1989

Invariant pattern recognition algorithm using the Hough Transform

Duwang Li
Portland State University

AN ABSTRACT OF THE THESIS OF Duwang Li for the Master of Science in Electrical and Computer Engineering presented June 9, 1989.

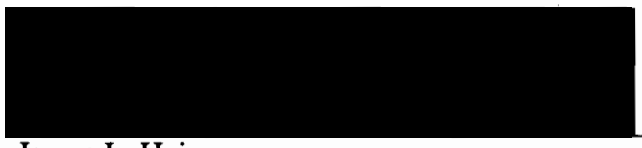
Title: Invariant Pattern Recognition Algorithm Using the Hough Transform

APPROVED BY MEMBERS OF THE THESIS COMMITTEE:

Faris Badi'i, Chairman

Rajinder P. Aggarwal

Lee W. Casperson



James L. Hein

A new algorithm is proposed which uses the Hough Transform to recognize two dimensional objects independent of their orientations, sizes and locations. The binary image of an object is represented by a set of straight lines. Features of the straight lines, namely the lengths and the angles of their normals, their lengths and the end point

positions are extracted using the Hough Transform. A data structure for the extracted lines is constructed so that it is efficient to match the features of the lines of one object to those of another object, and determine if one object is a rotated and/or scaled version of the other. Finally a generalized Hough Transform is used to match the end points of the two sets of lines. The simulation experiments show good results for objects with significant linear features.

INVARIANT PATTERN RECOGNITION ALGORITHM USING THE HOUGH TRANSFORM

by

DUWANG LI

A thesis submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE
in
ELECTRICAL AND COMPUTER ENGINEERING

Portland State University
1989

TO THE OFFICE OF GRADUATE STUDIES:

The members of the Committee approve the thesis of Duwang Li presented June 9, 1989.

Faris Badi'i, Chairman

Rajinder P. Aggarwal

Lee W. Casperson

James L. Hein

APPROVED:

Rolf Schaumann, Chairman, Department of Electrical Engineering

C. William Savery, Interim Vice Provost for Graduate Studies and Research

ACKNOWLEDGEMENTS

I would like to acknowledge my advisor at Portland State University, Dr. Faris Badi'i, whose guidance, intelligence and helpful discussions throughout the years were of great assistance to me. I especially appreciated the encouragement that he gave me.

Also I would like to acknowledge Shirley Clark, secretary of the Department of Electrical Engineering at Portland State University, who helped me navigate my way through the departmental inroads. Her advise, assistance and always cheerful and helpful attitude was invaluable.

Special acknowledgement is given to Dottie and Dennis Feucht for their long term support and commitment to this project. Discussions with Dennis were helpful to the completion of my work.

At last, but not least, I would like to thank Linda Harter for her help in editing this thesis. Her help with my English language skill was of great assistance to my education in this country.

Portland, Oregon

Duwang Li

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The Hough technique for straight line detection [1, 2] in digitized images can cope with noise, gaps in outlines, and partial occlusion, even in complicated backgrounds. Its applications include vehicle guidance [3], gauge inspection [4], detection of seismic patterns [5], recognition of hand printed Hebrew characters [6], automated SEM inspection of VLSI resist [7], etc. Many fast and efficient parallel algorithms and architectures have been developed to detect straight lines using the Hough Transform. It has been generalized to detect high dimensional parametric curves [8, 9] and arbitrary complex shapes [10, 11]. Ballard and Brown [11, 12] even proposed a more general technique to detect rotated and scaled images. The generalization by Sklansky [10] can also be extended to detect scalings and rotations of objects.

We are concerned with the detection of objects independent of their size, orientation and location. Many methods have been developed for this purpose, such as moment invariant method [13], dynamic alignment [14] and complex log mapping related methods [15, 16]. So far I have not seen invariant recognition algorithms using the Hough Transform and its generalization. Since the Hough Transform and its generalization can be implemented efficiently in parallel architectures or algorithms and can cope with noise and gaps in images, it was our interest to explore the invariant recognition algorithms along this line.

Two of the generalized Hough Transforms [10, 11] can be used to detect the rotated and scaled versions of the original shapes by adding more dimensions to the

parameter spaces of these transforms. But the computation time and memory required would be prohibitive. Naturally, we would like to reduce the computation requirements by reducing the information to be processed in the recognition process.

In this thesis an algorithm is presented to reduce computation requirements and detect the 2-D objects independent of size, orientation and position. It decomposes an object into a set of straight line segments and uses the set of the parameters of these lines in the recognition process. Since the number of lines required to represent the original image is much less than the number of image pixels in the image, computation time and memory requirements are much less demanding. Also it is possible to use efficient existing parallel algorithms to extract the lines. The recognition process involves characterizing an image with a set of straight line segments, matching the set of line segments to detect the rotated or scaled version of an original image, and finally matching the end points of two sets of line segments. The inputs to this algorithm are the image pixels of the edges or sketches of the 2-D objects. The original Hough Transform is chosen to detect the straight line segments, and the generalized Hough Transform is used to perform the final end point matching. A brief review of the Hough Transform, its generalizations, and some fast algorithms to detect straight lines are presented in Chapter II. The theory of invariant recognition using the Hough Transform is described in Chapter III. Chapter IV gives a detailed explanation of the implementation of the invariant recognition algorithm and the experimental results. A brief summary and conclusion are given in Chapter V.

CHAPTER II

REVIEW OF THE LITERATURE

2.1 HOUGH TRANSFORM

The Hough Transform originates in a patent by Paul V. C. Hough [1]. It was intended for detecting straight lines in digital pictures.

A straight line in the image plane (x-y plane) can be represented by the equation

$$\rho = x\cos\theta + y\sin\theta \tag{2.1}$$

where θ is the angle of its normal, and ρ the length of its normal (see Figure 1).

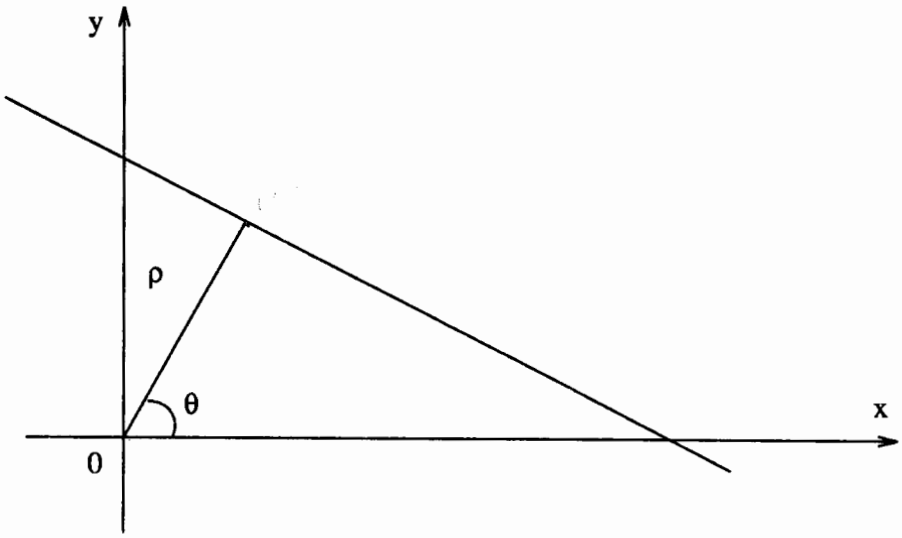


Figure 1. A straight line.

This straight line is characterized by a point (ρ, θ) in the parameter plane (ρ-θ plane). There are some relationships between the points in the (ρ-θ) plane and the points

in the (x-y) plane. These relationships have the following interesting properties:

Property 1.

A point in the image plane corresponds to a sinusoidal curve in the parameter plane (referred to as the corresponding curve *C*). For example, a point (5, 5) maps into a curve, represented by the following equation (see also Figure 2).

$$\rho = 5\sqrt{2}\cos(\theta + \pi/4)$$

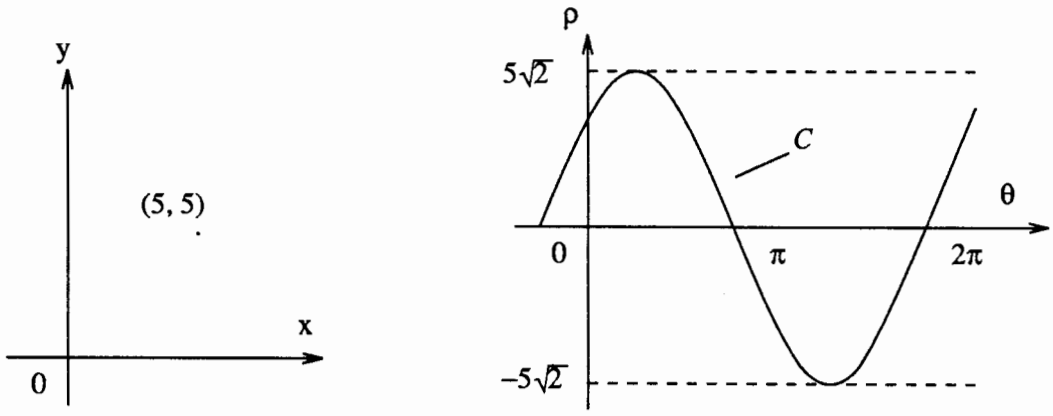


Figure 2. A point in the image plane and its corresponding curve in the parameter plane.

Property 2.

A point in the parameter plane corresponds to a straight line in the image plane. For example, $(5, \pi/6)$ in the parameter plane represents a straight line in the image plane (see Figure 3), represented by

$$\sqrt{3}x + y = 10$$

Property 3.

Points lying on the same straight line in the image plane correspond to curves through a common point in the parameter plane (referred to as the coincident

point (ρ_c, θ_c); see the example in Figure 4).

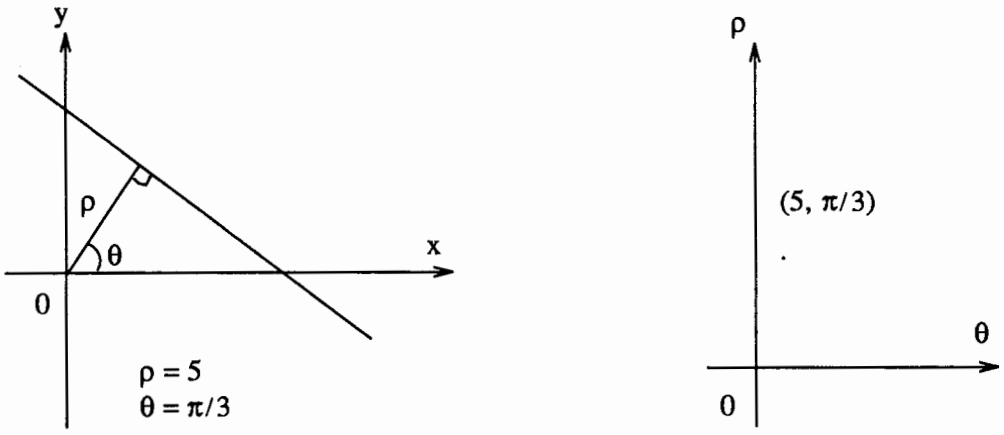


Figure 3. A point in the parameter plane and its corresponding line in the image plane.

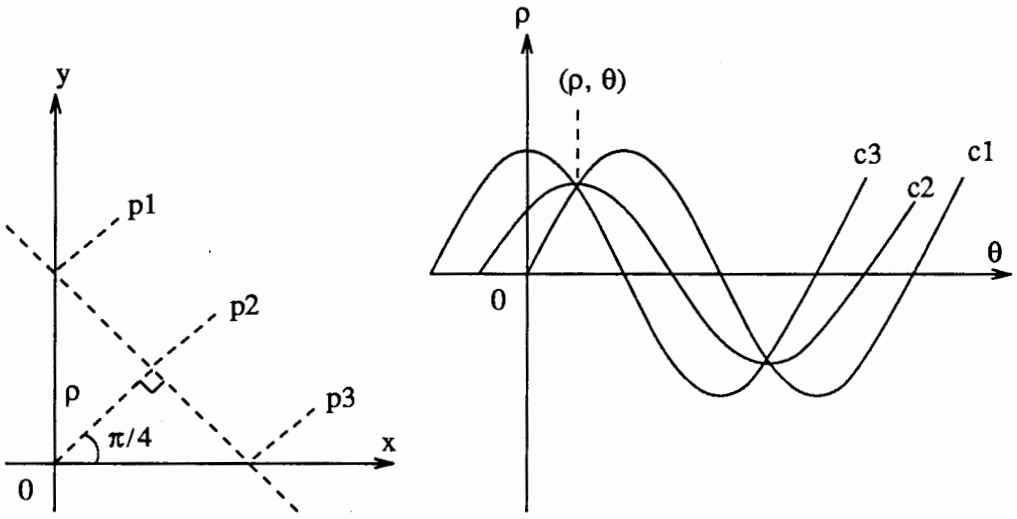


Figure 4. Points lying on a line in the image plane and their corresponding curves in the parameter plane.

Property 4.

Points lying on the same sinusoidal curve in the parameter plane correspond to lines through the same point in the image plane (see Figure 5).

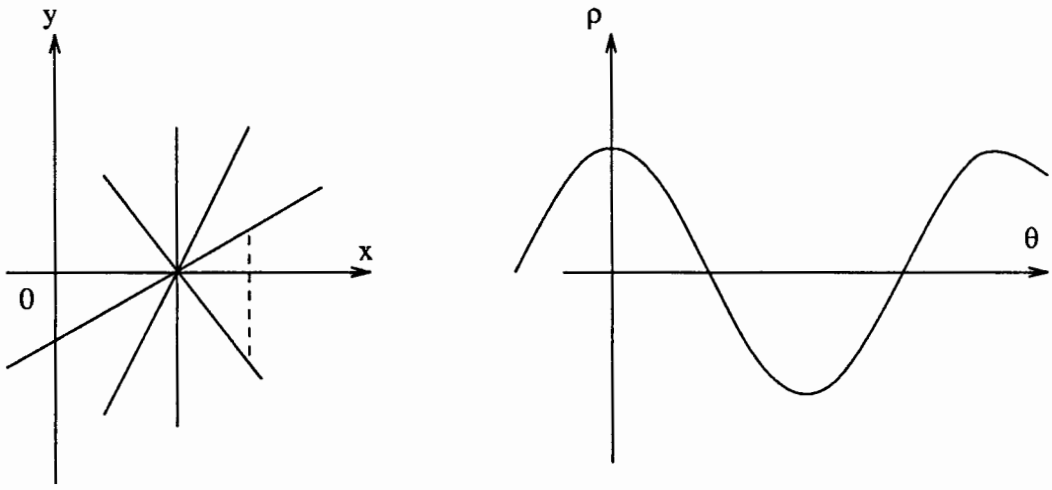


Figure 5. Points lying on the same curve in the parameter plane and their corresponding lines in the image plane.

Using property 3, collinear points in the image plane can be detected. First the mapping is performed from image points (x_i, y_i) to the parameter plane according to the equations (for different i's)

$$\rho = x_i \cos\theta + y_i \sin\theta \tag{2.2}$$

These equations (for different i's) represent many sinusoidal curves that pass through common points in the parameter space if the image points are collinear. Then this common point needs to be found.

In practical implementations we do not need to determine the parameters (ρ, θ) exactly; instead we specify the acceptable errors in θ and ρ and quantize the parameter plane into a quadruled grid [2]. This quantization can be confined to the region

$$-\pi < \theta < \pi$$
$$-R < \rho < R$$

where

$$R = (X_m^2 + Y_m^2)^{1/2}$$

X_m and Y_m are the maximum coordinates of the image space (see Figure 6).

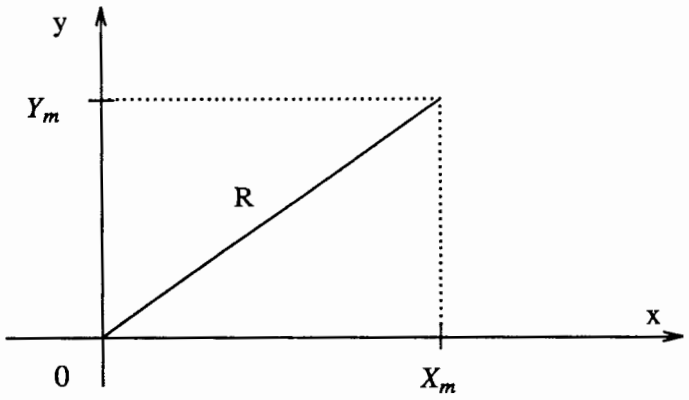


Figure 6. Image space.

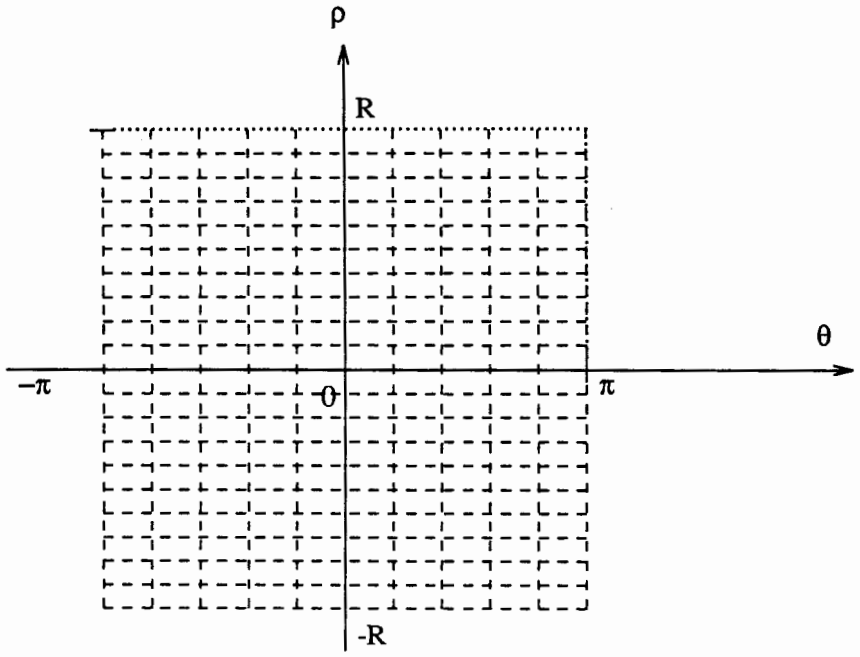


Figure 7. Parameter space.

The quantized region is treated as a 2-D array of accumulators (see Figure 7). Each feature point in the image plane is entered by doing a one-to-many mapping from the image plane to the parameter plane which then registers the mapping by incrementing the corresponding parameter accumulator cells. If the entered points are collinear, there will be a cell with a highest count whose coordinates indicate the parameters of the line.

The above procedures can be applied to the detection of multiple lines. Here a threshold should be specified to indicate the minimum number of feature points needed to make up a line segment. The parameter cells with counts higher than the threshold represent the parameters of the lines that are likely to exist in the image.

2.2 IMPLEMENTATIONS OF THE HOUGH TRANSFORM FOR STRAIGHT LINE DETECTION

The implementation of the Hough Transform requires incrementing an accumulator array and searching for peak counts, much of the computation can be done in parallel or in a pipeline. Several efforts have been made to speed up the computation by utilizing parallelism.

2.2.1 Systolic Array Structure

Chuang and Li [17] presented two systolic array processors for straight line detection; one of them implements the conventional Hough Transform procedure, and the other implements an improved procedure. The basic structure is shown in Figure 8.

Normal parameters (ρ,θ) are used here to represent straight lines, and ρ and θ are bounded and quantized by appropriate resolutions. The quantized angles θ_i are preloaded in the B-array cells. For each feature point entered, each of the B-array cells produces a quantized ρ value using

$$\rho = x_{in}\cos\theta_i + y_{in}\sin\theta_i$$

and sends the ρ value to the corresponding I-array cell. The coordinates of the feature point are sent to the next B-array cell. Each I-array cell has a w register to store a quantized ρ value and a c register to store the count of the pixel with the same ρ value. A comparison between the entered ρ value and the stored w value is made. If they are the same, the count register c is incremented by 1.

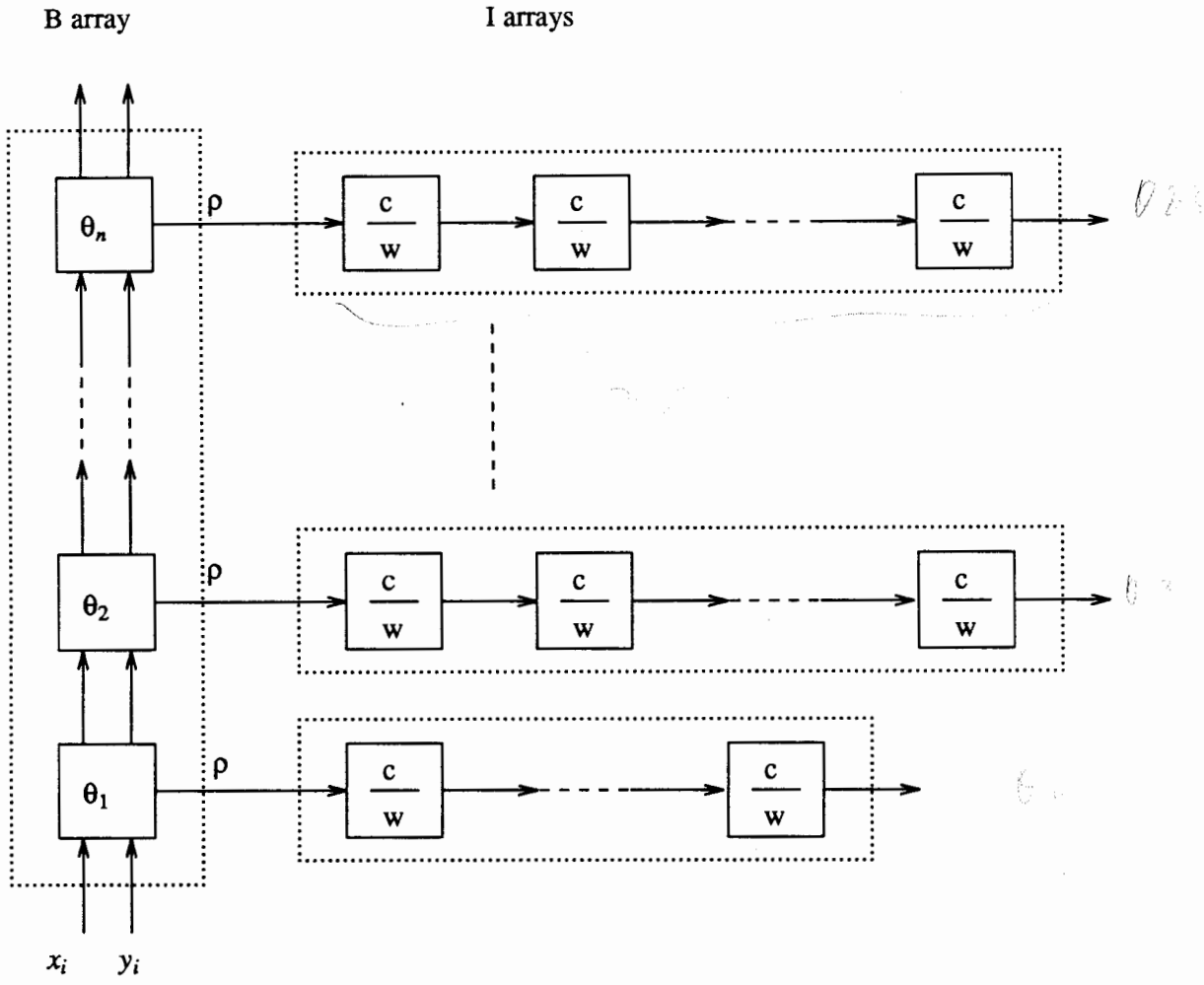


Figure 8. Architecture for the systolic array implementation of the Hough Transform.

Another processor is a modification of the Hough Transform procedure. It checks the connectivity of the edge pixels (feature points), filters out the noisy components and detects the connected line segments. This will also deal with the case in which feature points are collinear but meaningless to image analysis.

The function of the I-array cells is expanded so that it can compute and store the distance between each incoming point and a reference feature point. Each group of distances stored in an I-array cell whose count exceeds a given threshold is detected and sent to the array system. The array system consists of a sort array, a distributor and a

filter array. Each group of distances is sorted using a systolic multi-string sort array, which implements the bubble sort algorithm. The distributor's function is to align the groups of sorted distances and send them to the filter array in ascending order. Each filter cell filters out the noisy pixels, determines the connectivity of the feature points and the terminal points of a line or those of collinear line segments, and sends out the coordinates of the beginning and the end points of each line segment.

One application of this processor is the piece-wise linear approximation of curves. Work along this line of investigation was in progress in 1985 by Chuang and Li [17].

2.2.2 A Monolithic Hough Transform Processor

Recently Rhodes et al. designed a monolithic Hough Transform processor for line detection based on restructurable VLSI [18]. Normal parameterization is used, and the algorithm is similar to that of Chuang and Li [17]. Serial-pipelined multiply-add cells (MAC) are utilized to perform the mapping from image space to the parameter space. Computation time is saved by prestoring the values of $\sin\theta_i$ and $\cos\theta_i$ instead of angle θ_i in the MACs and computing the ρ values simultaneously for all the quantized angles θ_i. Another type of cell performs the format conversion both ways between the bit-serial format required by the MAC cells and the bit parallel format for system I/O. A memory array is used as an accumulator array to record the counts.

In their implementation, the parameter space is quantized into 64 columns for the angle and 256 rows for the normal distance. Throughput rate of the system is 3.2 ms per edge pixel. The completion of the Hough Transform is dependent on the size of the input image. For example, a full 256×256 image could be processed in about 200 ms. Typically edge pixels occupy about 1/10 of an image, so the average time to perform HT in this processor should be close to 20 ms. Several copies of this processor have been completed and have been, or are currently being, integrated into image processing systems.

2.2.3 Geometrical Arithmetic Parallel Processor

A somewhat different algorithm of the HT was implemented on the Geometric Arithmetic Parallel Processor (GAPP) by Silberberg [19]. The GAPP, a SIMD machine developed by Martin Marietta, is a chip of 72 PEs arranged in a 6×12 array. The chips can be cascaded to form an array of arbitrary size. Each PE can communicate with its four neighbors to the left, right, above and below; and has a one bit ALU; 128 bits of RAM and several one bit registers.

The architecture consists of a controller and a 2-D GAPP array of PEs. It is assumed that the GAPP array will accommodate the whole image and the whole accumulator array at once. Different parts of RAM in each PE are used to store the feature image points and the parameters.

The HT procedure is done in two steps: (1) mapping and count accumulating, and (2) the determination of the maximum valued cell in the array.

The mapping and count accumulating process can be summarized as the following. This results in a 2-D histogram of parameters.

```
load the coordinates of all nonzero pixels into PEs;
load quantized normal values into each column
   of the PE array;
for each θ_i, do
{
   broadcast θ_i to all PEs;
   compute ρ(θ_i) for all nonzero pixels;
   compute histogram for each row;
   sum up each column;
   put the total count of each column in
      the first row of the PE array;
   shift the total counts down by one row
}.
```

The parameter pair with maximum count can be detected by doing the following: find the maximum of each row and do this in parallel for all rows; then find the maximum of these row maxima. The parameters (ρ, θ) with this maximum count represent

the straight line in the image.

2.2.4 Fast Hough Transform and Adaptive Hough Transform

A straight line can also be represented by

$$y = mx + c \tag{2.3}$$

where m is the slope and c the intercept, which are the parameters of the line. The Hough Transform can also be performed in this parameter space. A point (x_i, y_i) maps into a straight line in the parameter space, and a point in the the parameter space corresponds to a straight line in the image space. After mapping all the points in the image space into the parameter space, if the image points are collinear, a coincident point (m, c) can be found that correspond to the line which the image points lie on. The following Hough Transform algorithms are based on this parameterization.

In the above algorithms for the HT, the number of accumulator arrays grows exponentially with the quantization resolution and dimension of the parameters. To save space, hierarchical data structures and recursive procedures have been proposed for the HT.

Li et al. developed a Fast Hough Transform (FHT) [20] that reduces the computational complexity and the storage requirements simultaneously. They adopted the generalized quad- and oct-tree, called a k-tree, to represent the parameter space hierarchically. The objects to be detected here are straight lines in 2-D image data, planes in 3-D image data, or hyperplanes. A point in the parameter space corresponds to a hyperplane in the parameter space. The parameter space is divided into hypercubes.

The parameter space is at first divided into 2^k hypercubes for k parameters. This means each parameter range is divided into two halves in each dimension. The mappings, from all feature points into corresponding hypercubes, are performed. In the count collecting process, the count of a hypercube increments if a hyperplane intersects the hypercube. Hypercubes with counts exceeding a predetermined threshold are selected

for subdivision, and the HT mapping and the counting process is performed in new hypercubes. Other hypercubes are not considered any more. This increases the parameter space resolution in those areas where the parameters are likely to be detected. This process continues until the desired parameter resolution is met (see Figure 9).

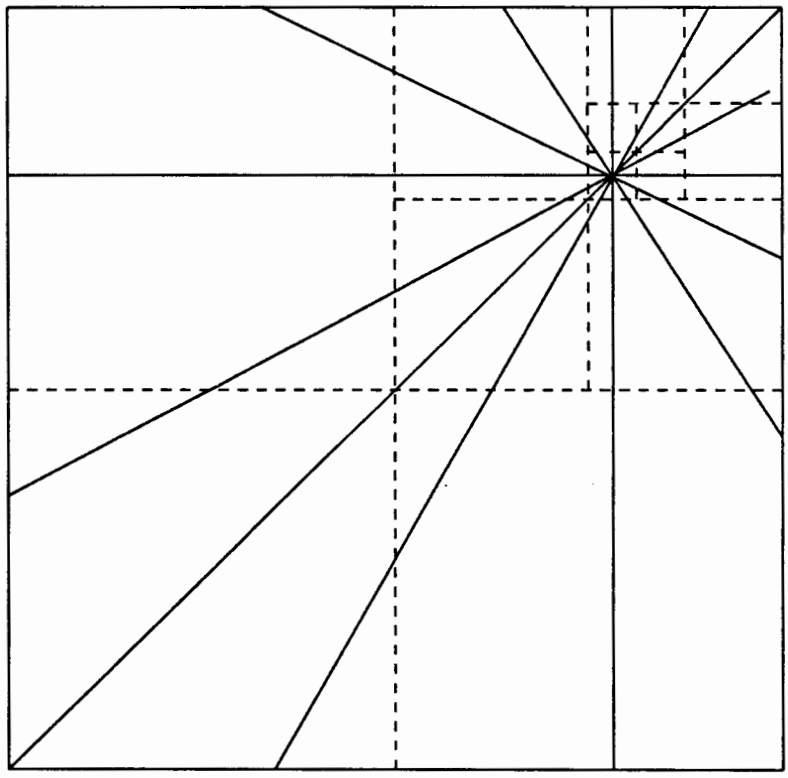


Figure 9. Parameter space division in the Fast Hough Transform.

To decide whether a hyperplane intersects a hypercube, the distance from the center point of the hypercube to the hyperplane is tested. They intersect if the distance is less than one half of the diagonal length of the hypercube. The k-tree data structure is used to represent the hypercubes and their subdivisions, and the intersection testing of hyperplanes with new hypercubes can be done incrementally without multiplication.

Li also proposed a SIMD architecture for the FHT procedure [21]. The architecture consists of a controller, which is a sequential processor, and N identical PEs (N is

the number of feature points in the image space). The controller issues one instruction at a time to all PEs and can broadcast one datum to all PEs through the instruction. Each PE is assigned the coefficients of one hyperplane, which are obtained from the coordinates of each feature point. The center coordinates of a hypercube are broadcast to all PEs, and the distances from this center to the hyperplanes are computed in the PEs simultaneously. Each PE sets/resets a flag to indicate the increment (1 or 0) by checking if the distance is less than one half of the diagonal distance of the hypercube. The controller sums all the increments. Then the total count is compared to the threshold to determine if the hypercube is to be subdivided. This algorithm is intended for detection of multiple collinear instances.

In this algorithm, the computation and time complexity is reduced compared to those of the HT algorithms using uniform quantization. Its disadvantages include: in some cases peak counts may occur in the boundaries between hypercubes, and this parameter space needs further analysis; the intersection testing is not exact; and it can detect only planar or linear objects.

In another work [22] Li et al. solved the intersection testing problem by checking the signs of the distances from the hypercube vertices to the hyperplanes. This algorithm, the Fast Hough Transform algorithm based on Bintree (FHTB) is a generalization of the FHT. The basic idea is still to quantize the parameter space from coarse to fine resolution; and also perform mapping, intersection testing, vote counting, and subdivision (see Figure 10). Except that in the FHTB, the subdivision process is represented by the Bintree and the intersection testing is exact.

For a k-D parameter space, the Bintree divides the space into two equal-size hypercubes alternating among all k dimensions. It was claimed that the total number of hypercubes visited (voted and selected for subdivision) by the FHTB is usually less than that of the FHT, which means that the amount of computation is reduced.

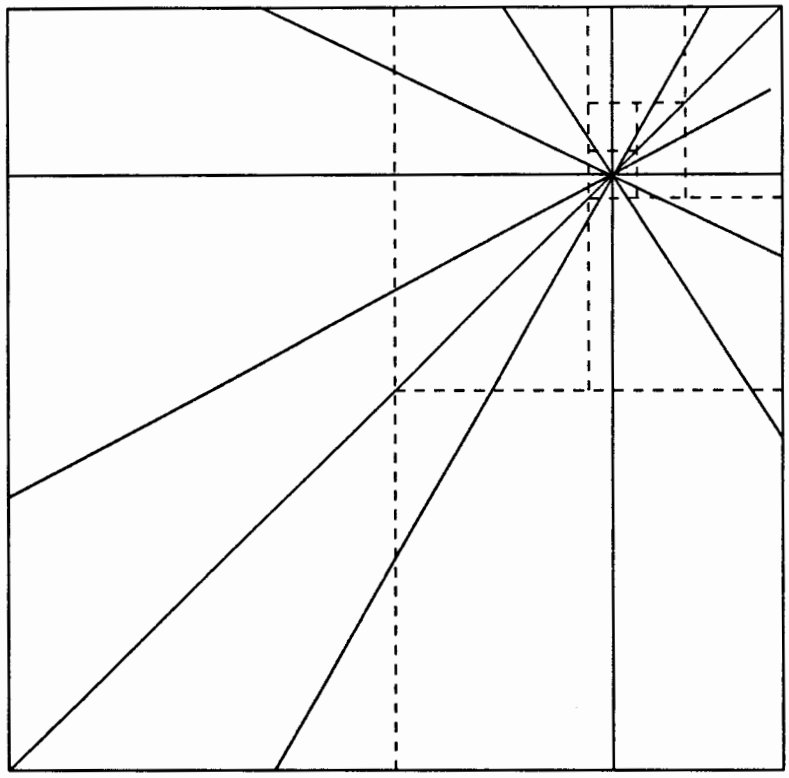


Figure 10. Parameter space division in the Fast Hough Transform Based on Bintree.

Illingworth and Kittler introduced the Adaptive Hough Transform (AHT) [23] to detect 2-D figures (straight lines, circles). The AHT is similar to the FHT in that the parameter space is investigated at several resolutions, but it is only evaluated in fine detail in those regions where a high density of points occurs.

In AHT, an M × M partition is performed for the 2-D parameter space. M is predetermined and is usually a small number, e.g. M is 9 in [23], and 5 in our example (see Figure 11). Mappings into straight lines in the parameter space are performed for all feature image points, and the intersection of these lines with the grid lines of the partitioning determines the votes to the neighboring grids. The grids with votes exceeding a threshold are selected. The connected components algorithm is used on these grids to

label and analyze the connected components. This information is used to guide the adaptive adjustment of the parameter range. Then the new parameter space is partitioned by $M \times M$ grids again. The process continues until the quantizing error is small enough. This algorithm is intended for the detection of a single line.

If the feature points belonging to a detected instance are removed, multiple objects can be identified by reinitializing the process at the coarsest resolution and searching for other objects one by one.

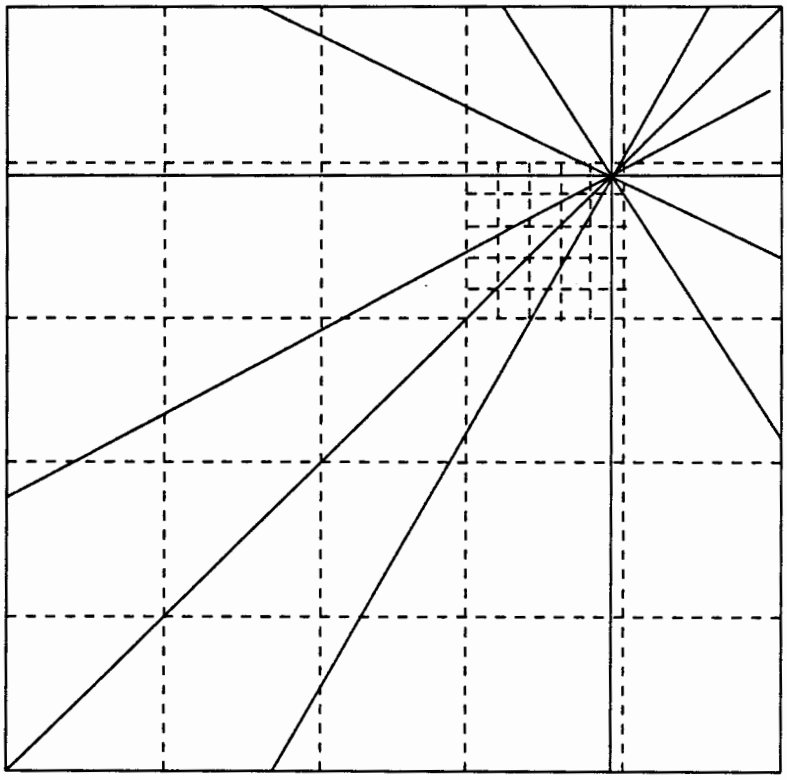


Figure 11. Parameter space division in the Adaptive Hough Transform.

The AHT will also decrease the computation and storage complexity substantially. Compared to the FHT or FHTB, AHT allows greater flexibility in the redefinition of parameter limits and can handle the case when peak counts occur in the grid lines. But the FHT can exploit a high degree of parallelism because of its overall regularity and

simplicity; therefore it is better suited for implementation in special purpose architectures.

2.2.5 A Comparison of the Implementations

The implementation of the Hough Transform are, so far, largely for straight line detection. In this sub-section, a comparison of some of the straight line detection algorithms is given.

For comparison purpose, the following tasks are considered in the Hough Transform process to detect a single line:

i) mappings from image space into parameter space,

ii) count accumulation in the parameter space, and

iii) detection of parameters with the maximum count.

Other tasks, such as connectivity analysis in the algorithm discussed in section 2.2.1, are not considered. It is also assumed that:

The dimension of the input image is : $X_m \times Y_m$

The number of feature points is : F

The size of the accumulator array is : $\rho_m \times \theta_m$

In the systolic array processor in [17], θ_m B- array cells are needed to perform the mapping. B-array cells must be able to compute triangular functions and to do multiplication and addition. The number of I-array cells needed is $\rho_m \times \theta_m$. Each of the I-array cells must have a comparator, an accumulator (it can be a counter) and several registers. The computation time depends mainly on the number of feature points (F) and the time to compute

$$\rho = x \cos\theta + y \sin\theta$$

It is fairly expensive to construct such a processor.

A lot of time and storage elements can be saved by storing values of $\sin\theta_i$ and

$\cos\theta_i$ instead of angles in the mapping cells, such as in [18]. Simple MACs are used in the Monolithic Hough Transform Processor [8] to perform the mapping, and only $2\theta_m$ MACs are needed. High speed RAM is used to accumulate the counts. The time to complete the HT also grows with the number of feature points. This processor will be faster and cost less than the systolic array processor in [17].

In [19] the GAPP cells store both the image and the parameters. At least $\rho_m \times \theta_m$ PEs are needed. The mapping and count accumulating are performed for one quantized angle at a time. The time complexity is proportional to $\rho_m \times \theta_m$. It costs more and is not necessarily faster than the processor in [18].

In the above architectures, the mapping unit and the HT array for count accumulating may not be large enough to accommodate the input image or the parameter space with high resolution. An alternative is to divide the parameter space into partitions and perform the transformation in these partitions one at a time. An input image may also be divided into many blocks, and the HT can be done in a time shift manner. There is a trade off between cost and performance in these architectures.

The Fast Hough Transform and the Adaptive Hough Transform use the slope-intercept parameters in the parameter space. It was claimed in [20] and [23] that this type of algorithms can save memory space and computing time. A serious drawback of this type of algorithms is that in the slope-intercept parameterization the parameter space would have to be unbounded in order to represent all the lines. For example, a vertical line has infinite slope and infinite intercept. This can increase the computing time significantly. To save computing time some method needs to be developed to deal with this problem.

In terms of performance, the AHT algorithm in [23] is roughly the same as FHT or FHTB for straight line detections. Compared to the FHT, it will take less iterations to get the desired parameter resolution, but it is more complicated and takes more time in

each iteration to determine the new parameter space for the next iteration. No hardware implementation has been developed for this algorithm.

Tree structures may be developed to make the dynamic quantizing process faster and easier to program in FHT [20, 22] and AHT [23]. The size of these tree structures is small, so it is possible to develop such an architecture for AHT or FHT.

The Hough Transform can detect multiple collinear line segments simply by finding the parameters receiving counts exceeding a certain threshold value. This is shown in an example in [2]. Thus we can modify the above algorithms, to find multiple instances of collinear segments, or do linear approximation of nonlinear curves. The major problems here are: (1) how should we decide the threshold, or how many votes must a parameter set receive so that it truly represent a line; and in what kinds of resolution must the parameter space be quantized; (2) when the noise can not be removed, how can we detect the true lines in the image. These are also problems for the generalized Hough Transforms. Some solutions have been proposed; for example, Cohen and Toussaint [24] provided two solutions to deal with the noise of known and unknown distributions; Brown [25] used the imaging process to characterize the Hough Transform and developed the Complementary Hough procedure to solve bias and noise problems. Depending on the purpose of the image analysis applications, these procedures can be incorporated into the Hough Transform algorithms or architectures. Other problems need further study, for example the threshold selection problem.

2.3 GENERALIZATIONS OF THE HOUGH TRANSFORM

In many cases, the object sought has no simple analytic form, but has a particular silhouette. The problem here is to find the best fit for a given curve; hence the existence and the location of the given curve in the image needs to be determined.

Sklansky [10] described a technique for the detection of any specified subset of

points or an arbitrary translation of this subset. A reference point is chosen in the given curve. For any feature point b in the image, when the given curve is translated to all the possible curves so that these curves will pass through point b, all the corresponding reference points of these curves form a locus, which is a translated reflection of the given curve about the origin and the reference point of this locus is in b. Thus an accumulator array of appropriate sizes and dimensions can be established. The locus of the reference points corresponding to each feature point is entered by incrementing the counts of the cells the locus passes through. The cell with the highest count indicates the most likely position, or the reference point of the specified curve.

This technique has been shown to be equivalent to template matching [10, 26]. A special case of this concept was described by Merlin and Farber [8]. This technique can also be applied to parametric curves.

To improve computation economy, directional information about the edges can be used in the Hough Transform, because that the gradient of each edge point is usually a by-product of edge extraction which is an earlier step in the image analyzing process.

The Hough algorithm generalized by Ballard [11] uses gradient information about the edges of an object to define a mapping from the the orientation of an edge point to a reference point of the shape. A point inside the given shape is chosen as a reference point. At the boundary points of the given shape, the gradient direction is computed and the reference point is stored as a function of this direction. Thus it is possible to compute the location of the reference point from the boundary points given the gradient angle. The set of such locations, indexed by gradient angle, comprises a table termed the R-table. Then for each feature point in the boundary compute the possible reference points and increment the parameter points in an accumulator array. The parameters here are the locations of the reference points. Two parameters, scale and rotation angle, can be included, so that the scaled and rotated version of a given object can be detected. This is

accommodated by expanding the accumulator array to 4 dimensions.

Use of the gradient information reduces the time to perform the mapping from image space to the parameter space and increases the accuracy of the Hough Transform by eliminating the irrelevant votes. But time is needed to construct the R-table, and efforts are needed to deal with data management problems. Gradient directions can also be used in the detection of parametric curves, and this will reduce the number of parameters. The examples are shown in [11].

CHAPTER III

INVARIANT RECOGNITION ALGORITHM USING THE HOUGH TRANSFORM

In this section the parameter space of the Hough Transform (the Hough Space) is analyzed to show how the invariant recognition algorithm evolves.

3.1 ANALYSIS OF THE HOUGH SPACE

In the last chapter we analyzed the properties of the mappings between the points in the image plane and the parameter plane. We would like to see what will happen in the parameter space (the Hough Space) if the image is is transformed geometrically.

From equation (2.1) which is shown below for convenience,

$$\rho = x\cos\theta + y\sin\theta,$$

a point (x_i, y_i) in the image plane corresponds to a sinusoidal curve (the corresponding curve) in the parameter plane (see Figure 2); and points lying on the same straight line in the image plane correspond to sinusoidal curves through a common point (the coincident point) in the parameter plane (see Figure 3).

Suppose in the image plane we rotate a point (x_i, y_i) about the origin by an angle ϕ. The new point will be (x_{ri}, y_{ri}), where

$$x_{ri} = x_i\cos\phi - y_i\sin\phi \tag{3.1a}$$

$$y_{ri} = x_i\sin\phi + y_i\cos\phi \tag{3.1b}$$

The corresponding curve of (x_{ri}, y_{ri}) in the Hough Space is represented by the equation

$$\rho_r = x_{ri}\cos\theta + y_{ri}\sin\theta \tag{3.2}$$

Substituting x_{ri} and y_{ri} by equations (3.1a) and (3.1b), we have

$$\begin{aligned}\rho &= (x_i\cos\phi - y_i\sin\phi)\cos\theta + (x_i\sin\phi + y_i\cos\phi)\sin\theta \\ &= x_i(\cos\theta\cos\phi + \sin\theta\sin\phi) + y_i(\sin\theta\cos\phi - \cos\theta\sin\phi) \\ &= x_i\cos(\theta - \phi) + y_i\sin(\theta - \phi)\end{aligned} \tag{3.3}$$

The new corresponding curve is a shift of the original curve corresponding to (x_i, y_i) by the angle ϕ along the angle axis of the parameter space (see Figure 12).

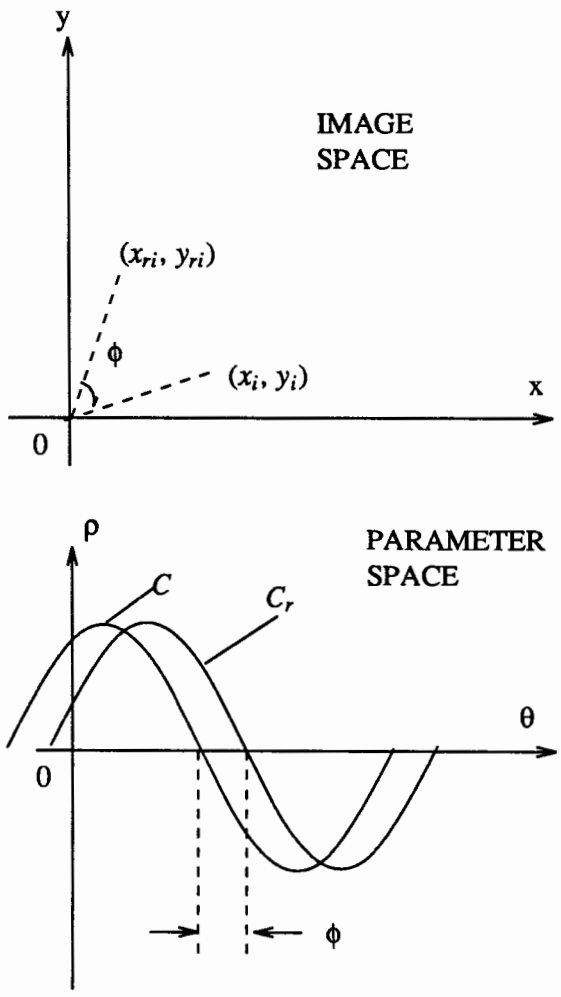


Figure 12. Effect of rotating an image point.

If we rotate a set of points lying on the same line in the image plane, their corresponding curves are also a shift of the original corresponding curves by the angle ϕ;

and the new coincident point is also a shift of the original coincident point (see Figure 13).

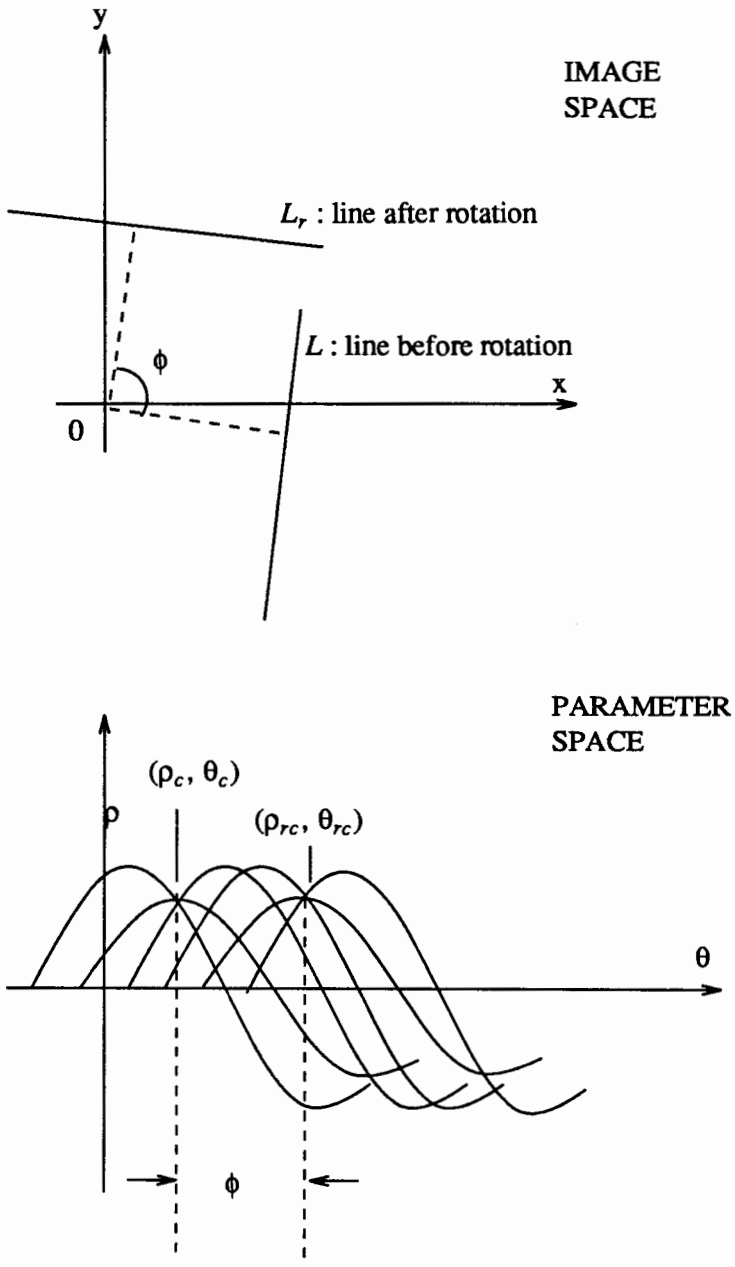


Figure 13. Effect of rotating a set of points.

Now let us examine the effect of scaling image points. If an image point (x_i, y_i) is scaled by factor S, the new point will be (x_{si}, y_{si}) where

$$x_{si} = S\, x_i \tag{3.4a}$$

$$y_{si} = S\, y_i \tag{3.4b}$$

The resulting corresponding curve in the Hough Space for the new point is represented by equation (3.5).

$$\rho_s = S(x_i \cos\theta + y_i \sin\theta) \tag{3.5}$$

This curve is just a scaling in ρ-axis of the original curve by a factor S without changing the phase (see Figure 14).

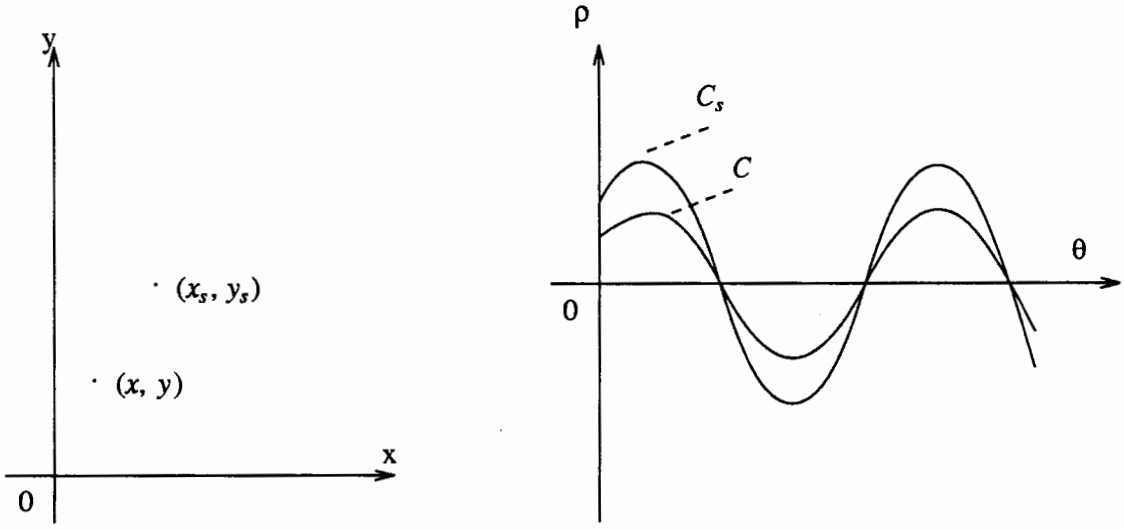


Figure 14. Effect of scaling an image point.

If a set of points lying on a straight line are scaled by a factor S, their corresponding coincident point will be (Sρ, θ) which means the orientation of the line does not change (see Figure 15).

Now consider the effect of translating a point. Suppose an image point (x_i, y_i) is translated into (x_{ti}, y_{ti}) where

$$x_{ti} = x_i + dx \tag{3.6a}$$

$$y_{ti} = y_i + dy \tag{3.6b}$$

The resulting corresponding curve is represented by

$$\rho_t = (x + dx)\cos\theta + (y + dy)\sin\theta$$
$$= \rho + (dx\cos\theta + dy\sin\theta) \quad (3.7)$$

There will be changes both to the amplitude and phase of the corresponding curve (see Figure 16).

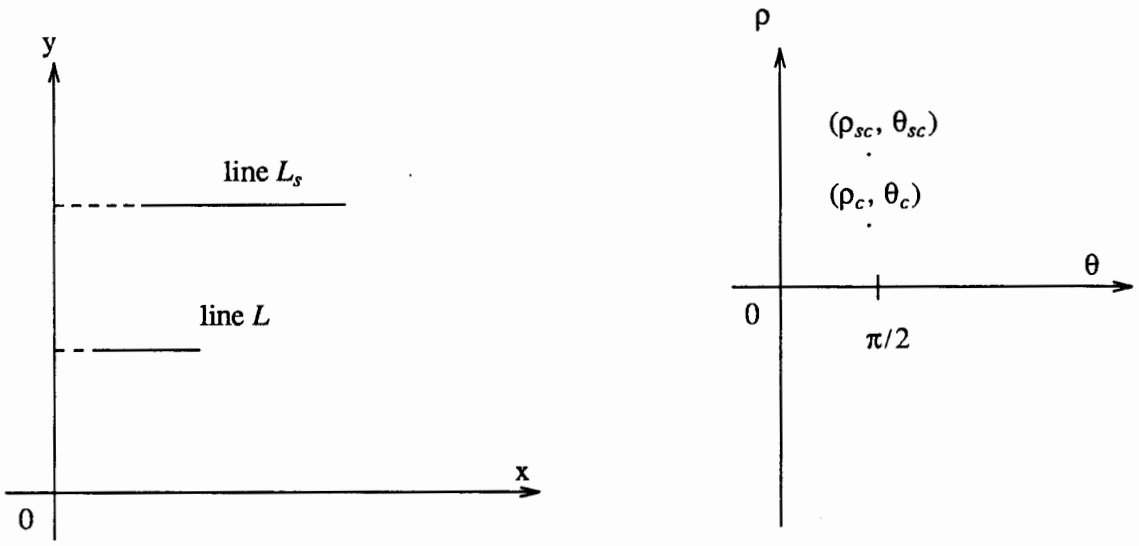


Figure 15. Effect of scaling a set of points.

For a set of points lying on the same straight line, their coincident point (ρ_c, θ_c) in the Hough Space will be changed to (ρ_{tc}, θ_{tc}), where

$$\rho_{tc} = \rho_c + dx\cos\theta_c + dy\sin\theta_c$$
$$\theta_{tc} = \theta_c$$

which means the length of the normal to the line will be changed, but the angle of the line will not be changed (see Figure 17).

The geometric transforms of a line, namely rotation about any point, scaling of its length, and translation, can be represented by a combination of the following transforms:

1) rotate the line about the origin,

2) scale all the points in the line, and

3) translate the line.

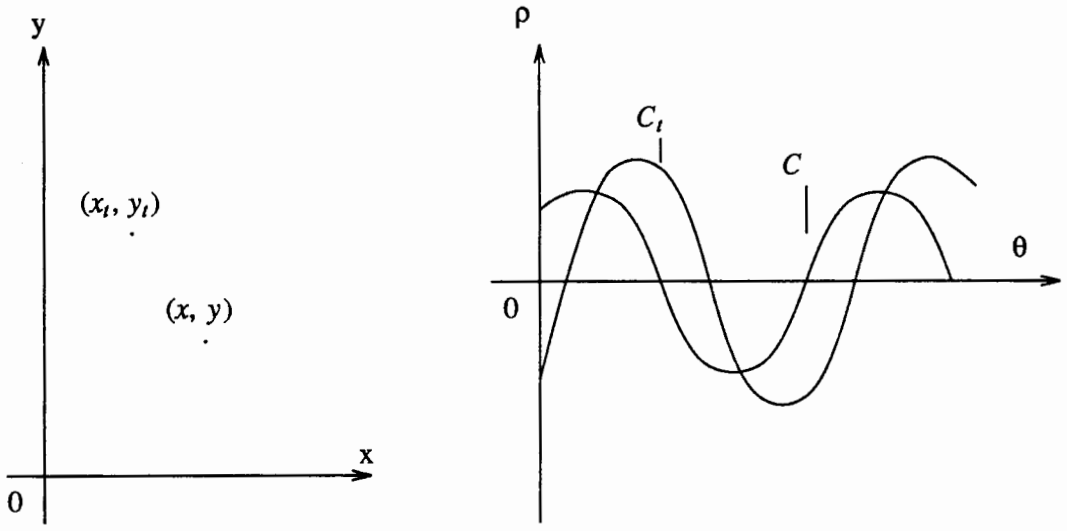


Figure 16. Effect of translating a point.

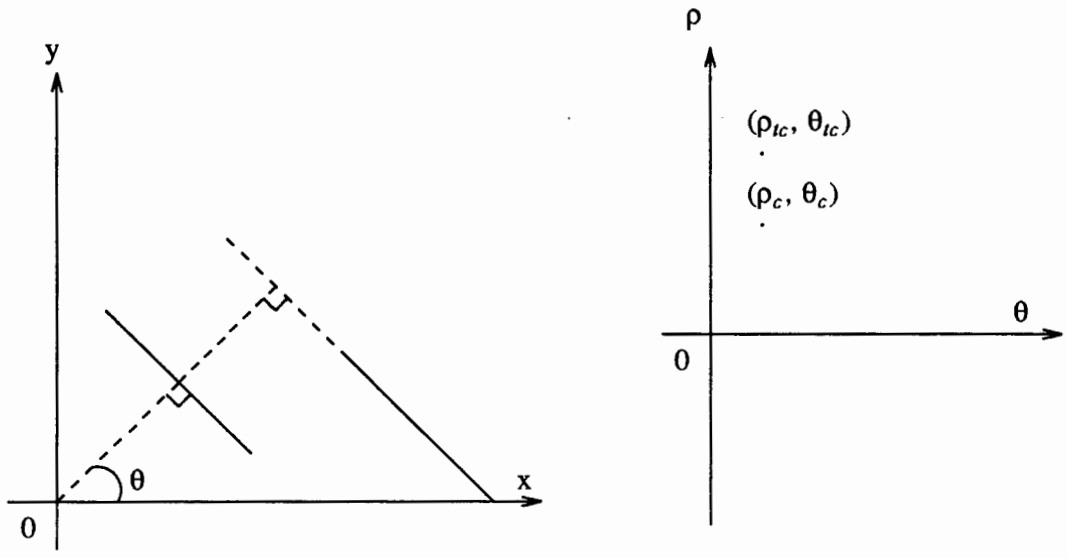


Figure 17. Effect of translating a line.

To rotate a straight line by an angle ϕ about any point P in the image plane two transforms are needed: first shift the image so that P coincides with the origin of the coordinates, then rotate the image, finally shift the image back. The rotation will change the θ value of the line's coincident point in the parameter plane from θ_c to $\theta_c + \phi$. The translation will not change the θ value, but the ρ value will be changed.

To scale a line two translations and a scaling are involved. This will not change the θ value of its coincident point but will change the ρ value.

From the analysis above, we know that to transform a line in the image plane corresponds to changes both in the angle and the length of the normal to the line. We are now interested to see the effect of changing a group of lines in the image plane.

Suppose we have a finite set of straight lines:

$\{L_o(i) \mid i = 1, 2, ..., n\}$,

with coincident points in the parameter plane:

$\{(\rho_c(i), \theta_c(i)) \mid i = 1, 2, ..., n\}$.

If all the lines are translated by (dx, dy), the new coincident points corresponding to these lines are:

$$(\rho_{tc}(i), \theta_c(i)) \mid i=1,2,...,n,$$

where

$$\rho_{tc}(i) = \rho_c(i) + dx\cos\theta_c(i) + dy\sin\theta_c(i) \mid i=1,2,...,n. \tag{3.8}$$

This means that the changes to the ρ_c values are different if the $\theta_c(i)$ are different for different i's, but the $\theta_c(i)$ remain the same. Since the changes to the ρ_c values are nonlinear, there is no easy way to identify the relationships among the lines by comparing the $\rho_{tc}(i)$ values to $\rho_c(i)$ values. Nor can we use the ρ values to recognize if a set of lines is a translation of another set of lines. The only information useful from $(\rho_c(i), \theta_c(i))$ for recognition purpose is that $\theta_c(i)$ values are not changed.

Since translations are involved to scale a set of lines in the image space, the changes to $\rho_c(i)$ are nonlinear for different i's. Both translation and scaling will not change the $\theta_c(i)$ values, so the relationships among them are preserved.

Supposing the set of lines are rotated by an angle ϕ, the new coincident points corresponding to these lines are:

$\{(\rho_{rc}(i), \theta_{rc}(i)) \mid i = 1, 2, ..., n\}$,

where

$$\theta_{rc}(i) = \theta_c(i) + \phi \tag{3.9}$$

for i = 1, 2, ..., n.

The relationships among the θ_c values are the same after the rotation. But the changes to the ρ_c values are very complicated.

In conclusion, the transformations of a set of lines preserve the angle relationships among the lines, but not the lengths of their normals.

3.2 INVARIANT RECOGNITION ALGORITHM USING THE HOUGH TRANSFORM

From the analysis in the last section, we would like to use the features extracted by the Hough Transform to represent an object and use the properties analyzed above to recognize objects independent of size, orientation and location.

The features extracted by using the Hough Transform are the parameters of collinear segments in the image. This to some extent will give the information on the orientation of the objects. For objects with certain linear features, the objects can be approximated by a set of straight lines. Depending on the application, only the orientation of the lines may be needed, such as in gauge inspections; in some other applications, we may need to have the information of the structure which is the relationships among the lines. Specifically we need to know the locations and lengths of all the lines in order to distinguish the objects in examples in Figure 18. The location of a line here means the starting point of the line.

To recognize an object we need to compare the input image to the prototype images. First the features are extracted from the prototype images and stored in a library by using the Hough Transform. These include the normal's lengths and angles of all the

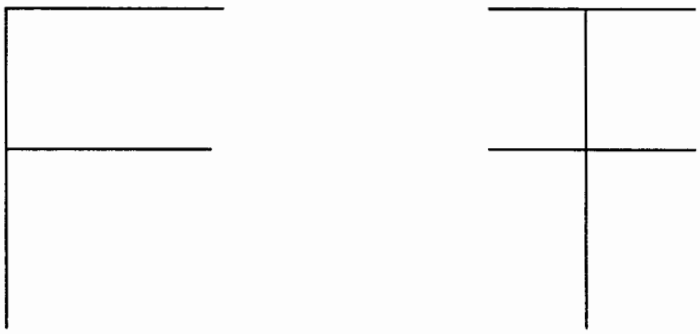


Figure 18. Examples of objects of the same orientation.

lines in the object, and the lengths and coordinates of their starting points. Then the features of the input image are extracted and compared to those of the prototype objects. The angles of the lines are compared to determine the orientation of the object; then the lengths of the lines are used to check the scale; finally the generalized Hough transform is used to match the locations of the lines and decide if the input object belongs to the same class as the prototype object. The processes involved are presented more formally as the following.

3.2.1 Representation of Objects

An object T can be approximated by a finite set of straight line segments

$\{L_o(i) \mid i = 0, 1, 2, ..., n\}$,

especially for the ones with significant linear features. For example, the letter 'F' consists of three segments (see Figure 19).

The features of a line segment are chosen as the following

a) the angle of its normal $\theta(i)$,

b) the length of the segment $d(i)$, and

c) the coordinates of its starting point (x_{si}, y_{si}).

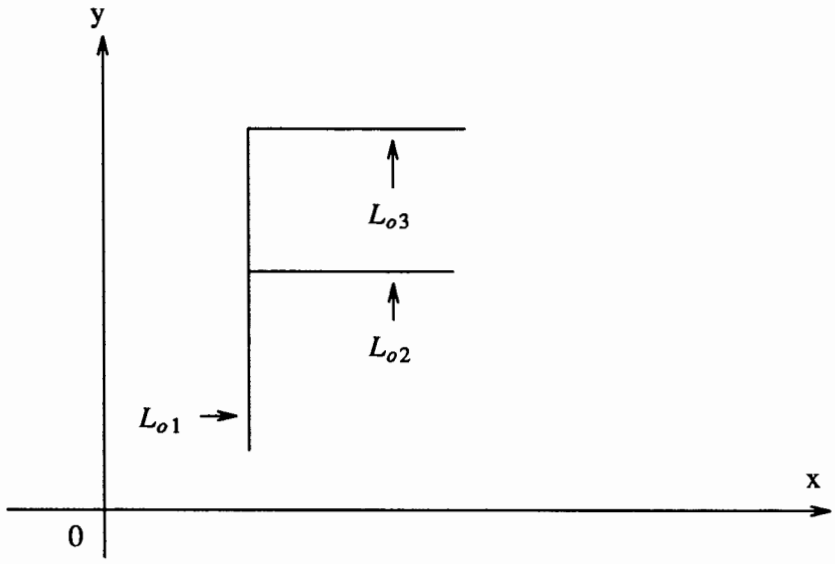


Figure 19. Representation of the letter 'F'.

3.2.2 Detection of Orientation

Suppose T_1 is a geometrically transformed version of the original object T, For example, the 45 degree rotation of the letter 'F' (see Figure 20). It will be represented by a new set of line segments:

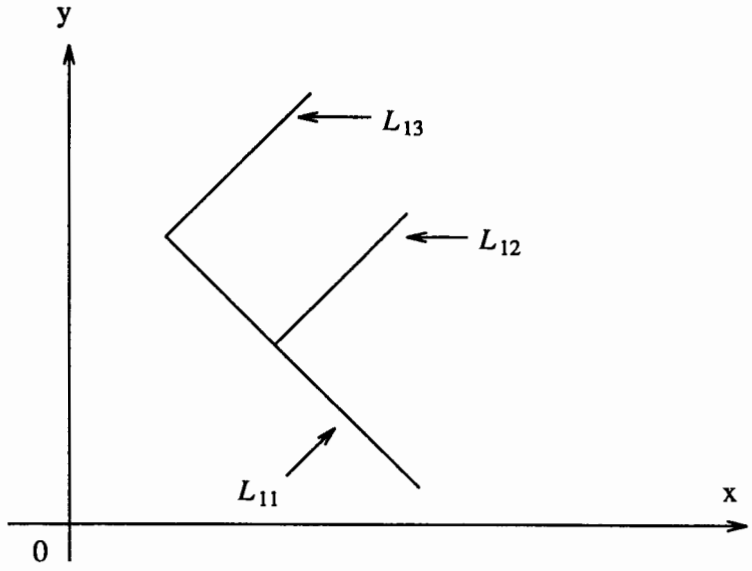


Figure 20. Representation of a rotated version of 'F'.

$\{L_1(i) \mid i = 0, 1, 2, ..., n\}$.

If its orientation differs from that of T_o by an angle ϕ, the angles of these line segments are

$$\theta_1(i) = \theta(i) + \phi$$

This is the important property used to recognize rotation. We can rotate T_o to become T_2 and try to match T_2 to the rotated version of the original object T_o. If the rotation angle is ϕ the two should be the same in terms of angle relations among the line segments:

$$\begin{aligned}\theta_2(i) &= \theta(i) + \phi \\ &= \theta_1(i).\end{aligned}$$

3.2.3 Detection of Scale

Once the orientation of T_2 and T_1 are the same, we can consider the size difference between the two. Since the line segments in the two object are known according to their normals, comparisons between the lengths of the corresponding lines, $L_2(i)$ and $L_1(i)$, can be made to determine the scale ratio between the two:

$$s(i) = \frac{d_1(i)}{d_2(i)}.$$

The ratios $s(i)$ are the same for i = 0, 1, 2, ..., n. This will also confirm the result of the angle matching in the previous step.

3.2.4 Detection of Location

Using the rotation angle and the scale ratio detected in the previous steps, the starting points of the object T_o are transformed; first by a rotation, then by a scaling. Since T_1 and T_2 are both the geometrically transformed version of T_o, the starting points of T_1 and T_2 are the same except that their locations are different. Shifting P_2, so that one point in P_2 overlaps its corresponding point in P_1, will make all the other corresponding points overlap. Hence the translation can be found by detecting the

coordinate difference between the corresponding starting points. This can be achieved in several ways, for example, using the Generalized Hough Transform. The result will confirm that the two objects are matched.

CHAPTER IV

IMPLEMENTATION OF THE INVARIANT RECOGNITION ALGORITHM

The invariant recognition algorithm is implemented as the following:

(a) input the prototype image; calculate the unit angle of the parameter space; set up the parameter space; supply the threshold, perform the Hough Transform and extract the lines $L_o(i) \mid i = 1, 2, ..., n$ and store them in the special data structure;

(b) input the image to be recognized; perform the Hough Transform for this image; and use the threshold derived in (a) to extract the line in this image $L_i(i) \mid i = 1, 2, ..., m$ and store them in the special data structure;

(c) set rotation angle = 0;

```
while (the rotation angle < 360 degrees)
{
        perform angle matching of the two representations;
        if (angle matching of the two representations are successful)
        {
                compare the lengths of the corresponding lines in
                each angle and obtain the average ratio S;
                compute the total rotation angle ϕ; rotate and scale
                the end points in the representation of the prototype
                image according to the rotation angle and the scale
                ratio obtained;
                use the generalized Hough Transform to match the
                end points of the input image to the rotated and
                scaled end points of the prototype image;
                if (the end points are matched)
                {
                        the program exits with the result that the
                        input image is a geometrically transformed
                        version of the prototype image;
                }
        }
```

```
        else
        {
                rotate the representation of the prototype image by
                one unit angle;
        }
}
```

the program exits with the result that the input image does not belong to the same class of the prototype image.

Some of the procedures of the above algorithm are explained below.

4.1 IMAGE SPACE AND PARAMETER SPACE

The image used in this algorithm is a two dimensional array of pixels with binary values. The size of this array is N × N ranging from 0 to N-1. The resolution of the image space is 1.

In order to represent all the possible lines in the image space, the parameter space is chosen as

ρ : [-R, R]

θ : $(-\pi, \pi]$

where

$$R = \sqrt{2}N$$

For the optimum resolution in the parameter space, the normal resolution is 1 and the angle resolution is $\Delta\theta$ with:

$$\Delta\theta = \frac{\sqrt{2}}{N}$$

This is based on the optimum sampling rule [9]. For less fine resolutions, so as to give more tolerance to the angle and the thickness of the lines, the resolution for the normal can be 2, 3, ... etc. with corresponding angle resolution as

$$\Delta\theta = \frac{\sqrt{2}}{N}$$

4.2 HOUGH TRANSFORM AND LINE EXTRACTION

The operations of the Hough Transform are the following:

```
reset the values in the parameter array;
for each nonzero image pixel (x, y)  DO
{
        for each quantized angle θ_i, i = 0, 1, 2, ..., ang_max - 1  DO
        {
                compute
                    ρ_i = x cosθ_i + y sinθ_i;
                increment the count value in (ρ_i, θ_i) by 1
        }
}.
```

Because the triangular functions for each quantized angle need to be calculated for every image point, they are precomputed and stored to save computation time significantly. If some special architecture is to be designed for this transform, they can be stored in a ROM.

To extract the lines a threshold needs to be chosen to indicate the minimum number of points needed to make up a line segment. The threshold is obtained from the prototype image. If the total points in the prototype image is N_p and the shortest line has N_m points, the relative threshold is N_m/N_p. This is done by human interface at this point of the research. The threshold of counts for an image to be recognized that has totally M_p points is

$$TH = N_s \frac{M_p}{N_p}$$

The parameter pairs with counts higher than TH represent the parameters of the possible line segments. From the experiments it is found that when there are relatively short lines in an image, TH will be small, and not every parameter pair with counts higher than TH represents a true line. An algorithm for line extraction in the following is

used to eliminate the "fault lines" because a nonzero point usually belongs only to one line, except for the end points which may belong to different lines:

search for a maximum of counts in the parameter space, and obtain the parameter values (ρ_i, θ_i);

while (the maximum $\geq$ TH)

{

find all the nonzero image pixels that fit the parameter pair (ρ_i, θ_i); modify the values of this pixels to zero. if the number of image points that fit the parameter pair is less than TH, reject this line;

find the coordinates of the two end points and the length of this line;

search for the next maximum count

}.

4.3 DATA STRUCTURE FOR THE SET OF STRAIGHT LINES

The features of each line segment are:

a) the angle of its normal θ,

b) the length of its normal ρ,

c) the coordinates of its two end points (x_e, y_e),

d) the length of the line segment d.

Compared to the feature proposed in Chapter III for the line segments, the length of its normal is added to sort the lines with the same orientations within one image. With this information, it would be possible to extend this algorithm to detect partially occluded objects or to detect an object from a scene. Also, two end points are used instead of the starting point of a segment because the algorithm does not have to identify which of the two ends is the starting point.

A data structure is implemented to store all the lines extracted from an image. It is an array of pointers pointing to a list of nodes whose contents are the features of the line segments. The array is indexed by the angles of the parameter space. If no line is

found in certain angle, the pointer in this angle points to NULL. If two or more lines are detected, the nodes are sorted according to the length of the normals. Examples are shown in Figure 21 and Figure 22 for the letter 'F' and its 90 degree rotation.

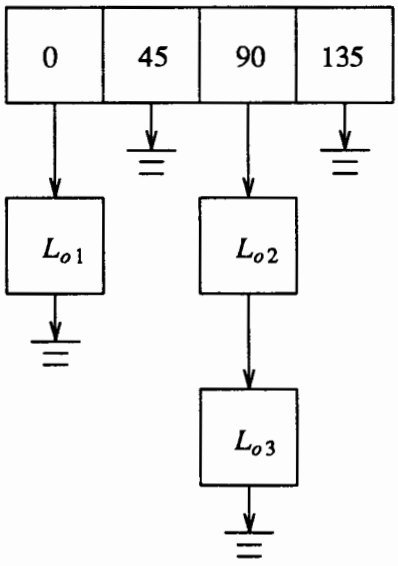


Figure 21. Data structure for the letter 'F'.

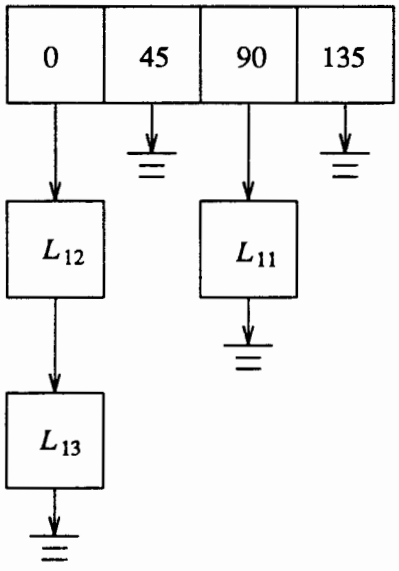


Figure 22. Data structure for the 90 rotation of the letter 'F'.

4.4 DETECTION OF ROTATION

The matching of the orientation is achieved by comparing the number of lines in each angle between the representations of the prototype image and that of the input image. This is best illustrated by an example. Figure 21 and Figure 22 are the representations of the letter is set to 45 degrees for this example. Comparing the two representations, we find that the number of lines is different in angles 0 degree and 90 degrees. The prototype is rotated 45 degrees by shifting the pointers to the right, as is shown in Figure 23. This is not a match either. The prototype is rotated once more until it appears as Figure 24. This is a good match in terms of the angle relations between the two. So the orientation of the input image is 90 degrees with reference to the prototype image. The program then rotates the end points of the prototype image representation 90 degrees from (x_i, y_i) into (x'_i, y'_i) (see the following equations):

$$x'_i = x_i \cos\phi - y_i \sin\phi$$
$$y'_i = x_i \sin\phi + y_i \cos\phi$$

ϕ is 90 degrees here.

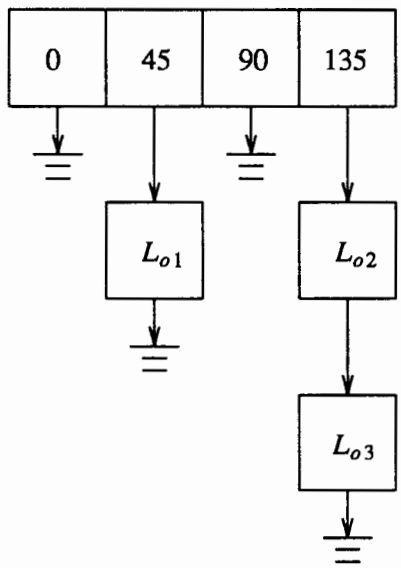


Figure 23. Data structure after the lines are rotated by 45 degrees in the data structure for 'F'.

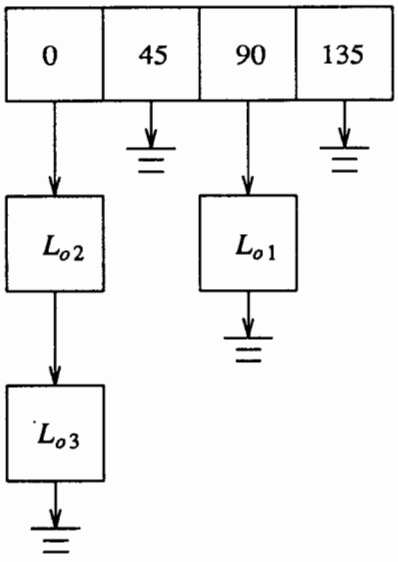


Figure 24. Data structure after the lines are rotated by 90 degrees in the data structure for 'F'.

4.5 DETECTION OF SCALE

The detection of the scale ratio is simply achieved by comparing the lengths of the corresponding lines between the representations of the input image and that of the rotated and scaled prototype image.

4.6 END POINT MATCHING USING A GENERALIZED HOUGH TRANSFORM

The number of the end points representing an image is far less than the points in the original image. Following is the implementation of the end point matching using the generalized Hough Transform [10]:

> set up a 2-D array to represent the generalized Hough Space; the array for the original Hough Transform in the previous procedures is used as the generalized Hough Space because it is no longer needed; reset the array to zero;

for (each rotated and scaled end point $(x_{2j},\ y_{2j})$ of the prototype image)
{

enter the reflections of the end points of the input image $(x_{ri},\ y_{ri})$ by incrementing the counts in the array elements indexed by

$$x_{2j} + x_{ri}\ ,\ y_{2j} + y_{ri},$$

};

find the count maximum in the generalized Hough Space;

if (the maximum equals the number of the end points in the prototype image)
a good match is found.

4.7 ITERATIVE PROCESS OF MATCHING

Since our main concern is to classify the input image, the procedures to detect the orientation and scale ratio are not intended for exact matching. If the minimum requirement for the angle matching is met, the program proceeds to compute the scale ratio and transform the end points in the prototype image. The matching result of the end points determines the final result of matching.

4.8 SIMULATION RESULTS

A program was written in C and tested on a SUN micro system under UNIX. The input images are arrays of 32×32 points created via the graphics tools in this system.

For each test a prototype image and an image to be tested are input to the program. The first results are the features of the lines extracted from the two images including their angles, lengths and locations. Then the program compares the two sets of lines to describe the possible angle matches and to decide if the image to be classified belongs to the same class of the prototype image. If it does, the orientation and the scale ratio of the input image, with respect to the prototype image, are also given. The translation can be easily calculated, but it is not given since it is meaningful only to the specific applications.

In the following, a list of recognition results are given including the lines extracted from the images and detection results. The test patterns are shown in Figure 25, Figure 26 and Figure 28.

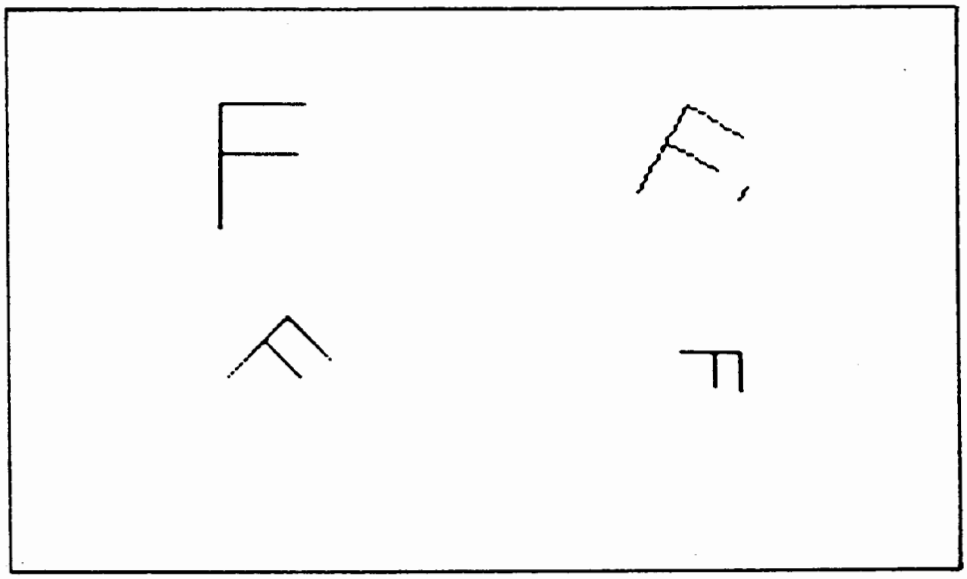


Figure 25. Test patterns for 'F'.

Table I shows the lines extracted from the prototype image of the letter 'F'.

TABLE I

LINES EXTRACTED FROM THE PROTOTYPE IMAGE OF THE LETTER 'F'

line #	angle (degrees)	length	end points (x1, y1) to (x2, y2)
1	0.0	20.0	(0, 0) to (0, 20)
2	0.0	17.0	(12, 1) to (12, 18)
3	90.0	30.0	(0, 0) to (30, 0)

Table II shows the lines extracted from a rotated and scaled version of the letter 'F' (30 degree rotation and 80% scaling); it is recognized as in the same class as the prototype image (see Table III).

TABLE II

LINES EXTRACTED FROM THE TRANSFORMED VERSION OF THE LETTER 'F' (SCALE : 0.8, ROTATION : 30 DEGREES)

line #	angle (degrees)	length	end points (x1, y1) to (x2, y2)
1	60	22.0	(0, 12) to (19, 1)
2	150	15.3	(0, 12) to (8, 25)
3	150	19.7	(10, 9) to (20, 26)

TABLE III

DETECTION RESULT OF THE TRANSFORMED VERSION OF THE LETTER 'F' (SCALE : 0.8, ROTATION : 30 DEGREES)

prototype	actual transform		detection result	
	scale	rotation	scale	rotation
F	0.80	30.0	0.85	30.0

Table IV and Table V show another successful recognition of a 45 degree rotation and 70% scaling.

TABLE IV

LINES EXTRACTED FROM THE TRANSFORMED VERSION OF THE LETTER 'F' (SCALE : 0.707, ROTATION : 45 DEGREES)

line #	angle (degrees)	length	end points (x1, y1) to (x2, y2)
1	45.0	19.8	(1, 16) to (15, 2)
2	135.0	14.1	(1, 16) to (11, 26)
3	135.0	9.9	(8, 12) to (15, 19)

TABLE V

DETECTION RESULT OF THE TRANSFORMED VERSION OF THE LETTER 'F' (SCALE : 0.707, ROTATION : 45 DEGREES)

prototype	actual transform		detection result	
	scale	rotation	scale	rotation
F	0.707	45.0	0.65	45.0

The program failed to recognize a 90 degree rotation and 50% scaling (see Table VI and Table VII) because two lines are extracted with a wrong angle, which is 85 degrees instead of 90 degrees. The program exits since it can not find an angle match.

TABLE VI

LINES EXTRACTED FROM THE TRANSFORMED VERSION OF THE LETTER 'F' (SCALE : 0.5, ROTATION : 90 DEGREES)

line #	angle (degrees)	length	end points (x1, y1) to (x2, y2)
1	0.0	14.0	(9, 10) to (9, 24)
2	85.0	9.0	(9, 24) to (18, 24)
3	85.0	7.0	(10, 18) to (17, 18)

TABLE VII

DETECTION RESULT OF THE TRANSFORMED VERSION OF THE LETTER 'F' (SCALE : 0.5, ROTATION : 90 DEGREES)

prototype	actual transform		detection result
	scale	rotation	
F	0.5	90	NOT MATCHED

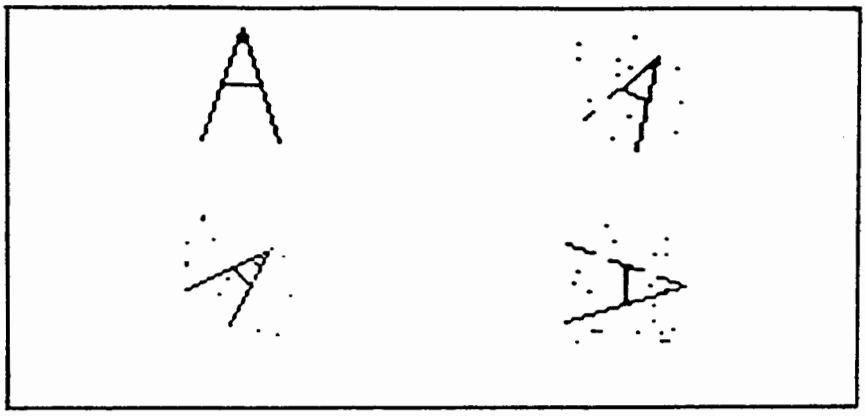


Figure 26. Test patterns for 'A'.

Because of the quantizing effect, lines may have certain thickness. For example the upper-case 'A' (see Figure 27) consists of five lines if the parameter resolution is 1 for the normal and 5 degrees for the angle (see Table VIII).

TABLE VIII

LINES EXTRACTED FROM THE PROTOTYPE IMAGE OF THE LETTER 'A' USING A FINE RESOLUTION

line #	angle (degrees)	length	end points (x1, y1) to (x2, y2)
1	0.0	6.0	(15, 12) to (15, 18)
2	70.0	27.5	(4, 14) to (30, 5)
3	70.0	24.4	(0, 15) to (23, 7)
4	105.0	9.5	(21, 22) to (30, 25)
5	110.0	30.7	(0, 15) to (29, 25)

```
00000000000000010000000000000000
00000000000000101000000000000000
00000000000000101000000000000000
00000000000001000100000000000000
00000000000001000100000000000000
00000000000010000010000000000000
00000000000010000010000000000000
00000000001000000010000000000000
00000000001000000010000000000000
00000000010000000001000000000000
00000000011111111110000000000000
00000000010000000001000000000000
00000000100000000000010000000000
00000000100000000000010000000000
00000001000000000000001000000000
00000001000000000000001000000000
00000010000000000000000100000000
00000010000000000000000100000000
00000100000000000000000010000000
00000100000000000000000010000000
00001000000000000000000001000000
```

Figure 27. Pixel arrays for the upper case 'A'.

If the resolutions are changed to 2 for the normal and 10 degrees for the angle, the lines extracted from the same image will be accurate (see Table IX).

TABLE IX

LINES EXTRACTED FROM THE PROTOTYPE IMAGE OF THE LETTER 'A' USING A COARSE RESOLUTION

line #	angle (degrees)	length	end points (x1, y1) to (x2, y2)
1	0.0	7.1	(15, 12) to (15, 18)
2	70.0	32.3	(0, 15) to (30, 5)
3	110.0	32.9	(0, 15) to (30, 25)

The same resolutions are used to extract lines from the transformations of the letter 'A'. The algorithm can also cope with noise and gaps. The results are shown in Table X and Table XI, Table XII and Table XIII, Table XIV and Table XV.

TABLE X

LINES EXTRACTED FROM THE TRANSFORMED VERSION OF THE LETTER 'A' (SCALE : 0.8, ROTATION : 30 DEGREES)

line #	angle (degrees)	length	end points (x1, y1) to (x2, y2)
1	40.0	21.9	(7, 20) to (22, 4)
2	80.0	26.3	(5, 22) to (30, 17)
3	160.0	4.5	(15, 15) to (16, 18)

TABLE XI

DETECTION RESULT OF THE TRANSFORMED VERSION OF THE LETTER 'A' (SCALE : 0.8, ROTATION : 30 DEGREES)

prototype	actual transform		detection result	
	scale	rotation	scale	rotation
A	0.8	30.0	0.74	30.0

TABLE XII

LINES EXTRACTED FROM THE TRANSFORMED VERSION OF THE LETTER 'A' (SCALE : 0.707, ROTATION : 45 DEGREES)

line #	angle (degrees)	length	end points (x1, y1) to (x2, y2)
1	30.0	24.2	(8, 24) to (19, 2)
2	30.0	5.7	(8, 24) to (11, 19)
3	60.0	21.9	(8, 24) to (27, 14)

The program failed to recognize this transformed version because the lines are not extracted accurately from this image (see Table XII and Table XIII).

TABLE XIII

DETECTION RESULT OF THE TRANSFORMED VERSION OF THE LETTER 'A' (SCALE : 0.707, ROTATION : 45 DEGREES)

prototype	actual transform		detection result
	scale	rotation	
A	0.707	45.0	NOT MATCHED

TABLE XIV

LINES EXTRACTED FROM THE TRANSFORMED VERSION OF THE LETTER 'A' (SCALE : 1.0, ROTATION : 90 DEGREES)

line #	angle (degrees)	length	end points (x1, y1) to (x2, y2)
1	20.0	30.1	(15, 29) to (25, 0)
2	90.0	9.1	(11, 15) to (19, 15)
3	160.0	32.0	(5, 0) to (15, 29)

TABLE XV

DETECTION RESULT OF THE TRANSFORMED VERSION OF THE LETTER 'A' (SCALE : 1.0, ROTATION : 90 DEGREES)

prototype	actual transform		detection result	
	scale	rotation	scale	rotation
A	1.0	90.0	0.984	90.0

The approach used in our program favors the longer lines. We feel that the treatment of the points should be determined by the characteristics of the objects. Other approaches, such as extracting any line that has more points than the threshold or allowing the points to be used for more than one line, can also be used. Using each point only one time is meant to extract the line that really exists, thus there will be less chance of misclassification and more chance of not being able to recognize the objects of the same class. The latter approach will have a better chance of misclassification.

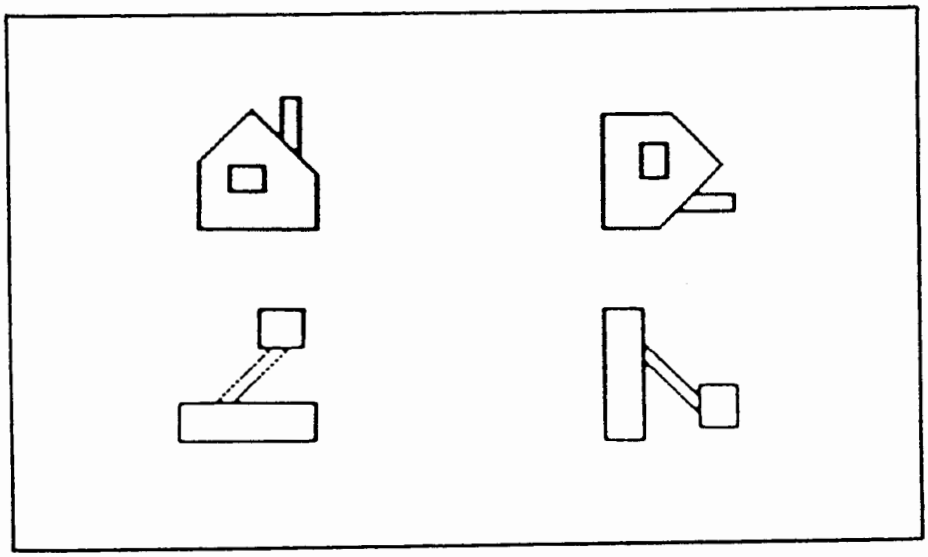


Figure 28. Test patterns for 'HOUSE' and 'PART'.

Since the algorithm extracts the longest line first, and the points fitting this line are not used for any other line, sometimes the short lines may not have enough points to be extracted, as shown in Table XII and Table XIII.

For more complicated objects, the program can also detect the transformation; but their may be some errors in the lines extracted and in the detection results. For example, Table XVI shows the lines extracted from the model 'HOUSE'.

TABLE XVI

LINES EXTRACTED FROM THE MODEL IMAGE FOR 'HOUSE'

line #	angle (degrees)	length	end points (x1, y1) to (x2, y2)
1	0.0	27.0	(31, 4) to (31, 31)
2	0.0	8.0	(16, 11) to (16, 19)
3	0.0	8.0	(22, 11) to (22, 19)
4	45.0	17.0	(3, 16) to (15, 14)
5	90.0	16.0	(15, 14) to (31, 4)
6	90.0	14.0	(0, 27) to (14, 27)
7	90.0	13.0	(18, 31) to (31, 31)
8	90.0	9.0	(0, 23) to (9, 23)
9	135.0	21.2	(3, 16) to (18, 31)

The 90 degree rotation of 'HOUSE' are represented by the lines in Table XVII and it is recognized as a transformation of 'HOUSE' (see Table XVIII).

TABLE XVII

LINES EXTRACTED FROM A 90 DEGREE ROTATION OF THE MODEL IMAGE FOR 'HOUSE'

line #	angle (degrees)	length	end points (x1, y1) to (x2, y2)
1	0.0	16.0	(4, 0) to (4, 16)
2	0.0	14.0	(27, 17) to (27, 31)
3	0.0	13.0	(31, 0) to (31, 13)
4	45.0	9.0	(23, 22) to (23, 31)
5	90.0	21.2	(16, 28) to (31, 13)
6	90.0	27.0	(4, 0) to (31, 0)
7	90.0	8.0	(11, 9) to (19, 9)
8	90.0	8.0	(11, 15) to (19, 15)
9	135.0	17.0	(4, 16) to (16, 28)

TABLE XVIII

RECOGNITION RESULT OF THE 90 DEGREE ROTATION OF THE MODEL IMAGE FOR 'HOUSE'

prototype	actual transform		detection result	
	scale	rotation	scale	rotation
HOUSE	1.0	90.0	1.0	90.0

Another example is the detection of the model 'PART'. The results are shown in Table XIX, Table XX, and Table XXI.

TABLE XIX

LINES EXTRACTED FROM THE MODEL IMAGE FOR 'PART'

line #	angle (degrees)	length	end points (x1, y1) to (x2, y2)
1	0.0	31.0	(22, 0) to (22, 31)
2	0.0	31.0	(31, 0) to (31, 31)
3	0.0	8.0	(0, 18) to (0, 26)
4	40.0	36.2	(8, 28) to (31, 0)
5	45.0	40.3	(1, 28) to (30, 0)
6	45.0	21.2	(6, 28) to (21, 13)
7	85.0	6.0	(22, 31) to (28, 31)
8	90.0	9.0	(0, 18) to (9, 18)
9	90.0	8.0	(1, 28) to (9, 28)

TABLE XX

LINES EXTRACTED FROM A 90 DEGREE ROTATION
OF THE MODEL IMAGE FOR 'PART'

line #	angle (degrees)	length	end points (x1, y1) to (x2, y2)
1	0.0	9.0	(18, 22) to (18, 31)
2	0.0	9.0	(28, 22) to (28, 31)
3	5.0	30.1	(28, 30) to (31, 0)
4	90.0	31.0	(0, 0) to (31, 0)
5	90.0	31.0	(0, 9) to (31, 9)
6	90.0	10.0	(18, 31) to (28, 31)
7	135.0	40.3	(0, 1) to (28, 30)
8	135.0	21.2	(13, 10) to (28, 25)

TABLE XXI

RECOGNITION RESULT OF THE 90 DEGREE ROTATION
OF THE MODEL IMAGE FOR 'PART'

prototype	actual transform		detection result	
	scale	rotation	scale	rotation
PART	1.0	90.0	1.02	90.0

The program can classify different objects as different. In our tests, the program classifies the several versions of letter 'F' to be different from letter 'A'; 'HOUSE' different from 'PART; 'HOUSE' different from 'F', etc.

TABLE XXII

COMPUTATION TIMES THE PROGRAM TAKES TO MATCH EACH PAIR OF OBJECTS

recognition task	computing time (seconds)
match 'A' and its tranformation (30 degree rotation and 80% scaling)	10.9
match 'F' and its transformation (45 degree rotation and 70% scaling)	11.0
match 'A' and its transformation (90 degree rotation and no scaling)	14.5
match 'PART' and its transformation (90 degree rotation and no scaling)	43.4
match 'HOUSE' and its transformation (90 degree rotation and no scaling)	41.5
distinguish 'HOUSE' from 'PART'	20.1
distinguish 'F' from 'A' ('A' as model)	3.1
distinguish 'A' from 'F' ('F' as model)	3.7

Usually the program takes less time to recognize simpler objects than complicated objects because there are more lines to be extracted and processed in the complicated objects. It takes far less time to distinguish different objects. Table XXII gives the computation times it takes to perform the tasks in some of the examples shown above. The tasks include: performing the Hough Transform for each image, extracting lines

from each image, and matching the two sets of lines. The computing time is the total time during the program execution, including time spent on the system and the time to execute the program.

CHAPTER V

DISCUSSION AND CONCLUSION

We proposed an algorithm to recognize objects invariant of size, orientation and position based on the straight line features of the objects. The Hough Transform and its modifications are used to extract the lines. Instead of matching the object to every rotation and scale of the prototype image, as in the generalized Hough Transform [10], the algorithm starts from the orientations of the line segments to search for a possible angle match and then carries out scale detection and position matching. If the matching in the latter steps fail, it continues to search for another possible angle match.

Our tests show that the algorithm proposed here is able to do the invariant recognitions with correct results most of the time for simple objects.

In our tests only 32×32 image arrays are used. If the images are of higher dimensions better results should be achieved. For exact matchings, it is important to have an accurate representation of an object. For example, the features of the line segments can be represented in different resolutions. If the resolution for the angle of the lines is coarse, some objects may not be distinguished from some that are different. But if the resolution is too fine, there will be many more peaks in the Hough Space, which gives rise to difficulties in extracting the correct lines and the matching process. Computing time will also increase with finer resolutions in the image space and parameter space.

In another respect, if the lines are not accurately represented, the program has to accept errors in angle matching, scale detection and end point matching. If errors are not accepted, the program tends to be unable to recognize the objects of the same class

because of representation differences. But if the errors are not set appropriately, the program tends to recognize something really different as the same. This causes a misclassification.

The major variables that influence the recognition results are: the dimensions of the image space, the quantizing resolution of the parameter space, threshold selection for line extractions, errors allowed in angle matching and end point matching. These variables have to be different for different classes of objects. In our tests, usually these variables depend on how the lines in each object are distributed. If the line lengths are uniform, these variables can be set easily; if there are very long lines and very short lines present in the same object, it is difficult to set a threshold so that the lines will be extracted correctly. For practical applications of a special set of objects, the representation problem deserves further study.

So far the algorithm has been implemented to recognize a complete object without other objects present. A natural extension of this is to detect an object from a scene, or to detect a partially occluded object. By the virtue of the generalized Hough Transform, it is possible to achieve this with minor modifications. We could use more information in orientation detection, and also give a certain degree of tolerance to the number of lines unmatched. In scale detection, only the corresponding lines of the lines in the prototype image are used to calculate the scale ratio. Finally, the generalized Hough Transform can be used to pick a set of points from a scene easily.

There are potential applications of this algorithm for example motion detection and industrial part recognition for which special architectures can be developed.

A SELECTED BIBLIOGRAPHY

[1] P. V. Hough, "Method and Means for Recognizing Complex Patterns", U. S. Patent 3,069,654, 1962.

[2] R. O. Duda and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in the Pictures", Commun. ACM, vol. 15, no. 1, pp. 11-15, 1972.

[3] R. M. Inigo, E. S. McVey, B. J. Berger, and M. J. Wirtz, "Machine Vision Applied to Vehicle Guidance", IEEE T-PAMI, vol. 6, no. 6, pp. 820-826, 1984.

[4] C. R. Dyer, "Gauge Inspection Using Hough Transform", IEEE T-PAMI, vol. 5, no. 6, pp. 621-623, 1983.

[5] K. Y. Huang, K. S. Fu, T. H. Sheen, and S. W. Cheng, "Image Processing of Seismograms: (A)Hough Transformations for the Detection of Seismic Patterns", Pattern Recognition, vol. 18, no. 6, pp. 429-440, 1985.

[6] M. Kushnir, K. Abe, and K. Matsumoto, "Recognition of Handprinted Hebrew Characters Using Features Selected in the Hough Transform Space", Pattern Recognition, vol. 18, no. 2, pp. 103-114, 1985.

[7] D. B. Shu, C. C. Li, J. F. Mancuso, and Y. N. Sun, "A Line Extraction Method for Automated SEM Inspection of VLSI Resist", IEEE T-PAMI, vol. 10, no. 1, pp. 117-120, 1988.

[8] P. M. Merlin and D. J. Farber, "A Parallel Mechanism for Detecting Curves in the Pictures", IEEE Trans. Comput., vol. 24, no. 1, Jan., 1975.

[9] S. Tsuji and F. Matsumoto, "Detection of Ellipses by Modified Hough Transformation", IEEE Trans. Comput., vol. 27, pp. 777-781, 1978.

[10] J. Sklansky, "On the Hough Technique for Curve Detection", IEEE Trans. Comput., vol. 27, no. 10, pp. 923-926, 1978.

[11] D. H. Ballard, "Generalizing the Hough Transform to Detect Arbitrary Shapes", Pattern Recognition, vol. 13, no. 2, pp. 111-122, 1981.

[12] D. H. Ballard and C. M. Brown, COMPUTER VISION, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1982.

[13] M-K, Fu, Visual Pattern Recognition by Moment Invariants", IRE Transactions on Information Theory, pp. 179-187, February, 1962.

[14] L. Gupta and M. D. Srinath, "Invariant Planar Shape Recognition Using Dynamic Alignment", Pattern Recognition, vol. 21, no. 3, pp. 235-239, 1988.

[15] R. A. Messner and H. H. Szu, "An Image Processing Architecture for Real Time Generation of Scale and Rotation Invariant Patterns", Computer Vision, Graphics, and Image Processing, vol. 31, no. 1, pp. 50-66, July 1985.

[16] H. Wechsler and G. L. Zimmerman, "2-D Invariant Object Recognition Using Distributed Associative Memory", IEEE T-PAMI, vol. 10, no. 6, pp. 811-821, November 1988.

[17] H. Y. H. Chuang and C. C. Li, "A Systolic Array Processor for Straight Line Detection by Modified Hough Transform", Proc.IEEE CS Workshop Computer Architecture for Pattern Analysis and Image Management, Miami Beach, FL, pp. 300-304, Nov., 1985.

[18] F. M. Rhodes, J. J. Dituri, G. H. Chapman, B. E. Emerson, A. M. Soares and J. I. Raffel, "A Monolithic Hough Transform Processor Based on Restructurable VLSI", IEEE Trans. PAMI, vol. 10, no. 1, pp. 106-110, Jan., 1988.

[19] T. M. Silberberg, "The Hough Transform on the Geometric Arithmetic Parallel Processor", Proc. IEEE CS Workshop Comp. Architect. for Pat. Anal. and Image Manag., Miami Beach, FL, pp. 387-393, Nov., 1985.

[20] H. Li, M. A. Lavin and R. J. LeMaster, "Fast Hough Transform", Proc. 3rd Workshop Computer Vision: Representation and Control, Bellair, MI, pp. 75-83, 1985.

[21] H. Li, "Fast Hough Transform for Multidimensional Signal Processing", Proc. ICASSP86, Tokyo, Japan, pp. 2063-2066, 1986.

[22] H. Li and M. A. Lavin, "Fast Hough Transform Based on Bintree Data Structure", Proc. Conf. Computer Vision and Pattern Recognition, Miami Beach, FL, pp. 640-642, 1986.

[23] J. Illingworth and J. Kittler, "The Adaptive Hough Transform", IEEE Trans. PAMI, vol. PAMI-9, no. 5, pp. 690-698, Sept., 1987.

[24] M. Cohen and G. T. Toussaint, "On the Detection of Structures in Noisy Pictures", Pattern Recognition, vol. 9, pp. 95-98, 1977.

[25] C. M. Brown, "Inherent Bias and Noise in the Hough Transform", IEEE Trans. PAMI, vol. PAMI-5, no. 5, pp. 493-505, Sept., 1983.

[26] G. C. Stockman and A. K. Agrawala, "Equivalence of Hough Curve Detection to Template Matching", Comm. ACM, vol. 20, no. 11, pp. 820-822, Nov., 1977.