point A for the four reference points. Note that the traces of the boundary point A for these four parameters are four consecutive points starting at $(r_{x1}+i,r_{y1}+j)$ in the same column, where $(-r_{x1},-r_{y1})$ is the relative position of the model boundary point A to the reference point. In general, if we trace a boundary point (r_{x1},r_{y1}) for p consecutive parameter points starting at (i,j), then the data required for the operation is p consecutive edge points from $(i+r_{x1},j+r_{y1})$ to $(i+p+r_{x1},j+r_{y1})$ in the edge picture. Since the value of the picture element is either 1 or 0, we can update the p accumulator cells for the boundary point by adding p consecutive edge points starting at $(i+r_{x1},j+r_{y1})$ to the p accumulator cells. Thus, the number of memory accesses can be reduced by accessing more than one data point in a single clock cycle when a small portion of the accumulator array is computed by using existing SIMD structures like Illiac-IV. If an N by N accumulator array for an N by N input image is computed by using Algorithm 3.2, the accumulator can be obtained by adding the translated edge picture to the accumulator array for each model point. For example, to update accumulator array for the model point



a) A Model point
to be Traced.



b) Four Parameter Points to
be computed.



c) Locus of the Model Point A for Four
Consecutive Parameter Points.

Figure 23. Trace of a Model Point in the Image Space

$(-r_{xi},-r_{yi})$, the edge picture is translated r_{xi} pixels in the x direction and r_{yi} pixels in the y direction and each pixel in the translated picture is added to a corresponding accumulator cell. During the translation, the portion of the picture shifted out and shifted into the frame boundary is set to zero.

In the next chapter, we will discuss three parallel structures specifically designed for the reformulated Hough transform in terms of ease of parallel implementation, size of the system, and speed. Like any other parallel structure with a large number of PEs, the performance of the structures to be discussed is limited by the I/O bandwidth of the system. The time spent to load and unload the data from the system will be a major portion of the total time spent to evaluate the Hough transform in those structures. Thus, we can increase the efficiency of the system by reducing the I/O requirements for a given algorithm. In the Hough transform, a 2D accumulator array must be computed for all possible orientations if the orientation of the object to be detected is not known. For example, if the orientation is divided into 360 steps, a 2D accumulator array will be computed for for every 1 degree. Thus, the number of I/O operations will be 360 times more than that of the HNGD with fixed orientation. One way of reducing these I/O operations will be discussed in the next section. This method is specifically advantageous for the parallel structures to be discussed in the next chapter.

3.2 Orientation Determination by the Hough Transform

In most industrial applications, the orientation of the object being sought is not known beforehand. To determine the location and orientation of an object, the object must be described by three parameters: two for the location and one for the orientation. In the previous discussion we used an N by N accumulator array to find the location of an object with a fixed orientation. If the orientation is

equally divided into 360 steps, the parameter space for the object with an arbitrary orientation is represented by an N x N x 360 accumulator array. Then, the number of computations required to detect the object is proportional to $360N^2$. A 2D accumulator is computed for every 1 degree and maxima are searched in the 3D accumulator. For the orientation *o*, the model is rotated *o* degree with respected to the reference point, and a 2D accumulator array for the orientation is computed for the rotated model. Since the model is rotated with respected to the reference point, the position of the reference point in the model is not changed. If we add all 2D accumulators computed for different orientations and generate a 2D accumulator, then maxima in the 2D accumulator will represent locations of the objects being sought. Maxima in the 2D accumulator do not give any information about the orientation of the object, but the orientation can be determined by dividing the model into two or more segments and finding the location of each segment. After the positions of all segments are found, the orientation of an object can be determined by comparing computed positions with known relationship among segments in the model. We will call this algorithm "Hough transform detecting arbitrary shape with arbitrary orientation (HASAO)" for convenience.

Algorithm 3.2 may be the best choice for the HASAO. Since the number of computations in the modified algorithm is proportional to the number of model points, dividing the boundary of a model into several segments does not increase the number of computations. Thus, we will discuss the HASAO using the modified algorithm in this section.

The HASAO is an excellent algorithm for the special purpose systems to be discussed in the next section. Specifically, the linearly connected parallel structure in the section 4.3 will be a perfect match for this algorithm. The number of computations of the algorithm in the linearly connected structure is not much

greater than the number of computations needed to detect an object with known orientation. The HASAO has been implemented based on Algorithm 3.2 and tested with some industrial objects including pliers and wrenches.

The first step in the HASAO is selecting a set of boundary segments describing a model. The boundary of the model is divided into several segments and a set of segments which corresponds to a unique shape for a given model are selected. The best set of segments for a model will be represent a shape which is hard to find in boundaries of other objects to appear in a scene. After a set of boundary segments for an object is determined, a reference table is generated for each segment and the location of the reference point chosen for the segment is stored with the table.

The transformation is performed by adding the contributions from all model points generated by equation (3.1) for all possible orientations. If the number of entries in the reference table is T and the number of steps used to quantize the orientation is Q, then the number of pixels added to an accumulator cell will be QT. Thus, the HASAO requires QTN^2 operations to compute a 2D accumulator array for a segment consisting of T boundary points. As we can see from this number, there is no savings in computation by using this algorithm over the Hough transform using a 3D accumulator. The only saving is achieved in the maxima search. In the HASAO, maxima are found in a 2D accumulator array instead of a 3D accumulator. However, in the parallel machines like those discussed in the chapter IV, the HASAO and the Hough transform with fixed orientation will take almost the same amount of time to compute a 2D accumulator array. Furthermore, the HASAO drastically reduces the I/O time compared to the Hough transform using a 3D parameter space. The Hough transform using a 3D accumulator needs to compute a 2D accumulator array for all possible orientations. Therefore, the contents of the accumulator array in a

special purpose system must be unloaded S times to the main memory in the host computer, where S is the number of steps used to digitize the orientation. The number of unloading operations can be reduced significantly by using the HASAO. The HASAO will be described in detail. Some experimental results will be presented.

After a 2D accumulator array is computed for a model segment, the location of the segment is determined by searching maxima in the accumulator. If there is more than one candidate for the location, all candidates will be registered. The orientation of the object being sought can be determined by comparing the relative positions of candidates in the input image with known relationships between model segments. The false locations of a segment are eliminated during the orientation computation process by making use of the known relationship among boundary segments such as relative position of a segment with others.

The HASAO has been implemented in C on the IRI D256 computer vision systems based on Algorithm 3.2. It was applied to actual data consisting of some industrial parts including a pair of pliers and wrench to demonstrate the performance of this algorithm. However, it should be mentioned here that the program used to produce the following results is designed only to show the basic concept of the HASAO. More work will be necessary for practical use.

Figure 24 show a pair of pliers and a wrench. The white dots on the boundary of the pair of pliers are points used to encode R tables. As shown in the figure, two segments of the pliers are selected to determine the orientation. An R table is generated for each segment and used to compute the accumulator array for the segment. Figures 25a and 25b show the accumulator for segment 1 and segment 2 for the image in Figure 13. The accumulators in Figure 25 are

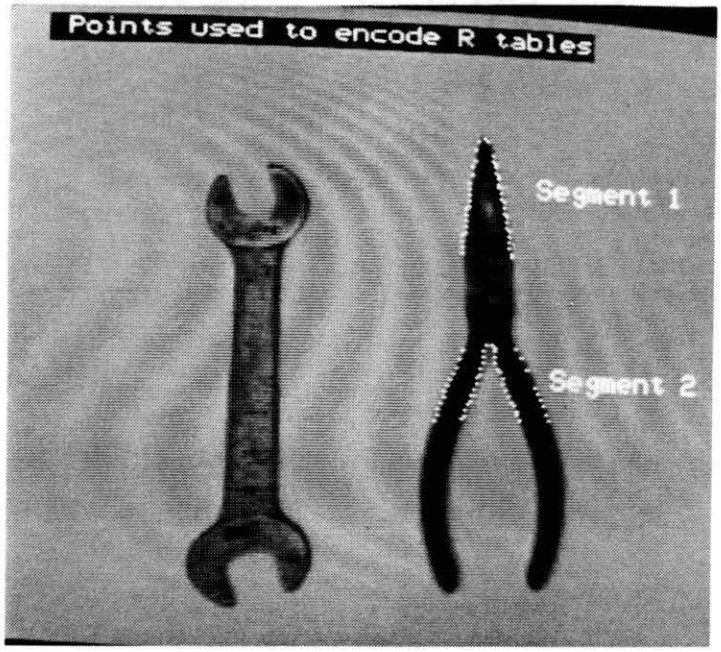
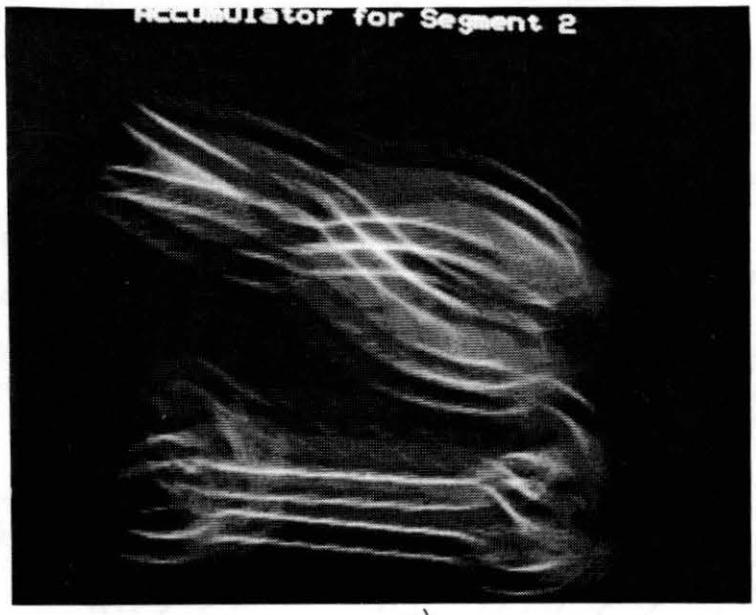


Figure 24. Boundary Points used to Encode R Table



a)

b)

Figure 25. Accumulators Computed for Selected Segments

obtained by adding all accumulators computed for every angle in the range between 43° and 88°. The number of boundary points used to describe segment 1 and segment 2 is 45 and 53, respectively. For the computation of the HASAO, thinning operation is applied to the edge picture obtained by thresholding a Sobel image. In practice, a 2D accumulator computed for all possible orientation may have peaks other than the peak representing the parameters of a given segment due to the edge pixels from other objects in the image. If the number of points used to describe a segment is increased, then the possibility of having false peaks will be reduced. Another way of avoiding this problem is computing more than one accumulator array for a given segment. To find the orientation of pliers, an accumulator array is computed for every 45° and possible parameters are extracted from each accumulator array. After all possible parameters extracted from four accumulator arrays, the best candidate for the parameter representing the segment is determined among those parameters. The orientation of the pliers is then determined by comparing computed locations of these two segments with known relationship between them. Figure 26a shows the parameters found from Figure 25 and detected segments on the object. The point p1 and p2 are the parameter representing the segment1 and 2, respectively. These parameters are found by thresholding the accumulators in Figure 25 at the gray level (peak value-1). The peak value in an accumulator array is obtained from the histogram of the array. The orientation of pliers determined by this algorithm is 66° which is the actual orientation of the pliers. The same algorithm has been used to determine the orientation of a wrench in Figure 13. Figure 26b shows detected segments of the wrench. For wrench detection, a 2D accumulator array is generated by adding all accumulator arrays computed for every angle in the range between 67.5° and 112.5°. The number of boundary points used to describe model

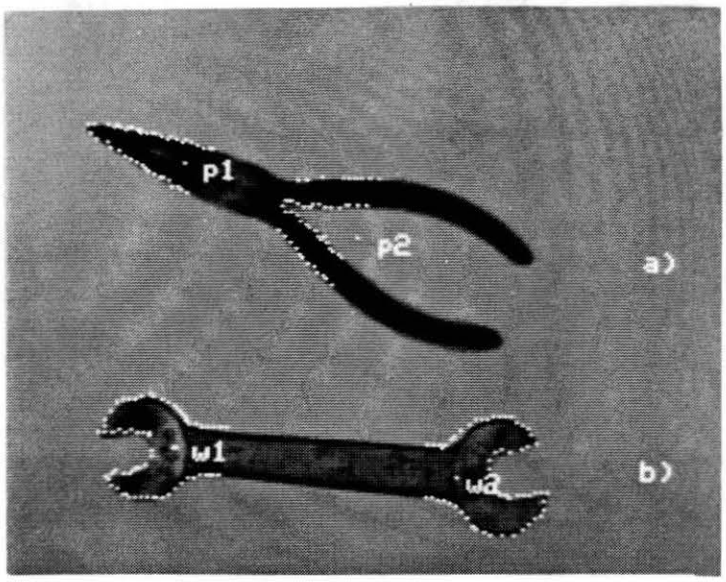


Figure 26. Detected Edge Segments Superimposed upon a Gray-scale Picture

segments 1 and 2 are 61 and 68, respectively. The orientation of the wrench computed by the HASAO is 93°. Observed orientation of the wrench is 90.

As we can see from the above experiments, the HASAO can determine the position and orientation of an object very accurately. This algorithm may not be practical in the existing computer systems, but it has a great potential for the special computer structures discussed in the next chapter. Here, we only show the basic idea of the HASAO. To be used for actual industrial applications, more work is necessary. First, a systematic way of selecting a set of boundary segments of the object should be investigated to provide unique identification of the object of interest. Second, more elaborated ways of deciding the orientation must be found to overcome false peaks or missing peaks.

CHAPTER IV

HARDWARE IMPLEMENTATION

In this chapter we will examine three parallel structures for algorithms derived in the previous chapter. One of these structures is the SIMD structure which is similar to Illiac IV, and the other two are structures specifically designed for Algorithm 3.2. The existing SIMD structure is discussed in section 4.1 to give an idea of how the modified algorithm can be implemented in this structure. The main objective of this research is designing specialized VLSI structures for the Hough transform which leads to inexpensive and compact functional units for industrial applications. The SIMD machine is flexible in the sense that it can be used not only for the Hough transform but also for a variety of image processing algorithms. However, its high cost limits the use of this system for industrial applications, such as bin picking, part verification and location, and robotics control. To be feasible for simple industrial computer vision applications, the Hough transform must be implemented in a simple and inexpensive device. In section 4.2 and 4.3, we will discuss two architectures which are specifically designed for VLSI implementation of Algorithm 3.2. One structure is a mesh connected structure where each PE is connected to its nearest four neighbors, and the other is a linearly connected structure where a PE is connected to its right and left neighbors. These two structures will be discussed in terms of their performance and VLSI implementation.

4.1 Existing Parallel Structure

Two important issues in the parallel implementation of any given algorithm are optimal distribution of workload among PEs to get the maximum hardware utilization and the minimum memory contention for the same operands. As mentioned, each accumulator cell is computed independently in Algorithm 3.2. Thus, the accumulator array can be divided into several subarrays, and each subarray can be assigned to a PE. If the new approach is implemented in a SIMD structure, like Illiac IV [33], a copy of the binary edge picture **P** will be stored in the local memory of each PE to avoid memory contentions for the same operands. Illiac-IV consists of a control unit, 64 processing elements each of which has its own local memory, and an interconnection network. The capacity of the local memory in the Illiac-IV is 2048 words. The network scheme used to interconnect PEs is a near neighbor mesh. We will investigate the implementation of Algorithm 3.2 in a parallel system similar to Illiac-IV but with a larger local memory.

4.1.1 Computation. If the capacity of the local memory is large enough to hold a whole input picture, contention for the same operands can be eliminated. By storing a copy of the input edge picture **P** in its local memory, a PE no longer requires operand accesses in the main memory. Since each pixel in the input image is represented by one bit, the amount of memory required to store an edge picture is N^2 bits which is within affordable range for a reasonable size image for industrial applications. Once **P** is copied to the local memory of a PE and an evenly divided segment of an accumulator array is assigned to a PE, there will be no contentions for the same operands. Furthermore, this algorithm is ideally regular for parallel implementation. Thus, it is expected that the speed of this algorithm will be increased almost linearly proportional to E, where E is the number of PEs. In such a system, Algorithm 3.2 can be computed as follows.

Step 1: Divide the accumulator **M** into E windows each of which consists of m^2 accumulator cells and send the coordinate of the upper left hand corner point of each window to the designated PE. (If the local memory of a PE is not big enough to hold all m^2 accumulator cells, **M** can be divided into more than E segments)

Step 2: Send the i'th model point to each PE.

Step 3: Update the accumulator segment in a PE for a given model point.

Step 4: Repeat Step 2 and 3 until all model points are traced.

Step 5: Read in the contents of the sub-accumulator-array in each PE and store them in the main memory.

Step 6: Update the model for different other possible orientation and repeat Step 2 through Step 5.

Step 7: Find maxima in the 3D accumulator array (or 2D accumulator for the fixed orientation).

* For an object of fixed orientation and size, Step 6 is not necessary.

All steps except Step 3 are executed in the control unit. Step 3 is executed in PEs. In Step 1, each PE receives the coordinate of the upper left hand corner point of the accumulator segment assigned to the PE and stores the coordinate in the registers "xoffset" and "yoffset". In Step 2 each PE receives the relative coordinates of the model point to be traced and stores them in registers Z1 and Z2. The starting position of the edge picture segment to be used as operands for the current operation is computed by adding xoffset and yoffset to Z1 and Z2, respectively. Then, the set of operations listed in Algorithm 4.1 is performed in Step 3. When all model points for the current orientation are processed, a subarray of the 2D accumulator in each PE is moved to the main memory. After the contents of all subarrays are moved to the main memory, the subarray in a

Algorithm 4.1

```
/* Receive the relative position of the current model point (Z1,Z2)
from the control unit and add offset values to the relative position. */
R1 = Z1 + xoffset;
R2 = Z2 + yoffset;
/* Add an m by m edge picture window starting at (R1,R2) to an m by m
accumulator window assigned to the PE */
for i = 1 to m
    begin
    R3 = R2;
    for j = 1 to m
        begin
        A(i,j) = A(i,j) + P(R1,R3);
        R3 = R3+1;
        end;
    R1 = R1+1;
    end;
```

PE is set to 0 and a new subarray is computed for other possible orientations. If a PE has enough memory to hold subarrays computed for all possible orientations, it is not necessary to move the contents of the subarray for each orientation to the main memory. Maxima in the 3D accumulator can be found directly in 3D subarrays in local memories by using E PEs.

4.1.2 Performance. The time spent performing computations in a SIMD structure consisting of E PEs is

$$P_E(M) = s_0 TM^2/E \quad (4.1)$$

(or Tm^2 if $m^2 = M^2/E$)

where T is the number of points used to represent the model and s_0 is the time spent to update one accumulator cell in Algorithm 4.1. The overhead required to manage the parallelism $O_E(M)$ is sum of the time spent to broadcast the relative position of boundary points of a model to all PEs, O_b, and the time spent to set up operand windows for each of the PEs, O_s.

$$O_E(M) = Ts_1 \quad (4.2)$$

where $s_1 = O_b + O_s$. In general, s_0 will be much larger than s_1. The execution time of the algorithm 4.1 on a system with E PEs is the sum of the overhead and the time spent performing actual computations.

$$T_E(M) = P_E(M) + O_E(M) = T(s_0M^2/E + s_1). \quad (4.3)$$

Other performance measures defined in chapter I are given by the equation (4.4) through (4.8):

$$S_E(M) = T_1(M)/T_E(M) = C_0EM^2/(s_0M^2 + s_1E) \quad (4.4)$$

where C_0 is the time spent to complete one operation in a sequential computer;

$$V_E(M) = M/T_E(M) = ME/(s_0TM^2 + Ts_1). \quad (4.5)$$

When the number of parameters assigned to a PE is large, the effect of the overhead on the total execution time is minimal. For a large m, the speed up S_E and the efficiency E_E are close to E and 1, respectively. If the number of PEs

increases, m should be decreased. Thus, the performance of the system reaches a point of diminishing returns when the number of PEs increases. However, the reformulated algorithm can be implemented very efficiently in the SIMD structure with a large local memory without contentions for the same operands.

Another important performance measurement for the parallel architecture is the cost effectiveness $C_E(M) = V_E(M)/c_E$, where c_E is the cost of the system. The cost of the system is approximately the cost of the control unit(CU), E times the cost of a PE, the design cost, and the cost of the interface network among PEs and CU. For a large E, the cost of PEs will be a dominating factor. Specifically, in the SIMD structure with large local memory the cost of PEs will be significantly greater than that in the structure with small local memory. Since our goal is computing the Hough transform detecting arbitrary shape in real time by using a large number of PEs, the cost of a PE must be small. Furthermore, the structure of each PE must be simple so that a large number of PEs can be implemented in a single chip by taking advantage of VLSI technology. However, the SIMD structure discussed is very flexible. PEs in this structure can be used not only to compute a parameter array but also to find maxima in the parameter array.

4.2 Mesh Connected Structure (MCS)

As mentioned in chapter III, Algorithm 3.2 is equivalent to adding a translated edge picture to an accumulator array for each model point in the table. For a model point (x_i,y_i), the edge picture **P** is translated $-x_i$ in the x direction and $-y_i$ in the y direction and added to the accumulator. Thus, if there is an N by N shift register array where the data in a cell can be passed on to one of its four neighbors, the translation of the edge picture in this array will be simple

and fast. This is the idea behind the MCS. In the MCS, processing elements are interconnected with the near-neighbor network so that edge data in a PE can be sent or received from one of its four neighbors. The PE at (i,j) in the 2D network is assigned to the accumulator cell M(i,j). After an edge picture in the processor array is translated to the desirable position for a given model point, the contents of a data register in each PE is added to the accumulator cell assigned to the PE.

The MCS designed for an N by N image consists of N^2 PEs. The system for a 3 by 3 image is illustrated in Figure 27. Figure 28 shows the definitions of symbols used in Figure 27. In the MCS, one bit of data representing an edge pixel can be shifted to one of its four neighbors. When the data are shifted, the data shifted out from the PEs on one end are wrapped around to the PEs on the other end. A portion of the PEs which are not used for a certain operation are disabled by the row enable and column enable signals from the row mask bit shift register (RMBSR) and the column mask bit shift register (CMBSR). The RMBSR and CMBSR are bi-directional shift registers which consist of linearly connected N shift cells. We are assuming that the image frame is N by N square. Data in the CMBSR can be shifted right or left depending on the direction signals from the control unit (CU). Both registers are set to high when the signal Reset from the CU is high.

An edge picture shifts into the system through 2-to-1-multiplexers. The input load (IL) signal from the CU connects N input bus lines to N south input (SI) lines of PEs on the bottom row of the 2D array processors. The following N shift operations in the north direction will move an N by N edge picture into the system. After all data are shifted into the system, the contents of both mask registers are set to high. Whenever the data are shifted, the contents of the CMBSR and RMBSR are shifted to the same direction that the data are being

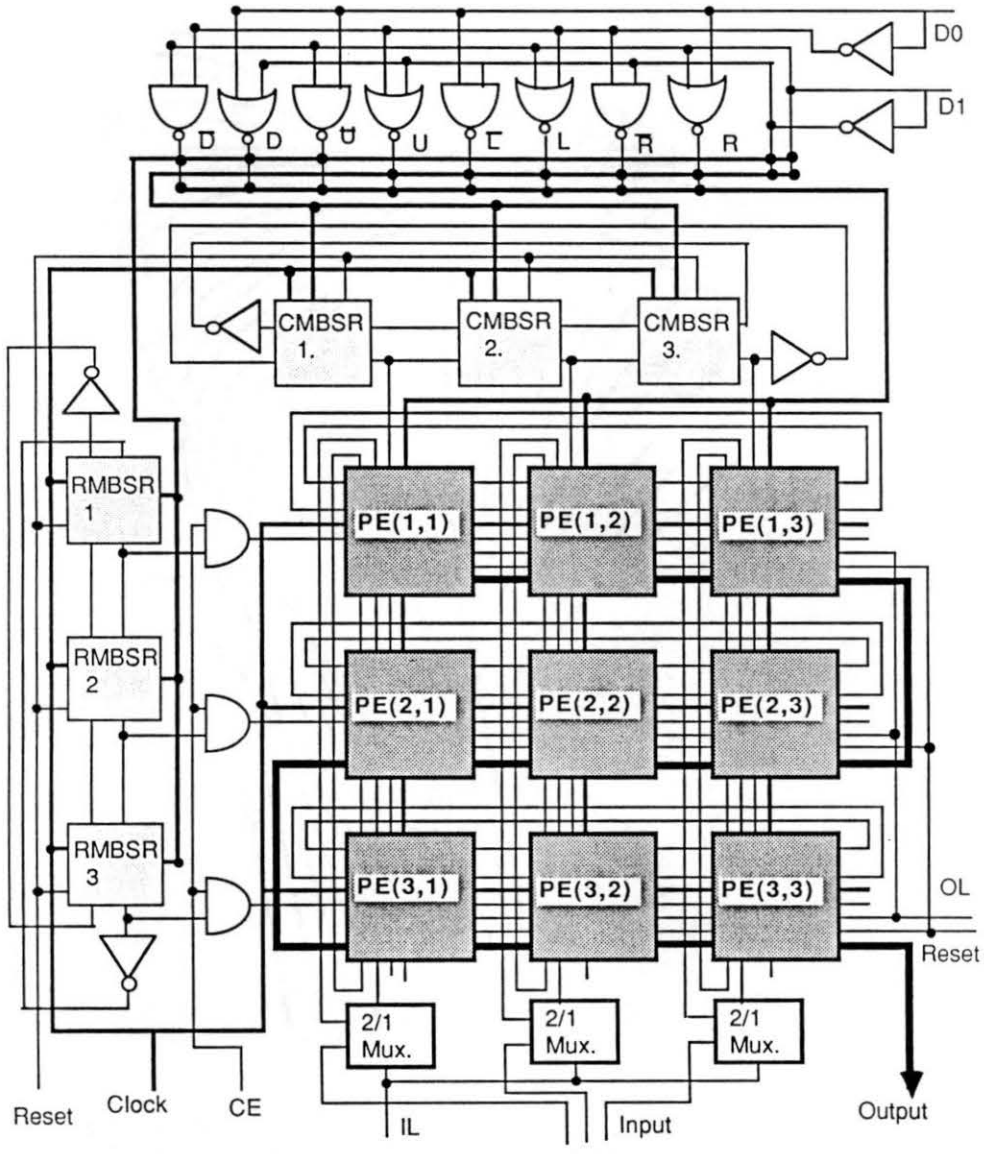
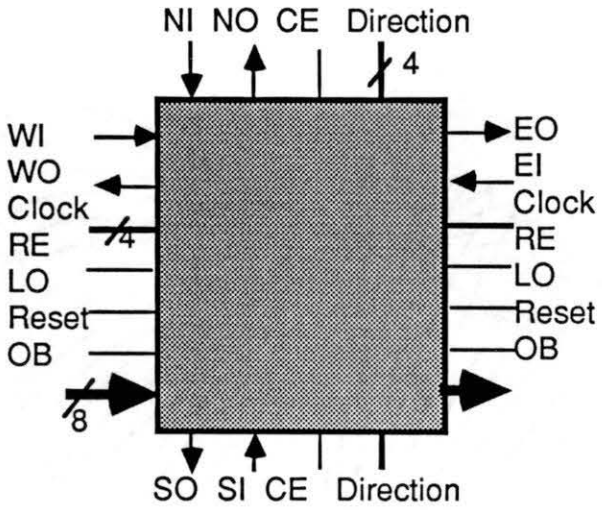
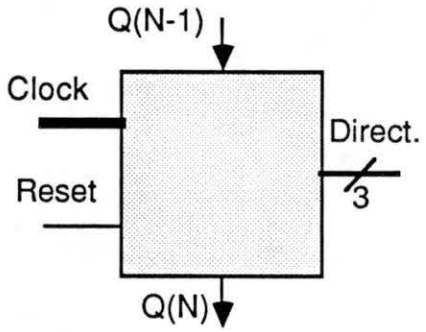


Figure 27. Overall Structure of the MCS



Processing Element



Mask Bit Shift Register Cell

RE : Row Enable
CE : Column Enable
OB : Output Bus
OL : Output Load
IL : Input Load
WI : West In
WO : West Out

EI : East In
EO : East Out
NI : North In
NO : North Out
SI : South In
SO : South Out

Signals

Figure 28. Symbol Definition for the MCS

shifted. If the data are shifted east or west, the CMBSR is also shifted east or west, while the RMBSR is unchanged. In the same manner, the RMBSR is shifted north or south when the data is shifted north or south. A mask bit shifted out from one end is inverted and fed back to the other end. Thus, if a column of the PE array contains wrapped around portion of an edge picture, the contents of the mask bit representing the column will be 0. If the output of the i'th RMBSR cell is low, then the i'th row will be disabled. In the same manner, if the output of the i'th CMBSR cell is low, the i'th column will be disabled. In this way, the image data is preserved during shifts but only the appropriate part contributes to the accumulator array.

Figure 29 shows a PE, a RMBSR cell, and a CMBSR cell for the MCS. The CMBSR and RMBSR are not a part of the PE in the MCS. Only one RMBSR cell or CMBSR cell is necessary for a row or a column. A PE, shown in Figure 29, consists of one **c**-bit counter, a **c**-bit output shift register (OSR), a one-bit data register, a count up generation logic (CGL), and a routing network. Since an element in the edge picture is represented by one bit, the data register can hold one edge pixel. The register can receive one bit of data from and send to any of its four neighbors through the routing network. The routing network is a 4-to-1-multiplexer which connects one of its four input lines to an output line depending on two bit direction signals from the CU. The CGL is a three input AND gate. Its three inputs are the data bit from the data register, the row enable from the RMBSR cell, and the column enable from the CMBSR cell. When the count up signal from the CGL is high, one is added to the counter.

The counter is a **c**-bit ripple counter with a reset option. The gate level description of a one-bit counter cell and the block diagram of an 8-bit counter are shown in Figure 30. A counter represents an accumulator cell assigned to the PE. The size of the counter will limit the maximum number of boundary

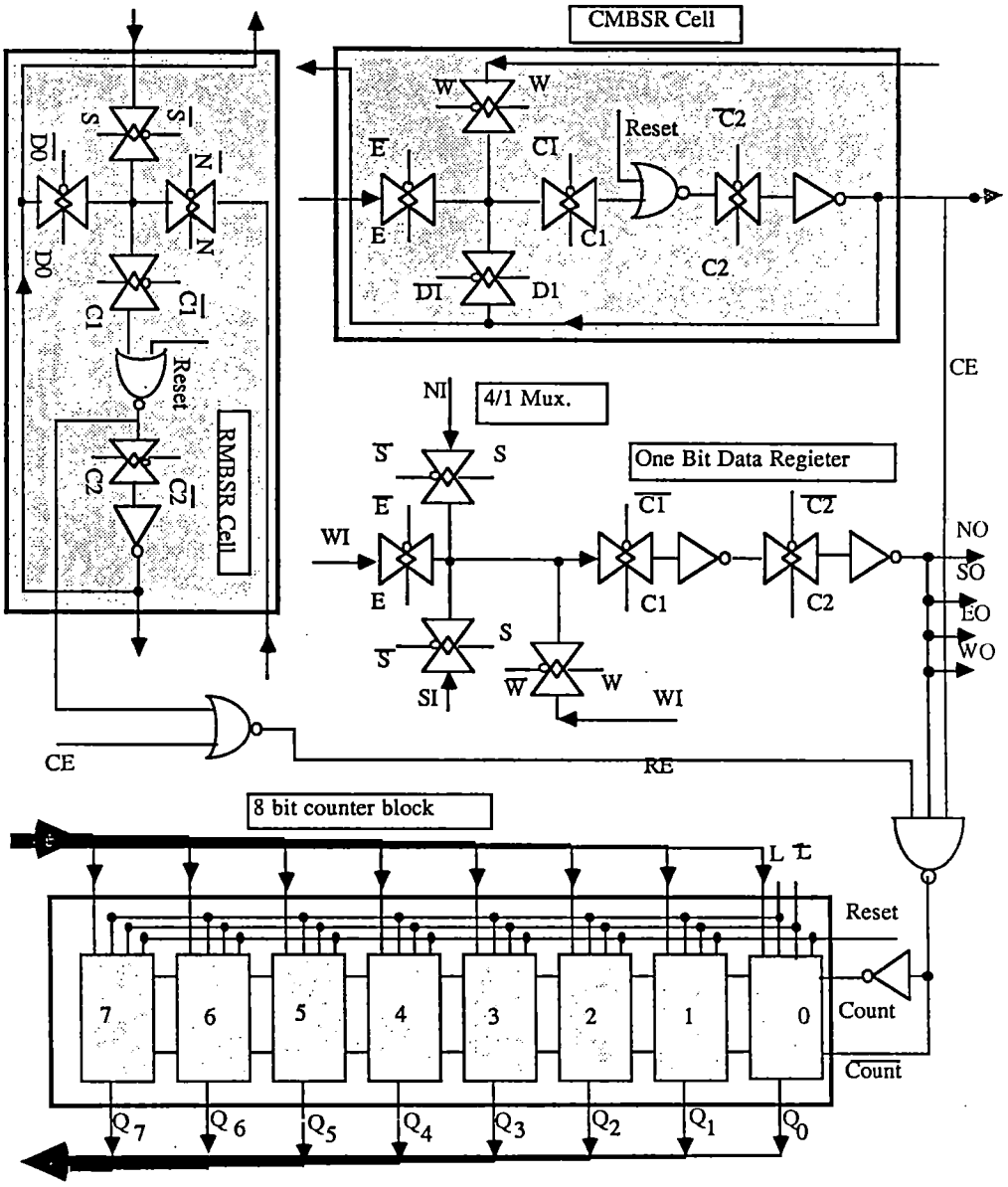
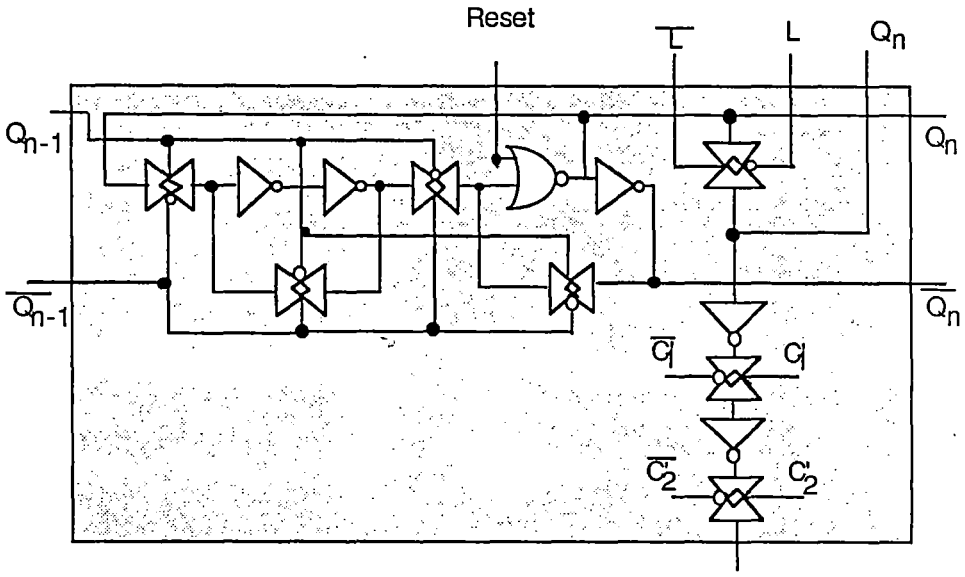
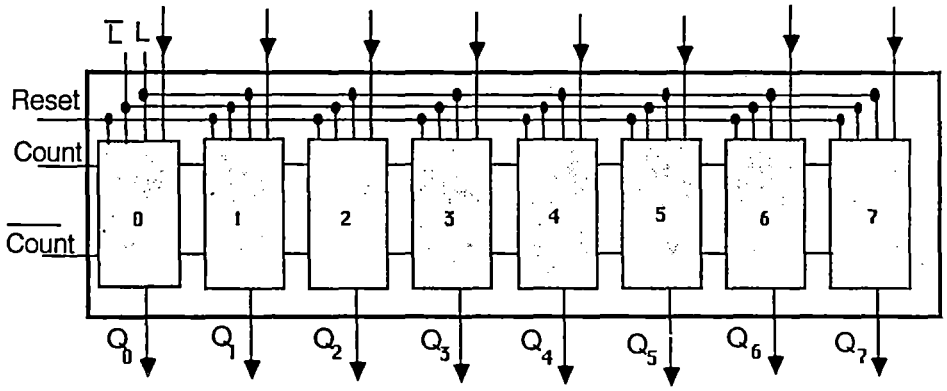


Figure 29. A Processing Element with Mask Bit Registers



a) One Bit Asynchronous Counter with an One Bit Output Shift Register.



b) The Block Diagram of an 8 Bit Counter with Output Shift Registers.

Figure 30. An Eight Bit Asyncronous Ripple Counter

points which can be used to describe a model. Because the number of transistors required for a PE is mainly determined by the size of the counter, the counter also determines the number of PEs per chip. For example, if **c** = 8, the maximum number of model points is 256. If the orientation of an object is fixed so that it can be detected by computing a 2D accumulator, an 8 bit counter will be large enough for most applications. However, If we use the MCS to compute the HASAO discussed in section 3.2, it may be necessary to have a counter with more than 8 bits. The ripple carry propagation time of the counter is not critical in this structure because the output is accessed only after all computation is done.

After all model points are processed, the contents of each accumulator cell is loaded into the OSR. The information in the OSR is shifted out while new input is shifted into the system. All shift operations including the mask bit, data bit, and output are controlled by the clock signal from the CU. The shift operation can be disabled by disabling the clock signal.

The mask bit shift register in Figure 29 is a one-bit shift register with a 3-to-1-multiplexer for the input. This register is set to 1 when the reset signal from the CU is high. It receives one bit data from three sources. For the column mask bit shift register, it can receive the data from its left or right neighbor when the data are shifted west or east. When the data are shifted north or south, the column mask bit shift register will hold its current status by feeding its output back to its input.

4.2.1 Computation. After an image is loaded into the system, the RMBSR and CMBSR are set to 1 and all counters are cleared. To update all accumulator cells for the first model point $(-x_1,-y_1)$, the image is translated by x_1 pixels in the x direction and by y_1 pixels in the y direction. Then the CU issues the count enable command. For the subsequent model point $(-x_2,-y_2)$, the

image is translated by $|x_1 - x_2|$ in the x direction and by $|y_1 - y_2|$ in the y direction. When the image is shifted horizontally, contents of the RMBSR are also shifted in the same manner with the exception that the signal shifted out from one end is inverted and fed back to the other end. The contents of the RMBSR are shifted likewise when the image is shifted vertically. The signals from the RMBSR and CMBSR disable the PEs having picture elements wrapped around. Figure 31 shows the arrangement of edge data in the 256 by 256 PE array when a count up command is issued for the model point (-100,-128). The PEs with data points in the highlighted area will be enabled for the count up operation.

A functional simulation of the MCS has been implemented in the IRI D256 Computer Vision Systems in C. The IRI D256 system has a translation function built into its coprocessor. This function translates an image in four directions: north, south, east, and west. The portion of the image shifted out from one end can be shifted into the other end. By using this routine, the edge picture is translated for each model point. The wrapped around portion of the image is disabled by setting pixels in this portion to zero. Since only edge elements in the picture have a value of 1, adding this translated image to the accumulator is equivalent to the count up operation in the MCS. In fact, the same procedure has been used to compute the HNGD in the section 2.3. In other words, the results for the HNGD given in section 2.3 were obtained by the simulation of the MCS.

4.2.2 Hardware Implementation. When we consider the actual implementation of this MCS, the most important factor affecting the performance of the system is the number of I/O pins available in a chip. The number of transistors required for a PE with an 8 bit counter is about 250, which is small enough to integrate a large number of PEs in a single chip. However, the MCS may not be able to take advantage of the simplicity of a PE if the size of the desirable system

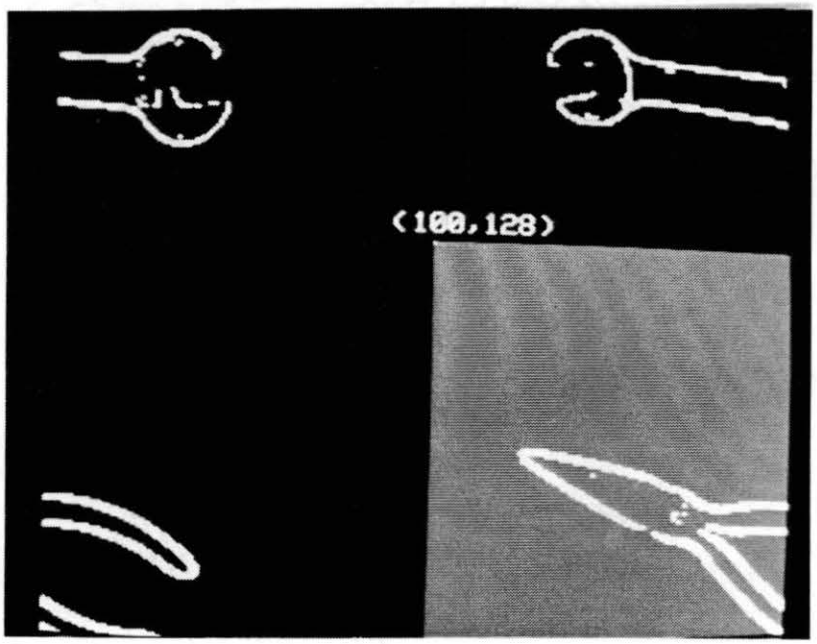


Figure 31. Arrangement of the Edge Picture in the Model Point (-104,-128)

is too large to implement in a single VLSI chip. In such a case, the system must be built by connecting several chips each of which contains a block of PEs. Thus, a chip must have enough I/O pins to connect a PE in a chip with its four neighbors. For example, if a chip contains m^2 PEs, the number of I/O lines required to form a proper communications network among PEs in other chips will be at least 8m lines. It means that the number of PEs per chip is limited by the number of I/O pins available in a chip. If the size of the system for a given application is small enough so that it can be implemented in a single chip, the number of interconnections among PEs is no longer the major limiting factor. The size of the system which can fit in a VLSI chip is only limited by VLSI technology. We will discuss these two systems: a single chip mesh connected structure (SCMCS) and multiple chip mesh connected structure (MCMCS), separately.

The block diagram of the SCMCS and MCMCS are shown in Figure 27 and 32, respectively. In the MCMCS consisting of multiple building blocks, the number of I/O pins per chip required to interconnect the blocks is 8m. For example, the number of I/O lines per chip required to interconnect 16 by 16 PE blocks without considering control lines is 128. If the row mask bits and column mask bits are added, the number will be 160. To reduce the number of pins required for mask bits, each PE block contains its own RMBSR and CMBSR. In this structure, the number of pins required to connect mask bit shift registers in a PE block to mask bit shift registers in other PE blocks is reduced to 4. To construct the data path among PEs with the minimum number of I/O pins, PEs on the boundary of a PE block interchange the data with PEs in other blocks through bi-directional switches. The direction of the switches is controlled by the direction signal from the CU. Now, the number of I/O pins required for data and mask bits is reduced to 4(m+1) for an m by m PE block.

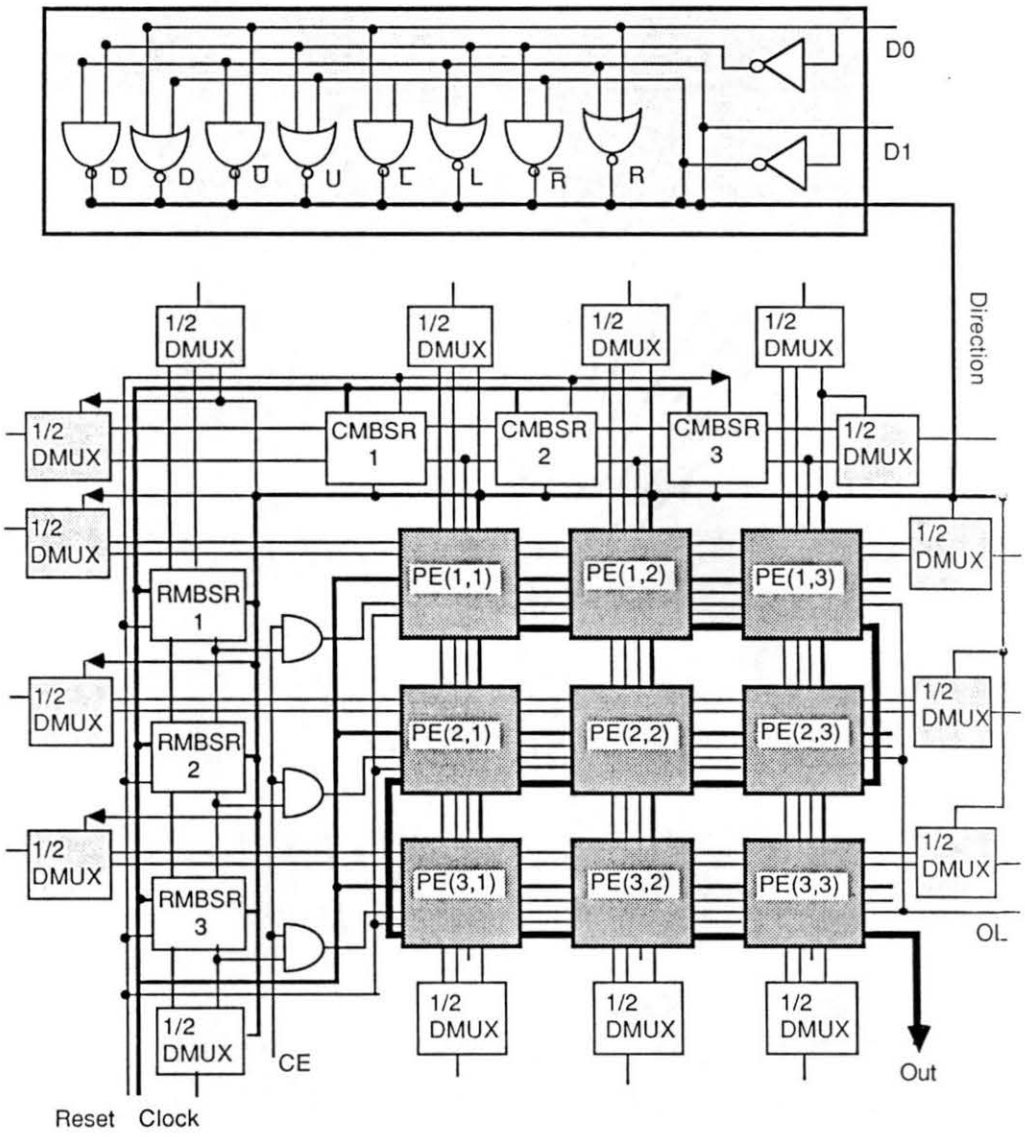


Figure 32. A Building Block of the MCMCS

The performance of the MCMCS and SCMCS depend on the number of lines designated for output and input. Both systems can compute the Hough transform in time proportional to T, where T is the number of model points. However, the time spent to load and unload data from the system may be a more significant factor than the computation time in the MCS. This problem is more significant in the SCMCS than the MCMCS. In the SCMCS, the data must be loaded and retrieved from the system through the small number of I/O pins available in a single chip. In the MCMCS, each PE block can have its own input and output data bus. However, a large price must be paid for the large I/O bandwidth.

In the SCMCS the total number of control lines from the CU is eight including Reset, Direction(2), input load (IL), count enable (CE), output load (OL), input/ouput (IO), and Clock(2). If the number of pins available per chip is P, (P-8) pins can be used to load and retrieve the data from the system. All (P-8) pins will be used as either an input or output bus depending on the IO signal. (However, the circuit necessary for the bus sharing is omitted in Figure 27 and 32.). Thus, the time required to load the edge picture to the system is proportional to $N^2/(p-8)$ and the time required to unload the accumulator is proportional to $\mathbf{c}N^2/(P-8)$, where N is the size of the edge picture and **c** is the size of the counter.

In the MCMCS, control signals from the CU are the same as those in the SCMCS. However, a part of the pins used to send data among chips are also used as the output data bus to unload counters. The input data and output data can be loaded or unloaded from each PE block through its own I/O bus. If the size of the PE block in a chip is m by m, the time required to load the edge picture to the system and the time required to unload counters will be as small as $m^2/(P-8)$ and $\mathbf{c}m^2/(P-8)$, respectively. However, it requires more complex

circuitry to accommodate the high data rate from each PE block. The number of I/O lines must be determined by considering the system requirements such as cost and speed.

The estimated number of transistors for a PE with an 8 bit counter is about 250. Since the current VLSI technology makes it possible to implement 10^6 transistors per chip, it is possible to implement 64^2 PEs in a single chip. The number of PEs in a single chip is expected to be increased by a factor of four by the end of this decade. Thus, a practical size of the SCMCS will be possible within the next few years. Though the size of the SCMCS is expected to increase greatly, the number of pins available for a VLSI chip will not be improved as much as the size of the PE array. It means that the time required for loading and unloading data will be the major limiting factor in the SCMCS. This problem can be alleviated by adapting the approach discussed in the section 3.2. On the other hand, the MCMCS can not take advantage of the advanced VLSI technology as much as the SCMCS can. The number of PEs in the MCMCS chip is limited by the number of pins on a chip. For the VLSI chip with P pins, the maximum number of MCMCS PE blocks that can be fit into the chip is (P-12)/4. If a chip has 78 pins, the number of the PE blocks that fit in the chip is 16^2. However, the size of the system can be made arbitrarily large by connecting as many MCMCS chips as are needed. Since each MCMCS chip can have its own I/O bus, the time required to load and unload data is no longer problem in the MCMCS although a larger system must pay a bigger price for the I/O circuits.

4.2.3 Performance. The time spent performing computations which are actually a part of the task depends on the boundary length of the model in the MCS. If the model is described by a set of points: $\{ (x_i,y_i) \mid i=1,T\}$, it can be expressed by the following equation.

$$P_E(M) = s_0\Sigma(|x_i - x_{i+1}| + |y_i - y_{i+1}|) , i = 1, T \quad (4.6)$$

where s_0 is the time taken for one shift operation. E represents the number of PEs. In the MCS, the number of PEs is N^2. If we define a new variable B to denote the boundary length of the model, then the equation (4.6) can be rewritten by

$$P_E(N) = s_0B \quad (4.7)$$

The overhead required to manage the parallelism is the time required to broadcast the control signals to all the PEs and it can be represented by

$$O_E(N) = s_1B \quad (4.8)$$

where s_1 is the time required to broadcast the relative position of a model point and control signals. However, broadcasting control signals and data and shifting operation are done in a single clock cycle. The clock period will be determined depending on the constant s_0 and s_1. If the clock period is represented by s_c, the execution time of the algorithm in the MCS is

$$T_E(N) = s_cB \quad (4.9)$$

Other performance measures including speed (V_E), speed up (S_E), efficiency (E_E), utilization (U_E), and cost effectiveness (C_E) are given by following four equations.

$$V_E(N) = N^2/s_cB \quad (4.10)$$

$$S_E(N) = s_sTN^2/ s_cB \quad (4.11)$$

$$C_E(N) = V_N(N)/c_E \quad (4.12)$$

s_s in the equation (4.11) is the time required to process one data point in a sequential computer and c_E in the equation (4.12) is the cost of the system.

The length of the boundary B is proportional to T. If the model points are continuous, the maximum distance between two points is $\sqrt{2}$. In this case, the speed up is $S_E = s_sN^2/\sqrt{2}s_c$. Since shifting and counting are done in a single clock cycle in the MCS, s_2 will be much bigger than s_c. It means that the speed

up will be much bigger than N^2. Another important performance measure of the special purpose system is the cost effectiveness. As mentioned in chapter I, the major portion of the cost of special purpose systems is the design cost. In the MCS as well as the LCS to be discussed in the next section, a whole system can be build by connecting copies of a PE. Thus, the design cost can be minimized. Since a PE consists of a small number of transistors, a large number of PEs can be implemented in a chip. All these factors will contribute to reducing the system cost. The cost of the SCMCS is much smaller than that of the MCMCS since it does not require any interconnection circuits.

If an L-bit common bus is used for the input and output, the I/O time can be expressed by

$$I(N) = s_b(1+\mathbf{c})N^2/L \qquad (4.13)$$

where s_b is the clock period running the bus and **c** is the number of bits used to represent an accumulator cell. In the SCMCS, the number of I/O lines L is limited by the number of pins available on a single chip. But, in the MCMCS, it can be increased by adding I/O circuits to every building block. However, increasing the number of I/O lines will also increase the cost and complexities of the system. As a matter of fact, a special purpose system with a multibus structure is not practical for current industrial computer vision systems. Without expensive supporting devices, current computer vision systems can not take advantage of such a system. Even if the supporting devices are available, using such a special purpose system for industrial computer vision systems will not be economically feasible.

More practical choice for industrial computer vision systems will be the special purpose systems with a single bus structure which does not require extra hardware for I/O operations. If we use a one-bit input data bus and a **c**-bit output data bus, the I/O time will be proportional to s_bN^2 , which is a dominating factor

in the total computation time. In this case, the speed of the SCMCS will be almost equal to that of the MCMCS.

4.2 Linearly Connected Structure

The mesh connected structure discussed in the previous section can compute the Hough transform in a time proportional to T. But it requires complex I/O circuitry to supply data to and retrieve data from the system. It also requires a large number of I/O pins which is the major limiting factor in the VLSI implementation. Specifically, in the MCMCS, the number of I/O pins available on a chip determines the number of PE blocks per chip. Thus, the MCMCS can not take advantage of advanced VLSI technology. In general large a number of pins in a package means high packaging cost and large system size.

The linearly connected structure (LCS) is not as fast as the MCS, but it has other important advantages over the MCS for simple VLSI implementation. Since it has only one data path, a PE block can be connected to other PE blocks with a small number of I/O pins. The number of PE blocks in a chip is determined only by the maximum number of transistors which fit in a chip and the packaging and interconnection cost of an LCS chip containing a PE block will be small compared to that of the MCS.

The block diagram of this structure and the description of a PE are given in Figure 33 and Figure 34, respectively. In the LCS, PEs are connected as a linear array and one bit of data in a PE can be passed only to its right neighbors. The picture data is loaded into the system one pixel at a time through the first PE in the first row. Thus, the time required to compute the Hough transform in the LCS is proportional to N^2.

A PE consists of a one-bit data register, a one-bit row mask bit shift register (RMBSR), a **c**-bit ripple counter, **c** one-bit output shift registers (OSR), and a

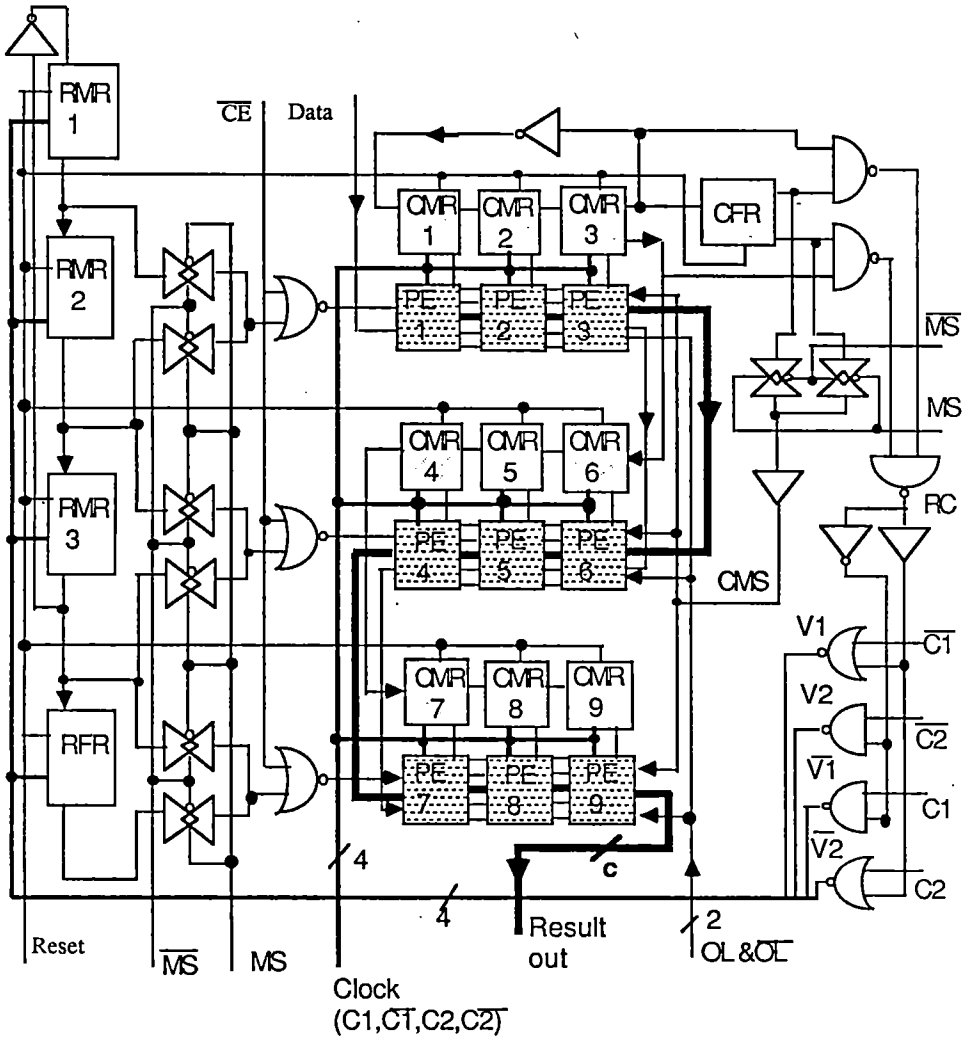


Figure 33. Block Diagram of the LCS

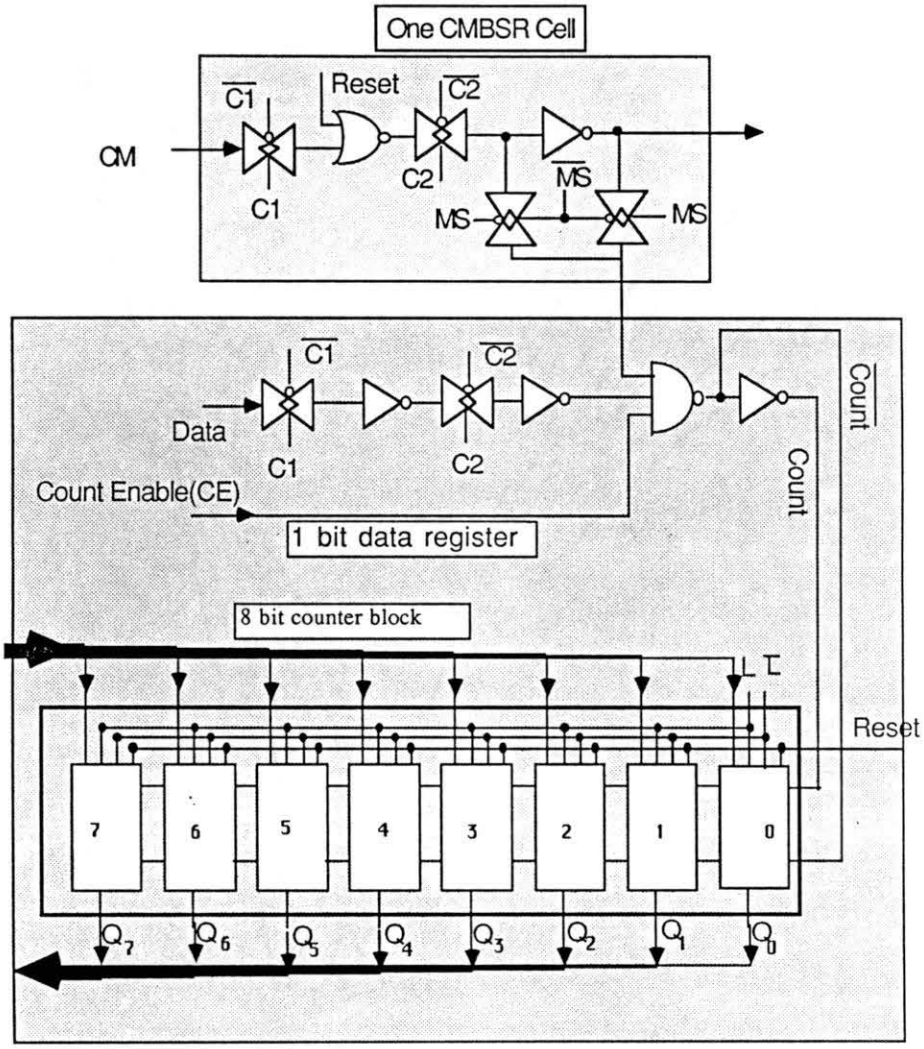


Figure 34. A Processing Element for the LCS

count up generation logic (CGL). The block diagram of a PE is shown in Figure 34. The counter used in the LCS is identical to the counter shown in Figure 30. The size of the PE is also determined by the size of the counter. The number of transistors required for a PE with an 8 bit counter is approximately 254. A PE in the LCS is basically the same as that in the MCS. One difference is that the LCS has its own CMBSR representing the status of the current data element in the data register. The content of the CMBSR is shifted whenever the data bit is shifted.

Figure 33 illustrates the overall structure of the LCS for a 3 by 3 image. In this structure, data bits and column mask bits are shifted one step per clock cycle in the same direction. A data bit shifted out from the last column of a row is fed to the first column of the next row, and a column mask bit shifted out from the last column of a row is fed to the first column of the next row. The column mask bit shifted out from the first row is also fed to the column flag register (CFR). The signal fed to the first column mask register of the next row is the inverse of the signal shifted out from the last column mask bit register of the current row.

Two phase clock signals from the controller are used to shift the CMBSR and data bit registers. When the shift enable (SE) from the CU is low, the shift operation can be disabled by disabling the clock signals. The clock disable circuit can be implemented either as a part of the building block or as a separate device. If it is implemented as a separate device, an I/O pin for the SE signal will not be necessary in each building block. The row mask bit registers are shifted when the first column of a new row of edge picture is shifted in. To generate the row change (RC) signal automatically, the column flag register (CFR) is used. This register receives the data from the last column of the first row, which means that it holds the column mask bit shifted out from the first column for one clock cycle. One difference between the CFR and other CMBSRs is that the CFR is set

high while others are set low when the reset from the CU is high. Thus, whenever a new row of an edge picture is fed into the system, the contents of the last CMBSR in the first row and the contents of the CFR are not the same. Thus, the RC can be generated By passing these two signals through an EXCLUSIVE OR circuit. This circuit is shown in the upper right hand corner of Figure 33. The RC enables the clock signals from the CU for one clock period to shift the RMBSRs. The row mask bit shifted out from the last row is inverted and fed back to the RMBSR in the first row. It is also fed to the row flag register (RFR).

Like the MCS, counters in a certain portion of the PE array are disabled for a given count up operation. The PEs to be disabled are determined by the data arrangement among PEs at the time of the count-up operation and the relative position of the model point being updated with respect to the reference point. The CU issues the mask select (MS) signal along with the CE signal for the count up operation. The MS signal and the row and column mask bits are used to determine the portion of PEs to be disabled. A detailed description of the MS signal will be given in the next section.

Unlike the MCS, the i'th row mask bit is not necessarily used to enable the i'th row of the PE array in the LCS. Depending on the MS from the CU, the enable signal for the i'th row is selected between the i'th and (i+1)'th row mask bit. The RFR is necessary for this operation. If MS=1, the (i+1)'th mask bit will selected. Otherwise, the i'th mask bit will be selected. If the selected signal and the count enable (CE) signal from the CU are both high, then the i'th row will be enabled for the count up operation. This row enable signal is denoted as RE in Figure 33.

The enable signal for the i'th column is selected between the mask bit from the i'th CMBSR and the inverse of the mask bit depending on the column mask selection signal (CMS). The CMS is generate by using the MS and the signal

from the CFR. If MS = 1, the flag bit in the CFR is selected as a CMS; otherwise, the inverse of the flag is selected as a CMS. The circuit generating the CMS is located under the EX-OR circuit in Figure 33. If CMS = 1, the mask bit in the i'th CMBSR will be used directly to enable the i'th column. Otherwise, the inverse signal of the i'th CMBSR is used.

Through the OSR, the parameters in the counters are unloaded. The output load (OL) from the CU will copy the content of a counter in a PE onto the OSR in the PE. When the OL is high, the shift operation is disabled for one clock cycle. Each subsequent shift operation will shift out one parameter from each PE block.

4.3.1 Computation. The idea behind the LCS is basically the same as that behind the MCS. As we discussed in the previous chapter, the Hough transform for a given edge picture can be computed by adding a translated edge picture for each model point to an accumulator. For example, for a model point (x_i,y_i), the edge picture is translated $-x_i$ in the x direction and $-y_i$ in the y direction, and a portion of the translated picture is added to the accumulator. During the translation operation, the picture elements shifted out from the image frame are discarded and the elements shifted into the image frame are set to zero. For the model point (x_i,y_i), column $(N-y_i)$ through N and row $(N-x_i)$ through N of the translated picture are set to zero by this procedure. In the MCS, data are shifted in four different directions to reach the right translation for a given model point. In the LCS, data are shifted in only one direction. That is the reason why the LCS is slower than the MCS.

To illustrate how Algorithm 3.2 is processed in the LCS, let us consider a 4 by 4 image given in Figure 35a. The picture contains a square consisting of eight boundary points. The model is shown in Figure 35b and the relative position of each boundary point with respect to the reference point R is given in

Table V.

a) Edge Picture b) Model

Figure 35. An Example for the LCS

TABLE V

RECTANGULAR COORDINATE SYSTEM REPRESENTATION OF THE REFERENCE TABLE FOR THE MODEL IN FIGURE 22

Model Pntr.	1	2	3	4	5	6	7	8
Position	(1,1)	(1,0)	(1,-1)	(0,1)	(0,-1)	(-1,1)	(-1,0)	(-1,1)

The LCS consists of 16 PEs arranged for a 4 by 4 image. PEs in the LCS are connected linearly but they are arranged as an N by N array for more clear illustration of row and column mask bits. Each row of the PE array is assigned to a corresponding row in the accumulator array.

The 4 by 4 input image is shifted into the system one pixel in a time

starting from the pixel at the last column of the last row. While the input image is shifted into the system, all the counters in the PE array are disabled until the portion of the data shifted in is aligned to appropriate PEs for a count up operation. When the data is aligned, the CU issues the count enable signal and the MS to each PE. Figure 36 show the arrangement of pixels in the system and control signals at the moment a count up is issued for each model point. Bits in the column heading of each table represent the column mask bits and bits in the row heading represent row mask bits at the moment when the counter-up command is issued for a given model point. An entry of the table represents the content of the data register of a PE. X represents "don't care". N and MS shown above each table is the number of shift operations required to reach the data arrangement for the model point and the mask bit selection signal which determines the portion of the PE block to be enabled, respectivly.

To generate control signals, the CU must know when all data points are aligned to appropriate PEs for a given model point and which part of the data will be disabled. The number of shift operations required to reach the right translation for the i'th model point (x_i,y_i) can be computed by using the following equation.

$$N_i = N(N-x_i+1) - y_i \qquad (4.14)$$

where N^2 is the size of the system. Thus, the CU issues a CE signal at the N_i'th shift operation for the i'th model point. Generation of the MS is simpler than that of the CE, but it is not easy to show how the MS is used to select the right portion of data for a given count up operation. Since the data point shifted out from the last column of a row is fed to the first column of the next row, a row of PEs may hold data elements belonging to more than one row of the input picture. If the first column of the (N-r)'th row is fed to the system, row 0 through r of the PE array will hold row N through N-r of the input image. If the c'th column of the r'th

row is fed to the system, row 0 through r of the PE array will hold picture elements belonging to two rows of the input picture. Figure 36 illustrates this for the given input picture. For example, Figure 36a shows the data arrangement when the second column of the input picture is fed to the system. In the first row in the table, the first three elements belong to the third row of the input picture and the last element belongs to the fourth row of the input picture. Figure 36a actually shows the moment that the CE signal is to be issued for model point 1. For model point 1, the PEs in the shaded area will be enabled for the count up operation. The MS signal determining the portion of data to be enabled is generated according to the following rule:

$MS = 0$, if $y_i \geq 0$

$MS = 1$, otherwise

TABLE VI

USE OF THE MASK SELECTION SIGNAL

MS	Column		Row
	CFB = 0	CFB=1	
0	CMB(i)	$\overline{CMB(i)}$	RMB(i)
1	$\overline{CMB(i)}$	CMB(i)	RMB(i+1)

* CFB : Column Flag Bit
CMB(i): Ouput of the i'th Column Mask Bit Register
RMB(i): Output of the i'th Row Mask Bit Register

As we can see in Figure 36, data pixels in a row of the LCS are divided

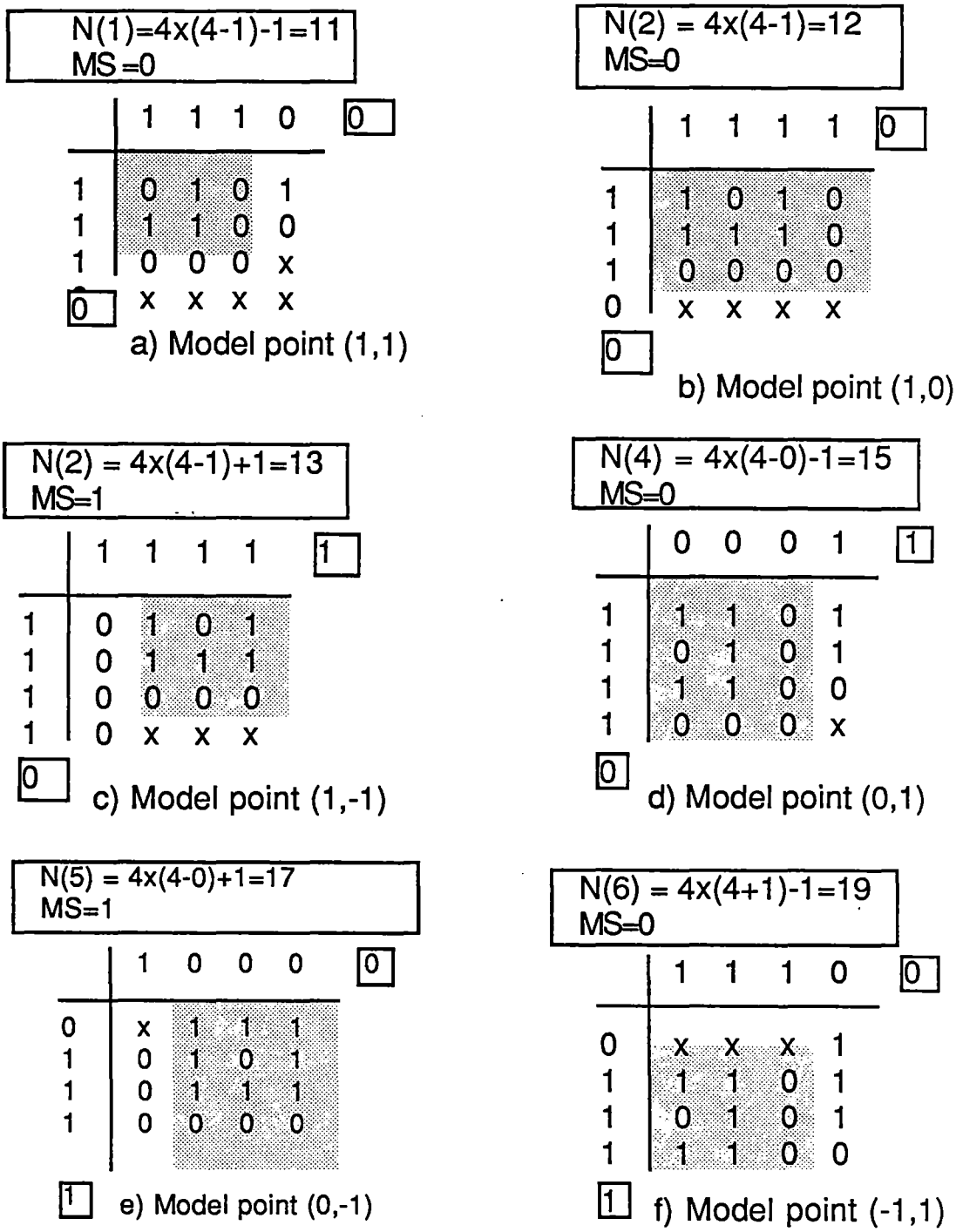


Figure 36. Arrangement of the Edge Picture and Control Signals Necessary for Each Model Point.

into two segments. Column mask bits on the table are used to separate these two groups. If MS=0, the first group in each row is enabled. Otherwise, the second group is enabled. Table VI shows how the MS is used to determine the portion of PEs to be enabled. To reduce the propagation delay in the selection circuit and the transmission delay, the MS is issued a half clock cycle earlier than the CE signal.

After a whole image is shifted into the system, the CU starts feeding 0s into the system until the pixel at the first column of the first row is shifted out from the system. The content of each counter is shifted out through output shift registers after all model points have been processed. To load parameters from the counters into output shift registers, the CU issues the output load (OL) signal to the LCS. When the OL is high, the shift operation is disabled for one clock cycle time. After parameters are loaded to output shift registers, a new input can be shifted into the system while parameters are shifted out from the system.

A functional simulation of the LCS has been implemented on the IRI D256 system and tested with both the machine generated image and the real image containing industrial parts. Unlike the MCS, the functional simulation of the LCS does not use any short cut. All the logic circuits except counters and register cells in the LCS are represented by independent subroutines. Counter and register cells are represented by 2 byte memory elements. Figure 37 shows the procedure of computing the Hough transform for the problem defined by Figure 35 and Table V. It shows the arrangement of the edge data and control signals right before the count up operation is performed for each model point. It also shows the contents of the accumulator array after the count up operation. The result shown in Figure 37 is an actual output of the simulation program for the shape given by Figure 35. An actual image containing nuts has been processed by using this simulation program. Figure 38a shows the original image and

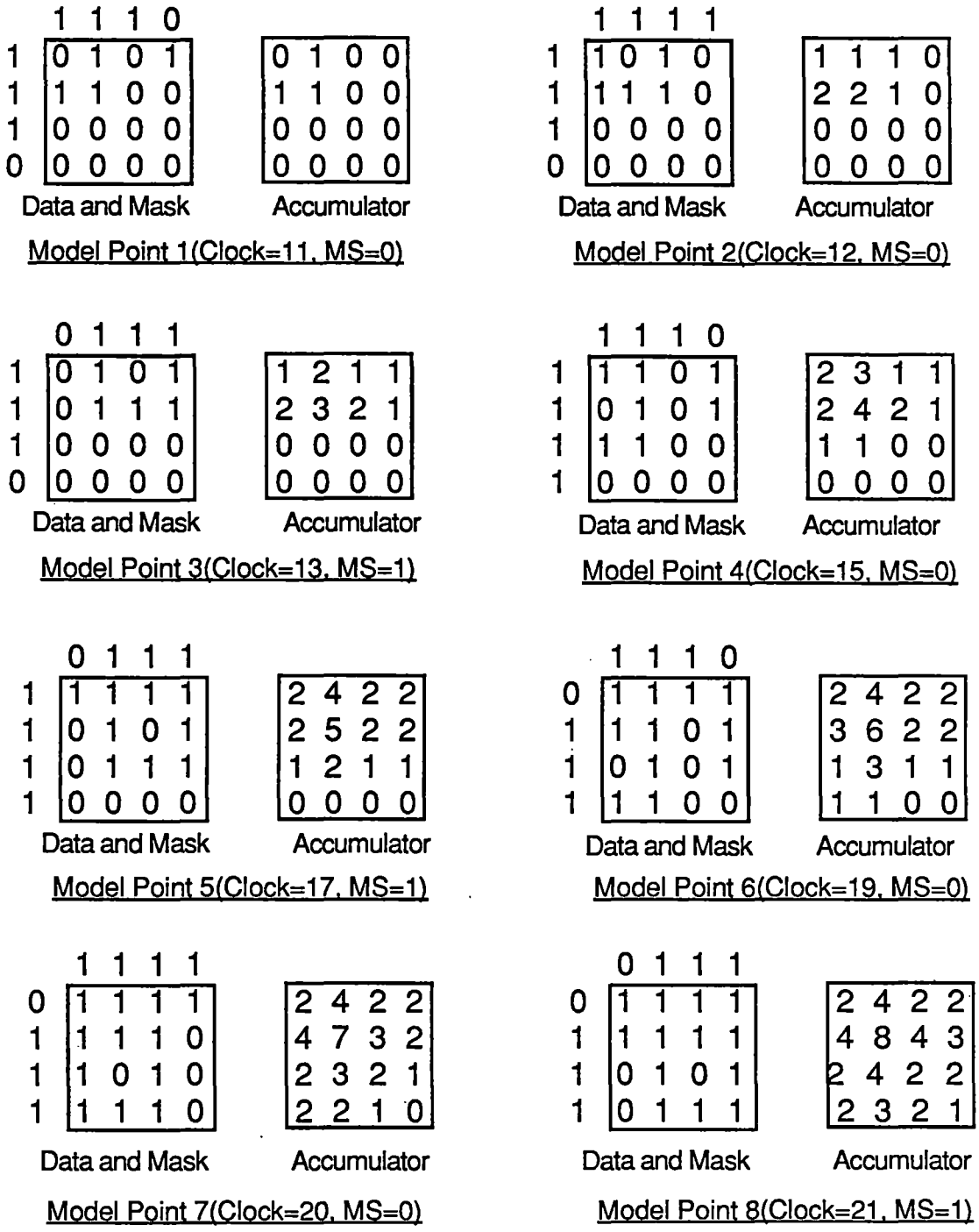


Figure 37. Arrangement of the Data and the Contents of Accumulator after Each Count-up Operation

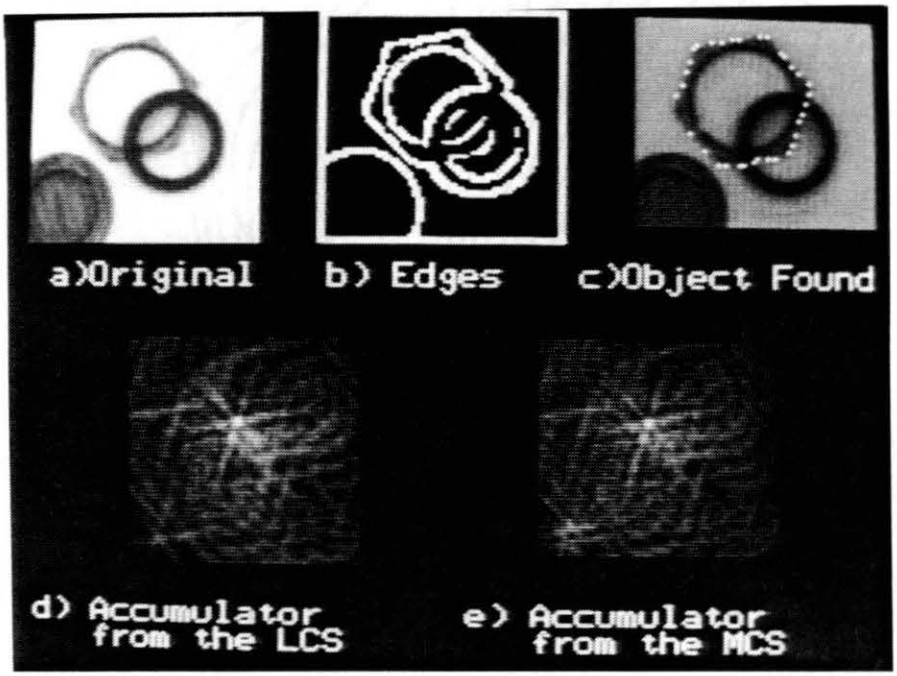


Figure 38. Results Obtained by the Computer Simulation of the LCS

Figure 38b shows the edge picture detected from the original. The contents of the accumulator array computed by the simulation program is shown in Figure 38d. The accumulator array computed by the LCS is identical to the accumulator array by the MCS which is shown in Figure 38e. The parameter detected from the accumulator in Figure 38d is displayed in Figure 38c. This experiment proves that the results from the LCS and the MCS are identical. Thus, the Hough transform computed by the LCS will have the same properties discussed in section 2.3.

4.3.2 Hardware Implementation. The LCS has two important advantages over the MCS. First, it can be easily adjusted for any size of image by adding simple building blocks. Second, the number of I/O pins required for each building block is small compared to the MCS. Thus, the number of building blocks in a chip is only limited by the density of the fabrication technology. In other words, the linearly connected structure will enable us to build more compact and less expensive systems for industrial computer vision systems than the mesh connected structure.

The gate level description of a PE is given in Figure 34. The number of transistors required for a PE with **c** bit counter is about 28**c**+24. With the currently available semiconductor fabrication technology (10^6 transistors per chip), the number of PE blocks in a chip is 2642 for the PE with an 8 bit counter. Thus, we can construct the LCS consisting of 256^2 PEs with 16 chips.

The number of pins required for an m by n building block varies with the packaging scheme. If the building block is packaged for the system with a fixed number of columns, then the building block needs (**c** + 10) pins, where **c** is the size of a counter. Ten pins are used for Reset, Clock, CE, OL, MS, Data In (DI), Data Out (DO), Row Mask In (RMI), and Row Mask Out (RMO). The RMI is the row mask bit shifted out from the upper block and the RMO is the row mask bit

shifted out from the current block. DO and DI connect the data path among three building blocks. If the building block is packaged for the system with an arbitrary number of columns, the building block requires (c + 12 + 2n), where n is the number of rows in the building block. In addition to those required for the building block with the fixed column size, it needs (2n+2) lines for DI, DO, Column Mask In (CMI), Column Mask Out (CMO), Row Change(RC), and Column Mask Selection. CMI and CMO are used to connect the last row mask bit register cell in the upper block and the first row mask bit register cell in the lower block. For a 256 by 16 building block consisting of PEs with an 8 bit counter, 52 pins are necessary. However, the second packaging scheme is not necessary for most industrial computer vision systems. The size of the image used in industrial vision systems is pretty much fixed. They usually have either 256 by 256 or 512 by 512 frame buffers.

4.3.3 Performance. In the LCS, the time needed to compute the Hough transform for an N by N image is proportional to N^2. The number of shift operations required to compute the Hough transform in the LCS may vary depending on the size and shape of the model, but it will not be larger than $2N^2$. If we do not consider the I/O time, the LCS is slower than the MCS where the same operation can be done in time proportional to B, where B is the length of the model boundary. However, the LCS has very important characteristics for the simple and inexpensive systems.

First, no extra time and circuits are necessary to load the edge picture into the system. In the LCS, an edge picture is loaded to the system one bit at a time. Thus, no extra hardware is necessary to feed the data into the system. The input time for the LCS is included in the computation time because the computation is performed while the input data are being shifted into the system. The contents of the accumulator array are shifted out from the system while a new edge picture

is being loaded. Thus, the LCS does not require any extra time to unload the accumulator array although the output is delayed one frame time (N^2).

Second the building block for the LCS needs a small number of pins so that the size and cost of the system will be small compared with the MCS. As mentioned in the previous section, the LCS for a 256 by 256 image can be built by connecting 16 chips with 18 pins.

Third, almost any size system can be built in this structure by connecting as many building blocks as are needed.

Fourth, since the execution speed is proportional to N^2, the LCS can compute the HASAO discussed in the section 3.2 with small additional cost. Let's assume that the orientation of an object is known with a possible error $\pm E^\circ$ and the model of the object is described with T boundary points. If we compute the HASAO to find this object in a N by N image, the number of computation is proportional to ET in the MCS and N^2 in the LCS. To compute the HASAO in these structures, a new reference table is generated by rotating the model for all possible orientations. For every possible orientation, the model is rotated and all boundary points of the rotated model are added to the table. In the LCS, if there is a model point duplicated d times, only the count-up operation is performed d times without shifting any operation to update the accumulator for the model point. The speed difference between these two structures will be smaller when T and E are getting bigger. This is very advantageous for a locating industrial part with arbitrary orientation. The constant execution time of the LCS is also important when a vision system is used as a feedback sensor for robotics.

Although the LCS executes the Hough transform in time proportional to N^2, it is fast enough for many industrial applications. Most systems in the current computer vision system market deal with 256 by 256 images. The LCS running

at 10 MHZ can compute the Hough transform for an 256 by 256 image in less than 14 milisecond. 10 MHZ is not an unrealistic value. Since most critical paths have eight gate delays, a higher clock rate should be possible.

If we compute performance measures for the LCS, the performance of the LCS may look very poor compared to the MCS. The execution time in the LCS is proportional to N^2 while the execution time in the MCS is proportional to B. However, the execution time in the MCS does not include the time required to load the data into the system, which will be the dominating factors in determining the total computation time and the cost of the system. The loading time can be reduced by increasing the number of input lines, but it increases the complexity and cost of the system. On the other hand, the LCS does not require extra time to load the data into the system.

In summary, by the figures representing the performance of the system, the MCS is superior to the LCS. But, for a small, inexpensive, and reasonably fast Hough transformer for an industrial computer vision system, the LCS is a better choice.

CHAPTER V

CONCLUSION AND SUGGESTIONS

5.1 Summary

In this thesis, two multiple specialized parallel structures are designed for efficient VLSI implementation of the Hough transform for arbitrary shape detection. The two structures-the mesh connected structure(MCS) and the linearly connected structure(LCS) promise compact and fast Hough transformers. These special purpose functional uints will boost the performance of the current computer vision systems for many industrial applications, such as bin picking and partially occluded parts location. The computer simulations show that these structures can compute the Hough transform as accurately as any sequential computer. To select the correct algorithm for these specialized structures, two types of the Hough transforms are studied in terms of efficiency and accuracy in detecting industrial parts and ease of parallel implementation: the Hough transform with gradient direction information (HWGD) and the Hough transform with no gradient direction information (HNGD).

The study shows that the HNGD is better suited for parallel implementation than the HWGD although its performance in detecting industrial parts is slightly inferior to that of the HWGD. The HNGD consists of a few simple operations for each data point, but the performance of this algorithm in a parallel machine is still limited by the I/O bandwidth of the system. To solve this problem, the HNGD is reformulated. Based on the reformulated algorithm, the two special purpose

functional units are developed for inexpensive and compact VLSI implementation.

The MCS can compute the Hough transform for an N by N image in time proportional to T, where T is the number of boundary points used to describe a model. With the advent of VLSI technology, this structure promises a compact single chip Hough transformer for a practical size image within a few years. By using many Hough transformer chips assigned for different tasks, we can construct a powerful and flexible system.

The LCS can compute the Hough transform for an N by N image in time proportional to N^2. Although the LCS is slower than the MCS, it can compute the Hough transform for a 256 by 256 image in real time and has very important characteristics for inexpensive parallel implementation. First, the LCS is not bound by the number of pins available in a chip. Thus, it can be implemented inexpensively and compactly by using existing VLSI technology. This system will add great power to current industral computer vision systems which suffer from lack of computation power for high-level and mid-level vision algorithms.

To take advantage of these specialized architectures (specifically for the LCS), we also developed an algorithm detecting objects with arbitrary orientations. Experimental results of this algorithm on industrial scenes are presented. The results show that it can be used to find an industiral part with unknown orientation in the image with little noise. This algorithm can be computed very efficiently in the LCS.

5.2 Suggestions for Further Research

5.2.1 Necessary Work for Actual VLSI Implementation. In this research, we prove that both the LCS and MCS will compute the Hough transform accurately; we also give system level descriptions of these structures for VLSI

implementation. But, much work remains to be done for actual VLSI implementaion.

First, the system must be defined more specifically. For example, the number of I/O lines must be determined by considering the speed and cost required for given tasks.

Second, although the way of generating control signals necessary for each structure is discussed, the description of the hardware generating these control signals are not given in this thesis. More work will be necessary to design an efficient control unit for each structure.

Third, to take advantage of the proposed functional units, ways of finding maxima in the computed accumulator array must be studied.

5.2.2 Suggestions. This thesis discusses two parallel structures computing the Hough transformation along with an algorithm taking advantage of these structures. Some extentions to the present effort are suggested.

1. The LCS can be extended for general 2D convolution by replacing the counter with an adder. Of course, the data register must be expaned to accommodate a multibit grayscale pixel. This structure may be significantly simpler than existing 2D convolver architectures if the size of templates are arbitrary large.

2. A hardware histogram generator attatched to the special purpose functional units discussed in this thesis will speed up the process of finding maxima in the accumulator array. The histogram generator can generate the histogram for the accumulator while the contents of the accumulator are shifting out from the functional uint. The histogram will give informations about the peak value and the number of possible peaks. The histogram information is useful when the special purpose functional unit is used to detect the object with arbitrary orientation. By comparing the peak values in histograms generated for

all possible orientations, the right orientation of the object can be determined without searching a 3D accumulator. The location of the object is then determined by searching the largest peak value in the 2D accumulator array computed for the correct orientation.

3. The histogram generator described above may give global information about the contents of accumulator arrays necessary for peak detection. If we are looking for only one object that matches best with the given model in the image, which is the case for bin picking problems, a specialized hardware can be built to find the peak in the accumulator array. A peak detection system can be designed by using a comparator and two registers: one for the maximum value and the other for the location of the maximun value. The comparator compares each parameter shifting out from the system with the current maximun value in the peak register. If a parameter from the system is larger than the current maximum value, the peak value and location in registers will be replaced by new values.

4. The 2D bit serial convolver which has the structure similar to the 2D convolver discussed in the section 1.6 may be used to compute the Hough transform. Such a system will be much slower and require a significantly larger amount of hardware than the LCS, but it can be also used to compute the grayscale convolution for large templates. If one needs a system capable of computing both the Hough transform and the grayscale convolution for a large template, this system may be a good choice.

5. The HASAO discussed in section 2.3 must be polished for practical use. Fisrt, a systematic way of determining an optimal set of model segments must be developed to minimize the performance degradation due to the noise from other objects. Second, more a sophisticated orientation determination algorithm using known information is necessary to minimize the effect of false peaks found

in 2D accumulators.

SELECTED BIBLIOGRAPHY

1. W. A. Perkins, "A Model-Based Vision System for Industrial Parts," IEEE Trans. on Computers, Vol. c-27, (February, 1978), pp. 126-143.

2. W. A. Perkins, "Simplified Model-Based Part Locator," IEEE Conference on Pattern Recognition and Image Processing, Vol. 1, (1978), pp. 260-263.

3. Mark W. Koch and R. L. Kashyap, " A Vision System to identify Occluded Industrial Parts," IEEE International Conf. on Robotics and Automation, (March, 1985), pp. 55-60.

4. P. V. C. Hough, "Method and Means for Recognizing Complex Patterns," U. S. Pattent 3,069,654, (Dec. 18, 1962).

5. R. O. Dudda and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," Communications of the ACM, Vol. 15, No. 1, (1972), pp. 11-15.

6. Frank O. O'Gorman and M. B. Clowes, "Finding Picture Edges Through Collinearity of Feature Points," IEEE Trans. on Computers, Vol. c-25, No. 4, (April, 1972), pp. 449-456.

7. Carolyn Kimme, Dana Ballard, and Jack Sklansky, "Finding Circles by an Array of Accumulators," Communication of the ACM, Vol. 18, No. 2, (Feb., 1975), pp. 120-122.

8. D. H. Ballard, "Generalizing the Hough Transform to Detect Arbitrary Shapes," Pattern Recognition, Vol. 13, No. 2, (1981), pp. 111-122.

9. Stephen D. Shapiro, "Feature Space Transforms for Curve Detection," Pattern Recognition, Vol. 10, (1978), pp. 129-143.

10. Stephen D. Shapiro, "Properties of Transforms for the Detection of Curves in Noisy Pictures," Computer Graphics and Image Processing, Vol. 8, No. 2, (Oct., 1978), pp. 219-236.

11. Stephen D. Shapiro, "Geometric Constructions for Predicting Hough Transform Performance," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-1, No.3, (July, 1979), pp. 310-317.

12. Stephen D. Shapiro, "Generalization of the Hough Transform for Curve Detection in Noisy Digital Image," Proc. 4'th Int. Joint Conf. Pattern Recognition, Kyoto, Japan, (1978), pp. 710-714.

13. Christopher M. Brown, "Inherent Bias and Noise in the Hough Transform," IEEE Trans. on Pattern Analysis and Machine Intellengence, Vol. PAMI-5, No. 5, (Sep., 1983), pp. 493-505.

14. C. M. Brown, "Modelling the Sequential Behavior of Hough Transform Schemes," Proc. DRAPA Image Understanding Workshop, Palo Alto, CA, (Sep., 1981), pp.737-739.

15. Philip R. Thrift and Stanley M. Dunn, "Approximating Point Set Image by Line Segments using a Variation of the Hough Transform," Computer Vision, Graphics, and Image Processing, Vol. 21, (1983), pp. 383-394.

16. G. C. Stockman, "Equivalence of Hough Curve Detection to Template Matching," Communication of the ACM, Vol. 20, No. 11, (Nov., 1977), pp. 820-822.

17. C. M. Brown, "Advanced Hough Transform Implementaions," Proc. 8'th Int. Joint Conf. on Artificial Intellegence, West Germany, (1983), pp. 1081-1085.

18. W. A. Perkins and T. O. Binford, "A Corner Finder for Visual Feedback," Computer Graphics and Image Processing, Vol. 2, (Dec., 1973), pp. 355-376.

19. A. E. Cowart, W. E. Snyder, and W. H. Ruedger, "The Detection of Unsolved Targets using the Hough Transform," Computer Vision, Graphics, and Image Processing 21, (1983), pp. 222-238.

20. D. G. Falconer, "Target Tracking with the Hough Transform," Algorithms for Image Processing, Percific Grove, (Feb., 1978).

21. Charles R. Dyer, "Gauge Inspection using Hough Transform," IEEE Trans. on Pattern Analysis and Machine Intellegence, Vol. PAMI-5, No. 6, (Nov., 1983), pp. 621-623.

22. H. Wechsler and J. Sklansky, "Automatic Detection of Ribs in Chest Radiographs," Pattern Recognition 9, (1977), pp. 21-30.

23. P. Bastian and L. Dunn, "Global Transformations in Pattern Recognition of Bubble Chamber Photographs," IEEE Trans. on Computer, c-20, (Sep., 1971), pp. 995-1001.

24. P. M. Merlin and D. J. Farber, "A Parallel Mechanism for Detecting Curves in Pictures," IEEE Trans. on Computers, c-24, (Jan., 1975), pp.96-98.

25. V. Cantoni, M. Caviglione, G. Musso, and G. Pnnunzio, "Location and Orientation Detection of Mechanical Parts using Hough Transform," Applications of Digital Image Processing, SPIE 397, (April, 1983), pp. 229-233.

26. J. L. Turney, T. N. Mudge, and R. A. Voltz, "Recognizing Partially Hidden Objects," IEEE International Conf. on Robotics and Automation, (March, 1985), pp. 55-60.

27. Bjo"rn Kruse, "A Parallel Picture Processing Machine," IEEE Trans. on Computers, Vol. c-22, No. 12, (Dec. 1973), pp. 1075-1073.

28. S. D. Shapiro, "Aspects of Transform Method for Curve Detection," Proc. of the J. Workshop on Pattern Recognition and Artificial Intelligence, Hyannis, (Jun, 1976), pp. 90-97.

29. Thomas Gross, H. T. Kung, Monica Lam, and J. Webb, "Warp as a Machine for Low-Level Vision," IEEE International Conf. on Robotics and Automation, (March, 1985), pp. 790-799.

30. H. T. Kung and C. E. Leisurson, "Systolic Array (for VLSI)," in Sparse Matrix Preceedings, (1978), ed. by I. S. Duff and G. W. Stewart, pp. 256-282.

31. H. T. Kung, "Why Systolic architectures," Computer, (Jan., 1982), pp. 37-46.

32. G. Bongiovanni, "Two VLSI Structures for the Discrete Fourier Transfrom," IEEE Trans. on Computers, Vol. c-32, No. 8, (August, 1983), pp. 750-753.

33. Kai Hwang and Faye A. Griggs, Computer Architecture and Parallel Processing, McGraw-Hill, Inc., 1984.

34. L. S. Haynes, R. L. Lau, D. P. Siewiorek, and D. W. Mizell, "A Survey of Highly Parallel Computing," Computer, (Jan., 1982), pp. 9-26.

35. Leah J. Siegel, Philip H. Swain, "Parallel Algorithm Performance Measures," in Multicomputers and Image Processing Algorithms and Programs, ed. by Preston, Jr. and L. Uhr, Academic Press, Inc., (1982), pp. 241-252.

36. Dennis Parkinson and Heather M. Liddell, "The Measurement of Performance on a Highly Parallel System," IEEE Trans. on Computers, Vol. c-23, No. 1, (1983), pp. 32-37.

37. H. T. Kung and S. W. Song, "A Systolic 2-D convolution Chip," in Multicomputer and Image Processing Algorithms and Programs, ed. by Preston, Jr. and L. Uhr, Academic Press, Inc., (1982), pp. 373-384.

38. K. R. Sloan, Jr. and D. H. Ballard, "Experience with the Generalized Hough Transform," Proc. 5'th Int. Conf. on Pattern Recognition and Image Processing, (1980), pp. 174-179.

39. Kenneth I. Laws, The GHOUGH Generalized Hough Transform Package: Description and Evaluation, Technical Note No. 288, AI Center, SRI Int., (Dec. 1982).

VITA

Ok Sam Chae

Candidate for the Degree of

Doctor of Philosophy

Thesis: SPECIALIZED PARALLEL STRUCTURES FOR VLSI IMPLEMENTATION OF THE HOUGH TRANSFORM FOR ARBITRARY SHAPE DETECTION

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Junnam, Korea, May 17, 1954, the son of Jang B. and Jum D. Chae. Married to Bok Y. Jung on July 19, 1981.

Education: Graduated from Mokpo High School, Mokpo City, Korea, in February, 1973; received Bachelor of Science degree in Electronic Engineering from Inha University in February, 1977; received Master of Science degree from Oklahoma State University in July 1982; completed requirements for the Doctor of Philosophy degree at Oklahoma State University in July, 1986.

Professional Experience: Research Assistant, Fluid Power Research Center, Oklahoma State University, January, 1982, to May, 1982; Teaching Assistant, School of Electrical and Computer Engineering, Oklahoma State University, August, 1982, to December, 1982; Research Associate, School of Electrical and Computer Engineering, Oklahoma State University, January, 1983, to May, 1986.