PARALLEL OCCLUDED OBJECT RECOGNITION
USING EXTENDED LOCAL FEATURES
ON THE HYPERCUBE MULTI-
PROCESSOR COMPUTER

By

JOONG HWAN BAEK

Bachelor of Science in Engineering
Hankuk Aviation College
Seoul, Korea
1981

Master of Science in Electrical Engineering
Oklahoma State University
Stillwater, Oklahoma
1987

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, 1991

Thesis
1991D
B139p
cop.2

Oklahoma State Univ. Library

PARALLEL OCCLUDED OBJECT RECOGNITION USING EXTENDED LOCAL FEATURES ON THE HYPERCUBE MULTI-PROCESSOR COMPUTER

Thesis Approved:

Keith A. Teague
Thesis Adviser

C. M. Bacon

Ronald P. Rhoten

J Chandler

James E Baker

Norman N Durham
Dean of the Graduate College

1407016

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to my advisor, Dr. Keith Teague, for his excellence in guiding and encouraging his student; I am indebted to him for his inspiration and knowledge. The suggestions and assistance he has provided have helped to make my work satisfying and enjoyable. I would also like to thank Dr. Charles Bacon, Dr. Ronald Rhoten, and Dr. John Chandler for their helpful comments and discussions; they were willing to help me and shared their precious time with me.

I extend my appreciation to my mother. She encouraged and supported me all the way and helped me keep the end goal constantly in sight. This work is dedicated to her.

TABLE OF CONTENTS

Chapter Page

I. INTRODUCTION 1

Previous Work in Object Recognition 2
    Global Feature Method 3
    Local Feature Method 5
Problem Definition 7
Objectives and Proposed Research 9
Overview 10

II. PARALLEL PREPROCESSING ON THE HYPERCUBE MULTIPROCESSOR COMPUTER 15

The iPSC/2 Hypercube 15
Parallel Edge Detection 17
    Review of the Edge Detection Techniques 17
    Input Image Distribution Method 21
    Parallel Edge Detection Algorithm 23
Parallel Straight Line Extraction 23
    Background 23
    Development of Parallel Straight Line Extraction Algorithm 24
Parallel Thinning 28
    Parallel Boundary Detection and Object Extraction 30
    Parallel Object Thinning 32
Experimental Results and Speedup 35

III. PARALLEL FEATURE EXTRACTION 41

Corner Detection 41
    Data Structure 41
    Parallel Corner Detection 43
    Computation of Corner Angle 44
Arc Detection 46
Parallel-Line Detection 53
Corner-Arc Detection 55
Parallel Line and Arc Drawing 57
    Parallel Line Drawing 57
    Parallel Arc Drawing 62

IV. PARALLEL MATCHING AND VERIFICATION 65

Modeling and Hypothesis Generation 66
Parallel Matching 72

Chapter Page

Parallel Implementation 72
Compatibility Between a Model Feature and an Image Feature 73
Coordinate Transform 75
Cluster Seeking 77
Hypothesis Verification of the Matching 80

V. EXPERIMENTAL RESULTS 82

Experiment 1 (Preliminary Test) 82
Experiment 2 (Precision Test) 102
Experiment 3 (Comprehensive Test) 106

VI. SUMMARY AND CONCLUSIONS 115

Summary of Results 116
Suggestions for Future Research 117

REFERENCES 121

APPENDIXES 128

APPENDIX A - DERIVATION OF EQUATIONS (III.5), (III.6), AND (III.7) 129

APPENDIX B - DERIVATION OF EQUATIONS (IV.20), (IV.21), AND (IV.22) 131

APPENDIX C - RESULTING DATA OF EXPERIMENT 3 133

LIST OF TABLES

Table Page

I. Comparisons of the Processing Times for Thinning 40

II. Determination of the Octants 60

III. Data of the Corners of the Long Nose Plier 91

IV. Comparisons of Matching Ratios 99

V. Comparisons of Processing Times on 16 Nodes 101

VI. Results of the Cluster Seeking for the Precision Test 104

VII. Number of Features 107

VIII. Overall Processing Times for the Example Shown in Figure V.14 114

LIST OF FIGURES

Figure Page

I.1. Block Diagram of the Proposed Object Recognition System 11

I.2. Sample Objects Used in This Research 12

II.1. Hypercube Architecture in d-Dimensions 16

II.2. Numbering for 3 x 3 Edge Detection Operators 18

II.3. Sobel Operators 19

II.4. Laplacian Operators 19

II.5. Laplacian of Gaussian Operator (9 x 9, $\sigma = 1.7$) 22

II.6. Input Image Distribution Method 22

II.7. Pseudocode for Parallel Segment Linking Algorithm 25

II.8. Parallel Pass-Segment Linking Algorithm 27

II.9. A C Program for Polygonal Approximation 27

II.10. An Example of the Polygonal Approximation Method 29

II.11. Eight-Neighbor Defintion 31

II.12. Bound, Bbe, and Tbe 32

II.13. Eight Boundary-Following Directions 33

II.14. Results of Shrinking a Simple Cross Object After One Iteration 34

II.15. Parallel Line and Single Line. 35

II.16. Processing Times on VAX 11/750 and iPSC/2
(9 x 9 Laplacian of Gaussian Operator, 8 Nodes) 36

II.17. Speedup When Using Laplacian of Gaussian Operator 37

II.18. Processing Times for Straight Line Extraction According to Various
Number of Nodes 38

II.19. Result of the Thinning 39

Figure Page

III.1. Parameters of a Straight Line 42

III.2. A Corner and Its Parameters 42

III.3. Global Straight Line Division Method 44

III.4. Deviations of the Straight Lines According to the Different Directions 45

III.5. Parameters of an Arc 47

III.6. Error Between the Arc and One Vertex of the Polygonal Approximated Line 47

III.7. A Method for Estimating the Initial Center Point and Radius of the Candidate of an Arc 51

III.8. A Block Diagram of Parallel Arc Detection 52

III.9. Parameters for the Significance of Parallelism Between Two Lines . . . 54

III.10. A Degree of Alignment (*doa*) of Two Lines 55

III.11. Parameters of a Corner-Arc 56

III.12. Notation for Bresenham's Algorithm 58

III.13. A C Program for Bresenham's Algorithm 61

III.14. Global and Local Image Planes 62

III.15. A C Program for the Arc Drawing 63

IV.1. Structures of a Model Database 66

IV.2. Ranges of the Parameters 68

IV.3. Formats of the Keys for the Hashing 69

IV.4. A Flow Chart for the Insertion of a Feature Data 71

IV.5. Compatibility Between a Model Feature and an Image Feature 74

IV.6. Coarse-Verification Method 81

V.1. Images of Model Objects and Input 83

V.2. Edge Detected Image from Figure V.1 (d) 87

V.3. Straight Lines Extracted from Figure V.2 88

V.4. Types of Arrangement of Two Straight Lines 90

Figure Page

V.5. Corners of the Long Nose Plier 92

V.6. Arcs of the Long Nose Plier 95

V.7. Parallel-Lines of the Allen Wrench 96

V.8. Corner-Arcs of the Adjustable Wrench 97

V.9. Verification of the Matching 100

V.10. Straight Lines of the Combinational Wrench for the Precision Test . . . 102

V.11. A Tree-Dimensional Plot of the Coordinate Transforms. 103

V.12. Verification of the Matching for the Precision Test 105

V.13. An Example of Smoothing 110

V.14. An Example of Matching and Verification of a Synthetic Image 112

CHAPTER I

INTRODUCTION

One goal of industrial robot vision research is to give robots humanlike visual capabilities so that the robots can sense the environment in their field of view, understand what is being sensed, and take appropriate actions as programmed. Over the past three decades, a number of serious efforts have been devoted to this challenging subject, especially to the problem of recognizing and locating objects in the workspace of the robot.

Some commercial or non-commercial vision systems for robotics and industrial automation have been developed. Some examples of such systems are SIGHT-I [Bai78], VS-100 [Che82], HYPER [Aya86], and SCERPO [Low87]. However, their capabilities are still very primitive. One good reason for this slow progress is that sophisticated visual interpretation is needed in an industrial robot vision system.

Basic requirements for an industrial robot vision system are speed, accuracy, and flexibility [Chi86] [Koc87] [Han89]. The processing speed for acquiring and analyzing an image must be high enough to execute a specific task while the system must recognize objects and determine their positions and orientations in high accuracy. Also, the vision system must be flexible enough to accommodate variations of the objects and their environment.

However, realization of these requirements is very difficult because extensive computations are involved in the processing of the image, and hardware is usually limited. Also, the objects can usually overlap or occlude in the actual environment. This problem can cause severe degradation in the rate of object recognition and accuracy.

Since the recognition of occluded objects is a very important problem to be solved for industrial robot vision applications, our research is focused on this topic. The main goal of the research is to develop a fast, accurate, and reliable system which can recognize partially occluded objects. Unlike other object recognition systems [Aya86] [Bol82] [Koc87] [Bha87] [Li89], we propose to use *extended local features* consisting of corners, arcs, parallel-lines, and corner-arcs to increase the accuracy of the system as well as the recognition rate. We also propose to use *hashing* in constructing a knowledge-base and searching feature data from the knowledge-base to reduce the searching time. In order to increase the speed of the system, we develop parallel algorithms for all procedures in the system and implement them on a hypercube-topology multiprocessor computer, the Intel iPSC/2.

In this research some reasonable assumptions are made to make the system realizable: 1) The shapes of the objects are unique, 2) The objects are flat, rigid, and hardly deformable, and 3) The objects are translated and rotated only in x and y planes. In the following sections, we review previous work in object recognition and define some problems in the work. Then we describe our objectives and proposed research in detail.

Previous Work in Object Recognition

The most straightforward way to recognize the objects would be to match models for each possible combination of identity, position, and orientation to the image. Besl and Jain [Bes85] show that such an exhaustive search is hopeless for the case of three-dimensional objects. Because of the complexity of this task, researchers have instead turned to recognizing objects by their features.

The features can be classified into two types: *global* and *local* features. The classic technique for object recognition is pattern recognition using global feature vectors. The idea is to describe the object with a list of numerical values that are invariant to translation, rotation, and possibly scaling. Examples are perimeter, centroid, area, and moments of inertia. On the other hand, a local feature is a feature that depends only on portions of the object. The local features describe more complex properties of the object, usually in terms of line and curve segments defining the object's boundary. Typically, the local features are organized as a highly structured and abstracted representation.

Global Feature Method

An *ad hoc* set of global features is used by the SRI vision module [Gle79]. The module includes a camera which sees a binary silhouette of the object. From this silhouette several global descriptors are computed. These include area, perimeter, number of holes, area of holes, maximum and minimum radii from the object's centroid, the ratio of these two radii, and others. These descriptors are formed into a feature vector, and the object is recognized using a nearest neighbor classifier.

Agin has developed a computer vision system for industrial inspection and assembly in [Agi80]. As implemented in the SRI vision module [Gle79], the connectivity analysis program builds a description of each blob as the image is processed. An array, which he calls a *blob descriptor*, is created to hold information about the blob and its shape. The types of information in a blob descriptor derive a number of shape and size descriptors characterizing the blob. The size and shape descriptors include area, perimeter length, the ratio perimeter^2/area, moments of inertia, number of holes, sum of the areas of the holes, etc. A storage array called a *prototype descriptor* is associated with each prototype or model. The mean and variance of each measure of size or shape (feature value) are stored in the descriptor. Given several prototype descriptors and the image of an unknown part shape, it is determined to which object class the part belongs by the nearest neighbor procedure and the binary decision tree procedure.

Birk *et al.* [Bir81] model objects by a set of coarse shape features for each possible viewpoint. For a given viewing position, the threshold object is overlaid with a 3 x 3 grid (each grid square's size is selected by the user), centered at the object's centroid, and oriented with respect to the minimum moment of inertia. A count of the number of above threshold pixels in each grid square is used to describe and recognize the object.

Persoon and Fu [Per77] use shape discrimination method using Fourier descriptors in character recognition and machine parts recognition. Objects are described as the Fourier descriptors of their boundary curves (shapes). They use the Euclidean metric in the space of the Fourier descriptors to compute the distance between the unknown sample and the nearest sample in the training set. They scale, rotate, and adjust the starting point of each sample in the training set in order to minimize the resulting Euclidean distance. In order to classify an unknown numeral, the distance between that numeral and each numeral in the training set is computed.

The normalized Fourier descriptor has been used to recognize aircraft from silhouettes [Wal80]. An object's Fourier descriptor is found by taking the Fourier transform of its boundary curve. The boundary is treated as a periodic complex function with real and imaginary parts corresponding to the x and y coordinates. The descriptor is then normalized to a standard location, rotation angle, size, and boundary curve trace starting point. The unknown object's normalized descriptor is compared to each normalized descriptor in the object library. The object is identified by the closest match.

Kulkarni *et al.* present a neural network model for invariant object recognition in [Kul90]. The model consists of two stages. In the first stage features are extracted from an image of an object, and the second stage is used to recognize the object. For invariant feature extraction, moment invariants [Hu62] are applied. Moment invariants are defined by a set of seven moment invariant functions that are invariant to translational, scale, and rotational differences in input patterns. In the recognition stage, a neural network is used as a classifier.

Wechsler and Zimmerman [Wec88] [Wec89] describe an approach to two-dimensional object recognition using distributed associative memory which can modify the flow of information. Stimulus vectors are associated with response vectors and the result of this association is spread over the entire memory space. Complex-log invariant image mapping is combined with the distributed associative memory to yield a system able to recognize a memorized object regardless of the scale and rotation. Recalled information from the memorized database is used to classify an object, reconstruct the memorized version of the object, and estimate the magnitude of changes in scale or rotation.

Local Feature Method

Perkins [Per78] has developed a vision system which can determine the position and orientation of complex curved objects. He constructs two-dimensional models from boundary segments, called concurves, constructed from line segments and arcs that are extracted from training images of each stable view of each part. All possible combinations of image and model concurves are matched and ranked according to general properties, such as total length, number of lines and arcs, etc.

Bolles and Cain [Bol82] describe the local-feature-focus method, which is an algorithm to recognize and locate partially visible two-dimensional objects. First the image is scanned to detect local features. In the example presented in their paper, three types of features were detected in the binary images: round holes, convex 90° corners, and concave 90° corners. The position of each feature is recorded, along with its size and orientation, if any. The matching process utilizes clusters of mutually consistent features to hypothesize objects and also uses templates of objects to verify these hypotheses.

Ayache [Aya83] [Aya86] has used polygons to represent objects and uses the polygon line segments as features. The longer line segments determine privileged features. These privileged segments determine an initial hypothesis when matched to scene line segments. Each hypothesis receives a quality score and a coordinate transform that takes the privileged segment onto the scene. Matching other model segments revises the quality score and a Kalman filter updates the coordinate transform. The quality score essentially determines how much the model resembles the scene objects.

Turney *et al.* [Tur85] develop an algorithm to identify and locate partially visible two-dimensional objects using a sub-template based on the Hough transform. Instead of using edge points as in the normal Hough transform, overlapping segments of boundary are used. For each segment, a vector is stored pointing to the object's centroid. Whenever a subtemplate matches in the image, an accumulator at the position pointed to by the subtemplate's vector is incremented. After all the object's subtemplates have been matched against the edges in the image, the accumulator with the largest value that is above a threshold is considered to be the position of the object.

Koch and Kashyap [Koc85] [Koc87] present a vision system that recognizes and locates partially occluded objects. The vision system uses stored models to locate and identify the objects in the scene. The models are based on the boundary of the objects. From the polygon approximation of the boundary, vertices of high curvature are identified as corners. These corners are used as features in detecting the model in the image. A globally consistent coordinate transform that takes the model into the image is found by using a Hough-like transform and the corner features.

Bhanu and Ming [Bha86] [Bha87] present a method based on a cluster-structure approach for the recognition of two-dimensional partially occluded objects. Basically, the technique consists of the following steps: 1) determine the initial disparity matrix, 2) cluster the samples in the disparity matrix using the K-means algorithm, 3) find and thin the sequences in the current cluster, 4) cluster the sequence averages using K-means algorithm, and 5) use the maximum distance algorithm to select the final set of matching segments from the cluster with the best maximum distance results.

Mehrotra *et al.* [Meh90] has developed an industrial part recognition method using a component-index. A component corresponds to a convex section of the object boundary. A component of the unanalyzed portion of the scene is identified using *k-d* tree-based component-index. The objects that contain the identified component are hypothesized to be present in the scene. A given hypothesis is verified by matching the transformed boundary of the hypothesized object against the scene.

Problem Definition

The global features are easy to compute from binary images, and their ordering in the model is unimportant. But the global features depend on the whole object being visible. Hence, they are of little value when the objects may be touching or overlapping. On the other hand, the local features are extracted from only part of the object, but when combined describe the whole object. Therefore, local features can be used for the recognition of occluded objects since the objects can be recognized by using only a subset of the features [Koc85].

Some researchers have used local features for recognizing partially occluded objects. Ayache *et al.* [Aya86] and Rummel *et al.* [Rum84] used lines, but Perkins [Per78] used arcs. On the other hand, Koch *et al.* [Koc85] [Koc87] and Li *et al.* [Li89] used corners. However, the recognition rate and accuracy of their systems could be decreased severely as the occluded portion of the objects is increased because only a single type of local feature is used.

Most object recognition systems [Koc85] [Koc87] [Bha87] [Han89] convert a gray-level image to a simple binary image by using a thresholding technique. If the intensity of the pixel is above a certain threshold, that pixel is assigned to the background. Otherwise it is labeled as belonging to the object. A closed contour of the object is extracted from the binary image by following the edges. However, it is very difficult to separate the objects from the background by using the thresholding technique in the case of low contrast or noisy images. Therefore, more sophisticated algorithms would be required to extract a closed contour of the object in an image taken under poor lighting conditions.

Usually, before the features are extracted, the digitized image undergoes several preprocessing (or early processing) steps such as median filtering, edge detection, thinning, boundary detection, and straight line extraction. The preprocessing steps are usually the most time-consuming components in the object recognition system, because the digitized images to be processed generally consist of two-dimensional arrays of pixels. The processing requirements grow as n^2 for an n x n image.

For example, McIntosh and Mutch presented a new approach to match straight line segments extracted from image pairs in [McI88]. According to their experiments, line extraction took 28 seconds while matching lines using constraints took 2.5 seconds on an Apple McIntosh computer. Note that the size of the image they used was only 128 x 128. In Ayache *et al.* [Aya87], edge extraction for a 512 x 512 image required an average of 120 seconds of CPU time on a Sun 3 workstation.

Real-time vision applications demand much higher speed than this. Video frame rate is typically either 30 or 60 frames per second, so preprocessing must be measured in milliseconds, not seconds. Preprocessing requires extensive computations and thus the overall system is very slow. We could increase the speed of the system by reducing the complexity of the preprocessing and/or by exploiting parallelism inherent in the process.

Most industrial object recognition systems are *knowledge-based* systems in which objects in the input image are matched with a set of predefined models of objects. Before on-line matching, distinctive features are extracted from the model objects and then those features are usually stored in a *knowledge-base.* For the matching, compatible features between the input object and model object are searched from the knowledge-base. The searching time grows linearly as the number of the model objects increases. To reduce the searching time, some researchers construct the knowledge-base in the form of a decision tree [Gri88] [Meh90]. But, the depth of the tree becomes greater as the number of model objects increases, or as the objects become more complex. Thus, the searching time would be increased. Also the decision trees are not easily extended to use local features [Kno86].

Objectives and Proposed Research

The goal of this research is to develop a new vision system that recognizes and locates partially occluded objects. We first propose to use extended local features consisting of corners, arcs, parallel-lines, and corner-arcs. These extended local features will increase the accuracy of our system. An object is simply modeled by the local features and then the feature data is stored in a knowledge-base. For recognizing an object, we test the compatibility between the model feature data and the image feature data. Koch *et al.* develops a matching method which brings a model point to the corresponding image point using coordinate transformation and cluster-seeking in [Koc85]. However his method is only for corners. We expand his method for arcs, parallel-lines, and corner-arcs.

Second, we propose to develop robust preprocessing techniques such as edge detection and straight line extraction so that our system handles gray-level noisy images. The reliability of the system depends on how well the edges or straight lines are extracted from the input image because the features are extracted from the preprocessed image. Since we have already studied edge detection and straight-line extraction in [Bae89-1], [Bae89-2], and [Bae90], we expand the previous work for this research.

Recently, several commercial systems offering 100 - 1,000 processors in a hypercube configuration have been announced in the super-mini computer price range. Intel's personal supercomputer, the iPSC/2, is an example. It comprises 32, 64, or 128 processing nodes connected in a regular hypercube topology [Int86]. As mentioned earlier, preprocessing steps such as median filtering, edge detection, straight-line extraction, thinning, and boundary detection, etc., are the most time consuming procedures in an object recognition system. However, those processing steps might be implemented on a multiprocessor machine to reduce the processing time, since most image processing tasks (particularly in early processing) exhibit a high degree of inherent parallelism.

Third, we propose to develop parallel algorithms for the preprocessing steps and implement them on the Intel iPSC/2 hypercube machine, and then analyze our parallel algorithms. In addition to the development of the parallel algorithms for the preprocessing, we also propose to increase the processing parallelism for the remaining procedures such as feature extraction, matching, and verifying, then implement them on the hypercube.

Fourth, we propose to construct the knowledge-base using *hashing*, which can reduce the searching time significantly because a key (or feature) can be searched in one access time most of the time. The space of each type of the extended local features is discretized by equally dividing the space. Each feature data is converted into a unique key, and then a hashing function generates a home address for the key. Figure I.1 shows a block diagram of the proposed object recognition system. Note that the whole system is implemented on the iPSC/2.

Finally, we test the system for a variety of combinations of industrial objects such as nippers, screwdrivers, pliers, wrenches, etc. Sample objects used in this research are shown in Figure I.2. Also we perform a precision test for the system by measuring the difference between the desired coordinate transform and the actual coordinate transform.

Overview

Chapter II presents some parallel algorithms for the preprocessing steps such as edge detection, straight line extraction, and thinning. The iPSC/2 hypercube multiprocessor is introduced first. Then an overview of some conventional edge detection techniques using operators such as Sobel, Laplacian, and Laplacian of Gaussian is provided and a parallel edge detection algorithm is developed. For parallel straight line extraction, a parallel pass-segment linking algorithm is developed. Also, a new parallel

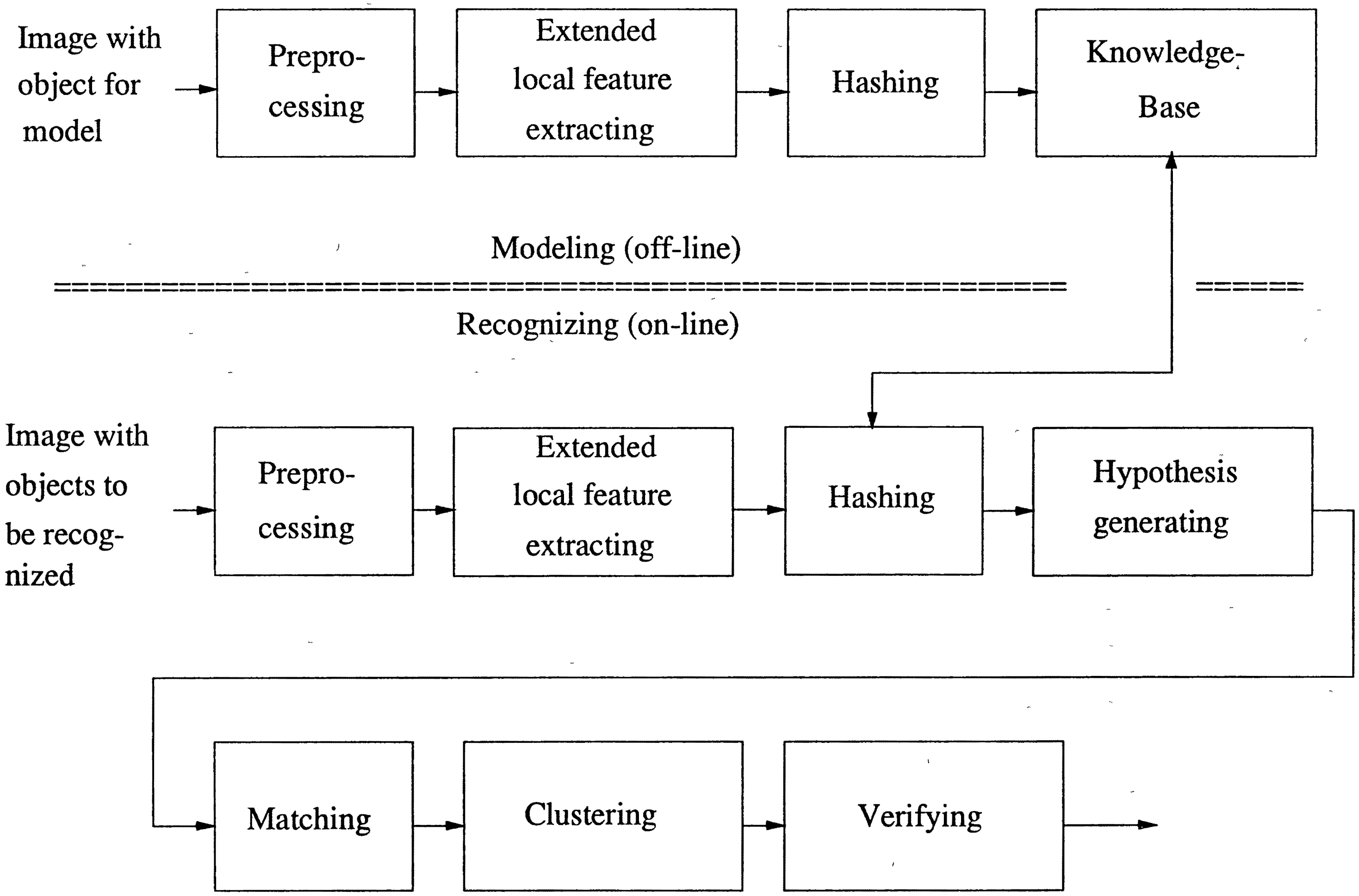


Figure I.1 Block Diagram of the Proposed Object Recognition System

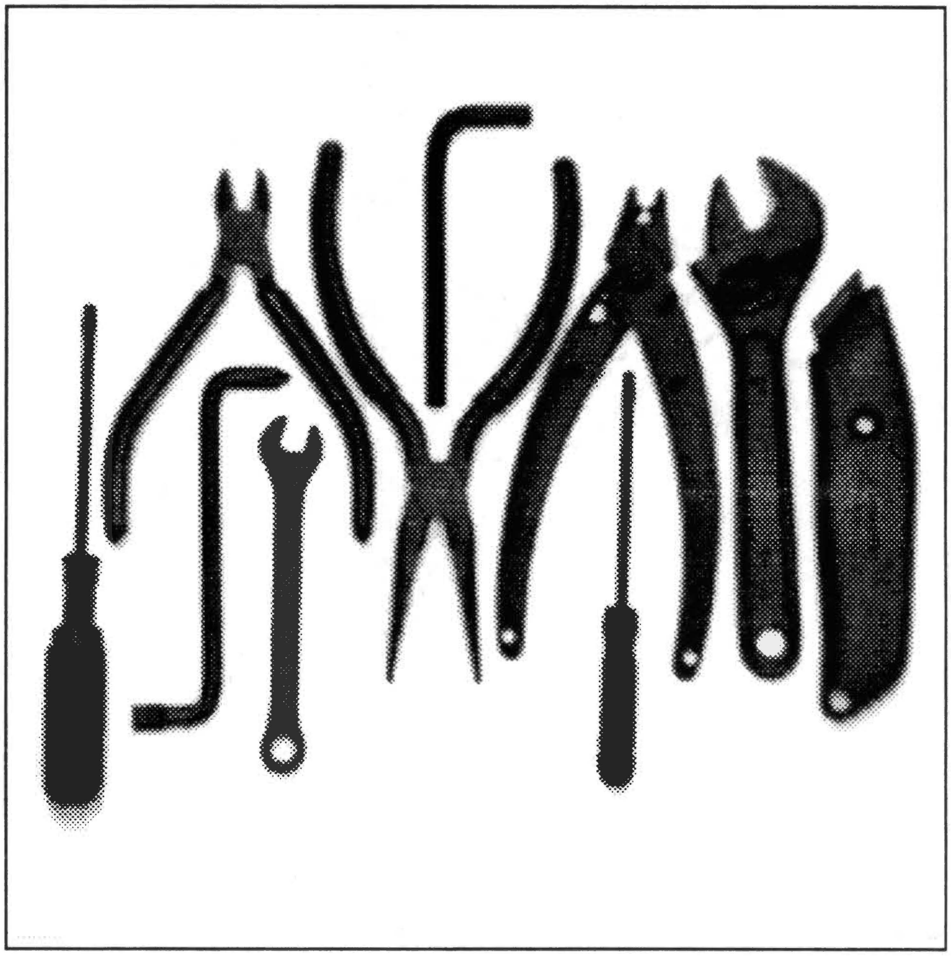


Figure I.2 Sample Objects Used in This Research

thinning algorithm based on boundary following and shrinking is proposed. The parallel preprocessing algorithms are implemented on the hypercube and their performance is analyzed.

Chapter III describes how to extract the extended local features such as corners, arcs, parallel-lines, and corner-arcs in parallel. The structures for the feature data are declared and their parameters are shown. For parallel implementation of the corner detection, a feature data distribution method is developed which can be extended to the other features. Equations for estimating the center point and radius of an arc candidate are derived by the least square error method, and then the Newton's method for solving the equations is described. Also, a method for accurately estimating an initial center point and radius is developed because the initial values are very important for reducing the number of iterations in the Newton's method. To detect parallel-lines, a significance of parallelism and a degree of alignment between two straight lines are defined and depicted. Finally the corner-arc detection method is described. At the end of the chapter, a parallel line and arc drawing algorithm is developed.

Parallel matching and verification methods are presented in Chapter IV. First of all, the object modeling and hypothesis generation schemes using hashing are explained in detail. Formulas for testing the compatibility between a model feature and an image feature are given according to the feature type. Then a coordinate transform, which brings a model point to the corresponding image point, is derived from the compatible features. The coordinate transforms will be clustered if the objects are correctly matched. Several existing cluster seeking algorithms are reviewed, and then the maximin-distance algorithm is selected for our use. In order to verify the matching, two methods are developed: coarse-verification method and fine-verification method.

In Chapter V, three experiments are performed and their results are reported. In the first experiment a simple occluded object image is used for a preliminary test. Some resulting images of the extended local feature extraction and the verifications of the matchings are shown. A precision test is performed in the second experiment. Differences between the desired coordinate transform and the actual coordinate transform are provided in terms of the pixels for the translation and the angles for the rotation. Also, the resulting data of the cluster seeking is tabulated. In the third experiment, a comprehensive test is performed with 20 synthetic occluded object images. The method for constructing the synthetic images is described. Also, an occlusion ratio for each object is computed. Finally, the hypothesis and matching rates are reported, and the overall processing times are summarized.

Chapter VI provides a summary of the results of this research. Some conclusions and a motivation for further research are presented. Suggested topics for future investigation are included at the end.

CHAPTER II

PARALLEL PREPROCESSING ON THE HYPERCUBE MULTIPROCESSOR COMPUTER

As we discussed earlier, one of the requirements for a successful robot vision system is *speed*. However, preprocessing such as edge detection, straight line extraction, and thinning is time-consuming because large data sets and extensive computations are involved. Since the proposed system involves numerous local features, this becomes even more of a problem. Our attempted solution to this problem is to increase the processing parallelism and implement the processing on a hypercube-topology multiprocessor computer, the Intel iPSC/2.

In this chapter, we first introduce the iPSC/2 hypercube multiprocessor. Then we review some edge detection techniques and develop efficient parallel algorithms for edge detection and straight line extraction on the hypercube. Also, we develop a new parallel thinning algorithm based on boundary following and shrinking. Finally, we test the algorithms on the Intel iPSC/2 with some images and analyze their performance.

The iPSC/2 Hypercube

The hypercube is a MIMD (Multiple Instruction stream and Multiple Data stream) machine with a distributed memory model that characterizes loosely coupled processing. A hypercube represents a geometrical interconnection scheme having 2^d identical processing nodes, in which *d* represents the dimension of the hypercube. For example, a five-dimensional hypercube is a 32-node (2^5) system with each node connected to its five nearest neighbors. Figure II.1 shows graphically hypercubes of various sizes. Each

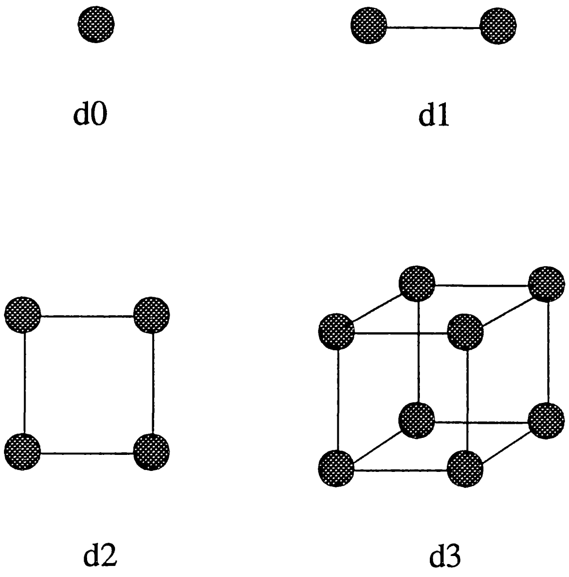


Figure II.1 Hypercube Architecture in d-Dimensions

processor has its own memory and communicates with others by message-passing.

The iPSC/2 is a second generation machine with improved computation and communication capability. The system consists of two subsystems: the system resource manager (SRM, sometimes called the *host*), and the hypercube itself (sometimes called the *cube*). The SRM is a standalone microcomputer connected to one node of the hypercube. Each node consists of an Intel 80386/Weitek 1167 processor set along with special hardware for efficient message passing using a form of worm-hole routing [Nug88]. Each node provides processing power approximately equal to a VAX 11/780. Additionally, *I/O nodes* providing parallel disk I/O directly from the processing nodes may be added, but are not part of this system.

iPSC/2 programs are generally written in two parts: a host part that runs on the SRM, and a node part that runs on each node of the cube. Although this structure is not mandatory, it provides a convenient paradigm from which to work. This paradigm is usually referred to as Single Program Multiple Data (SPMD). Usually, the host program handles such non-parallel functions as terminal I/O, disk I/O, or network access. Meanwhile, the node program or programs handle all problem specific processing.

Parallel Edge Detection

Edge detection is a very important step in image processing and pattern recognition. An image can be simplified or converted into a line representation through the edge detection step, and then significant features of the image can be extracted from the result [Dud73]. In this section, we provide an overview of some conventional edge detection techniques using operators such as Sobel [Sob72] [Smi87], Laplacian [Pra84] [Gon83] [Ros82], and Laplacian of Gaussian [Mar80] [Gri85] [Tor86] [Lun86]. Then we discuss the image distribution method and the load balancing problem. Finally, we develop a parallel edge detection algorithm and implement it on the iPSC/2.

Review of the Edge Detection Techniques

The most commonly used method of edge detection in image processing applications is the gradient. Given a function $f(x,y)$, the gradient of f at coordinates (x,y) is defined as the vector

$$G[f(x,y)] = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{II.1}$$

The magnitude of $G[f(x,y)]$ is given by

$$\text{mag}[G] = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \tag{II.2}$$

For a digital image, the derivatives in Equation (II.2) are approximated by differences. Consider the 3 x 3 image region shown in Figure II.2. The $\partial f/\partial x$ is estimated by

$$\frac{\partial f}{\partial x} = (P_0 + 2P_1 + P_2) - (P_6 + 2P_5 + P_4) \tag{II.3}$$

Similarly, the $\partial f/\partial y$ is estimated by

$$\frac{\partial f}{\partial y} = (P_4 + 2P_3 + P_2) - (P_6 + 2P_7 + P_0) \tag{II.4}$$

Now, Equation (II.2) can be implemented by using two 3 x 3 orthogonal directional (vertical and horizontal) operators, which are often referred to as Sobel operators. The operators are shown in Figure II.3.

P_6	P_7	P_0
P_5	f(x,y)	P_1
P_4	P_3	P_2

Figure II.2 Numbering for 3 x 3 Edge Detection Operators

Another edge detection technique is to use a linear derivative operator given by

$$\nabla^2 f \equiv \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{II.5}$$

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

Figure II.3 Sobel Operators

which is called the Laplacian operator. For a digital image, the discrete analog of the Laplacian could be approximated as

$$\begin{aligned} \nabla^2 f(i,j) &\equiv \Delta_x^2 f(i,j) + \Delta_y^2 f(i,j) \\ &= 4f(i,j) - [f(i+1,j) + f(i-1,j) + f(i,j+1) + f(i,j-1)] \qquad \text{(II.6)} \end{aligned}$$

which can be represented as a single 3 x 3 operator. Figure II.4 shows this approximation to the Laplacian operator.

0	-1	0
-1	4	-1
0	-1	0

Figure II.4 Laplacian Operator

The edge detection technique using the Sobel or Laplacian operator is simple to implement, and relatively fast in terms of processing time. But these operators do not give thin edges of the image. Therefore, some extra processing such as edge thinning or edge following is needed when line edges are desired. The Laplacian is also more susceptible to noise.

Marr and Hildreth [Mar80] [Gri85] proposed an optimal edge detection method using the Laplacian of Gaussian. It makes use of significantly larger convolution masks, and its output consists of a set of connected edge contours which may be used directly for feature extraction. The basic idea is that an edge, or sharp intensity change, gives rise to a zero-crossing in a second directional derivative of the image.

Marr and Hildreth proposed the use of a Gaussian smoothing filter to reduce the sensitivity to noise. The Gaussian function is given by

$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-x^2}{2\sigma^2}\right) \tag{II.7}$$

In two dimensions,

$$G(r) = \frac{1}{2\pi\sigma^2} \exp\left(\frac{-r^2}{2\sigma^2}\right) \tag{II.8}$$

where r is the radial distance from the origin and σ is the standard deviation of the Gaussian smoothing function. Now we can write

$$\nabla^2 G(x) = -\frac{1}{\sigma^3\sqrt{2\pi}} \left(1 - \frac{x^2}{\sigma^2}\right) \exp\left(\frac{-x^2}{2\sigma^2}\right) \tag{II.9}$$

For two dimensions, an alternative representation of Equation (II.9) is

$$-\nabla^2 G(r) = \frac{1}{\pi\sigma^2} \left(1 - \frac{r^2}{2\sigma^2}\right) \exp\left(\frac{-r^2}{2\sigma^2}\right) \tag{II.10}$$

This equation is circularly symmetric. Note that the r in Equation (II.10) is set to $2\sqrt{2}\sigma$.

Although line edges can be obtained by the optimal edge detection technique, one degree of freedom is left: the σ in Equation (II.9) or the r in Equation (II.10). Those variables vary the width and height of the Laplacian of Gaussian function, and thus they affect the resolution of the resulting edges. In this research, the magnitude of the gradient at every zero-crossing point is computed by the Sobel operators. Then only the points with magnitude above a threshold are considered as edges. Equation (II.10) is transformed to be an 9 x 9 or larger operator to provide adequate smoothing. Figure II.5 shows a 9 x 9 Laplacian of Gaussian operator when $\sigma = 1.7$. Note that the values are scaled by 1000.

Input Image Distribution Method

Load balancing is a crucial factor in parallel processing because imbalance can cause severe degradation of performance. In order to maximize performance, the amount of work performed by each node must ideally be equal. In many cases this implies that the amount of data loaded to each node must also be equal. These algorithms exhibit this property. In this research the input image plane is divided into rows as evenly as possible according to the number of nodes being allocated. Each resulting sub-image is distributed to a node. If the size of the input image is M x M, the original computational complexity is $O(M^2)$. It is obvious that the processing time can be reduced in theory to $O(M^2/N)$ by using the distribution method, where N is the number of nodes being used.

Note that some rows on the borders of each node need to be shared with its adjacent nodes for convolution with a kernel. The number of shared rows depends on the number of rows of the kernel: $r/2$ rows need to be shared if r is even and $r/2 - 1$ if r is odd (more common), where r is the number of rows of the kernel used. The input image distribution method is depicted in Figure II.6.

0	1	2	3	4	3	2	1	0
1	3	5	4	4	4	5	3	1
2	5	3	-2	-5	-2	3	5	2
3	4	-2	-17	-26	-17	-2	4	3
4	4	-5	-26	-38	-26	-5	4	4
3	4	-2	-17	-26	-17	-2	4	3
2	5	3	-2	-5	-2	3	5	2
1	3	5	4	4	4	5	3	1
0	1	2	3	4	3	2	1	0

Figure II.5 Laplacian of Gaussian Operator (9 x 9, $\sigma = 1.7$)

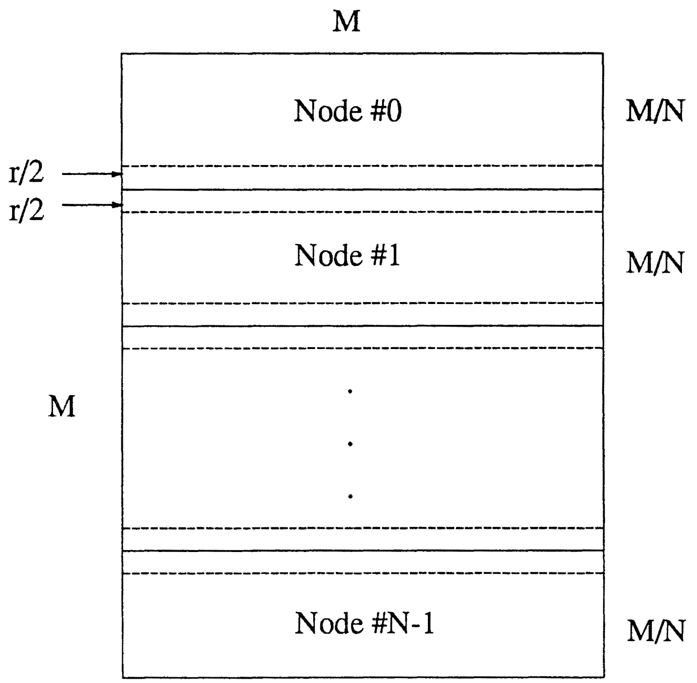


Figure II.6 Input Image Distribution Method

Parallel Edge Detection Algorithm

By the input image distribution method discussed earlier, we can load the image evenly into the nodes. This is a form of static load balancing, which is very important for efficient use of the hypercube. The parallel edge detection algorithm then runs on the data as follows:

Step 1: Read in the file header of the input image and get the size of the image.

Step 2: Calculate the sub-image size including overlapped rows.

Step 3: Read in the sub-image and send it to each node.

Step 4: On each node detect the edges by convolving the sub-image with the edge operator(s) in parallel.

Step 5: Receive the edges of the sub-images from the nodes.

Step 6: Write out the edges of the input image.

Note that only Step 4 executes completely in the hypercube. The other steps are performed in the host. However, Step 4 is by far the most complex so the potential benefit due to a parallel solution could be large.

Parallel Straight Line Extraction

In this section, we discuss several object representation schemes and develop a parallel object representation scheme using straight lines, which can be implemented on the hypercube.

Background

Much research has been performed on the development of compact algorithms for object representation as well as data reduction of the image of objects. One instance is a hierarchical representation of spatial occupancy, which is called octree or quadtree representation of objects [Jac80] [Mea81]. Occupied and unoccupied regions are represented in the form of a hierarchical tree, where the size of a region depends on the level in the tree. The size of a region in a high level in the tree is larger than the size of a region in a low level. This scheme is very compact, but it requires complicated algorithms for some computations [Bes85].

Another scheme is a line representation of objects. The vertex points of the objects are detected, then each pair of vertices is connected with a line. In practice, the edges or contours of the objects are often detected and approximated by straight lines. Then the objects can be represented simply by such approximated straight lines [Tro80]. This scheme is frequently used in object recognition problems because of its simplicity and fast display of the lines even though it has some limitations with complex objects [Hat83] [Low87] [McI88].

In recent years, Lowe [Low87] and Rosin [Ros89] proposed a straight line extraction method using polygonal approximation. In the following section we modify their methods and develop a parallel algorithm implemented on the hypercube.

Development of Parallel Straight Line Extraction Algorithm

Since the edge detection technique using the Laplacian of Gaussian operator gives us a set of connected edge contours, it is easy to list the coordinates of the edge points in each set into *segment* arrays: first, scan row by row from the topmost left point until a non-zero point (i.e., zero crossing point) is found. If a non-zero point is found, it is a beginning point for that segment. Second, keep searching for a non-zero neighbor point and moving to that point until no more non-zero points are detected. Note that once a non-zero point is detected, then it is assigned to zero after listing its coordinates into a segment array so that the point will never be listed again. Finally, a set of edge contours is stored into a segment array. Since the number of segments depends on the image, the segment array should be allocated dynamically to save memory space. This segmenting procedure is performed in parallel.

As the result of the segmenting procedure, the segment arrays in each node contain contour points of the objects only in its own sub-image. In other words, the segmenting is only a local procedure. In order to link the local segments globally, the segments that contain any points on the top or bottom row in the node must be sent to their adjacent nodes, then linked one after the other for all nodes. Such a segment is defined as a *pass-segment.*

One might think that the linking is a sequential procedure because it should be done for one after the other node. But we can increase the procedure parallelism and reduce the processing time significantly by studying the procedure and developing a parallel algorithm. Figure II.7 shows pseudocode of the algorithm for the node program when 8 nodes are used.

```
Procedure Global_Linking

Begin

   /*  Step 1  */
   IF (my_node_number mod (4)) = 0 THEN
      search pass-segments
      send the pass-segments to (my_node_number + 1)
   ELSE IF ((my_node_number mod (4)) - 3) = 0 THEN
      search pass-segments
      send the pass-segments to (my_node_number - 1)
   ELSE
      receive the pass-segments
      link the pass-segments locally
   END IF

   /*  Step 2  */
   IF ((my_node_number mod (4) - 1) = 0 THEN
      update the top and bottom row positions
      search pass-segments
      send the pass-segments to (my_node_number + 1)
   END IF
```

Figure II.7 Pseudocode for Parallel Segment Linking Algorithm

```
   IF ((my_node_number mod (4)) - 2) = 0 THEN
      receive the pass-segments
      link the pass-segments locally
   END IF

   /*  Step 3  */
   IF ((my_node_number mod (8) - 2) = 0 THEN
      update the top and bottom row positions
      search pass-segments
      send the pass-segments to (my_node_number + 4)
   END IF
   IF ((my_node_number mod (8)) - 6) = 0 THEN
      receive the pass-segments
      link the pass-segments locally
   END IF

End
```

(Figure II.7 *Continued*)

my_node_number is the logical position of a node. So, in this case, it is one number between 0 through 7, inclusive. *mod* stands for modulo operation. We can see that only three sequential steps are needed to link the pass-segments in 8 nodes, and it is easy to expand this algorithm for more than 8 nodes: simply add one more step as the number of nodes is increased by a factor of 2. The complexity of the parallel algorithm is thus $O(\log n)$ rather than $O(n)$. Figure II.8 depicts this algorithm.

In order to extract an approximated straight line from the global segments, we use the polygonal approximation method by Lowe [Low87] and Rosin [Ros89]. The coordinates of the first and last points of the segment are listed into a *straight-line* array as the beginning and ending points of the straight line. Then the approximated straight line is recursively split at the maximum deviation point if the ratio of the length of the line divided by the deviation is greater than a threshold value. A C program for this procedure is given in Figure II.9.

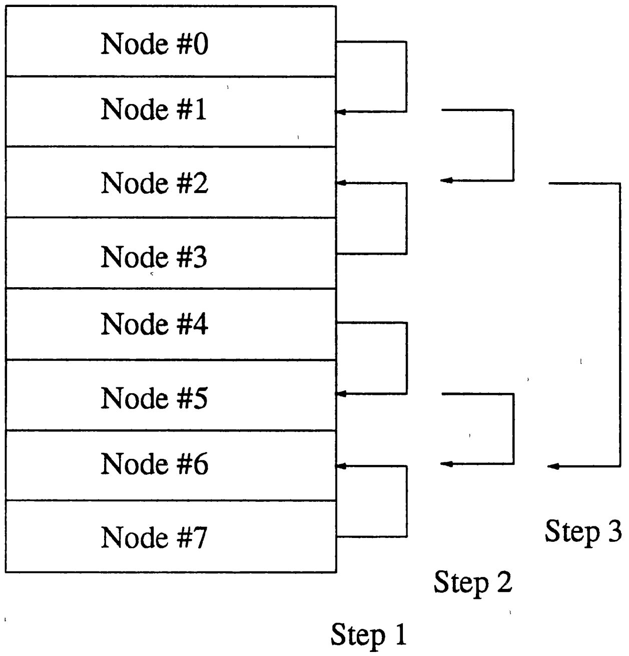


Figure II.8 Parallel Pass-Segment Linking Algorithm

```
Polygonal_Approximation(segment, begin, end, stline)
   struct  SEGMENT  seg_array; /* Array of global-segment       */
   int     begin, end;         /* Beginning and ending position */
   struct  STLINE  st_array;   /* Array of straight-line        */
{
   /* Initialize the variable for maximum deviation */
   max_deviation = 0;

   /* Calculate the length between beginning and ending points */
   /* of the segment                                           */
   length = calculate_distance(seg_array[begin], seg_array[end]);

   /* Loop for finding the position with maximum deviation */
   for (pos = begin; pos < end; pos++) {
      d = calculate_deviation(seg_array[begin], seg_array[end],
              seg_array[pos]);
      if (d > max_deviation) {
         max_deviation = d;
         max_pos = pos;
      }
   }
```

Figure II.9 A C Program for Polygonal Approximation

```
   /* If the ratio of the maximum deviation to the length  */
   /* is greater than threshold, split the segment at that */
   /* position and then call the function for two split    */
   /* segments recursively                                 */
   if ((max_deviation / length) > threshold) {
      Polygonal_Approximation(segment, begin, max_pos, stline);
      Polygonal_Approximation(segment, max_pos, end, stline);
   }

   /*  If the segment is not split, list the coordinates of   */
   /*  the beginning and ending positions into the straight-  */
   /*  line array                                             */
   else {
      list_stline(stline, begin, end, segment);
   }
}
```

(Figure II.9 *Continued*)

If (x_1, y_1) and (x_2, y_2) are the coordinates of the beginning and ending points of a segment respectively, and (x, y) is the coordinates of a point in the segment, then the deviation at the point, d, can be calculated by the following equation:

$$d = \frac{|ax + by + c|}{\sqrt{a^2 + b^2}} \tag{II.11}$$

where

$$a = y_1 - y_2 \qquad b = x_2 - x_1 \qquad c = -ax_1 - by_1$$

Figure II.10 shows an example of the straight line extraction using the polygonal approximation method for three levels of recursion.

Parallel Thinning

Thinning is one of the most important procedures in pattern recognition and image data reduction, but it is a very time consuming procedure. In most existing thinning

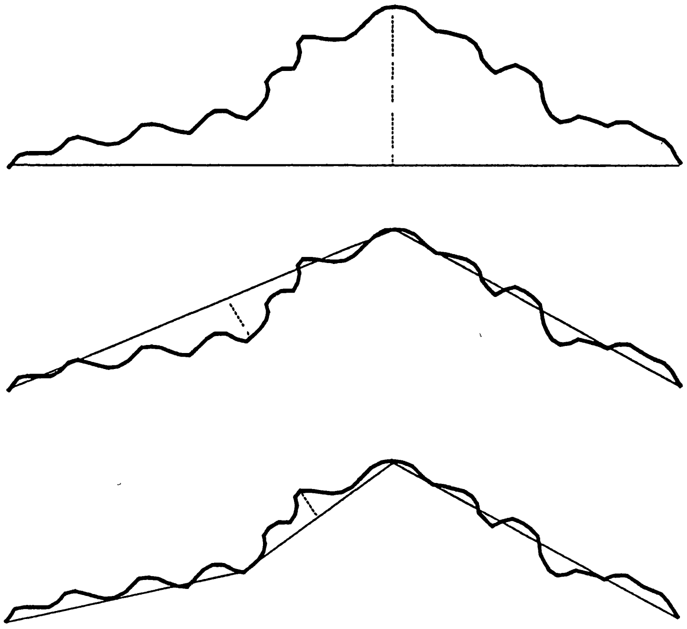


Figure II.10 An Example of the Polygonal Approximation Method

algorithms, several templates (usually 3 x 3) are scanned on the image for deleting boundary points but not the skeleton of an object. This procedure is repeated until no points are deleted. The complexity of the whole procedure is found to be $O(n^3)$ [Mar87].

In order to reduce processing time, many parallel algorithms have been proposed which can be easily implemented on currently available mesh computers. Examples are [Zha84], [Lu86], [Chi87], [Hol87], [Hal89], and [Guo89]. However, those parallel thinning algorithms are undesirable for implementation in distributed memory computers because the global shapes of the objects in an image might be affected when the image is divided and distributed to each node. To avoid this problem, data swapping between nodes, that is, communication, must be performed at every iteration [Kue85].

Parallel Boundary Detection and Object Extraction

In this section, we develop a new parallel thinning algorithm based on boundary following and shrinking. Each object boundary in the image is extracted and linked in parallel. The number of objects is divided based on the number of nodes being used and object size. Then each of the sub-objects is distributed to the nodes and thinned in parallel by following the boundaries and shrinking them in the direction perpendicular to the boundary and pointing toward the inside of the object. This algorithm reduces the complexity from $O(n^3)$ to $O(n^2)$.

Let us assume that we have a digitized binary image. Then an object region in the image can be simply represented by the set of boundary points, or connected edge points of the object. The connectedness of two boundary points in a binary image depends on the definition of the neighborhood: four-neighbor (N_4) or eight-neighbor (N_8) [Shi87]. (N_4) and (N_8) neighborhoods of a point at (i,j) consist of the following points:

$$N_4 = \{(i,j-1),\ (i,j+1),\ (i-1,j),\ (i+1,j)\} \tag{II.12}$$

$$N_8 = \{N_4,\ (i-1,j-1),\ (i-1,j+1),\ (i+1,j-1),\ (i+1,j+1)\} \tag{II.13}$$

In this research we have chosen eight-neighbor as the definition of the neighborhood. Two points are eight-connected if they are eight-neighbors of each other. Figure II.11 shows a 3 x 3 template according to the eight-neighbor definition.

Let us assume that the size of the input binary image is n x n, and no object points touch the periphery. Then the boundaries can be detected by the following procedure:

```
Procedure  Boundary_Detection;

begin
   for j = 0 to n-1 do begin
      for i = 0 to n-1 do begin
         if I(i,j) = 1 and I(p,q) = 1 for all (p,q) ∈ $N_8$, then
            B(i,j) = 0;
         else
            B(i,j) = 1;
      end;
   end;
end;
```

P_8 (i-1, j-1)	P_1 (i, j-1)	P_2 (i+1, j-1)
P_7 (i-1, j)	P_0 (i, j)	P_3 (i+1, j)
P_6 (i-1, j+1)	P_5 (i, j+1)	P_4 (i+1, j+1)

Figure II.11 Eight-Neighbor Definition

where *I* is an array representing the input image, and *B* is an array representing the output which contains only boundary points. This procedure is performed in each node for its sub-image in parallel.

As the result of the boundary detection, each node has boundary points only for its own sub-image. Now, we extract the objects locally by the boundary-following method which is described in [Shi87] (see chapter 4). The data structure for an object might be the following:

```
typedef struct {
   int    numofbound, /* Number of boundary points          */
          numoftbe,   /* Number of points on top border      */
          numofbbe;   /* Number of points on bottom border   */
   XYCRD *bound,      /* A pointer for boundary data         */
          *tbe,       /* A pointer for top border elements   */
          *bbe;       /* A pointer for bottom border elements */
} OBJECT;
```

where XYCRD is another data structure for a data position. Figure II.12 illustrates the 'bound', 'tbe', and 'bbe'. In order to link the local objects in the nodes globally, the local objects which have top border elements or bottom border elements are sent to their adjacent nodes and linked in parallel by the method described in the previous section.

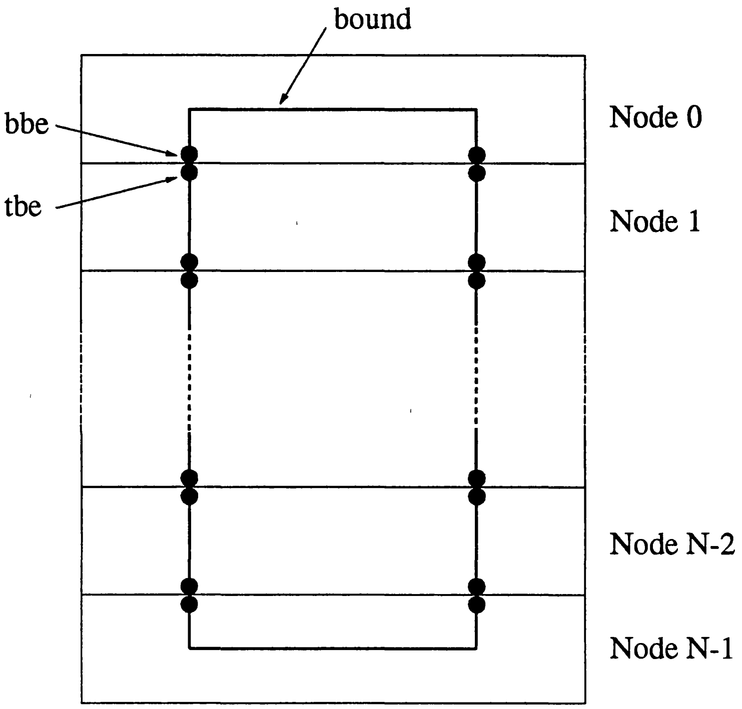


Figure II.12 Bound, Bbe, and Tbe

Parallel Object Thinning

For load balancing, the root node collects the information of the number of objects and their sizes from all nodes. The root node divides the number of the objects according to their sizes as well as the number of nodes, and then redistributes the sub-objects to all nodes. In the global object-linking step, the boundary data of the objects might be shuffled. To arrange the data, we project the objects on a *working plane* and perform the boundary-following step once again.

Now, we thin the objects in each node in parallel by following the object boundaries clockwise and by shrinking them in the direction perpendicular to the boundary and pointing toward the inside of the object. This procedure is repeated until the number of boundary points is not changed. Note that we find the direction for shrinking based on the eight boundary-following directions shown in Figure II.13 and defined by the following equation:

$$shrink\_dir = (follow\_dir + 2) \bmod 8 \tag{II.14}$$

where if the *shrink_dir* is zero, then *shrink_dir* is reassigned to eight.

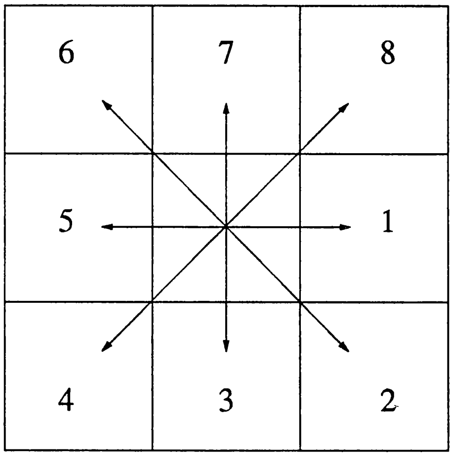


Figure II.13 Eight Boundary-Following Directions

Figure II.14 shows the results of shrinking a simple cross object after one iteration. The starting point is the top-left position of the object, and the arrows represent the boundary-following direction which is clockwise. 'x' and '.' denote the boundary of the original object and the boundary of the shrunken object respectively. Note that the circled points are inserted to make the shrunken object boundary connect. The connectivity of the shrunken-object boundary is essential for the next iterations. If a boundary point is an element of a parallel line or a single line (overlapped) then the point is just copied without shrinking. The parallel line and the single line are depicted in Figure II.15(a) and II.15(b), respectively.

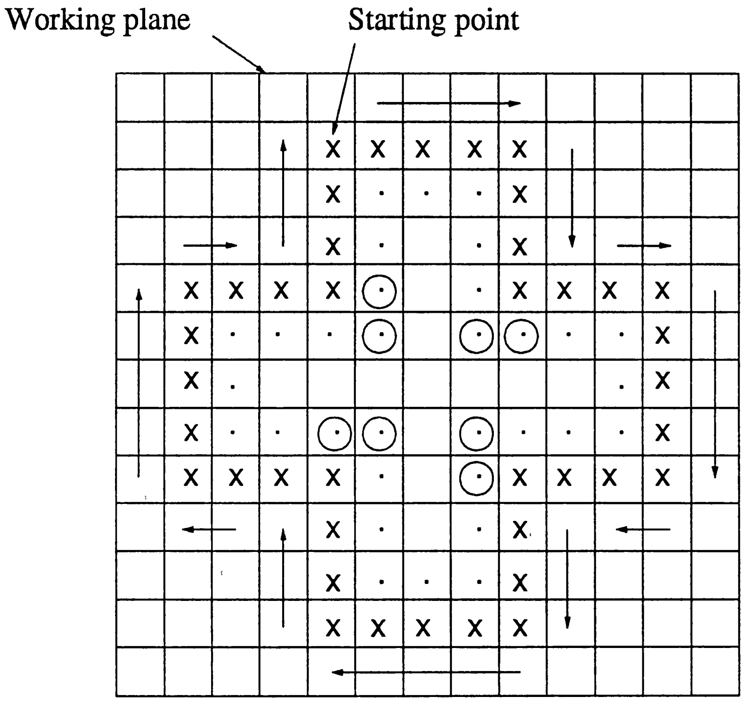


Figure II.14 Results of Shrinking a Simple Cross Object After One Iteration

The shrinking step produces the skeletons of the objects, which are at most two-pixels wide. To make single-pixel wide skeletons, we use the Zhang and Suen algorithm [Zha84] which preserves the connectivity of the skeletons. Note that we can remove two-pixel wide points by following the skeleton data points instead of scanning all over the working plane. Also note that we need only one iteration, that is, two subiterations, because one layer of the object is peeled off at each iteration.

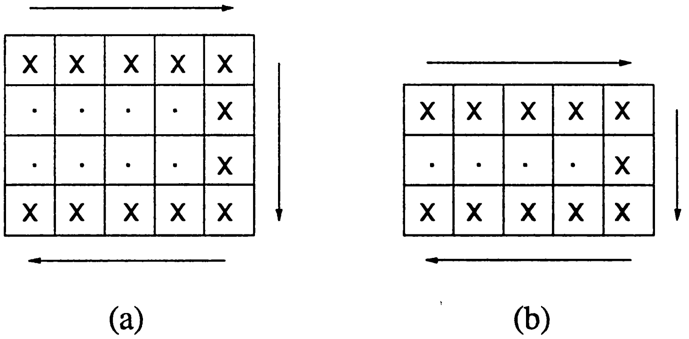


Figure II.15 Parallel Line and Single Line. (a) Parallel Line, (b) Single Line

Experimental Results and Speedup

The three parallel algorithms of edge detection, straight line extraction, and thinning developed in the previous sections were implemented on the iPSC/2 and tested for a variety of combinations of image size and number of nodes.

For the edge detection, the processing time on a VAX 11/750 and the hypercube when using Laplacian of Gaussian operator are compared in Figure II.16. Here, we can see that edge detection on a 512 x 512 image on the hypercube with 8 nodes is about 30 times faster than on the VAX 11/750.

Speedup (S_p) can be defined as

$$S_p = \frac{\text{Processing time on a single processor}}{\text{Processing time on P processors}} = \frac{T_a}{T_p} \qquad \text{(II.15)}$$

Figure II.17 shows the speedup when using Laplacian of Gaussian operator. We can see that the speedup increases almost linearly as the number of nodes increases, and we

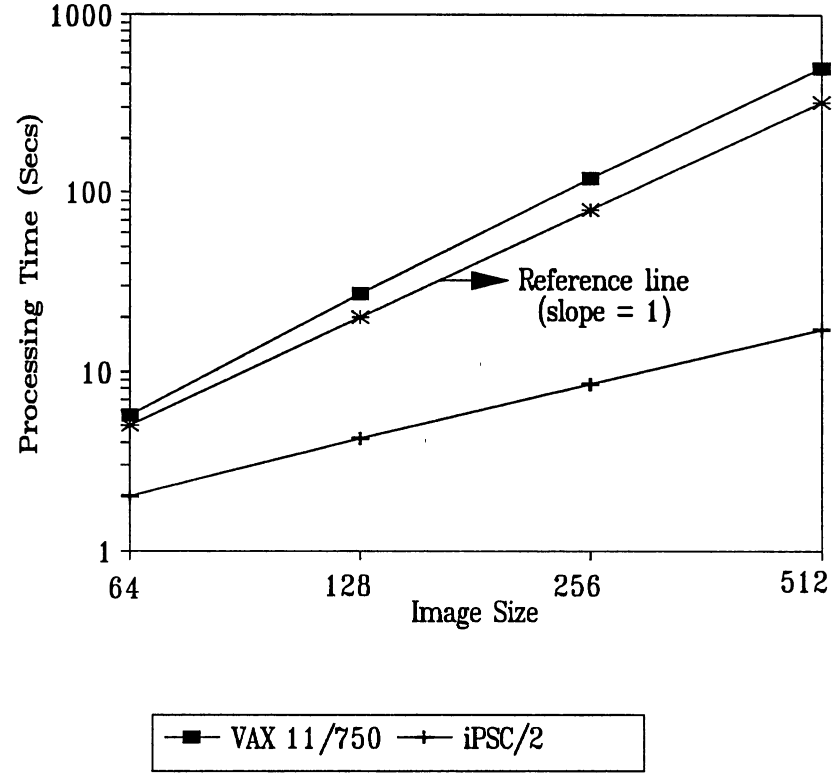


Figure II.16 Processing Times on VAX 11/750 and iPSC/2 (9 x 9 Laplacian of Gaussian Operator, 8 Nodes)

obtain more speedup at larger image sizes. This is an indication that, for reasonable (large) sized images, useful increases in performance can be realized for much larger order hypercubes than shown here.

To test straight line extraction, an image of simple industrial objects was used. The size of the input image was 512 x 512. The processing times according to different number of nodes are shown in Figure II.18. Note that the processing time includes times for the edge detection using Laplacian of Gaussian operator, the zero crossing points detection, and the straight line extraction. We can see that the processing times are decreased approximately linearly as the number of nodes is increased.

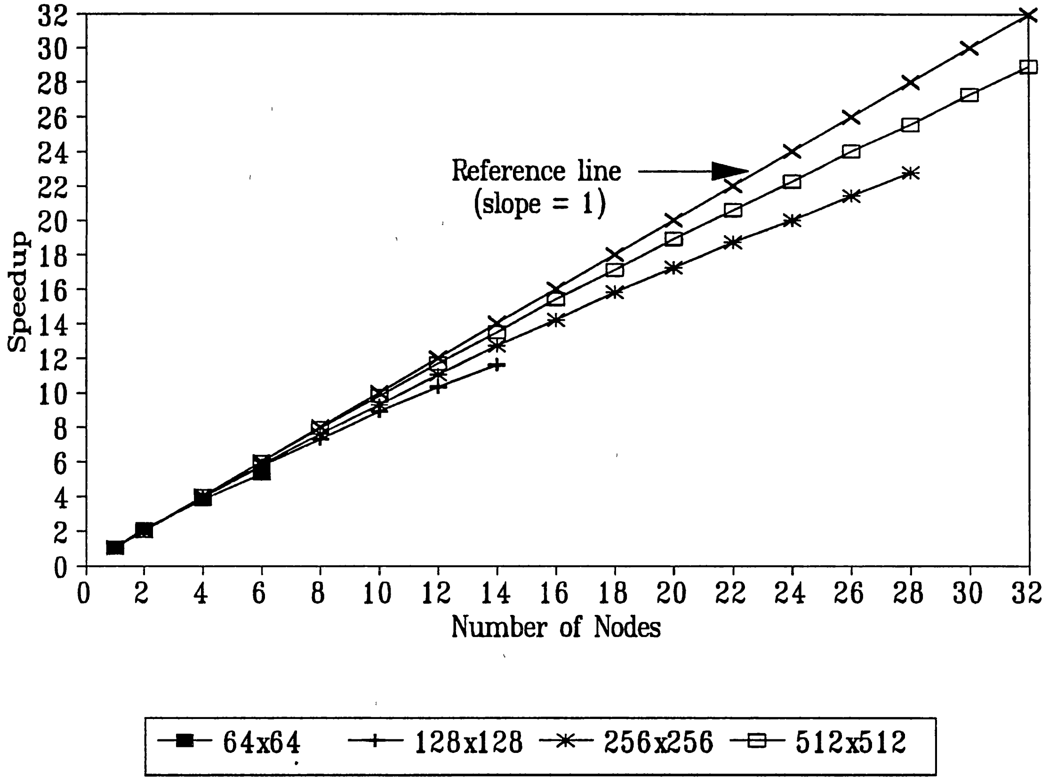


Figure II.17 Speedup When Using Laplacian of Gaussian Operator

It is obvious from our experimental results that the processing times for the preprocessing steps such as edge detection and straight line extraction are approximately inversely proportional to the number of nodes. This means our parallel algorithms have a high degree of parallelism without serious degradation.

Our parallel-thinning algorithm is also implemented on the Intel iPSC/2. Figure II.19(a) shows a test image which contains sixteen 'H's. The size of the image is 512 x 512. According to the input image distribution method, the input image is divided by the number of nodes and each sub-image is distributed to each node. Then the boundary detection for each sub-image is performed in parallel. Through the parallel linking

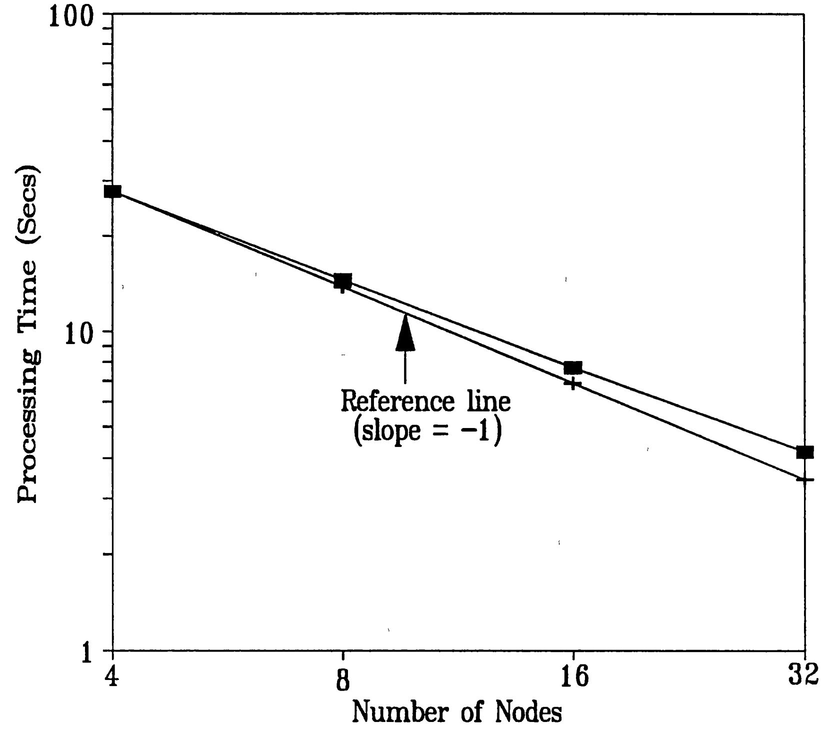


Figure II.18 Processing Times for Straight Line Extraction According to Various Number of Nodes

procedure, the sixteen objects are extracted and redistributed to the nodes as evenly as possible. Finally, the objects are thinned by boundary-following and shrinking. Figure II.19(b) is the final result. The skeletons are single-pixel wide.

For the comparison, we also implement Zhang and Suen's algorithm on the iPSC/2. As we discussed in the previous section, we need to swap data between nodes at every iteration. The processing times of Zhang and Suen's algorithm as well as our algorithm according to different numbers of nodes are shown in Table I. We can see that our algorithm is much faster than Zhang and Suen's. But our algorithm has some degradation when using 32 nodes because there are only 16 objects in the image and more communication time is needed for object extraction as more nodes are used. That

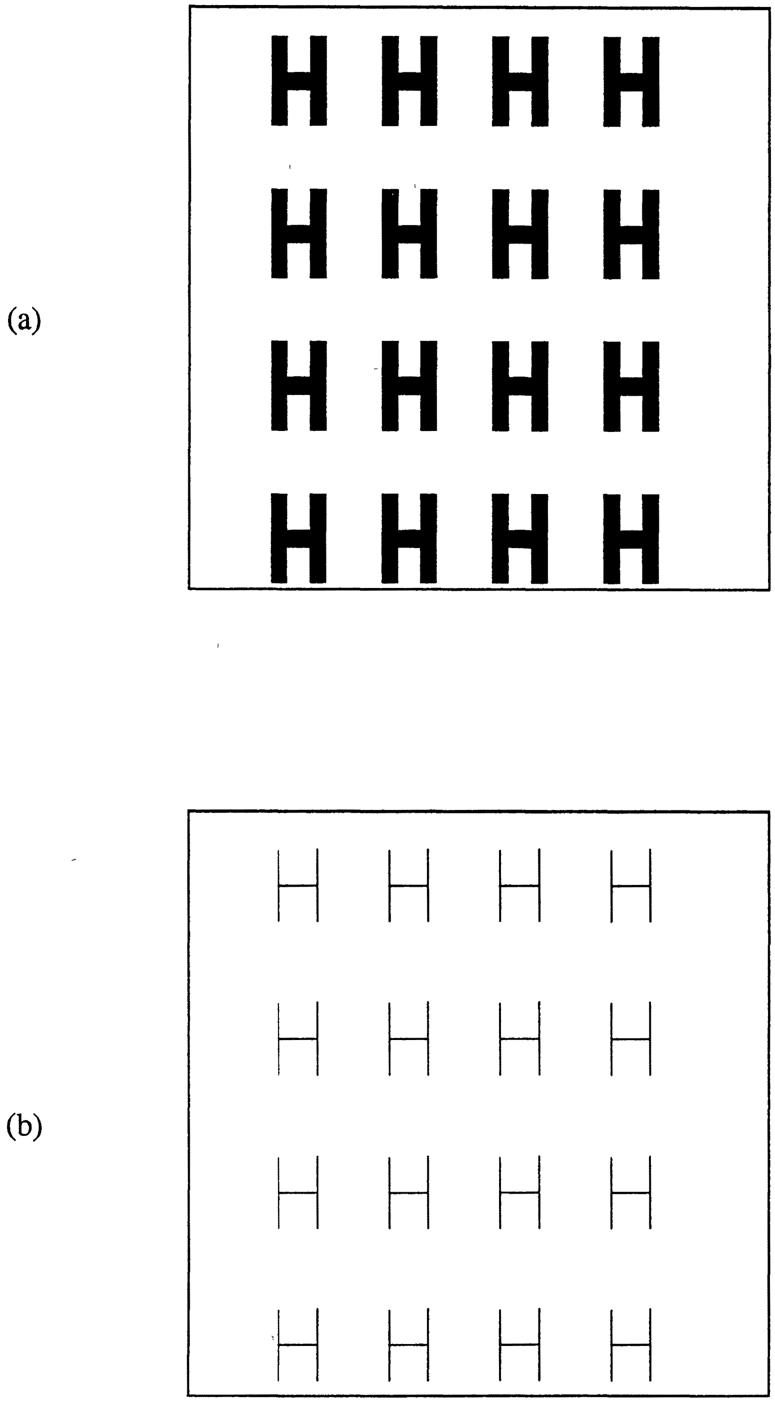


Figure II.19 Result of the Thinning. (a) An Original Input Image (512 x 512), (b) A thinned Image

is, our algorithm depends on the number of objects in the input image. The more objects in the scene, the more parallelism and the more efficient the algorithm becomes. Another problem is that, so far, our algorithm works only for solid objects. But this problem can be solved by some modification of the algorithm: If an object contains some holes inside, then we detect the boundaries of the holes as well as the boundary of the object. In the shrinking procedure, we shrink the boundary of the object inward and shrink the boundaries of the holes outward.

TABLE I

COMPARISONS OF THE PROCESSING TIMES FOR THINNING (IN SECONDS)

Algorithms	Number of Nodes			
	4	8	16	32
Zhang and Suen's	56.1	34.4	23.6	13.0
Ours	10.2	6.4	3.7	4.5

CHAPTER III

PARALLEL FEATURE EXTRACTION

Feature extraction is the most important step in a pattern recognition system. Accuracy of the system relies on how well the features are extracted. In this chapter, we will discuss how to extract the local features such as corners, arcs, parallel-lines, and corner-arcs from the straight lines, which are obtained through the preprocessing steps discussed in Chapter II. The local feature detection is also performed on the hypercube in parallel. At the end of this chapter, a parallel line and arc drawing algorithm is provided.

Corner Detection

Data Structure

As the result of the preprocessing such as edge detection and straight line extraction, each node has the data of the approximated straight lines, which consists of coordinates of starting and ending points, lengths and deviations of the lines. The data structure for a straight line is the following:

```
typedef struct {
   int  x1, y1,     /*  Coordinates of starting point  */
        x2, y2;     /*  Coordinates of ending point    */
   float   len,     /*  Length of the line             */
           dev;     /*  Deviation of the line          */
} STLINE;
```

Figure III.1 shows the parameters of a straight line.

A corner can be defined as a point where two straight lines merge. The data structure for a corner is the following:

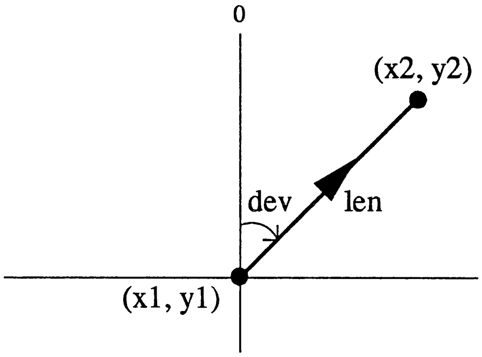


Figure III.1 Parameters of a Straight Line

```
typedef struct {
   STLINE  st1,     /*  Straight line 1       */
           st2;     /*  Straight line 2       */
   float   ang;     /*  Angle of the corner   */
} CORNER;
```

Figure III.2 depicts a corner and its parameters. Note that the length of *st1* is greater than or equal to the length of *st2*. This condition is desired for matching discussed in the next chapter.

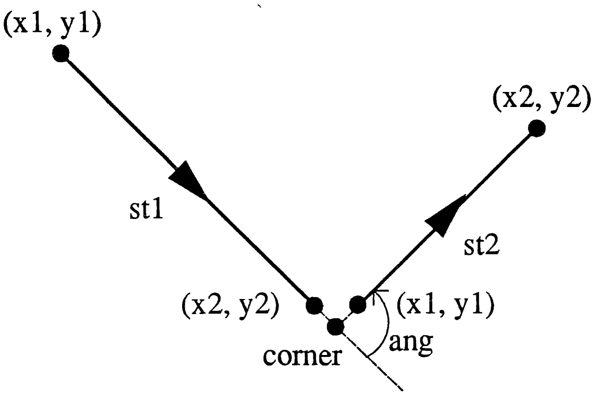


Figure III.2 A Corner and Its Parameters

Parallel Corner Detection

In [Koc85] and [Koc87], Koch and Kashyap assumed that the contour of the object was closed. In practice, it is very difficult to get a closed contour due to variation in illumination, reflections, and shadows. In our research, we assume that the contour is not closed. Hence, we have to compare every straight line to all other straight lines in order to detect corners. The complexity of this step is $O(n^2)$, where n is the number of straight lines. In order to reduce the complexity of the comparison, we split the data and then implement the corner detection in parallel on the hypercube.

First, the root node (node 0) collects the local straight lines from each node, and merges them into an array, *global_straight_line.* Second, the root node redistributes the global_straight_line array to all nodes except the root node. Third, each node computes the beginning and ending positions of the global_straight_line by the following equations for its own straight lines to be concerned.

$$begin = \frac{\text{my\_node\_number}*n}{N} \tag{III.1}$$

$$end = \frac{(\text{my\_node\_number} + 1)*n}{N} - 1 \tag{III.2}$$

where N is the number of nodes allocated. In this way, parallelism can be introduced to reduce the complexity of the corner detection to $O(n^2/N)$. Figure III.3 illustrates the global straight line division method.

Now each node detects corners in parallel by comparing every pair of straight lines from the global_straight_line array and its own straight lines. If the proximity of two straight lines is less than a threshold value, they are saved into an array *corner.* Koch and Kashyap [Koc85] [Koc87] detected only corners with high curvature, that is, large corner

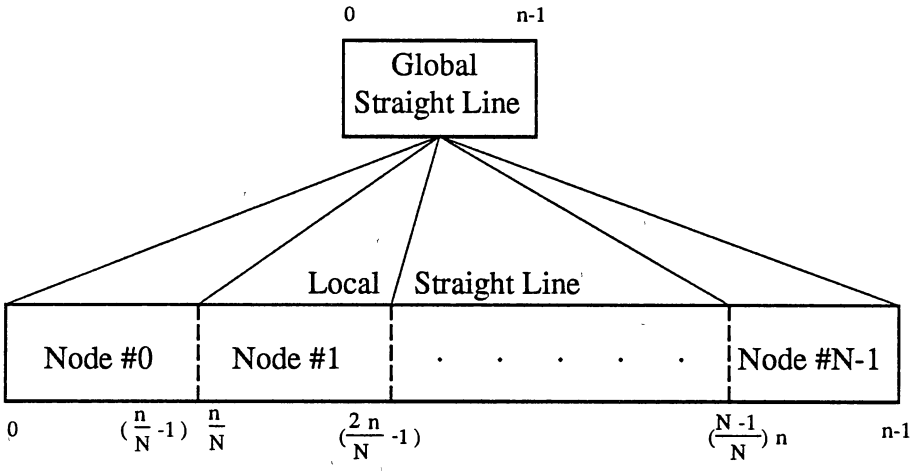


Figure III.3 Global Straight Line Division Method

angle in our case. Although high curvature gives more significant features than low curvature, corners with low curvature but long straight lines can give us valuable information of the objects. In this research we detect and use all of the corners.

Computation of Corner Angle

According to the data structure of straight line, each straight line has information of its deviation. The deviation is computed by the following method using a library function, *atan2()*:

```
dx = x2 - x1
dy = y2 - y1

IF (dx = 0 and dy = 0) THEN
   dev = 0
ELSE
   dev = atan2(dy, dx)
      IF (dy < 0) THEN
         dev = dev + $2\pi$
      END IF
END IF
```

Figure III.4 illustrates the deviations of the straight lines according to the different directions. Note that the deviations are assigned clockwise.

The corner angle between two straight lines can be obtained by the following method:

```
diff = st2.dev - st1.dev

IF (diff > $\pi$) THEN
   ang = diff - $2\pi$
ELSE IF (diff < $-\pi$) THEN
   ang = diff + $2\pi$
ELSE
   ang = diff
END IF
```

Note that the corner angles range from $-\pi$ to π.

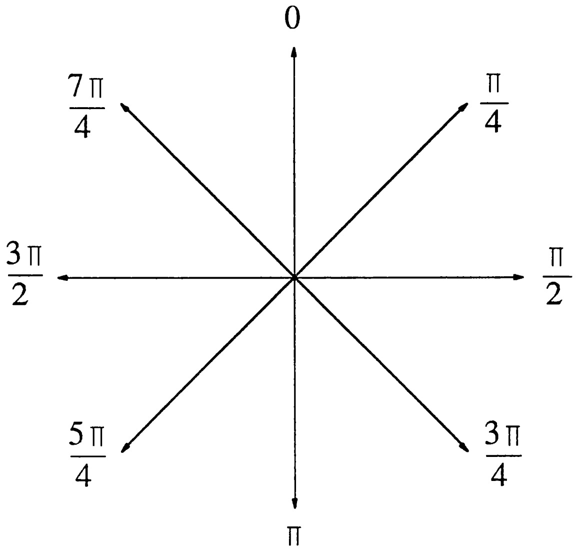


Figure III.4 Deviations of the Straight Lines According to the Different Directions

Arc Detection

The arc is another important local feature. However arc detection is much more difficult than corner detection since the function of an arc is non-linear and it requires more parameters than a corner. Recently, Rosin [Ros89] proposed an arc detection method, which is more accurate but complicated. Earlier, Perkins [Per78] developed a circular arc fitting algorithm based on a least squares error criterion.

An arc can be represented by eight parameters: radius R, center at (X_c, Y_c), starting at (X_s, Y_s), ending at (X_e, Y_e), and length L. Figure III.5 shows the parameters of an arc. The data structure of the arc is the following:

```
typedef struct {
   int      xs, ys,      /*  Starting point      */
            Xe, ye,      /*  Ending point        */
            xc, yc;      /*  Center point        */
   float    r,           /*  Radius of the arc   */
            len;         /*  Length of the arc   */
} ARC;
```

In order to fit an arc to polygonal approximated lines, we minimize the errors between the arc and the vertices of the lines. Figure III.6 illustrates an error between the arc and one vertex.

An arc with radius R and center at (X_c, Y_c) can be represented by the following equation:

$$(X - X_c)^2 + (Y - Y_c)^2 = R^2 \tag{III.3}$$

Since the coordinates of the vertices of the polygonal approximated lines are already known, the error function to be minimized will be the following:

$$J = \sum_{i=1}^{n} [(X_i - X_c)^2 + (Y_i - Y_c)^2 - R^2]^2 \tag{III.4}$$

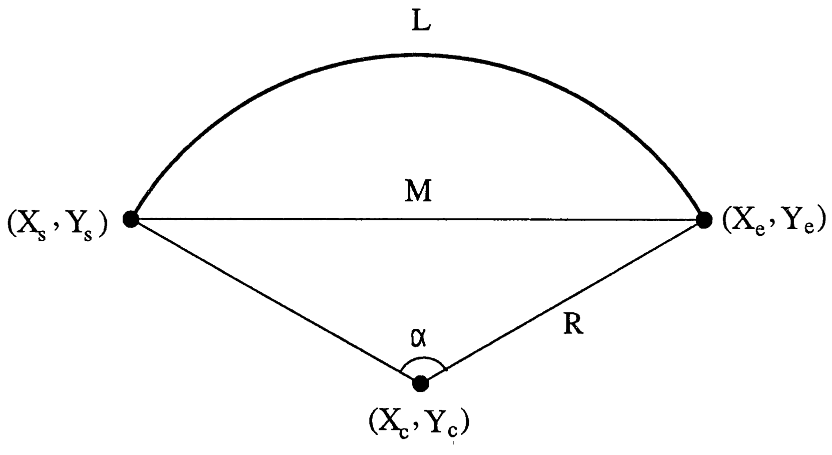


Figure III.5 Parameters of an Arc

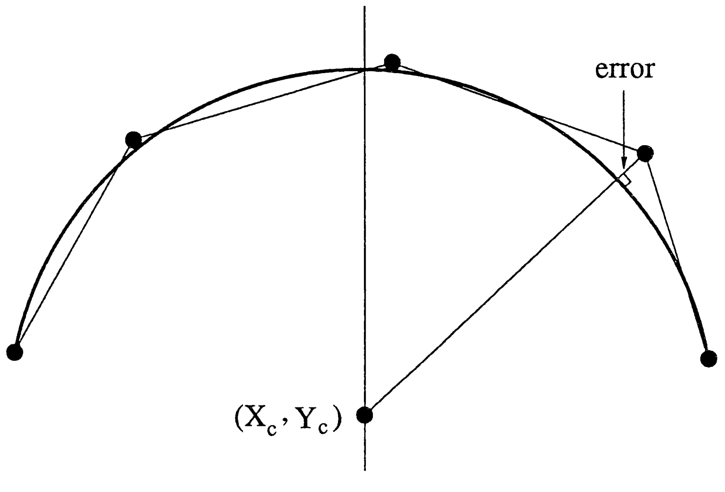


Figure III.6 Error Between the Arc and One Vertex of the Polygonal Approximated Line

where n is the number of vertices. From the conditions,

$$\frac{\partial J}{\partial X_c} = 0, \qquad \frac{\partial J}{\partial Y_c} = 0, \qquad \text{and } \frac{\partial J}{\partial R} = 0$$

we can get

$$\sum_{i=1}^{n}(X_i - X_c)^3 + \sum_{i=1}^{n}(Y_i - Y_c)^2(X_i - X_c) - R^2\sum_{i=1}^{n}(X_i - X_c) = 0 \tag{III.5}$$

$$\sum_{i=1}^{n}(X_i - X_c)^2(Y_i - Y_c) + \sum_{i=1}^{n}(Y_i - Y_c)^3 - R^2\sum_{i=1}^{n}(Y_i - Y_c) = 0 \tag{III.6}$$

$$\sum_{i=1}^{n}(X_i - X_c)^2 + \sum_{i=1}^{n}(Y_i - Y_c)^2 - nR^2 = 0 \tag{III.7}$$

See Appendix A for the derivations of the Equations (III.5) through (III.7).

The above three equations can be solved by Newton's method for the three unknown parameters, X_c, Y_c, and R. Let's represent the three unknown parameters as a vector,

$$\mathbf{x} = \begin{bmatrix} X_c \\ Y_c \\ R \end{bmatrix}$$

Then, the Equations (III.5), (III.6), and (III.7) can be represented by a system

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ f_3(\mathbf{x}) \end{bmatrix} = \mathbf{0} \tag{III.8}$$

Newton's method for the system is given by the following:

For $m = 0, 1, 2, \ldots,$ until satisfied, do:

$$\mathbf{x}^{(m+1)} := \mathbf{x}^{(m)} - \mathbf{f}'(\mathbf{x}^{(m)})^{-1}\mathbf{f}(\mathbf{x}^{(m)}) \tag{III.9}$$

where the matrix $\mathbf{f}'(\mathbf{x})$ called the Jacobian matrix is given by

$$\mathbf{f}'(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial X_c} f_1 & \frac{\partial}{\partial Y_c} f_1 & \frac{\partial}{\partial R} f_1 \\ \frac{\partial}{\partial X_c} f_2 & \frac{\partial}{\partial Y_c} f_2 & \frac{\partial}{\partial R} f_2 \\ \frac{\partial}{\partial X_c} f_3 & \frac{\partial}{\partial Y_c} f_3 & \frac{\partial}{\partial R} f_3 \end{bmatrix} \tag{III.10}$$

In order to get the inverse of the matrix $\mathbf{f}'(\mathbf{x})$, we use the following equation instead of a common numerical method such as factorization or *LU* decomposition method:

$$\mathbf{f}'(\mathbf{x})^{-1} = \frac{adj\ \mathbf{f}'(\mathbf{x})}{|\mathbf{f}'(\mathbf{x})|} \tag{III.11}$$

where $|\mathbf{f}'(\mathbf{x})|$ is the determinant of the matrix $\mathbf{f}'(\mathbf{x})$ and $adj\ \mathbf{f}'(\mathbf{x})$ is the adjoint matrix of $\mathbf{f}'(\mathbf{x})$ [Ger65] [Ful67].

An adjoint matix is defined as the transpose of the cofactor matrix [Ger65]. To illustrate the construction of a cofactor matix, consider the third-order matrix **A**:

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \tag{III.12}$$

The cofactor matrix obtained from **A** is denoted $\mathbf{A}^c$ and is as follows:

$$\mathbf{A}^c = \begin{bmatrix} A_{11}^c & A_{12}^c & A_{13}^c \\ A_{21}^c & A_{22}^c & A_{23}^c \\ A_{31}^c & A_{32}^c & A_{33}^c \end{bmatrix} \tag{III.13}$$

Each of the cofactors in the preceding matrix is obtained from the determinant of **A**, which is

$$|\mathbf{A}| = \begin{vmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{vmatrix} \qquad \text{(III.14)}$$

Thus, the cofactors are

$$A_{11}^{c} = \begin{vmatrix} A_{22} & A_{23} \\ A_{32} & A_{33} \end{vmatrix} \qquad A_{12}^{c} = -\begin{vmatrix} A_{21} & A_{23} \\ A_{31} & A_{33} \end{vmatrix}$$

and so forth for the remaining cofactors.

In Newton's method, the first guess $\mathbf{x}^{(0)}$ is a very important factor not only for converging to a root but also for reducing the number of iterations. In this research, we develop a method for estimating a "good" initial center point (X_c^0, Y_c^0) and radius R^0 of the candidate for an arc.

Let l_1 and l_2 be the first and last straight lines of the candidate. Then we find the equations for l_1' and l_2', which are perpendicular to l_1 and l_2, and pass through the center points of l_1 and l_2, respectively. Now, the initial center point (X_c^0, Y_c^0) is obtained by solving the two equations for l_1' and l_2'. The initial radius R^0 is simply the distance from the (X_c^0, Y_c^0) to the center point of l_1 or l_2. This method is demonstrated in Figure III.7. From the experimental results, the initial center point and radius of the candidate for an arc are very close to the actual values in most case. So, we can reduce the number of iterations significantly.

In order to find the length of the arc L, let α be the central angle of the arc and M be the length of the side opposite the angle (see Figure III.5). From the *Law of Cosines*,

$$M^2 = 2R^2 - 2R^2\cos\alpha \qquad \text{(III.15)}$$

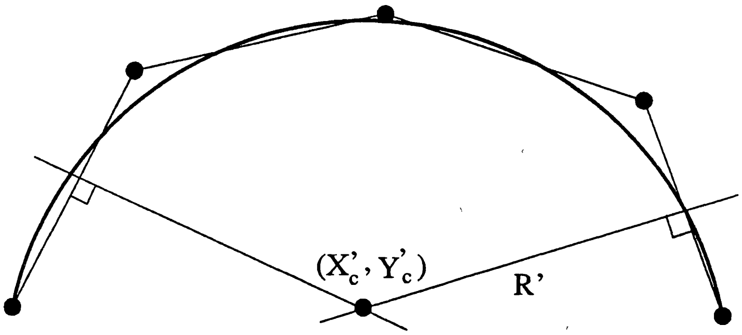


Figure III.7 A Method for Estimating the Initial Center Point and Radius of the Candidate of an Arc

or

$$\alpha = \cos^{-1}\left[\frac{2R^2 - M^2}{2R^2}\right] \quad \text{(radians)} \tag{III.16}$$

where

$$M = \sqrt{(X_e - X_s)^2 + (Y_e - Y_s)^2} \tag{III.17}$$

Thus, the length of the arc L is simply given by

$$L = \alpha \cdot R \tag{III.18}$$

Since all pairs of straight lines have already been compared to each other for testing their proximity in the corner detection, we use the corner data for searching a set of straight lines, or candidate for an arc: we recursively link the corners which are adjacent to each other. Before the linking, we drop the corners with high curvature (that is, sharp corners) or high ratio of the length of the longer line to the length of the shorter line for reducing the execution time. Those lines have little possibility to be an arc.

For parallel implementation of the arc detection on the hypercube, the root node collects the local candidates from the nodes and divides the local candidates as evenly as possible according to the number of the nodes being allocated. Then the root node redistributes the sub-candidates to the nodes, and each node estimates the parameters such as center point, radius, and length for each candidate in parallel. Figure III.8 shows a block diagram of parallel arc detection. Note that the *N* in the figure is the number of nodes allocated.

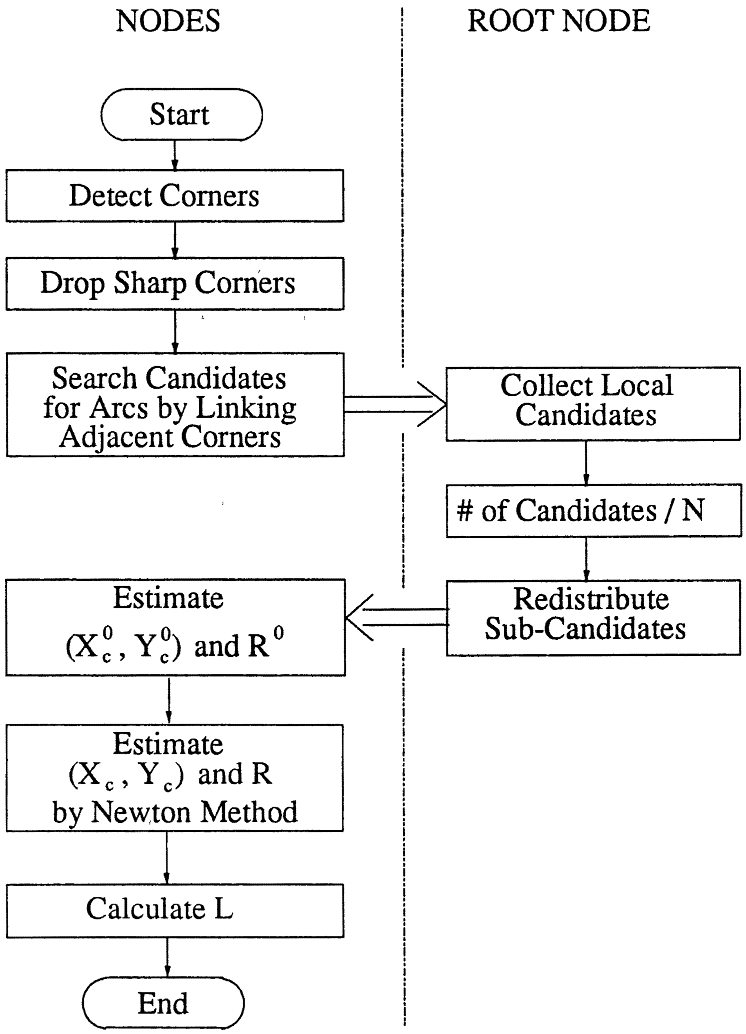

NODES
ROOT NODE
Start
Detect Corners
Drop Sharp Corners
Search Candidates for Arcs by Linking Adjacent Corners
Collect Local Candidates
# of Candidates / N
Redistribute Sub-Candidates
Estimate (X_c^0, Y_c^0) and R^0
Estimate (X_c, Y_c) and R by Newton Method
Calculate L
End

Figure III.8 A Block Diagram of Parallel Arc Detection

Parallel-Line Detection

Industrial objects or parts usually contain some parallel lines in their boundary. Thus parallel lines are potentially significant features for representing the objects. Lowe [Low87] used the perceptual grouping method based on proximity, parallelism, and collinearity of the straight lines extracted from an image to reduce searching space for object matching. In his work, a measure of the parallelism between two lines is defined as

$$E = \frac{4D\theta s l_1}{\pi l_2^2} \tag{III.19}$$

where D is a constant (set to 1), θ is the angular difference in radians between two lines, s is the average separation between two lines, l_1 is the length of the longer line, and l_2 is the length of the shorter line. The significance of the parallelism is inversely proportional to E.

In this research, we simplify Equation (III.19) and define a significance of parallelism between two straight lines, S_{par}, as the following:

$$S_{\text{par}} = C \cdot \left(\frac{1}{\theta \cdot d}\right)\left(\frac{l_2}{l_1}\right) \tag{III.20}$$

where C is a constant, θ is the angular difference in radians between two lines, d is the perpendicular distance from the longer line to the midpoint of the shorter line, l_1 is the length of the longer line, and l_2 is the length of the shorter line. Figure III.9 shows these parameters. Note that the significance of the parallelism is proportional to S_{par}.

Now we compute S_{par} for every pair of straight lines. If S_{par} is greater than a threshold value, then we list the pair of lines into an array *parallel-line.* The data structure of the parallel-line is the following:

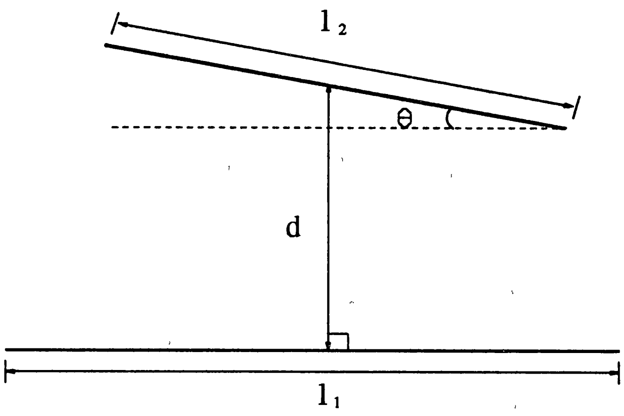


Figure III.9 Parameters for the Significance of Parallelism Between Two Lines

```
typedef struct {
   STLINE  st1,        /* Longer line            */
           st2;        /* Shorter line           */
   float   d;          /* Perpendicular distance */
                       /* between two lines      */
} PARLINE;
```

However, one more parameter might be added in the significance of parallelism defined in Equation (III.20). If two lines are parallel but too far away from one another, those lines have less significance in parallelism than closely aligned parallel lines. In order to eliminate the parallel lines not aligned, we compute a degree of alignment (*doa*). The *doa* is defined by perpendicular distance from the center point of the shorter line (*st2*) to the line which is perpendicular to the longer line (*st1*) and passes through the center point of the line. In this research, we set the *doa* as

$$doa < \frac{\text{length of } st2}{4}$$

- If the *doa* of a parallel line is greater than or equal to the threshold value, we drop the parallel line from the list. Figure III.10 shows degree of alignment of two lines. Note that the parallel-line detection is also performed in parallel on the hypercube.

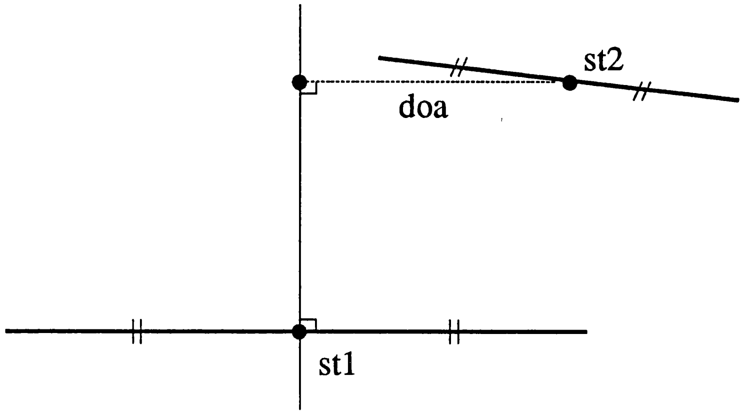


Figure III.10 A Degree of Alignment (*doa*) of Two Lines

Corner-Arc Detection

If an arc and a corner exist in an object, the information of the arc and the corner can be combined and used as a feature for the object. For example, when the angle of the corner, radius of the arc, and distance from the vertex of the corner to the center point of the arc are combined as a feature, it can form a significant new feature for the object. Let's call the feature "*corner-arc*".

The feature is invariant to rotation and translation of the object, so we can test the compatibility between the corner-arc of a model object and the corner-arc of an input object. Also, a coordinate transform can be obtained from the corner-arc feature because the coordinates of the vertex of the corner and center point of the arc are linearly transformed as the object is translated and rotated. The compatibility test and the coordinate transform are described in Chapter IV.

To detect corner-arcs, corner and arc detection should be performed in advance because corner-arcs are combined features of the corners and arcs. If any arcs are detected, we search the corners with sharp angle. Of course, all of the corners can be used, but only sharp corners are used because the combination of the arcs and all of the corners might result in a large number of the features. Now the information of the corner and the arc is listed into an array, *corarc*. The following is the data structure of the *corarc* array.

```
typedef struct {
   int    x,      /* X coordinate of the corner point        */
          y,      /* Y coordinate of the corner point        */
          xc,     /* X coordinate of center point of the arc */
          yc;     /* Y coordinate of center point of the arc */
   float  ang,    /* Angle of the corner                     */
          r,      /* Radius of the arc                       */
          d;      /* Distance from the center point of the   */
                  /* arc to the corner point                 */
} CORARC;
```

Figure III.11 shows the parameters of a corner-arc.

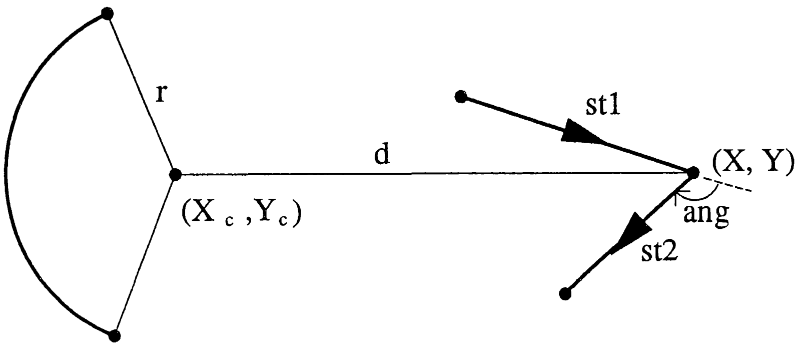


Figure III.11 Parameters of a Corner-Arc

Parallel Line and Arc Drawing

The extended local features basically consist of lines and arcs. The basic task of a line or arc drawing in computer graphics is to compute the coordinates of the pixels which lie near the line or arc on a two-dimensional raster grid. In this section, we study some existing line and arc drawing algorithms, and then we develop a parallel line and arc drawing algorithm which can be implemented on the iPSC/2.

Parallel Line Drawing

There exist several line drawing algorithms. Examples are *incremental algorithm* and *Bresenham's algorithm* [Fol82]. In the incremental algorithm, the slope of the line, m, is calculated by $m = \Delta y/\Delta x$, where $\Delta x = (x2 - x1)$ and $\Delta y = (y2 - y1)$ if the coordinates of the starting and ending points of the line are $(x1, y1)$ and $(x2, y2)$, respectively. Here if we assume that $\Delta x = 1$, then the m reduces to Δy, that is, a unit change in x changes y by m. Thus, for all points (x_i, y_i) on the line, we know that if $x_{i+1} = x_i + 1$, then $y_{i+1} = y_i + m$. That is, the next values of x and y are defined in terms of their previous values. When m is greater than 1, a step in x will create a step in y which is greater than 1. Thus we must reverse the roles of x and y by assigning a unit step to y and incrementing x by $\Delta x = \Delta y/m = 1/m$. The incremental algorithm is quite simple to implement, but it has some disadvantages: rounding y to an integer takes time, and the variables y and m must be real or fractional binary rather than integer, because the slope is a fraction.

Bresenham's algorithm [Bre65] is attractive because it may be programmed without multiplication or division instructions, and thus it is efficient with respect to speed of execution and memory utilization. For simplicity in describing the algorithm, we assume that the slope of the line is between 0 and 1. The algorithm uses a decision variable d_i which at each step is proportional to the difference between r and q shown in Figure III.12. The figure depicts the ith step, at which the pixel P_{i-1} has been determined to be closest to the actual line being drawn, and we now want to decide whether the next pixel to be set should be R_i or Q_i. If $r < q$, then R_i is closer to the desired line and should be set; else Q_i is closer and should be set. The black circles in Figure III.12 are pixels selected by Bresenham's algorithm.

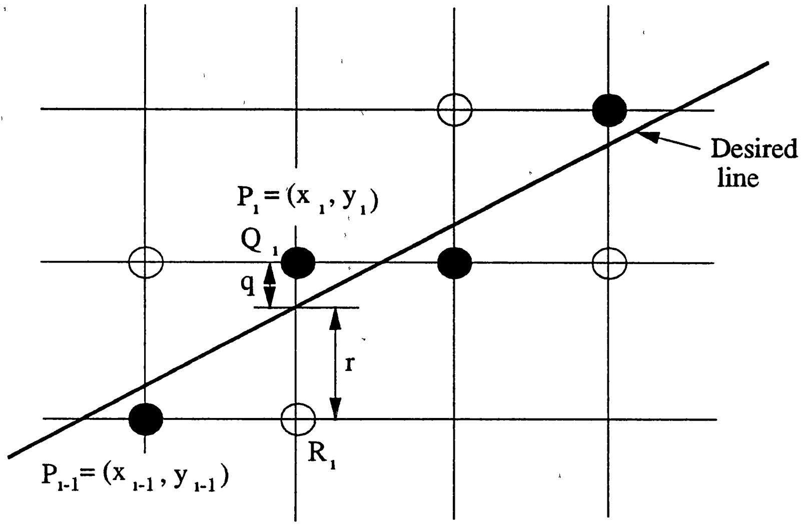


Figure III.12 Notation for Bresenham's Algorithm

The line being drawn is from $(x1, y1)$ and $(x2, y2)$. Assuming that the first point is nearer the origin, we translate both points by $(-x1, -y1)$, so it becomes the line from (0, 0) to (dx, dy), where $dx = (x2 - x1)$ and $dy = (y2 - y1)$. Now the decision variable d_i is defined as

$$d_i = dx(r - q) \tag{III.21}$$

Since dx is positive when the slope of the line is between 0 and 1, we choose R_i when $d_i < 0$, that is, $(r - q) < 0$. From the examination of Figure III.12 we can rewrite Equation (III.21) as

$$d_i = 2x_{i-1}dy - 2y_{i-1}dx + 2dy - dx \tag{III.22}$$

If $d_i \geq 0$, then Q_i is selected, so $y_i = y_{i-1} + 1$ and

$$d_{i+1} = d_i + 2(dy - dx) \tag{III.23}$$

If $d_i < 0$, then R_i is selected, so $y_i = y_{i-1}$ and

$$d_{i+1} = d_i + 2dy \tag{III.24}$$

Hence we have an iterative way to calculate d_{i+1} from the previous d_i and to make the selection between R_i and Q_i. The initial starting value d_1 is found by evaluating Equation (III.22) for $i = 1$, knowing that $(x_0, y_0) = (0,0)$. Then

$$d_1 = 2dy - dx \tag{III.25}$$

The arithmetic needed to evaluate Equations (III.23), (III.24), and (III.25) is minimal, because the equations consists of only addition, subtraction, and left shift to multiply by 2. This is important, because time-consuming multiplication is avoided. Moreover, the actual inner loop is quite simple. To generalize the algorithm, slopes of the lines are determined by one of the eight octants (see Table II). Note that the origin point is at the top-left corner of the image plane (refer to Figure III.14 in the next section). According to the different octants, only slight changes are needed in the actual inner loop. A C program for the Bresenham's algorithm is shown in Figure III.13. Note that this version works only for lines with slope between 0 and 1, that is, eighth octant.

Bresenham's algorithm is implemented on the hypercube in parallel. All of the lines to be drawn are sent to all nodes. In each node, the lines are tested to determine if they are inside the local image plane (LIP) or if they pass through the top and/or bottom borders of the LIP (see Figure III.14). If a line is inside of the LIP, then we draw the line

TABLE II

DETERMINATION OF THE OCTANTS

Δx	Δy	$\lvert \Delta x \rvert - \lvert \Delta y \rvert$	Octant
≥ 0	≥ 0	≥ 0	1
≥ 0	≥ 0	< 0	2
< 0	≥ 0	< 0	3
< 0	≥ 0	≥ 0	4
< 0	< 0	≥ 0	5
< 0	< 0	< 0	6
≥ 0	< 0	< 0	7
≥ 0	< 0	≥ 0	8

into the LIP. Otherwise, we compute the coordinates of the intersection point(s) of the line and the top and/or bottom borders of the LIP. The coordinates can be obtained by the following equations:

$$x_1' = x_1 + dev*(\text{top} - y_1) \tag{III.26a}$$

$$x_2' = x_1 + dev*(\text{bottom} - y_1) \tag{III.26b}$$

$$y_1' = \text{top} \tag{III.26c}$$

$$y_2' = \text{bottom} \tag{III.26d}$$

where

```
Line_Draw(D, x1, y1, x2, y2)
   Image D;              /* Image plane for the line drawing   */
   int   x1, y1, x2, y2; /* Coordinates of starting and ending */
                         /* points of the line                 */
{
   dx = x2 - x1;
   dy = y2 - y1;
   absdx = abs(dx);      /* Absolute value of dx               */
   absdy = abs(dy);      /* Absolute value of dy               */
   diff = absdx - absdy; /* Difference between absdx and absdy */

   /* Write the starting point */
   write_pixel(D, x1, y1);

   /* Determine the octant of the line */
   octant = determine_octant(dx, dy, diff);

   /* Switch according to the octant of the line */
   switch(octant) {
      case 8:
         d = 2*absdy - absdx;        /* Initial value for d */
         incr1 = 2*absdy;            /* Increment if d < 0  */
         incr2 = 2*(absdy - absdx); /* Increment if d >= 0 */

         /* Main loop */
         while(x1 < x2) {
            x1++;
            if (d < 0) {
               d = d + incr1;
            }
            else {
               y1--;
               d = d + incr2;
            }
            /* Write the selected point near the line */
            write_pixel(D, x1, y1);
         }
      break;
   }
}
```

Figure III.13 A C Program for Bresenham's Algorithm

$$dev = \frac{x_2 - x_1}{y_2 - y_1}$$

Now the lines inside of the LIP or newly generated line segments by the above equations are drawn in parallel on each node.

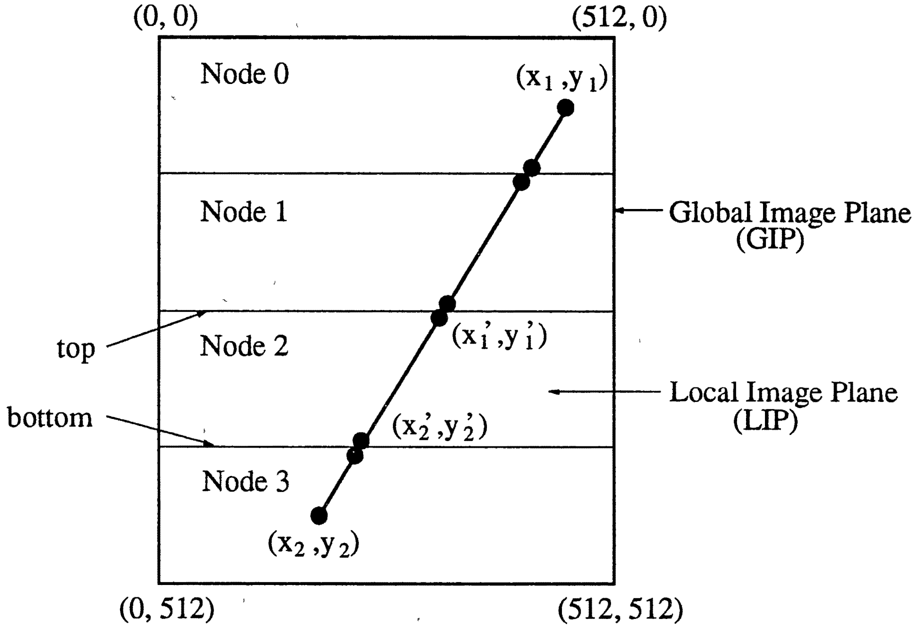


Figure III.14 Global and Local Image Planes

Parallel Arc Drawing

There exist several arc drawing algorithms [Dan70] [Jor73] [Sue79], but the simplest method is to create a suitable approximation to the arc by drawing it as a series of short straight lines [Dem84]. We benefit from the method because we have already implemented the Bresenham's line drawing algorithm in the previous section.

To produce good-looking arcs, it is obvious that the number of line segments needed for an arc should be increased as the radius increases. Very complex formulas can be developed to calculate the number of lines needed to produce an arc, but a simple one that works well on a typical microcomputer screen is

$$N = (min + const \times r) \times \frac{end\_angle - start\_angle}{2\pi} \tag{III.27}$$

where r is the radius of the arc, n is the number of lines required, and *start_angle* and *end_angle* are the angles of the ending and starting points of the arc, respectively. The minimum *min* and the constant multiplier *const* can be adjusted to produce acceptable resolution. In this research, we set the *min* to 20 and *const* to 0.4. A C program of the arc drawing method is given in Figure III.15. As we can see, an arc is converted into a series of the approximated line segments.

To implement the arc drawing in parallel on the hypercube, we divide the number of the arcs to be drawn by the number of nodes being used. Then the arc drawing procedure shown in Figure III.15 is performed in parallel on each node for each sub-arc.

```
Arc_Draw(xc, yc, xs, ys, xe, ye, r)
   int   xc, yc, /* Coordinates of the center point of the arc   */
         xs, ys, /* Coordinates of the starting point of the arc */
         xe, ye, /* Coordinates of the ending point of the arc   */
   float r;      /* Radius of the arc                            */
{
   /* Compute the angles of starting and ending points */
   start_angle = atan2(yc-ys, xc-xs);
   end_angle = atan2(yc-ye, xc-xe);

   /* Compute the number of the line segments for the arc */
   N = (20 + 0.4*r)*(end_angle - start_angle)/(2*pi);

   /* Compute the incremental angle */
   delta_angle = (end_angle - start_angle)/N;
```

Figure III.15 A C Program for the Arc Drawing

```
   /* Coordinates of the starting point */
   /* of the first line segment         */
   x1 = xs;
   y1 = ys;

   /* Generate the coordinates of the starting and ending points */
   /* of each line segment.  Then, draw the line segment         */
   for (k = 1; k <= N; k++) {

      /* Compute the coordinates of the ending point */
      x2 = cos(start_angle + k*delta_angle)*r + xc;
      y2 = sin(start_angle + k*delta_angle)*r + yc;

      /* Draw the line segment */
      Line_Draw(x1, y1, x2, y2);

      /* Set the coordinates of the starting point of the next  */
      /* line segment as the coordinates of the ending point of */
      /* current line segment                                   */
      x1 = x2;
      y1 = y2;
   }
}
```

(Figure III.15 *Continued*)

CHAPTER IV

PARALLEL MATCHING AND VERIFICATION

Most industrial vision systems are knowledge-based (or model-based) systems and usually employ the *hypothesis-test* paradigm in the matching phase. In the hypothesis generation step, the identities and locations of several objects are hypothesized; whereas in the hypothesis test step, the hypothesis is tested by checking if the hypothesized objects are matched to the input objects or not.

There are two types of approaches to hypothesize the identities and locations of objects in an input image: *model-driven* and *data-driven* [Meh90]. In the model-driven approach, all of the features of each model are tried in turn to the features of the input image for generating hypothesis. This approach has a disadvantage in terms of execution time because exhaustive searching is involved. Thus this approach is undesirable when the number of the model objects is large.

On the other hand, in the data-driven approach, the knowledge-base is searched for the features of the input image to hypothesize what objects are present in the input image. This approach is relatively efficient, but execution time still depends on the size of the knowledge-base. In order to eliminate this problem, a *hashing* scheme is applied in this research. The knowledge-base can then be searched in very few accesses in most cases.

In this chapter, we first describe how to model an object and generate the hypothesis by hashing. Then we develop a parallel matching strategy using coordinate transform and clustering. Finally, we describe how to verify the hypothesis.

Modeling and Hypothesis Generation

Once the extended local features such as corners, arcs, parallel-lines, and corner-arcs are extracted from the image with a model object, the model object is modeled by the extracted features. A database of model objects contains the name of the object, number of each feature type, and data of the feature type. The structures of a model database are shown in Figure IV.1.

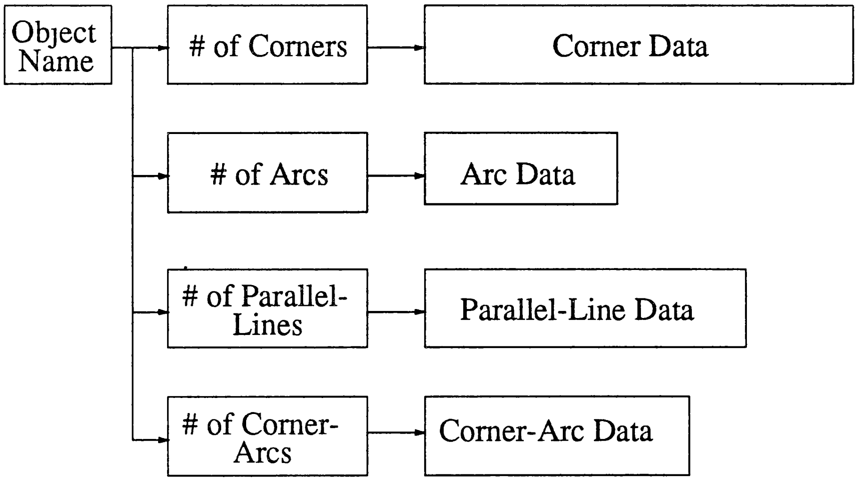


Figure IV.1 Structures of a Model Database

In order to recognize the objects in an input image, the feature extraction is also performed for the input image. Then one or more objects are hypothesized by searching the knowledge-base. In other words, the data-driven method is applied in this research. However, since the searching time increases linearly with the size of the knowledge-base, the *hashing* scheme is used to reduce the searching time.

Hashing is an important solution to the searching problem, because the search times can be independent of the number of records. That is, no matter how many records are stored, the average searching times remain bounded [Vit87] [Tre84]. Suppose we want to store m records in a contiguous area of memory containing at least n locations. A record is converted into a unique *key* K. Then a *hash function*, $H(K)$, transforms the key into the address space. This can be represented by the following equation:

$$H(K) \rightarrow \{0, 1, \ldots, n-1\} \qquad \text{(IV.1)}$$

For implementing the hashing, each feature data should be converted into a unique key first. How can we represent the data in numerical form? Fortunately, the spaces of the local feature data are finite. For example, in case of a corner, the range of the corner angles is from $-\pi$ to π and the range of the lengths of two straight lines is from 0 to $512\sqrt{2}$ when the size of the image is 512 x 512. Thus, the space can be digitized by equal division and a unique key can be assigned to each unit space. Figure IV.2 shows the range of each parameter according to the four different feature types. The numbers in the parentheses are unit spaces for the digitization.

Now, each feature data is converted into a number with seven digits, which becomes a key for the hashing. Formats of the keys for each feature type are shown in Figure IV.3. Conversion functions, which map the feature data to the keys, are given in the following:

1) *Corner*:

$$key = 1000000 + 10000 * \frac{(ang + \pi)}{0.1} + 100 * \frac{st1.len}{10} + \frac{st2.len}{10} \qquad \text{(IV.2)}$$

2) *Parallel-line*:

$$key = 2000000 + 10000 * \frac{d}{10} + 100 * \frac{st1.len}{10} + \frac{st2.len}{10} \qquad \text{(IV.3)}$$

3) *Arc*:

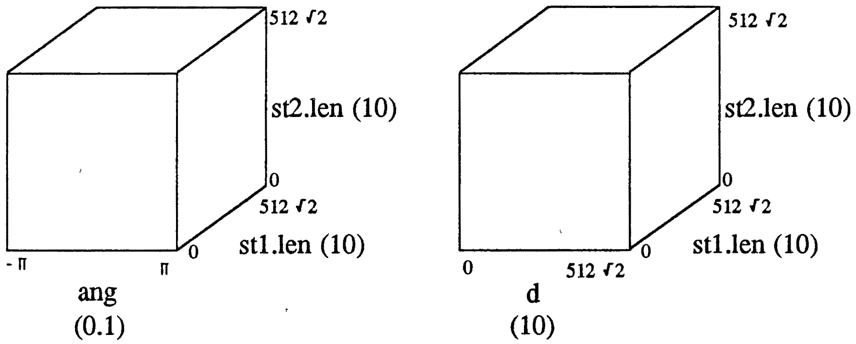
$$key = 3000000 + 1000*\frac{r}{10} + \frac{len}{10} \qquad \text{(IV.4)}$$

4) *Corner-arc*:

$$key = 4000000 + 10000*\frac{(ang + \pi)}{0.1} + 100*\frac{r}{10} + \frac{len}{10} \qquad \text{(IV.5)}$$

(a) Corner

(b) Parallel-Line



(c) Arc

(d) Corner-Arc

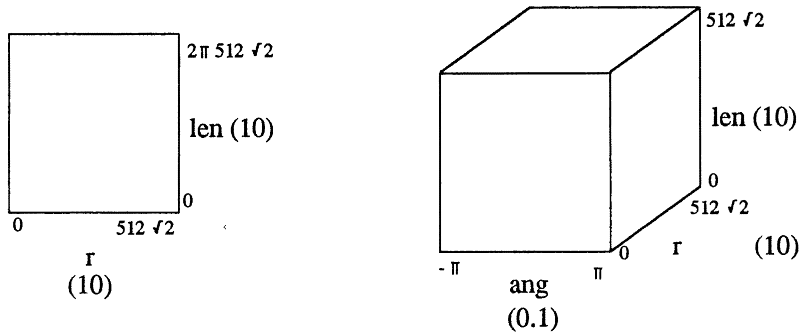


Figure IV.2 Ranges of the Parameters

(a) Corner

Type 1	ang	st1.len	st2.len

(b) Parallel-Line

Type 2	d	st1.len	st2.len

(c) Arc

Type 3	0	r	len

(d) Corner-Arc

Type 4	ang	r	len

Figure IV.3 Formats of the Keys for the Hashing

There are many different hashing functions, but a simple hashing function using the modulo operator is applied. A key is divided by a number that is the address size of the memory, and then the remainder will be the home address of the key. This can be represented by the following equation:

$$H(K) = K \ mod \ n \tag{IV.6}$$

The remainder produced by the modulo operator will be a number between 0 and n - 1.

When we store the feature data into a knowledge-base by using the above hashing function, the choice of the divisor n can have a major effect on how well the data are spread out. Usually, a *prime* number is used for the divisor because primes tend to distribute remainders much more uniformly than do nonprimes. Since the remainder is going to be the address of a data item, we choose a number as close as possible to the desired size of the address space. This number actually determines the size of the address space [Fol87].

If the hashing function hashes to an address that has already been occupied by a data item, that is, a *collision* between the data occurs, another hashing function is applied. We call this function the *rehashing function*, which is given by

$$H_i(K) = G(H_{i-1}(K)) \bmod n \tag{IV.7}$$

where $H_{i-1}(K)$ is the previous hashed or rehashed value, and $G(n)$ is the decimal value of $(n + 10)13$. For example,

$$G(53) = (53 + 10)13 = 63(13) = 6*13 + 3*1 = 81$$

If the previous rehashing function produces a *cycle* (i.e., visits an address a second time), then we switch to the following alternate rehashing function to complete the storing or searching data:

$$H_i(K) = (H_{i-1}(K) + 1) \bmod n \tag{IV.8}$$

where the initial value of $H_{i-1}(K)$ is the value produced by the previous rehashing function that caused the cycle. A flow chart for the insertion of a feature data is given in Figure IV.4. Note that only the key and object name for the feature data are inserted in the knowledge-base, and multiple object names might be listed under a key because the model objects could have the same feature data.

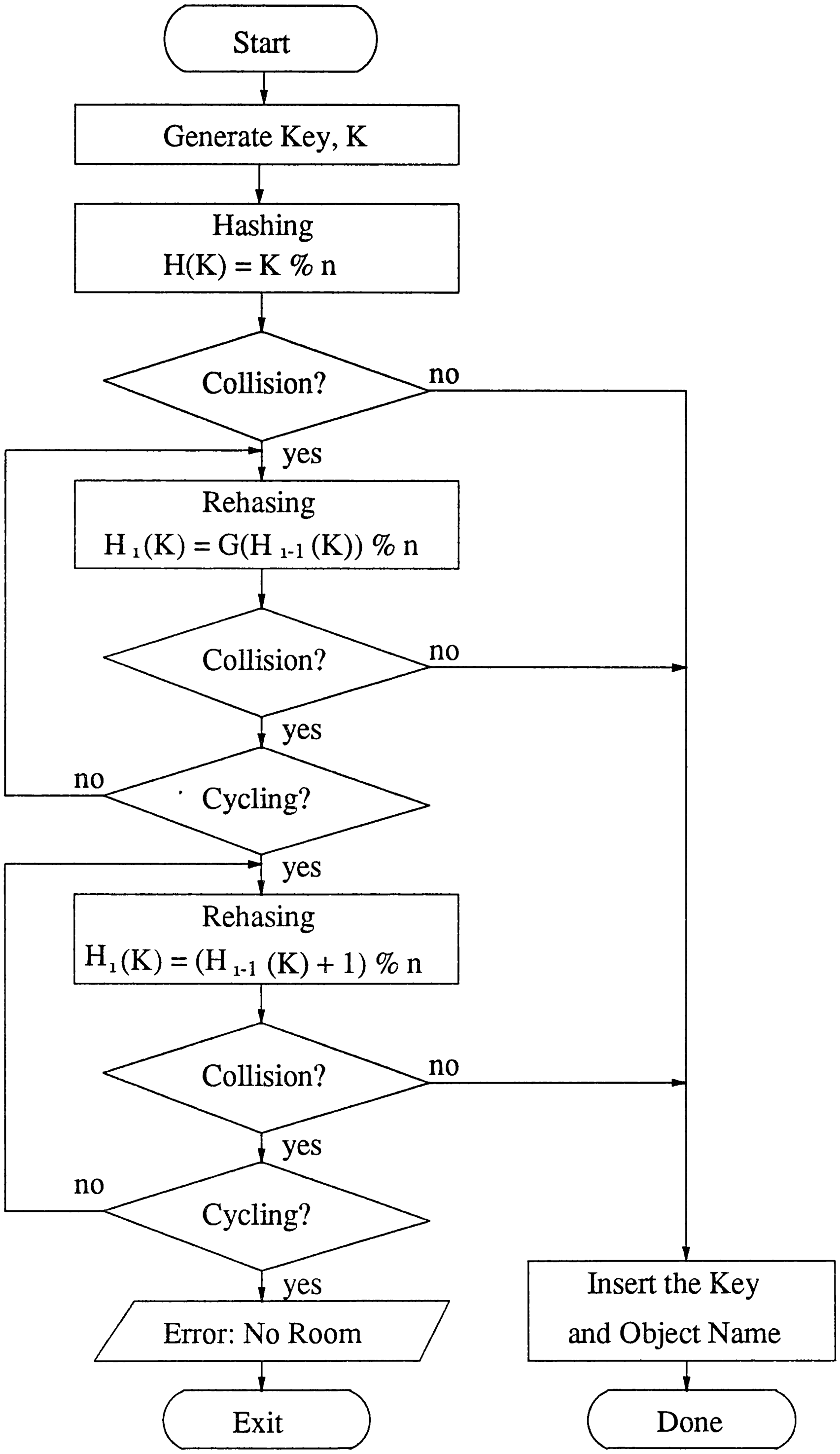


Figure IV.4 A Flow Chart for the Insertion of a Feature Data

For hypothesizing the objects in an input image, the feature data from the input image are converted into keys, and then each key is searched from the knowledge-base in the same manner as the insertion. If a key is found, the objects listed under the key are hypothesized. Now, the locations of these objects are hypothesized by the matching method discussed in the next section. For parallel implementation the number of hypothesized objects are divided by the number of the nodes being used, and the sub-objects are redistributed to the nodes.

Parallel Matching

If we assume that the objects to be recognized are rigid and hardly deformable, the objects could be rotated and translated in x and y directions. Therefore, the matching can be done by finding the translation and the rotation, and then transforming the coordinates of the model features. In this section, we will develop a parallel matching strategy using coordinate transform and clustering. Then we will verify the matching by transforming the model and overlapping the transformed model onto the image. The procedures are implemented in parallel on the iPSC/2 hypercube.

Parallel Implementation

The local features such as corners, arcs, parallel-lines, and corner-arcs described in Chapter III are extracted from both the model and the image. Every pair of model feature and image feature is compared to test their compatibility. If the number of model features is M and the number of image features is I, then the time complexity of the compatibility test is $O(MI)$. For large M and I, the time complexity of the compatibility test and subsequent procedures such as coordinate transformation and verification will be increased extensively. In order to solve this problem, we increase the processing parallelism by partitioning the feature data.

First, the root node (node 0) collects the data of corners, arcs, parallel-lines, and corner-arc from all nodes. Then the root node merges them and lists into the arrays *global_corner*, *global_arc*, *global_parallel_line*, and *global_corner_arc* respectively. Second, the root node redistributes the global data of the features to all nodes. Third, each node finds its own data to be concerned by dividing the number of global data by the number of nodes allocated. It should be noted that each node keeps only beginning and ending positions of the global data and not the actual data. In this manner we can save memory space. Now every pair of feature data from the global array and only its own data are compared in parallel in each node to test their compatibility. Then the coordinate transform is computed from each pair of compatible features.

Compatibility Between a Model Feature and an Image Feature

Since the objects to be matched are rigid and not easily deformable, some parameters of the features are invariant to rotation and translation. For example, in case of a corner, angle between two lines, or lengths of two lines are stable even if the corner (or the object) is rotated or translated. In this section, we represent compatibility between a model feature and an image feature as a distance function for each type of the features (see Figure IV.5).

Let us define *CC*, *AA*, *PP*, and *EE* as compatibility between corners, arcs, parallel-lines, and corner-arcs respectively. Then *CC*, *AA*, *PP*, and *EE* can be represented by the following:

1) *Corner to Corner* (*CC*):

$$CC = [C_1(\alpha_i - \alpha_m)^2 + C_2(l_{1i} - l_{1m})^2 + C_3(l_{2i} - l_{2m})^2]^{\frac{1}{2}} \tag{IV.9}$$

2) *Arc to Arc* (*AA*):

$$AA = [A_1(L_i - L_m)^2 + A_2(R_i - R_m)^2]^{\frac{1}{2}} \tag{IV.10}$$

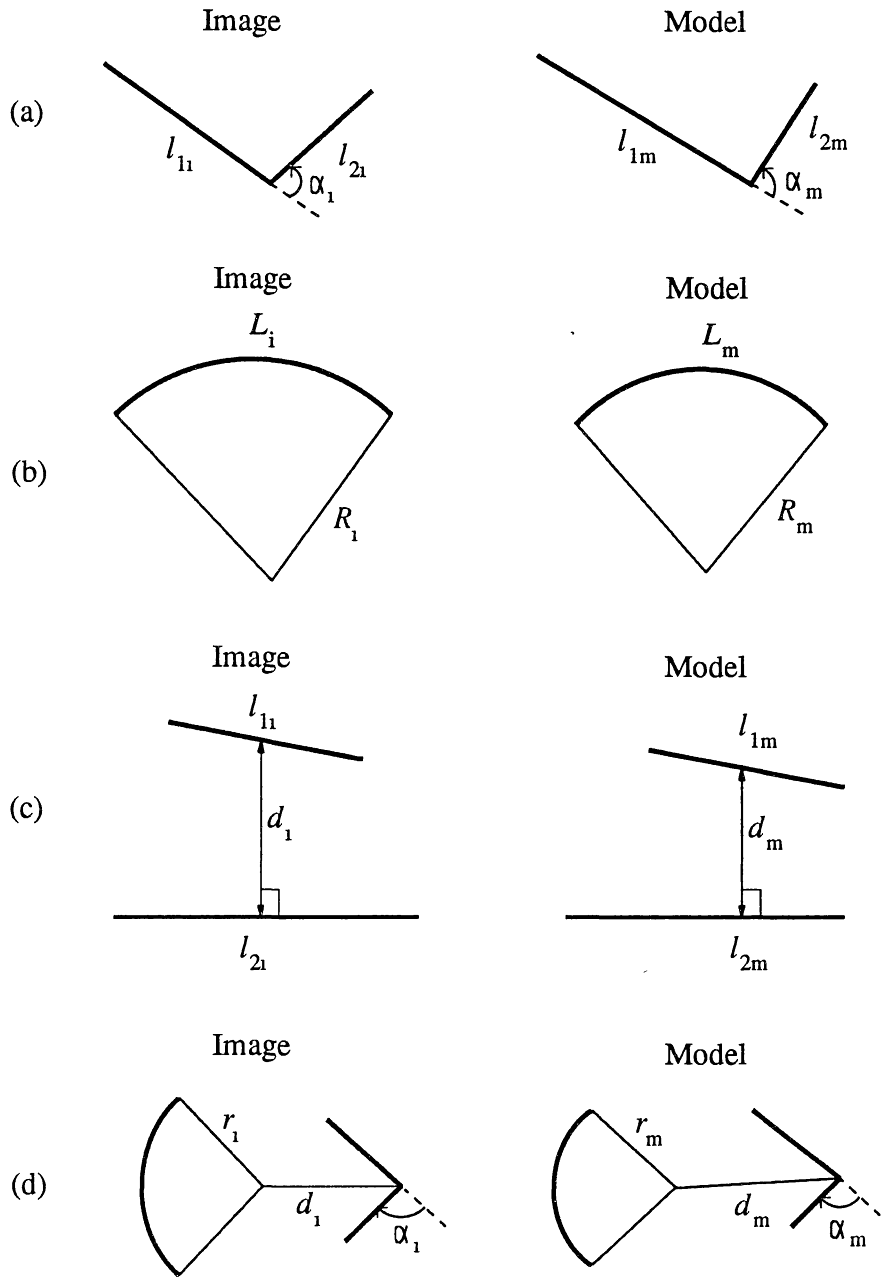


Figure IV.5 Compatibility Between a Model Feature and an Image Feature. (a) Corner to Corner, (b) Arc to Arc, (c) Parallel-line to Parallel-line, (d) Corner-arc to Corner-arc

3) *Parallel-line to Parallel-line* (*PP*):

$$PP = [P_1(d_t - d_m)^2 + P_2(l_{1t} - l_{1m})^2 + P_3(l_{2t} - l_{2m})^2]^{\frac{1}{2}} \tag{IV.11}$$

4) *Corner-arc to Corner-arc* (*EE*):

$$EE = [E_1(\alpha_t - \alpha_m)^2 + E_2(r_t - r_m)^2 + E_3(d_t - d_m)^2]^{\frac{1}{2}} \tag{IV.12}$$

where C_1, C_2, C_3, A_1, A_2, P_1, P_2, E_1, E_2, and E_3 are constants. If the values of the above compatibility functions are less than some threshold values, then we can say that the corresponding model and image features are compatible each other. Now we derive the coordinate transform from the set of the compatible features.

Coordinate Transform

In this section we derive the coordinate transform which brings the coordinates of a model feature into the coordinates of an image feature. Let θ and (t_x, t_y) be the rotation angle and the translation in x and y directions respectively. Let (M_{xi}, M_{yi}) and (I_{xi}, I_{yi}) be a point on a model feature and a point on an image feature respectively. The coordinate transform (θ, t_x, t_y) rotates a model point (M_{xi}, M_{yi}) by angle θ and translates by (t_x, t_y) in x and y directions, and then matches the transformed model point to the image point. This can be represented by the following equation in matrix form.

$$[I_{xi} \quad I_{yi}] = [M_{xi} \quad M_{yi}]\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} + [t_x \quad t_y] \tag{IV.13}$$

Solving for I_{xi} and I_{yi}, we have

$$I_{xi} = M_{xi}\cos\theta - M_{yi}\sin\theta + t_x \tag{IV.14}$$

and

$$I_{yi} = M_{xi}\sin\theta + M_{yi}\cos\theta + t_y \tag{IV.15}$$

Let us define Q_1 and Q_2 as

$$Q_1 = \sum_{i=1}^{n}(I_{xi} - M_{xi}\cos\theta + M_{yi}\sin\theta - t_x)^2 \tag{IV.16}$$

and

$$Q_2 = \sum_{i=1}^{n}(I_{yi} - M_{xi}\sin\theta - M_{yi}\cos\theta - t_y)^2 \tag{IV.17}$$

where n is the number of points taken along the feature segment. Let Q be the sum of Q_1 and Q_2. That is

$$Q = Q_1 + Q_2 \tag{IV.18}$$

Then we have to minimize Q since it is the sum of the squared errors between the points on the image and the points on the transformed model. In other words, set

$$\frac{\partial Q}{\partial t_x} = 0, \qquad \frac{\partial Q}{\partial t_y} = 0, \qquad \text{and } \frac{\partial Q}{\partial \theta} = 0 \tag{IV.19}$$

From the above conditions, we obtain

$$t_x = \frac{1}{n}\left(\sum_{i=1}^{n} I_{xi} - \cos\theta\sum_{i=1}^{n} M_{xi} + \sin\theta\sum_{i=1}^{n} M_{yi}\right) \tag{IV.20}$$

$$t_y = \frac{1}{n}\left(\sum_{i=1}^{n} I_{yi} - \sin\theta\sum_{i=1}^{n} M_{xi} + \cos\theta\sum_{i=1}^{n} M_{yi}\right) \tag{IV.21}$$

and

$$\theta = \tan^{-1}\left(\frac{-n \sum_{i=1}^{n} I_{xi}M_{yi} + \sum_{i=1}^{n} I_{xi} \sum_{i=1}^{n} M_{yi} + n \sum_{i=1}^{n} I_{yi}M_{xi} - \sum_{i=1}^{n} I_{yi} \sum_{i=1}^{n} M_{xi}}{n \sum_{i=1}^{n} I_{xi}M_{xi} - \sum_{i=1}^{n} I_{xi} \sum_{i=1}^{n} M_{xi} + n \sum_{i=1}^{n} I_{yi}M_{yi} - \sum_{i=1}^{n} I_{yi} \sum_{i=1}^{n} M_{yi}} \right) \tag{IV.22}$$

See Appendix B for the derivations of the Equations (IV.20), (IV.21), and (IV.22).

Since we assume that the objects are rigid, the coordinate transforms for the compatible features between the image and the model will produce a group of matches or cluster which has the same or similar coordinate transforms. To find the group of matches, we apply a cluster-seeking algorithm discussed in the next section.

Cluster Seeking

There exist several applicable cluster-seeking algorithms. Examples are *K-means algorithm*, *maximin-distance algorithm*, and *Isodata algorithm* [Tou74]. In this section we will evaluate the three cluster seeking algorithms and select one of them for our research. Then we will describe the selected algorithm step by step.

The K-means algorithm is based on the minimization of the sum of the squared distances from all points in a cluster domain to the cluster center. This procedure consists of the following steps:

Step 1: Choose K initial cluster centers $\mathbf{c}_1(1), \mathbf{c}_2(1),..., \mathbf{c}_K(1)$ arbitrarily.

Step 2: At the kth iterative step, distribute the sample $\{\mathbf{p}\}$ among the K cluster domains, using the relation,

$$\mathbf{p} \in S_j(k) \quad \text{if} \quad \|\mathbf{p} - \mathbf{c}_j(k)\| < \|\mathbf{p} - \mathbf{c}_i(k)\| \tag{IV.23}$$

for all $i = 1,2,\ldots,K, i \neq j$, where $S_j(k)$ denotes the set of samples whose cluster center is $\mathbf{c}_j$(k), and $\|\mathbf{p} - \mathbf{c}\|$ denotes the Euclidean distance between two patterns $\mathbf{p}$ and $\mathbf{c}$.

Step 3: From the results of Step 2, compute the new cluster centers $\mathbf{c}_j(k+1), j = 1,2,\ldots,K$, such that the sum of the squared distances from all points in $S_j(k)$ to the new cluster center is minimized. In other words, the new cluster center $\mathbf{c}_j(k+1)$ is computed so that the performance index

$$J_j = \sum_{\mathbf{p} \in S_j(k)} \|\mathbf{p} - \mathbf{c}_j(k+1)\|^2, \quad j = 1,2,\ldots,K \qquad \text{(IV.24)}$$

is minimized. The $\mathbf{c}_j(k+1)$ which minimizes the performance index is simply the sample mean of $S_j(k)$. Consequently, the new cluster center is given by

$$\mathbf{c}_j(k+1) = \frac{1}{N} \sum_{\mathbf{p} \in S_j(k)} \mathbf{p}, \quad j = 1,2,\ldots,K \qquad \text{(IV.25)}$$

where N_j is the number of samples in $S_j(k)$.

Step 4: If $\mathbf{c}_j(k+1) = \mathbf{c}_j(k)$ for $j = 1,2,\ldots,K$, the algorithm has converged and the procedure is terminated. Otherwise go to Step 2.

Bhanu and Ming used the K-means algorithm in [Bha86] and [Bha87] because the K-means algorithm is relatively simple to implement. However, the behavior of the K-means algorithm is influenced by the number of cluster centers specified before executing the algorithm and the choice of initial cluster centers. In practice, it is very difficult to predict the number of cluster centers desired.

The Isodata algorithm is similar in principle to the K-means procedure in the sense that cluster centers are iteratively determined by the sample means. However, Isodata represents a fairly comprehensive set of additional heuristic procedures which have been incorporated into an interactive scheme. For a set of N samples, Isodata consists of the following principal steps [And73]:

Step 1: Choose values for the process parameters:

N_c = number of current cluster centers;
K = number of cluster centers desired;

θ_N = a parameter against which the number of samples in a cluster domain is compared;
θ_e = standard deviation parameter;
θ_c = lumping parameter;
L = maximum number of pairs of cluster centers which can be lumped;
I = number of iterations allowed.

Step 2: If a set of seed points is not provided as part of the input, then generate a set of seed points.

Step 3: Assign each data unit to the cluster with the nearest seed point.

Step 4: Discard any cluster which contains fewer than θ_N data units.

Step 5: Perform either a lumping or a splitting iteration (details are specified below) according to the following rules:

a) A *lumping* iteration is mandatory if $N_c \geq 2K$.
b) A *splitting* iteration is mandatory if $N_c \leq K/2$.
c) Otherwise, alternate the processes by splitting on odd iterations and lumping on even iterations.

Step 6: Compute new seed points as cluster centers and perform the data unit reallocation/seed point recomputation cycles.

Step 7: Repeat steps 4, 5, and 6 until the process converges or until these three steps have been repeated I times.

During a *lumping* iteration, all pairwise distances between cluster centers are computed. If the distance between the two nearest centers is less than θ_c, then the associated clusters are merged and the distance to all other centers is computed. This process is continued up to a maximum of L merges in any iteration. During a *splitting* iteration, a cluster provisionally is chosen for splitting if the within cluster standard deviation for any variable exceeds the product of θ_e and the standard deviation of that variable in the original data set as a whole.

In the Isodata algorithm, it is necessary to specify the number of initial cluster centers, but the number is not necessary to be equal to the number of the desired cluster centers because the number of cluster centers is updated during execution. However, the Isodata algorithm is very complicated and requires extensive experimentation before arriving at meaningful conclusions.

On the other hand, the maximin-distance algorithm is based on the Euclidean distance concept. It is very simple to implement and it is not necessary to specify the number of initial cluster centers. Therefore we select the maximin-distance algorithm for our use. Let $\mathbf{p}_i$ be a sample pattern and let $\mathbf{c}_1$ be a cluster center. Then the maximin-distance algorithm consists of the following steps.

Step 1: Arbitrarily let $\mathbf{p}_1$ become the first cluster center $\mathbf{c}_1$.

Step 2: Determine the farthest sample from $\mathbf{p}_1$, and let it become the second cluster center $\mathbf{c}_2$.

Step 3: Compute the distance from each remaining sample to the existing centers while saving the minimum of the computed distances. Then select the maximum of the minimum distances. If the maximum distance is an appreciable fraction of the average distance between the current cluster centers (in our research, at least one half of the average distance), then let the corresponding sample be a new cluster center.

Step 4: If a new cluster center was created in Step 3, then go to Step 3. Otherwise the algorithm is terminated.

Hypothesis Verification of the Matching

Through the coordinate transform and the cluster-seeking procedures, we obtain the rotation angle θ and the translation in x and y directions (t_x, t_y). Now the points of the model object are rotated by θ and translated by (t_x, t_y). If (M_{xi}^T, M_{yi}^T) is a transformed model point, then this can be represented by the following equation.

$$[M_{xi}^T \quad M_{yi}^T] = [M_{xi} \quad M_{yi}]\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} + [t_x \quad t_y] \tag{IV.26}$$

Or

$$M_{xi}^T = M_{xi}\cos\theta - M_{yi}\sin\theta + t_x \tag{IV.27}$$

and

$$M^T_{yi} = M_{xi}\sin\theta + M_{yi}\cos\theta + t_y \tag{IV.28}$$

To verify the matching, the transformed model M^T is imposed onto the input image, and then the input image pixel at every transformed model point is checked if the pixel belongs to object area or background. If an input image pixel at a transformed model point belongs to object area, then we say that the point is "matched". The matching ratio R_m can be defined by the following formula:

$$R_m = \frac{\text{Number of matched points}}{\text{Number of model points}} \times 100\ (\%) \tag{IV.29}$$

If the matching ratio is greater than a predefined value, the hypothesis is accepted. Otherwise, it is rejected.

The verification method described above (we call that *fine*-verification method) is very time-consuming because all of the model points are transformed and the input image points corresponding to the model points are checked if they belong to object area or not. An alternative method (*coarse*-verification method) is considered. Instead of checking all of the model points, only the boundary points at equispaced locations are checked. Figure IV.6 demonstrates the coarse-verification method.

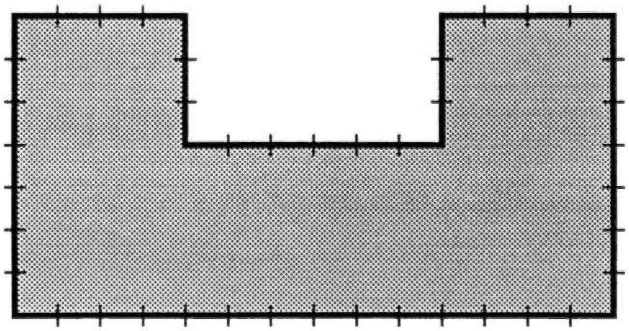


Figure IV.6 Coarse-Verification Method

CHAPTER V

EXPERIMENTAL RESULTS

The parallel occluded object recognition algorithm using extended local features such as corners, arcs, parallel-lines, and corner-arcs is implemented on a hypercube-topology multiprocessor computer, the Intel iPSC/2. For model objects, we use ten different industrial tools shown in Figure I.2.

Three experiments are made and their results are provided in this chapter. In the first experiment a preliminary test is performed with a simple occluded image. All of the procedures such as preprocessing, feature extraction, matching, and verification are performed to see how the system works. A precision test is given in the second experiment. We check how precisely an object whose location is already known is located. Also results of the cluster-seeking are provided. Finally, a comprehensive test is performed in the third experiment. Twenty synthetic occluded images are constructed, and the objects in each synthetic image are recognized. Overall recognition rate is provided.

Experiment 1 (Preliminary Test)

For preliminary experimental results, we use only three industrial tools as model objects; an adjustable wrench, a long nose plier, and an Allen wrench. The image of each model object is shown in Figure V.1(a), V.1(b), and V.1(c). The occluded image shown in Figure V.1(d) is an input image to be matched or recognized. The size of the image is 512 x 512. Note that the objects in the occluded image are rotated and translated arbitrarily.

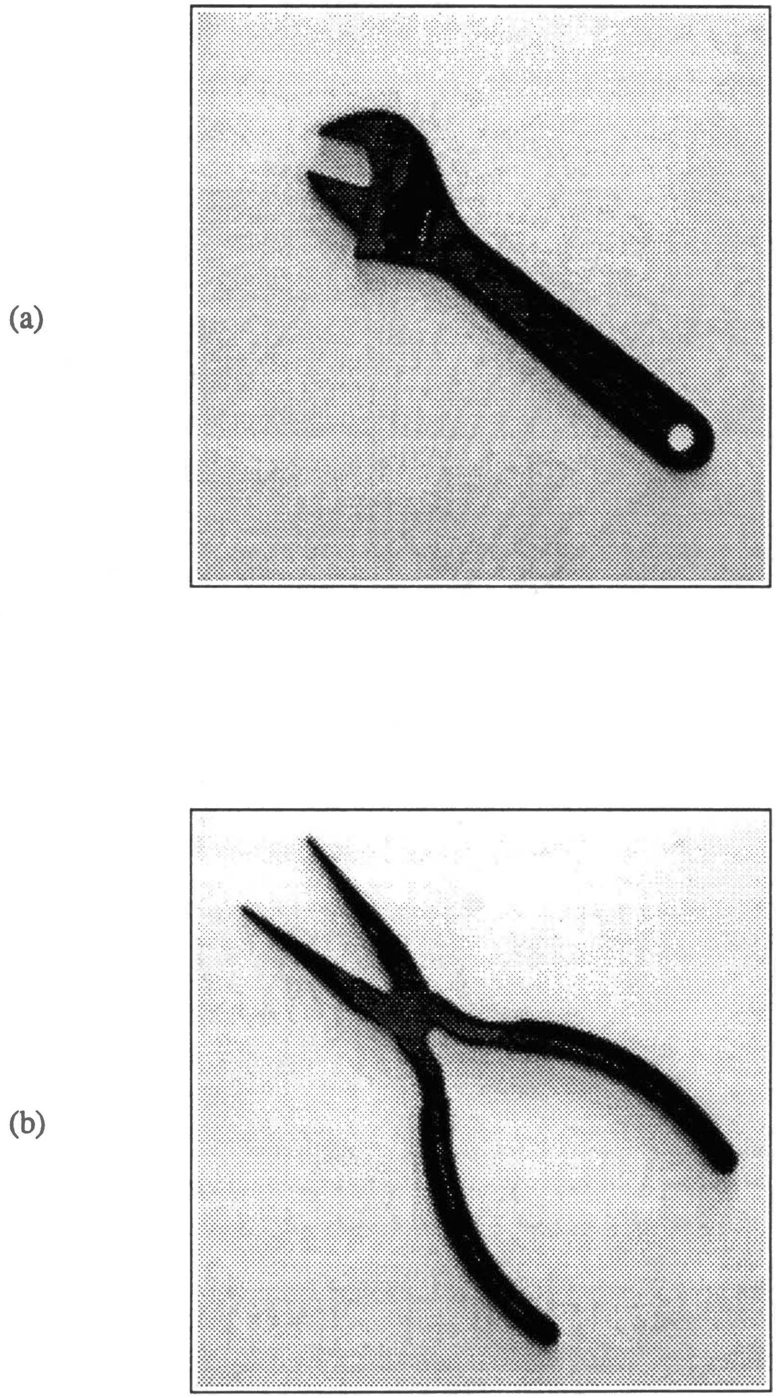
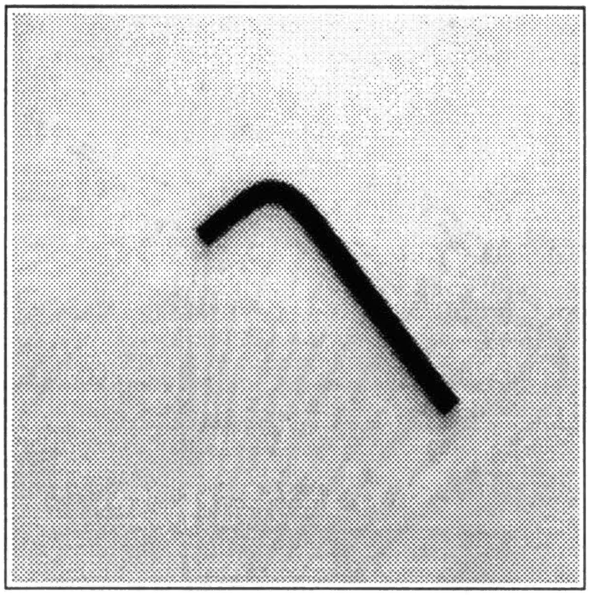
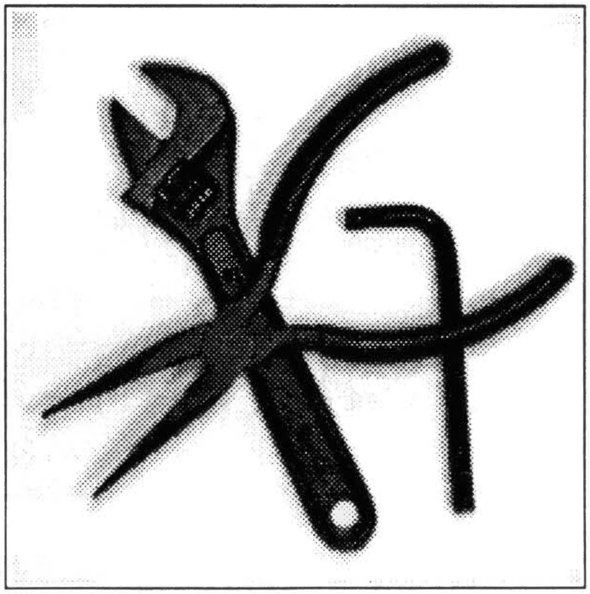


Figure V.1 Images of Model Objects and Input. (a) Image of Adjustable Wrench (model, 512 x 512), (b) Image of Long Nose Plier (model, 512 x 512), (c) Image of Allen Wrench (model, 512 x 512), (d) Image of Occluded Objects (input, 512 x 512)

(c)



(d)



(Figure V.1 *Continued*)

Before extracting features, we perform preprocessing such as edge detection and straight line extraction. First of all, according to the input image distribution method discussed in Chapter II, the input images are divided by the number of nodes being used and each resulting sub-image is distributed to one node. Each sub-image is convolved with a 9 x 9 Laplacian of Gaussian operator (see Figure II.5), and then zero-crossing points are detected in parallel on the hypercube.

The zero-crossing points detection is a task of assigning the changes in sign of a convolved image with a Laplacian of Gaussian operator as the location of the edge [Vli89]. There exist several algorithms for zero-crossing point detection. Examples are [Kim87] and [Vli89]. In [Kim87] a zero-crossing patterns is classified into one of 16 patterns by assigning its pattern value according to the direction of the connectivity. These 16 zero-crossing patterns represent all the possible types of the spatial connection of a zero-crossing point.

On the other hand, in [Vli89] the convolved image is segmented into positive, zero, or negative regions dependent upon the pixel value. Then the minimum distance for a zero value pixel to both the positive and negative regions is computed with Borgefors' distance transformation [Bor84]. A pseudo-Euclidean distance from the given (zero) pixels to the nearest (positive or negative) regions is computed by the transformation in two passes through the image. The zero value pixels are now assigned to the nearest adjacent region. This results in a binary image of positive and negative regions. Finally the 8-connected contour of the positive region is extracted by an exclusive-or with the 4-connect eroded image for getting closed contours.

The two zero-crossing algorithms described above have some disadvantages in terms of execution times. The searching of 16 zero-crossing patterns in [Kim87] is a time-consuming task, and the distance transformation in [Vli89] involves heavy computations. Also, it is not necessary to get closed contours because we are dealing with noisy images. This has been discussed in Chapter I. In this research, we develop a simple and fast zero-crossing points detection method.

After convolving an image with the Laplacian of Gaussian operator, the sign of each pixel is compared with the signs of the pixels in *east* and *south* neighbors. If any sign changes are detected, we consider the pixel with positive value as a zero-crossing point. Here we compute the magnitude of the gradient at the zero-crossing point by the Sobel operators for thresholding. Only when the magnitude is greater than a pre-defined threshold value, a value (255) is assigned to that point. The following is a C program for this method:

```
Zero_Crossing(S, D)
   Image S, D;    /* Source and Destination image buffers */
{
   /* Initialize the destination buffer with 0 */
   Initialize(D);

   for (i = 0; i < row-1; i++) {
      for (j = 0; j < column-1; j++) {

         /* Compare the sign of the current pixel */
         /* with the sign of the pixel in east    */
         if (S(i,j)*S(i,j+1) < 0) {

            /* Compute the magnitude of gradient by Sobel    */
            /* operators.  If the magnitude is greater than  */
            /* threshold value, val = 1.  Otherwise val = 0. */
            val = Sobel_mag(i,j);

            /*  Assign 255 to the point with positive value  */
            if (S(i,j) > 0)
               D(i,j) = 255*val;
            else
               D(i,j+1) = 255*val;
         }

         /* For the pixel in south */
         else if (S(i,j)*S(i,j+1) < 0) {
            val = Sobel_mag(i,j);
            if (S(i,j) > 0)
               D(i,j) = 255*val;
            else
               D(i,j+1) = 255*val;
         }
         else
            ;
      }
   }
}
```

Figure V.2 is the edge detected image from Figure V.1(d). We can see that the edges are one-pixel wide but not closed.

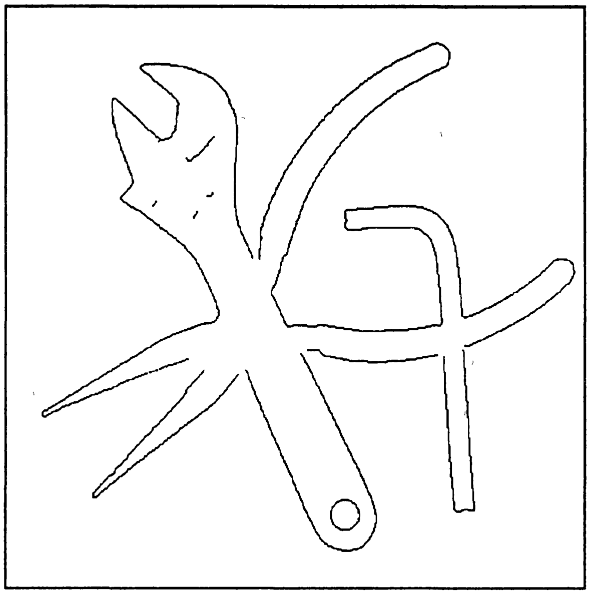


Figure V.2 Edge Detected Image from Figure V.1(d)

After the zero-crossing points detection, straight lines are extracted from the edge detected images by using the polygonal approximation method described in Chapter II. *Segments* (or connected edges) are extracted, and then each segment is approximated by several straight lines according to their lengths and deviations. The straight lines in each node (*local straight lines*) are linked globally in parallel by the algorithm given in Figure II.7. Figure V.3 shows the straight lines extracted from Figure V.2. 77 straight lines are extracted in this case. Also, 31, 39, and 9 straight lines are extracted from the images of the adjustable wrench, long nose plier, and Allen wrench, respectively. Now the objects can be simply represented by the straight lines. Thus the storage requirements for the objects can be reduced significantly. For example, only 0.8 Kbytes are needed for storing the straight lines of the adjustable wrench while 262 Kbytes are needed for the original raw image.

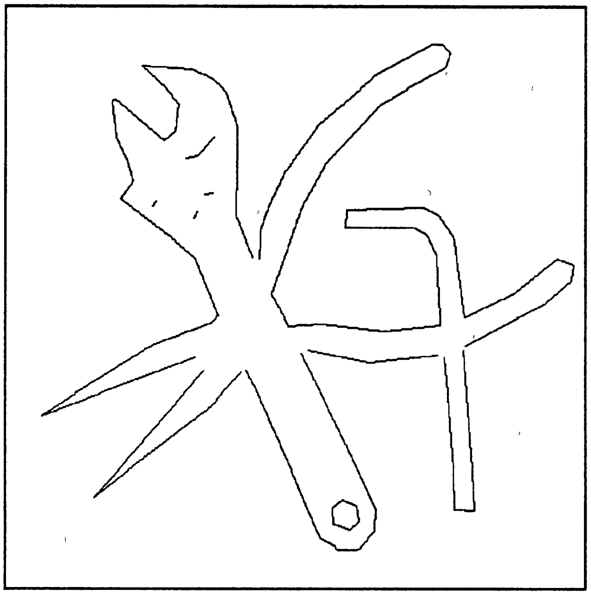


Figure V.3 Straight Lines Extracted from Figure V.2

In order to extract the local features in parallel, the straight lines are redistributed to the nodes by the global straight line division method given in Chapter III. The root node (node 0) collects the local straight lines in each node for the global straight lines, and sends the data of the global straight lines to all of the nodes. For load balancing, the global straight lines are split as evenly as possible according to the number of nodes being allocated. Note that the global straight lines are not split physically. Each node figures out only the beginning and ending positions for its own local straight lines. Now every pair of the global straight lines and the local straight lines are compared with each other, and then the extended local features such as corners, arcs, parallel-lines, and corner-arcs are extracted in parallel.

First of all, corners are detected from the straight lines. If the distance between the beginning or ending points of any two straight lines is less than three pixels, we regard the point where two straight lines merge as a *corner*. There are four different types of arrangement of the two straight lines, which are illustrated in Figure V.4. According to the type of the arrangement, we list the coordinates of the straight lines in different order so that the ending point of *st1* and the beginning point of *st2* should stand face to face with each other like type 3 in Figure V.4. This is important because the beginning or ending point of the straight line of a model is mapped to that of the input in the coordinate transform. The lengths of the two straight lines are computed. If the length of *st2* is greater than that of *st1*, we exchange *st1* for *st2*. Finally the angle of the corner is computed by the method described in the section of *Computation of Corner Angle* in Chapter III. Figure V.5 shows the corners of the long nose plier and their data is listed in Table III.

After detecting the corners, arcs are detected from the corners. Since the proximity or connectivity of every pair of straight lines has already been checked during the corner detection, we use the corner data for searching the candidates for the arcs. Each candidate should consist of three or more connected lines. In order to reduce the searching space, we drop the corners with high curvature or large difference in lengths of two straight lines, because those corners have little possibility to be a member of the candidates. If a corner angle is greater than $4\pi / 9$ radians (80 degrees) or the ratio of the length of the longer line (*st1*) to the length of the shorter line (*st2*) is greater than 2.5, then we drop the corner from the list. For searching the candidates, we check if any two corners are adjacent each other. If so, we link the straight lines of the corners. This procedure is repeated recursively until no lines are linked. Note that the lines are linked in a clockwise manner.

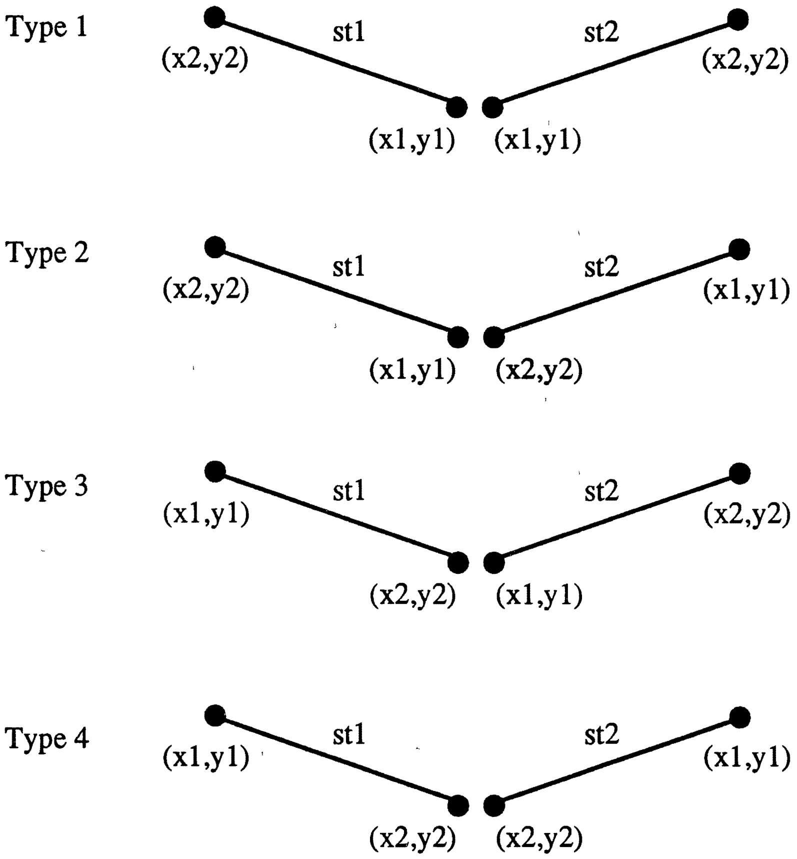


Figure V.4 Types of Arrangement of Two Straight Lines

For parallel implementation of the arc detection procedure, the candidates for the arcs are divided by the number of nodes being used. Each subset of the candidates is redistributed to each node, and then the parameters of the arc such as center point, radius, and length of the arc are computed in parallel. We estimate the center point and radius of the arc by solving the Equations (III.5), (III.6), and (III.7). To solve the equations we use Newton's method given in Equation (III.9). The initial values for the Equation (III.9) are crucial for reducing the number of iterations. So, we have developed a method for estimating good initial values in Chapter III (see Figure III.7).

TABLE III

DATA OF THE CORNERS OF THE LONG NOSE PLIER

Corner		Line 1					Line 2				
Num	ang	x1	y1	x2	y2	len	x1	y1	x2	y2	len
0	-1.15	45	77	170	150	144.8	170	150	177	145	8.6
1	0.59	462	253	485	282	37.0	485	282	486	294	12.0
2	0.87	485	282	486	294	12.0	486	294	481	299	7.1
3	-1.44	308	442	324	441	16.0	324	441	325	427	14.0
4	-0.78	263	403	308	442	59.5	308	442	324	441	16.0
5	-0.18	40	78	134	158	123.4	134	158	177	183	49.7
6	2.24	134	158	40	78	123.4	40	78	45	77	5.1
7	-0.73	170	150	45	77	144.8	45	77	40	78	5.1
8	0.77	178	142	100	22	143.1	100	22	101	17	5.1
9	-2.19	185	106	101	17	122.4	101	17	100	22	5.1
10	0.22	101	17	185	106	122.4	185	106	210	148	48.9
11	0.56	257	172	210	148	52.8	210	148	185	106	48.9
12	-0.30	391	198	322	174	73.1	322	174	257	172	65.0
13	0.23	322	174	391	198	73.1	391	198	427	221	42.7
14	0.16	427	221	462	253	47.4	462	253	485	282	37.0
15	-0.17	462	253	427	221	47.4	427	221	391	198	42.7
16	0.28	326	202	244	191	82.7	244	191	212	177	34.9
17	-1.32	244	191	212	177	34.9	212	177	205	186	11.4
18	-0.21	227	238	230	283	45.1	230	283	241	322	40.5
19	0.24	273	379	241	322	65.4	241	322	230	283	40.5
20	0.90	273	379	325	427	70.8	325	427	324	441	14.0
21	0.31	325	427	273	379	70.8	273	379	241	322	65.4
22	0.50	134	158	177	183	49.7	177	183	201	223	46.6
23	0.33	177	183	201	223	46.6	201	223	205	242	19.4
24	0.44	322	174	257	172	65.0	257	172	210	148	52.8
25	0.28	369	216	428	253	69.6	428	253	468	298	60.2
26	-0.25	428	253	369	216	69.6	369	216	326	202	45.2
27	0.18	244	191	326	202	82.7	326	202	369	216	45.2
28	1.21	219	209	205	186	26.9	205	186	212	177	11.4
29	-0.20	230	283	227	238	45.1	227	238	219	209	30.1
30	-0.28	227	238	219	209	30.1	219	209	205	186	26.9
31	-0.23	225	351	263	403	64.4	263	403	308	442	59.5
32	0.29	263	403	225	351	64.4	225	351	204	292	62.6
33	0.99	201	223	205	242	19.4	205	242	201	246	5.7
34	-0.77	428	253	468	298	60.2	468	298	481	299	13.0
35	-0.86	468	298	481	299	13.0	481	299	486	294	7.1
36	0.28	225	351	204	292	62.6	204	292	201	246	46.1
37	0.85	204	292	201	246	46.1	201	246	205	242	5.7

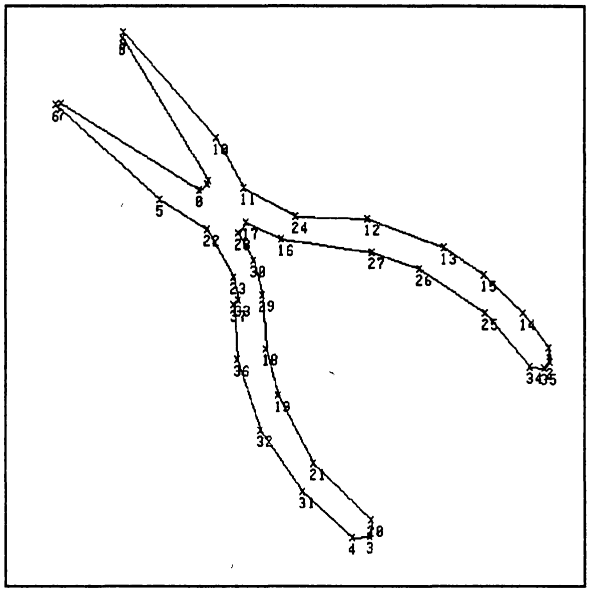


Figure V.5 Corners of the Long Nose Plier

Let (x_{11}, y_{11}), (x_{12}, y_{12}), (x_{nl}, y_{nl}), and (x_{n2}, y_{n2}) be the coordinates of the beginning and ending points of the first and last straight lines of the candidate, respectively. Also, let (x_{1m}, y_{1m}) and (x_{nm}, y_{nm}) be the coordinates of the middle points of the first and last straight lines respectively. Then the initial values for the center point (X_c^0, Y_c^0) are given by the following:

i) When $y_{11} \neq y_{12}$ and $y_{nl} \neq y_{n2}$:

$$X_c^0 = \frac{dev_2 * x_{2m} + y_{2m} - dev_1 * x_{1m} - y_{1m}}{dev_2 - dev_1}$$

$$Y_c^0 = -dev_1 * X_c^0 + dev_1 * x_{1m} + y_{1m}$$

ii) When $y_{11} \neq y_{12}$ and $y_{nl} = y_{n2}$:

$$X_c^0 = x_{2m}$$

$$Y_c^0 = -dev_1 {*} X_c^0 + dev_1 {*} x_{1m} + y_{1m}$$

iii) When $y_{11} = y_{12}$ and $y_{nl} \neq y_{n2}$:

$$X_c^0 = x_{1m}$$

$$Y_c^0 = -dev_2 {*} X_c^0 + dev_2 {*} x_{2m} + y_{2m}$$

where

$$dev_1 = \frac{x_{12} - x_{11}}{y_{12} - y_{11}}$$

$$dev_2 = \frac{x_{n2} - x_{nl}}{y_{n2} - y_{nl}}$$

The initial radius R^0 is the distance from (X_c^0, Y_c^0) to (x_{1m}, y_{1m}) or (x_{nm}, y_{nm}).

The Jacobian matrix given in Equation (III.10) is obtained by taking the partial derivative of f_1, f_2, and f_3 with respect to X_c, Y_c, and R. The followings are the elements of the Jacobian matrix:

$$\frac{\partial}{\partial X_c} f_1 = -3 \sum_{i=1}^{n} (X_i - X_c)^2 - \sum_{i=1}^{n} (Y_i - Y_c)^2 + nR^2$$

$$\frac{\partial}{\partial Y_c} f_1 = -2 \sum_{i=1}^{n} (Y_i - Y_c)(X_i - X_c)$$

$$\frac{\partial}{\partial R} f_1 = -2R \sum_{i=1}^{n} (X_i - X_c)$$

$$\frac{\partial}{\partial X_c} f_2 = -2 \sum_{i=1}^{n} (X_i - X_c)(Y_i - Y_c)$$

$$\frac{\partial}{\partial Y_c} f_2 = -\sum_{i=1}^{n} (X_i - X_c)^2 - 3 \sum_{i=1}^{n} (Y_i - Y_c)^2 + nR^2$$

$$\frac{\partial}{\partial R} f_2 = -2R \sum_{i=1}^{n} (Y_i - Y_c)$$

$$\frac{\partial}{\partial X_c} f_3 = -2 \sum_{i=1}^{n} (X_i - X_c)$$

$$\frac{\partial}{\partial Y_c} f_3 = -2 \sum_{i=1}^{n} (Y_i - Y_c)$$

$$\frac{\partial}{\partial R} f_3 = -2nR$$

where n is the number of vertices. The inverse of the Jacobian matrix is computed by Equation (III.11)

The final center point and radius are obtained by solving Equation (III.9) iteratively. At any iteration, if the difference between the previous value and the current value is less than ε, then an arc has been detected successfully from the candidate. Here ε is set to 0.001. However, if the number of iterations exceeds a certain number (100 in this experiment), then the iteration is terminated and the candidate is rejected. Finally, the length of the arc L is computed by Equation (III.18). Figure V.6 illustrates the arcs detected from the long nose plier, which consist of 8 arcs. Note that some beginning or ending portions of the arcs are overlapped with each other. This could happen because a straight line is usually shared by two adjacent corners. However, this is not a serious problem because the coordinates of the starting and ending points of the arcs are used in the matching.

Parallel-lines are also detected from the straight lines. In order to detect the parallel-lines, the significance of parallelism given in Equation (III.20) is checked between two straight lines. The constant C is set to 1 and the threshold value for the significance of parallelism S_{par} is set to 0.02. The greater S_{par}, the more significant in parallelism. So, if S_{par} between any two straight lines is greater than 0.02, they are regarded as a parallel-line.

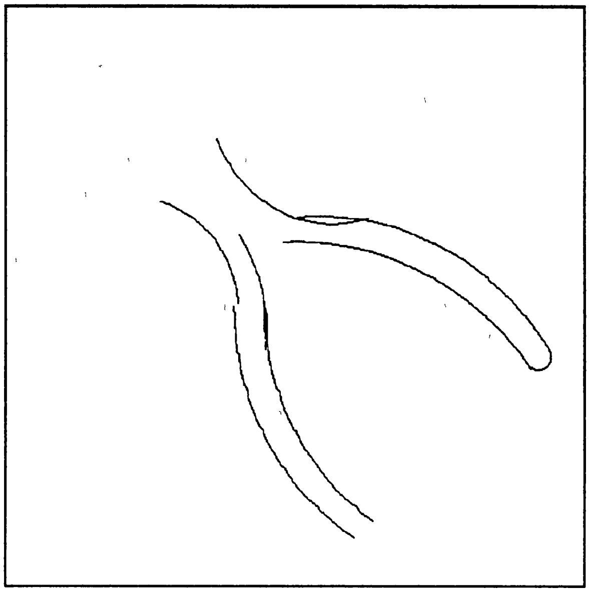


Figure V.6 Arcs of the Long Nose Plier

However, the parallel-line detection method does not work very well in practice because the S_{par} is composed of three parameters; angular difference θ, perpendicular distance d, and ratio of the lengths of two straight lines l_2 / l_1. If θ is large but d and l_2 / l_1 are small, the S_{par} could be greater than the threshold value, and thus a parallel-line could be detected. But the two straight lines are not parallel actually because they have large difference in angle.

To solve this problem, we check the three parameters for the S_{par} in a hierarchical fashion. We first check the angular difference θ between two straight lines. If the θ is less than $\pi / 20$ radians (9 degrees), then we check the perpendicular distance d. If the d is less than 100.0, then we check the ratio of the length l_2 / l_1. If l_2 / l_1 is less than 3.0, then the two straight lines are regarded as a parallel-line. Otherwise, they are rejected from the parallel-line testing. Here we check one more parameter, degree of alignment (*doa*) of the two lines. Let (x_{11}, y_{11}), (x_{12}, y_{12}), (x_{21}, y_{21}), and (x_{22}, y_{22}) be the coordinates of the beginning and ending points of the longer line and the shorter line, respectively. Also, let (x_{1m}, y_{1m}) and (x_{2m}, y_{2m}) be the coordinates of the middle points of the longer line and the shorter line, respectively. Then, the *doa* is given by following formula:

$$doa = \frac{(x_{1m} - x_{2m})(x_{11} - x_{12}) + (y_{1m} - y_{2m})(y_{11} - y_{12})}{\sqrt{(x_{11} - x_{12})^2 + (y_{11} - y_{12})^2}} \tag{V.1}$$

If the *doa* is less than (*length of longer line* / 4), the two straight lines are listed into an array for parallel-lines. The parallel-lines detected from the Allen wrench are shown in Figure V.7.

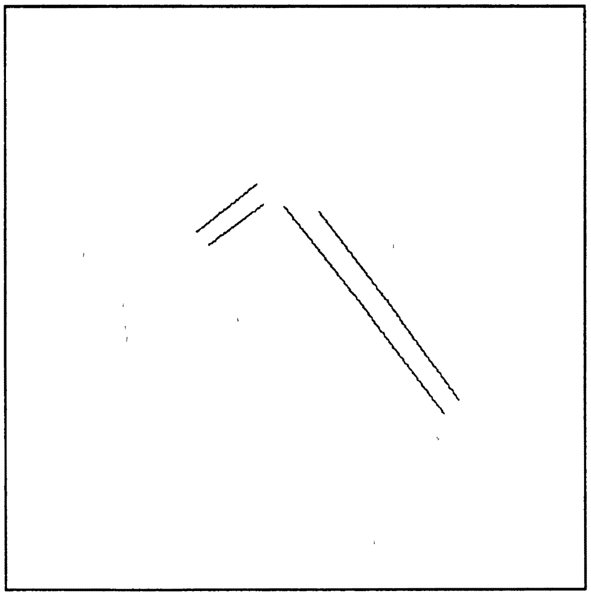


Figure V.7 Parallel-Lines of the Allen Wrench

Finally, corner-arcs are detected from the lists of corners and arcs. If any arcs are detected, then we search the corners with sharp angle. In this experiment we set a threshold value for the corner angle to 2.0 radians. So, if a corner angle is greater than 2.0, all pairs of the corner and the arcs become *corner-arcs*. Figure V.8 illustrates the corner-arcs detected from the adjustable wrench. The extended local feature extraction is also preformed for the input image in parallel. 67 corners, 6 arcs, 11 parallel-lines, and 36 corner-arcs are extracted. Refer to Table VII for the numbers of the features of the adjustable wrench, long nose plier, and Allen wrench.

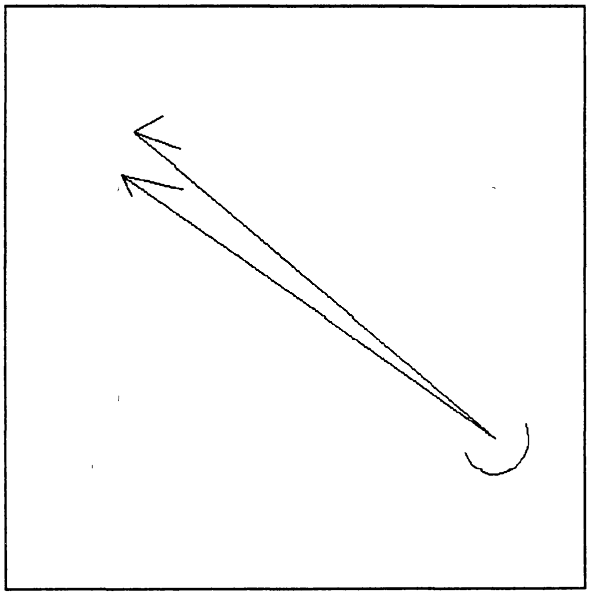


Figure V.8 Corner-Arcs of the Adjustable Wrench

Only for this preliminary experiment, let us assume that we have already known what objects are in the input image. So, we just try to match the model objects to the known objects in the input image by figuring out how much they are rotated and translated. Then we verify the matching by overlapping the transformed model objects onto the input image. For the matching, compatibilities between the features extracted from the model object and the features from the input image are checked in parallel (see Figure IV.5). If a model feature and an input feature are compatible with each other, we compute a coordinate transform (θ, t_x, t_y) from the compatible features by Equations (IV.20), (IV.21), and (IV.22). If the type of the features is a corner, we take 4 points each from the model and image features: $(x1, y1)$ and $(x2, y2)$ of $st1$, and $(x1, y1)$ and $(x2, y2)$ of $st2$. For an arc, we take 2 points each: (xs, ys) and (xe, ye). For a parallel-line, we take 4 points each: $(x1, y1)$ and $(x2, y2)$ of $st1$, and $(x1, y1)$ and $(x2, y2)$ of $st2$. For a corner-arc, we take 2 points each: (x, y) and (xc, yc). Refer to Figures (III.2), (III.5), (III.9), and (III.11).

Since the objects being used are rigid and not easily deformable, the coordinate transforms will be clustered if some model points are correctly mapped to the corresponding points of the input objects. We seek the cluster by the cluster-seeking algorithm (maximin-distance algorithm) described in Chapter IV, and the final coordinate transform for the object is given by the center of the cluster. The final coordinate transforms for the adjustable wrench, long nose plier, and Allen wrench are (0.459, 63.845, -92.071), (-1.348, -9.218, 433.584), and (0.607, 268.340, -80.677), respectively. The straight lines of the model object are rotated by θ and translated by (t_x, t_y), and the transformed straight lines are superimposed on the straight lines of the input image for the verification of the matching.

For comparison, we first use only corners as the feature. The adjustable wrench and the long nose plier are matched successfully, but the Allen wrench is not matched because no coordinate transforms are clustered. The lack of appropriate features causes the no-match of the Allen wrench. Next we use the extended local features. The Allen wrench is matched successfully now. Also the matching ratios of the adjustable wrench and the long nose plier are increased. Here the term *matching ratio* (R_m) was defined as

$$R_m = \frac{\text{Matched area}}{\text{Area of model object}} \times 100 \ (\%) \tag{V.2a}$$

or

$$R_m = \frac{\text{Number of pixels in matched area}}{\text{Number of pixels in model object}} \times 100 \ (\%) \tag{V.2b}$$

Table IV shows the comparisons of the matching ratios with only corners and with the extended local features. Figure V.9 (a), (b), and (c) show the verifications of the matchings of the adjustable wrench, long nose plier, and Allen wrench respectively.

We also compare the processing times for the feature extraction and matching with only corners and with the extended local features. The processing times are tabulated in Table V. Note that 16 nodes are allocated for this comparison. We can see that the processing times when using the extended local features are approximately 4 times larger than the processing times when using only corners.

TABLE IV

COMPARISONS OF MATCHING RATIOS

	With only corners	With extended local features
Adjustable wrench	95.7%	97.5%
Long nose plier	86.2%	87.6%
Allen wrench	No match	90.3%

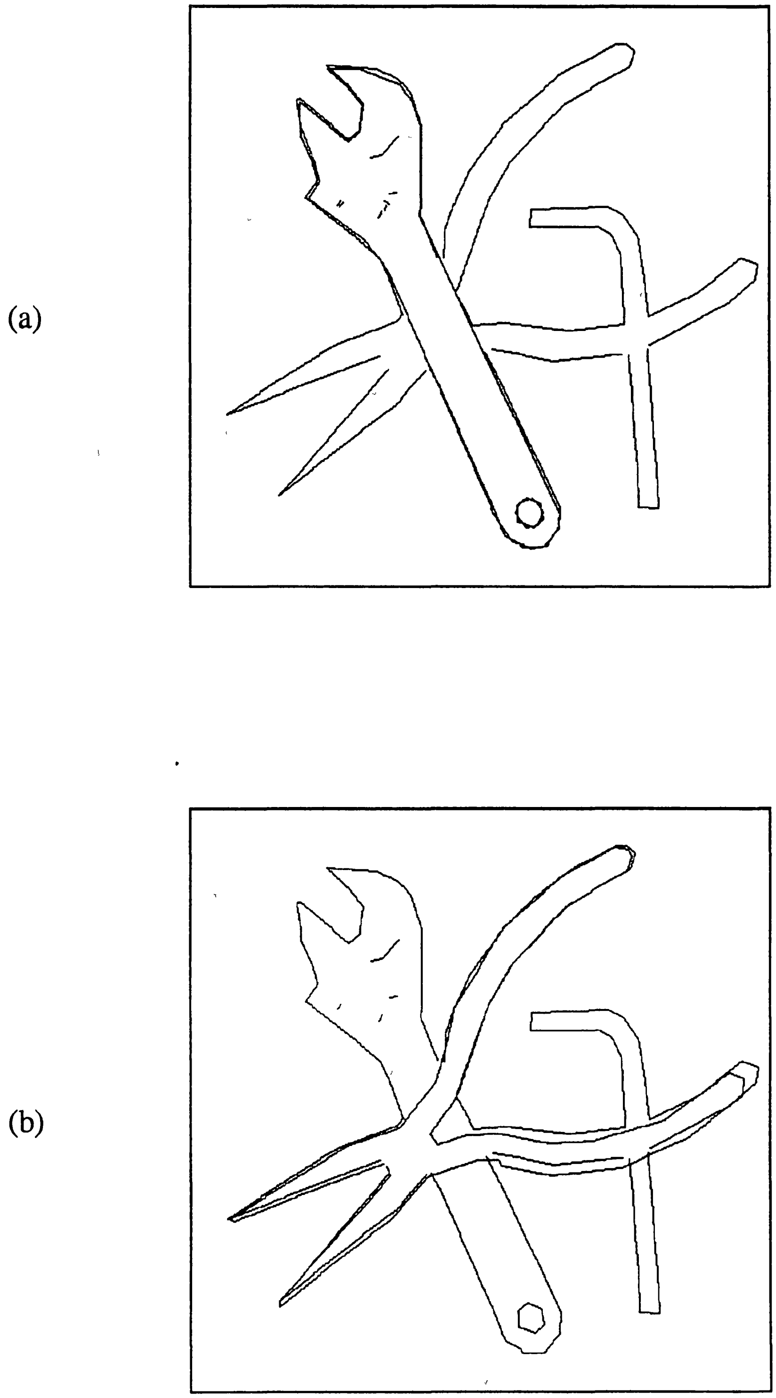
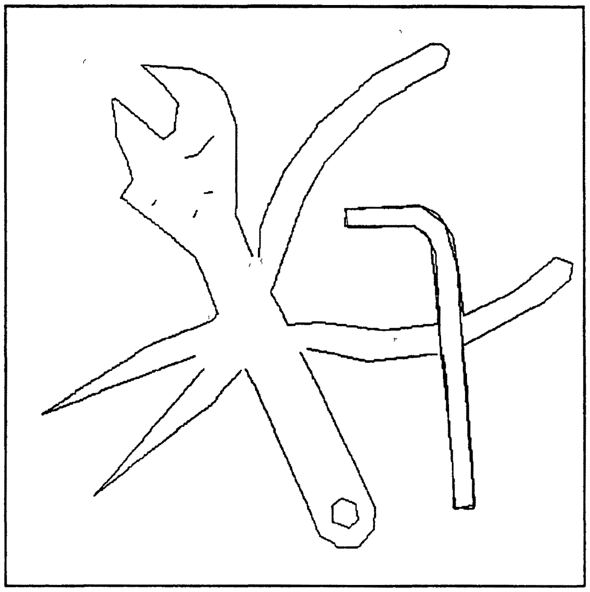


Figure V.9 Verification of the Matching. (a) Adjustable Wrench, (b) Long Nose Plier, (c) Allen Wrench

(c)



(Figure V.9 *Continued*)

TABLE V

COMPARISONS OF PROCESSING TIMES ON 16 NODES (IN SECONDS)

	With only corners	With extended local features
Adjustable wrench	0.42	1.39
Long nose plier	0.45	1.67
Allen wrench	0.21	0.76

Experiment 2 (Precision Test)

In this experiment we intend to test the precision of our system. In order to do this, we rotate and translate the straight lines of the combinational wrench shown in Figure V.10 (a) by $\theta = 1.0$ radians or 57.3 degrees, $t_x = 300$ pixels, and $t_y = -100$ pixels. The transformed straight lines of the combinational wrench is shown in Figure V.10 (b). Then we use the Figure V.10 (a) as a model and (b) as an input to be matched. After extracting the extended local features, we performed the matching.

35 pairs of the model and input features are compatible each. A coordinate transform is computed from each pair of the compatible features. Clusters are sought by the maximin-distance algorithm. The coordinate transforms are plotted in three-dimensional space (see Figure V.11). The results of the cluster-seeking procedure are provided in Table VI. 11 clusters are sought and the last cluster (cluster number 10)

(a)

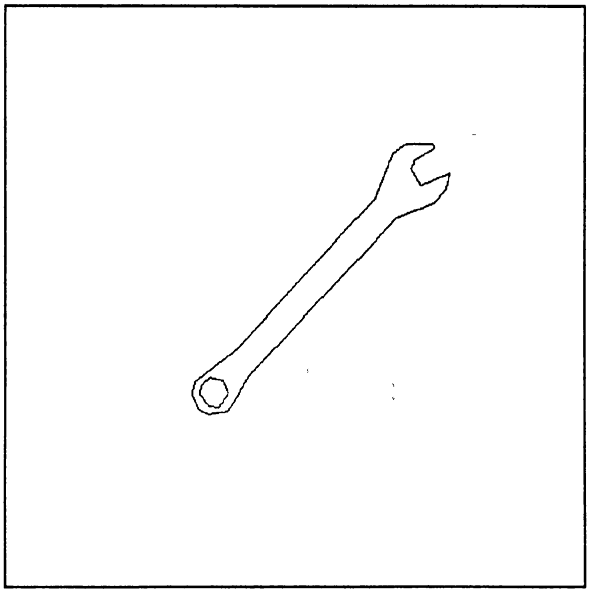
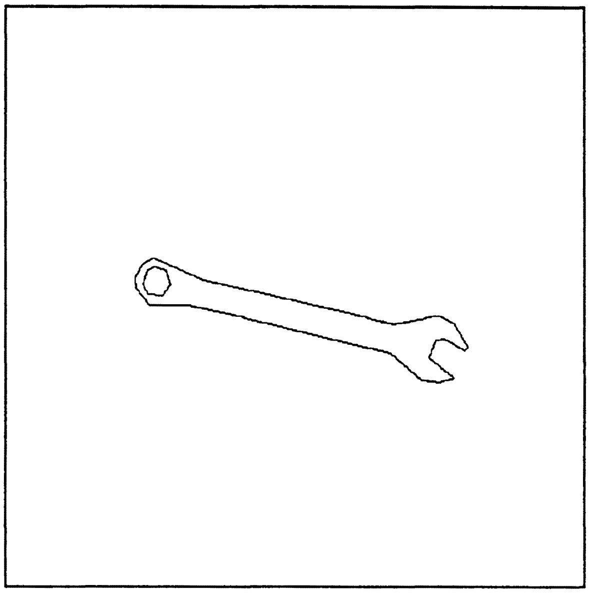


Figure V.10 Straight Lines of the Combinational Wrench for the Precision Test. (a) Original Straight Lines (model), (b) Transformed Straight Lines (input)

(b)



(Figure V.10 *Continued*)

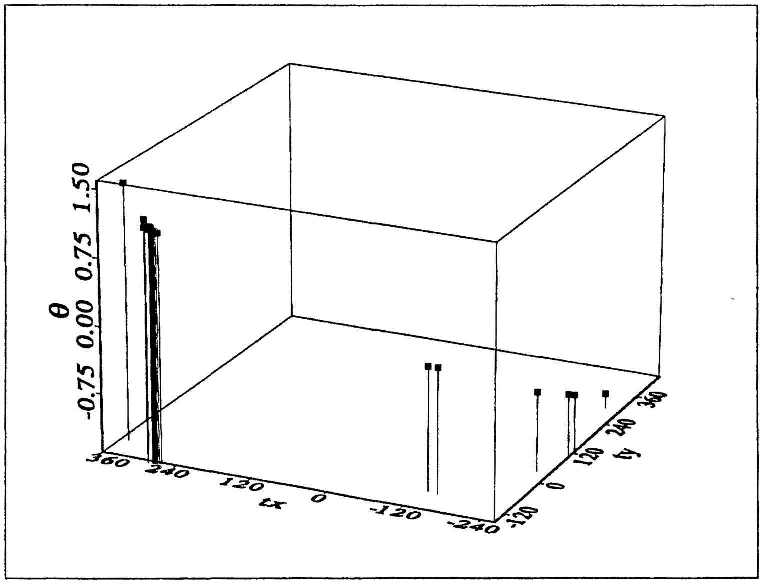


Figure V.11 A Three-Dimensional Plot of the Coordinate Transforms

TABLE VI

RESULTS OF THE CLUSTER SEEKING FOR THE PRECISION TEST

Data Number	θ	t_x	t_y	Cluster Number
1	-0.700196	-214.225357	99.327232	0
2	1.296954	380.323822	-38.560028	1
3	-1.429823	-210.693146	361.585510	2
4	-0.201902	-103.208069	-31.887503	3
5	0.992272	297.548035	-101.295464	4
6	0.991179	296.100647	-99.273170	4
7	0.980327	292.502777	-98.502090	4
8	-0.948638	-235.183762	187.503296	5
9	1.068916	321.799286	-87.157135	6
10	1.061309	319.188202	-89.056450	6
11	1.030517	312.360870	-102.683609	7
12	-0.903635	-232.449524	171.584518	8
13	-0.189480	-119.443680	-35.585964	9
14	1.017366	305.640411	-97.021362	10
15	0.996538	299.062988	-99.384239	10
16	0.998950	299.647919	-100.093613	10
17	0.999034	299.677856	-100.089813	10
18	1.025569	307.991455	-95.658203	10
19	1.009387	303.761871	-100.858192	10
20	1.011308	303.426422	-98.042068	10
21	1.002593	301.059723	-99.897430	10
22	1.005226	301.988953	-100.130119	10
23	1.001673	300.485901	-99.894585	10
24	0.996626	298.828918	-100.622444	10
25	1.000399	299.973755	-99.678383	10
26	1.019787	306.404999	-96.202690	10
27	0.996212	298.612335	-100.327347	10
28	1.011029	304.174316	-100.718086	10
29	0.997770	299.199829	-99.442734	10
30	1.013357	305.071564	-101.170715	10
31	1.001269	300.476074	-99.877029	10
32	1.015303	304.842896	-97.071106	10
33	0.999433	299.740112	-100.093460	10
34	1.001362	300.669128	-100.335258	10
35	1.015969	305.908875	-101.438835	10

has the largest number of coordinate transforms. The final coordinate transform is obtained by averaging the coordinate transforms in the cluster. The final coordinate transform is (1.006, 302.120, -99.457). Thus the differences between the desired coordinate transform and the actual coordinate transform are the followings:

Difference in θ = 0.006 radians or 0.344 degrees.

Difference in t_x = 2.120 pixels,

Difference in t_y = 0.543 pixels, and

To verify the matching, we rotate the input straight lines by 1.006 radians and translate by 302.120 pixels and -99.457 pixels in x and y directions. Figure V.12 shows the verification of the matching by superimposing the transformed input straight lines onto the original straight lines shown in Figure V.10 (a). We can see that the transformed combinational wrench is located and matched almost perfectly. Therefore, we can conclude that the precision of our system is very high for this test case.

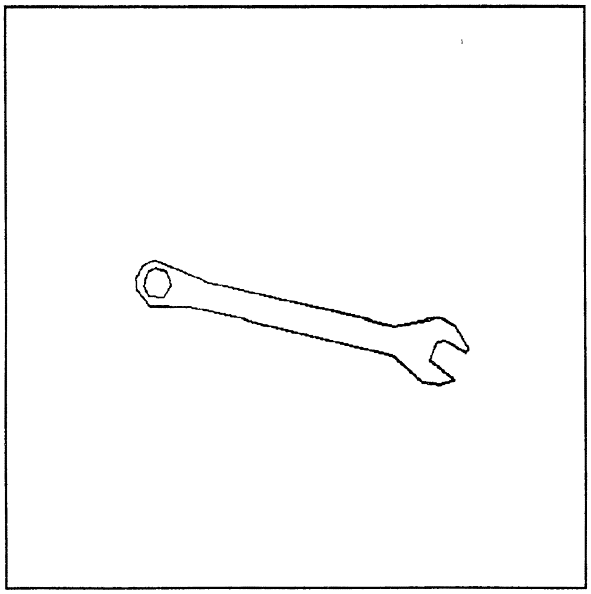


Figure V.12 Verification of the Matching for the Precision Test

Experiment 3 (Comprehensive Test)

In this experiment we test the system with all of the 10 model objects shown in Figure I.2. First of all, we extract the extended local features from the image of each model object, and then we model the object with the features. Each model consists of the name of the object, number of the features for each feature type, and feature data. The numbers of the features extracted from each model object image are summarized in Table VII.

During the modeling of the model objects, the knowledge-base is created or updated with the extended local features by hashing. We allocate the memory space for 997 addresses, which is a prime number. Each feature data is converted to a key represented as a 7 digit number using Equations (IV.2) to (IV.5) according to its feature type. For example, if *angle*, *st1.len*, and *st2.len* of a corner are -1.0, 200, and 100 respectively, the key will be

$$\begin{aligned} key &= 1000000 + 10000*\frac{(-1.0 + \pi)}{0.1} + 100*\frac{200}{10} + \frac{100}{10} \\ &= 1216169 \end{aligned}$$

A base address for the feature data is hashed by the Equation (IV.6). In the case of the previous example, the base address is 826. If the base address is empty, we insert the key there and add the name of the object into the object name list for the address. If the key to be inserted and the key in the base address are same, we search the object name list of the address. If no same object name is found, we add the object name into the list. On the other hand, if the two keys are different, that is, a collision has occurred, we use the rehashing functions given by Equation (IV.7) and (IV.8) until an available address is found. In this experiment 345 addresses are occupied. Let us define a term *packing density* as the ratio of the number of occupied addresses to the number of total addresses:

TABLE VII

NUMBER OF FEATURES

	Corner	Arc	Parallel-line	Corner-arc
Adjustable wrench	27	1	3	2
Long nose plier	38	8	7	14
Allen wrench	9	0	2	0
Combinational wrench	26	1	2	1
Diagonal plier	33	3	2	6
Offset screw-driver	17	2	3	4
Big screw-driver	20	1	3	1
Small screw-driver	13	1	2	1
Cutter	46	7	4	56
Knife	23	1	1	5

$$\textit{Packing density} = \frac{\text{Number of occupied addresses}}{\text{Number of total addresses}} \times 100\ (\%) \qquad \text{(V.3)}$$

In our case, the packing density is 34.6%. On the other hand, since the total number of features extracted from the 10 model objects is 396 (see Table V.4), the number of addresses with more than one object name is 51. Thus, the ratio of the addresses where multiple objects are assigned is 12.9%.

For input images, we try to generate synthetic images, in which some objects are occluded arbitrarily. In other words, we do not use the images taken by a CCD camera. Therefore, we do not have to generate the images of occluded objects by hand. Also we can easily create as many synthetic images as we want. Let's assume that the model objects are fairly dark and the background is light. Then we can extract the object areas from the model object images by thresholding. That is, if the gray level of a pixel is less than a threshold value, then we regard the pixel as one of pixels in the object area and we assign 0 (black) to that pixel. Otherwise, we assign 255 (white).

In order to make a synthetic image, we first determine the number of model objects to be used for the image by using a random number generator. In this experiment we set the lower and upper bounds of the random number generator to 3 and 5, respectively. Note that all of the objects in a synthetic image are unique. Now we extract the object areas from the images of the randomly chosen model objects. Then we rotate and translate the pixels in the object areas randomly by Equation (IV.26). Again, we used the random number generator for (θ, t_x, t_y). However, the transformed objects might be out of the bounds of the image plane. To avoid this problem, we transform only four corner points of the object: top-left, top-right, bottom-left, and bottom-right. If the four transformed corner points are inside of the bounds, we perform the transformation for all of the pixels. Otherwise, we generate another (θ, t_x, t_y) until the condition is satisfied. Now we overlap the transformed objects one after the other for making the synthetic occluded object image.

A term *occlusion ratio* (R_o) refers to the ratio of the occluded area of the object to the area of the object:

$$R_o = \frac{\text{Occluded area of the object}}{\text{Area of the object}} \times 100 \ (\%) \qquad \text{(V.4a)}$$

or

$$R_o = \frac{\text{Area of the object} - \text{Visible area of the object}}{\text{Area of the object}} \times 100\ (\%) \qquad \text{(V.4b)}$$

The visible area of the object can be obtained by subtracting all the other objects from the synthetic image.

One problem was found in the transformation of the pixels in the model object area: the pixels are not transformed linearly. In other words, more than one pixel could be transformed to the same position. This problem is caused by the truncation of the floating point representation to make integer coordinates in Equation (IV.26). Therefore, the number of pixels of the transformed object area is not identical to the number of pixels of the model object area. For example, we extract the object area of the long nose plier from the image shown in Figure V.13 (a). Then we rotate the object area by $\pi / 4$ radians and translate by -60 and 260 pixels in x and y directions. The transformed object is shown in Figure V.13 (a). We can see that the object area is not completely dark, which means some pixels in the object area are white (background). To solve this problem, we use a smoothing method: if a pixel is white, then check its 8 neighbors (see Figure II.11). If more than 4 neighbors are black, then convert the pixel to black. Figure V.13 (b) illustrates the results of the smoothing for the Figure V.13 (a). The following is pseudo code in C-like notation for the generation of the synthetic images:

```
for (i = 0; i < number_of_synthetic_images; i++) {
   choose_number_of_objects_randomly();
   for (j = 0; j < number_of_objects; j++) {
      while (choose_an_object() != UNIQUE)
         ;
      read_object_image();
      extract_object();
      detect_4_corner_points();
      while (transform(4_corner_points) == OUT_OF_BOUND) {
         choose_rot_tx_ty_randomly();
      }
      transform(object);
      smoothing(object);
      overlap(object, synthetic_image);
   }
   write_synthetic_image();
}
```

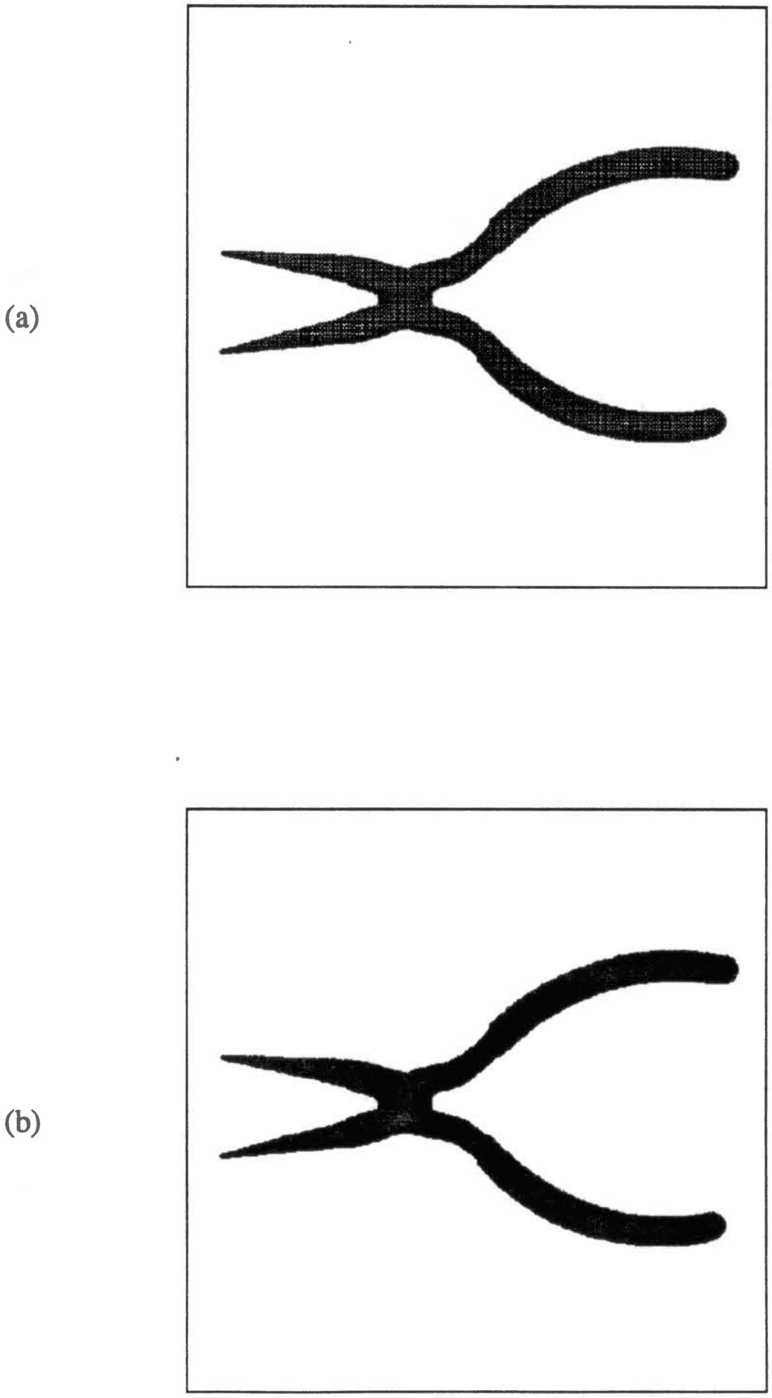


Figure V.13 An Example of Smoothing. (a) Before Smoothing, (b) After Smoothing

In this experiment we constructed 20 synthetic occluded object images and used them as the input images for the comprehensive test. The extended local features are extracted from the input image for the matching. Each feature data is converted to a key by one of the Equations (IV.2) to (IV.5) according to its feature type, and the key is sought from the knowledge-base by hashing. If the key is found, we hypothesize that the objects in the object name list of the key exist in the input image. The hypothesized objects are split evenly according to the number of nodes allocated, and each hypothesized object is matched in parallel using the compatibility test of the features, coordinate transform, and clustering. Then the matching is verified by the coarse and fine verification methods discussed in Chapter IV. In this experiment we do not try the only coordinate transform from the cluster with largest number of samples, but also we try the coordinate transforms from all the other clusters recursively to increase the matching rate.

The matching and verification are performed for the 20 synthetic images. One example is shown in Figure V.14. Figure V.14 (a) is a synthetic image in which 5 objects are occluding each other. The occlusion ratios of the adjustable wrench and big screwdriver are 26.3% and 48.1% respectively. On the other hand, the matching ratios of the adjustable wrench and big screwdriver are 97.1% and 77.8% respectively. The verifications of these matchings are shown in Figure V.14 (b) and (c). The big screwdriver is not matched because it is occluded heavily, so only a few features are extracted. The results of the comprehensive test for the 20 synthetic images are provided in Appendix C. The occlusion ratios range from 0.3% to 100% and the average occlusion ratio is 35.8%. The number of the matched objects (more than 80% in matching ratio) is 57 out of 74 objects. Thus overall matching rate is 77%. The status of the hypothesis for each object is also provided. 61 out of 74 objects (82.4%) are correctly hypothesized. Note that the previous example shown in Figure V.14 is about the synthetic image #4 in Appendix C.

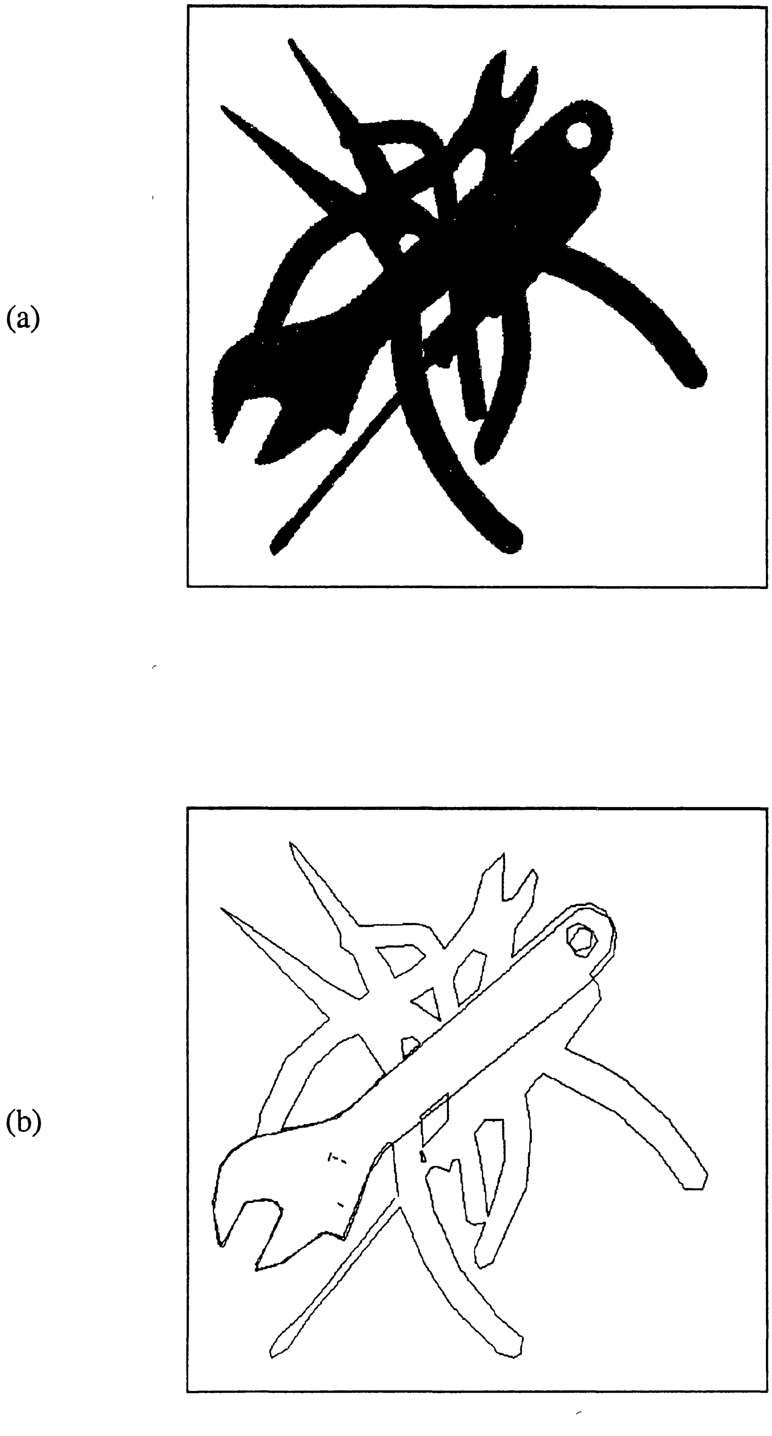
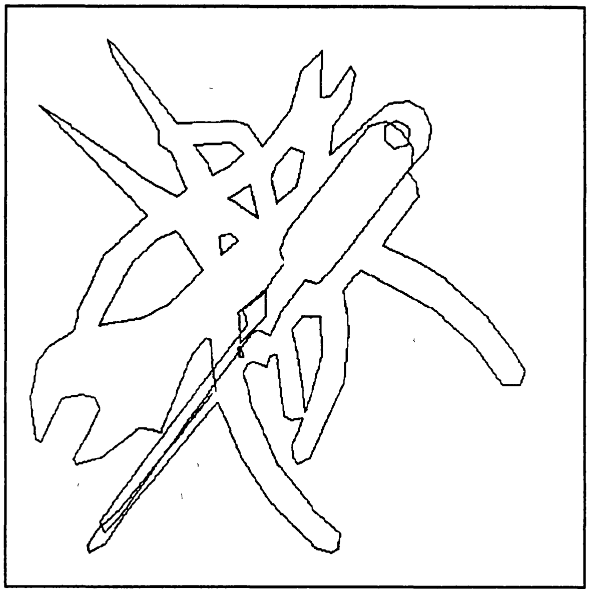


Figure V.14 An Example of Matching and Verification of a Synthetic Image. (a) A Synthetic Image, (b) Matching and Verification of the Adjustable Wrench, (c) Matching and Verification of the Big Screwdriver

(c)



(Figure V.14 *Continued*)

Table VIII shows the overall processing times for the example shown in Figure V.14 according to the different number of nodes. We can see that the processing times for the edge detection and straight line extraction are decreased approximately by a factor of 2 as the number of nodes is increased by a factor of 2. This means the speedup is almost linear. However, the processing times for the extended local feature extraction and the matching and verification are not so nearly linear. This is caused by the fact that communication times between nodes are increased as the number of nodes is increased.

TABLE VIII

OVERALL PROCESSING TIMES FOR THE EXAMPLE SHOWN IN FIGURE V.14 (IN SECONDS)

Procedures	Number of nodes			
	4	8	16	32
Edge detection and straight-line extraction	27.5	14.4	7.7	4.2
Extended local feature extraction	3.4	2.7	2.7	3.5
Matching and verification	3.5	3.1	2.9	1.8
Total	34.4	20.2	13.3	9.5

CHAPTER VI

SUMMARY AND CONCLUSIONS

In conclusion, a parallel occluded object recognition algorithm using extended local features and hashing has been proposed. The algorithm has been implemented and tested on a hypercube-topology multiprocessor computer, the Intel iPSC/2. The main goal of this research was to develop an accurate, reliable, and fast object recognition system.

In order to achieve the goal, we first proposed to use extended local features (corners, arcs, parallel-lines, and corner-arcs) to make our system more accurate and more reliable. We developed methods for extracting these extended local features from an image. An object was simply modeled by the extended local features. For hypothesizing and matching objects, compatible features between input and model objects were searched from the knowledge-base. Here we developed a scheme for constructing the knowledge-base using hashing, which can reduce the searching time significantly. Then we derived the coordinate transform from the compatible features, which brings the model features onto the image features.

Second, we developed new parallel algorithms for low-level image processing techniques such as edge detection, straight-line extraction, and thinning, because these steps are the most time-consuming in most existing object recognition systems. Then we implemented those algorithms on the iPSC/2 hypercube and analyzed their performance. We also increased the processing parallelism for the feature extracting and matching procedures and implemented them on the hypercube.

Finally, we tested the system with ten industrial tools and analyzed its performance. We performed three experiments: preliminary test, precision test, and comprehensive test.

In the preliminary test we used a simple occluded image where three objects were occluded arbitrarily. We implemented all of the procedures such as preprocessing, feature extraction, matching, and verification, and then verified their correct operation. In the precision test we translated and rotated an object in known directions. Then we matched the transformed object and analyzed the precision of the matching in terms of the pixels and angles. In the comprehensive test we tested our system with 20 synthetic images and collected statistical data such as the matching ratio, occlusion ratio, and overall matching rate. The experimental results show that our parallel occluded object recognition system using extended local features on the Intel iPSC/2 is fairly accurate, reliable, and fast. The results are summarized in the next section.

Summary of Results

The preliminary test given in Chapter V shows that our parallel preprocessing algorithms such as parallel edge detection using Laplacian of Gaussian operator and parallel straight-line extraction work very well. The extended local features of the corners, parallel-lines, arcs, and corner-arcs are extracted from the straight-lines successfully. From the matching procedure, all of the three known objects in the test image are successfully located, and the matching is verified. The comparison of the matchings with only corners and with extended local features reveals that the matching ratios and rate are improved when using the extended local features. However, the processing time when using extended local features is approximately 4 times as large as the processing time when using only corners.

The results of the precision test led us to believe that our system is very accurate. In that test, the average differences between the desired coordinate transform and the actual coordinate transform are only (0.006 radians, 2.120 pixels, 0.543 pixels). The transformed model is matched to the image almost perfectly (see Figure V.11). Also, it is shown that the clusters of the coordinate transforms are sought successfully by the maximin-distance cluster seeking algorithm in the precision test.

In the comprehensive test, the 10 model objects are modeled by the extended local features and a knowledge-base is constructed by the hashing scheme. The packing density is 34.6% when 997 addresses are used. For the comprehensive test, 20 synthetic images are constructed, in which 3 to 5 objects are overlapped with each other. The average occlusion ratio is 35.8%. From the hypothesizing procedure, 61 out of 74 objects are correctly hypothesized. Thus the true hypothesizing rate is 82.4%. On the other hand, 57 out of 74 objects are successfully located from the matching. So, the overall matching rate is 77%.

The processing times for the parallel preprocessing steps (see Table VIII) are approximately inversely proportional to the number of nodes. This result is very meaningful and desirable because of the fact that the hypercube architecture is easily scalable to many processors. In other words, for reasonably larger sized images, useful increases in performance can be realized for much larger order hypercubes than shown in this dissertation. However, the processing times for the feature extraction and matching are not decreased as the number of nodes is increased. This is caused by the fact that the communication times between nodes exceed the computing times when a large number of nodes is used.

Suggestions for Future Research

Although our system performs fairly well, there are some places to be studied further. Also, there are some possibilities for extending the ideas presented in this dissertation. Several points which could provide worthwhile results are listed below.

1. So far, our system works for only two-dimensional objects. In other words, relatively flat objects can be located and recognized. What about three-dimensional objects? Is it possible to use the extended local features for three-dimensional object recognition? This would seem to depend on how the objects are modeled. If the objects

are modeled by the features extracted from several views such that their shapes are consistent within some degree of the translation and rotation, for example, views from the top or bottom or sides, then the extended local features could be applicable because they are also consistent from the translation and rotation. The coordinate transform can be computed from the compatible features between the input and model objects. In this case, six parameters are required for the coordinate transform: translations in x, y, and z axes (t_x, t_y, t_z) and rotations about x, y, and z axes $(\theta_x, \theta_y, \theta_z)$. Now Equation (IV.13) can be modified as the following:

$$[I_{xi}\ I_{yi}\ I_{zi}] = [M_{xi}\ M_{yi}\ M_{zi}] \bullet R + [t_x\ t_y\ t_z] \tag{VI.1}$$

where R is a composition matrix of $R(\theta_x)$, $R(\theta_y)$, and $R(\theta_z)$. That is,

$$R = R(\theta_x) \bullet R(\theta_y) \bullet R(\theta_z) \tag{VI.2}$$

The $R(\theta_x)$, $R(\theta_y)$, and $R(\theta_z)$ are defined by the following matrices:

$$R(\theta_x) = \begin{bmatrix} \cos\theta_x & \sin\theta_x & 0 \\ -\sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{VI.3}$$

$$R(\theta_y) = \begin{bmatrix} \cos\theta_y & 0 & -\sin\theta_y \\ 0 & 1 & 0 \\ \sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \tag{VI.4}$$

$$R(\theta_z) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_z & \sin\theta_z \\ 0 & -\sin\theta_z & \cos\theta_z \end{bmatrix} \tag{VI.5}$$

Equation (VI.1) could be solved for the six parameters by a least square error method.

2. The extended local features consist of corners, arcs, parallel-lines, and corner-arcs. From our experiment, it was shown that the extended local features increase

the accuracy of the system as well as the overall matching rate of the system. Is there any possibility to add more feature types and further increase accuracy? One possibility is the features between the corners, that is, from each corner to the other corners. The distance between two corners and the angles of the corners are invariant to rotation and translation of the object. Thus compatible features between the model and input can be searched, and then coordinate transforms can be computed from the compatible features. In order to reduce the searching space, only the corners with sharp angles might be considered as the features.

3. MIMD machines can be classified into two types in terms of memory architectures: *distributed*-memory architectures and *shared*-memory architectures. The distributed memory architectures can be classified again in terms of the interconnection network topologies: *ring*-topology architectures, *mesh*-topology architectures, *hypercube*-topology architectures, *tree*-topology architectures, etc. [Dun90]. Our parallel algorithms for occluded object recognition were developed and implemented on the Intel iPSC/2, which belongs to the distributed-memory hypercube-topology architecture. Implementation on other multiprocessor systems would probably require modification of our parallel algorithms. However, it would be interesting to study the performance of the extended local feature method on a variety of multiprocessor architectures. Especially, the low-level image processing steps such as the edge detection, straight line extraction, and thinning might be interesting to implement on other multiprocessors because those steps are still time-consuming in our system, but a high degree of parallelism could be exploited from them.

4. Recently, many researchers have proposed to use neural networks, especially Hopfield-style neural network [Hop85] for the object matching or recognition phase in computer vision. Examples are [Leu88] and [Li89]. In [Leu88], a model and an input handwritten characters are thinned and approximated by straight segments. Features are simply represented by the corner angle of a pair of straight segments and the vector

joining mid-points of the segments. A compatibility function is defined to measure the degree of conformity with the relative positional relations between the input and model features. The compatibility function is mapped to the minimum energy function of a Hopfield neural network, and then the neural network is simulated to find the best matching segments. In [Li89], distinct features such as curvature points are extracted, and a graph consisting of a number of nodes connected by arcs is constructed. Object recognition is formulated as matching a model graph with an input image graph. A Hopfield binary network is implemented to perform a sub-graph isomorphism to obtain the optimal compatible matching features between graphs. The algorithm is extended to detect one object among several objects which could be touching or overlapping. Similarly, our occluded object recognition algorithm using extended local features could be solved by the neural network. The compatibility functions given by Equations (IV.9) through (IV.12) could be combined to form a new compatibility function. Then, the new compatibility function could be converted into the minimum energy function of a Hopfield neural network by rearranging the parameters. Better performance might result from using the extended local features.

REFERENCES

[Agi80] Agin, Gerald J., "Computer Vision Systems for Industrial Inspection and Assembly", *Computer*, Vol. 13, No. 5, May 1980, pp. 11-20.

[And73] Anderberg, Michael R., *Cluster Analysis for Applications*, Academic Press, New York, 1973.

[Aya83] Ayache, Nicholas, "A Model-Based Vision System to Identify and Locate Partially Visible Industrial Parts", *Proceedings of Computer Vision and Pattern Recognition*, June 19-23, 1983, pp. 492-494.

[Aya86] Ayache, Nicholas and Faugeras, Oliveier D., "HYPER: A New Approach for the Recognition and Positioning of Two-Dimensional Objects", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, No. 1, January 1986, pp. 44-54.

[Aya87] Ayache, Nicholas and Faverjon, Bernard, "Efficient Registration of Stereo Images by Matching Graph Descriptions of Edge Segments", *International Journal of Computer Vision*, 1987, pp. 107-131.

[Bae89-1] Baek, Joong H. and Teague, Keith A., "Parallel Edge Detection on the Hypercube", *Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, March 6-8, 1989, pp. 983-986.

[Bae89-2] Baek, Joong H. and Teague, Keith A., "Parallel Object Representation Using Straight Lines on the Hypercube Multiprocessor Computer", *Proceedings of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications*, March 6-8, 1989, pp. 987-990.

[Bae90] Baek, Joong H. and Teague, Keith A., "Parallel Thinning on a Distributed Memory Machine", *Proceedings of the Fifth Distributed Memory Computing Conference*, April 8-12, 1990. pp. 72-75.

[Bai78] Baird, M. L., "Sight I: A Computer Vision System for Automated IC Chip Manufacture", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 8, No. 2, February 1978, pp. 133-139.

[Bal81] Ballard, D. H., "Generalizing the Hough Transform to Detect Arbitrary Shapes", *Pattern Recognition*, Vol. 13, 1981, pp. 111-122.

[Bes85] Besl, Paul J. and Jain, Ramesh C., "Three-Dimensional Object Recognition", *Computing Surveys*, Vol. 17, No. 1, pp. 75-145, March 1985.

[Bha86] Bhanu, Bir and Ming, John C., "Clustering Based Recognition of Occluded Objects", *Proceedings of the 8th International Conference on Pattern Recognition*, 1986, pp. 732-734.

[Bha87] Bhanu, Bir and Ming, John C., "Recognition of Occluded Objects: A Cluster-Structure Algorithm", *Pattern Recognition*, Vol. 20, No. 2, 1987, pp. 199-211.

[Bir81] Birk, J. R., Kelley, R. B., and Martins, H., "An Orienting Robot for Feeding Workpieces Stored in Bins", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 11, No. 2, February 1981, pp. 151-160.

[Bol82] Bolles, Robert C. and Cain, Ronald A., "Recognizing and Locating Partially Visible Objects: The Local-Feature-Focus Method", *The International Journal of Robotics Research*, Vol. 1, No. 3, Fall 1982, pp. 57-82.

[Bor84] Borgefors, G., "Distance transformations in arbitrary dimensions", *Computer Vision, Graphics, and Image Processing*, Vol. 27, 1984, pp. 321-345.

[Bre65] Bresenham, J. E., "Algorithm for Computer Control of a Digital Plotter", *IBM Systems Journal*, Vol. 4, No. 1., 1965, pp. 25-30.

[Che82] Chen, M. J. and Milgram, D. L., "A development System for Machine Vision", *Proceedings of the IEEE Computer Society Conference on Pattern Recognition and Image Processing*, 1982, pp. 512-517.

[Chi86] Chin, Roland T. and Dyer, Charles R., "Model-Based Recognition in Robot Vision", *Computing Surveys*, Vol. 18, No. 1, March 1986, pp. 67-108.

[Chi87] Chin, Roland T. and Wan, Hong-Khoon, "A One-Pass Thinning Algorithm and Its Parallel Implementation", *Computer Vision, Graphics, and Image Processing*, Vol.40, 1987, pp. 30-40.

[Dan70] Danielsson, Per E., "Incremental Curve Generation", *IEEE Transactions on Computers*, Vol. C-19, No. 9, September 1970, pp. 783-793.

[Dem84] Demel, John T. and Miller, Michael J., *Introduction to Computer Graphics*, Wadsworth, Inc., Belmont, 1984.

[Dud73] Duda, Richard O. and Hart, Peter E., *Pattern Classification and Scene Analysis*, A Wiley-Interscience Publication, New York, 1973.

[Dun90] Duncan, Ralph, "A Survey of Parallel Computer Architectures", *Computer*, February 1990, pp. 5-16.

[Fol87] Folk, Michael J. and Zoellick, Bill, *File Structures*, Addison-Wesley Publishing Company, Massachusetts, 1987.

[Ful67] Fuller, E. Leonard and Bechtel, D. Robert, *Introduction to Matrix Algebra*, Dickenson Publishing Company, Inc., Belmont, California, 1967.

[Ger65] Gere, James M. and William Weaver, Jr., *Matrix Algebra for Engineers*, D. Van Nostrand Company, Inc., Princeton, New Jersey, 1965.

[Gle79] Gleason, G. J. and Agin, G. J., "A Modular Vision System for Sensor-Controlled Manipulation and Inspection", *Proceedings of 9th International Symposium on Industrial Robots*, 1979, pp. 57-70.

[Gon83] Gonzalez, Rafael C. and Wintz, Paul, *Digital Image Processing*, Addison-Wesley Publishing Company, Massachusetts, 1983.

[Gri85] Grimson, W. E. L. and Hildreth, E. C., "Comments on Digital Step Edges from Zero Crossing of Second Directional Derivatives", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-7, No. 1, January, 1985, pp. 121-127.

[Gri88] Grimson, W. Eric L., "On the Recognition of Parameterized 2D Objects", *International Journal of Computer Vision*, Vol. 3, 1988, pp. 353-372.

[Guo89] Guo, Zicheng and Hall, Richard W., "Parallel Thinning with Two-Subiteration Algorithms", *Communications of the ACM*, Vol. 32, No. 3, March, 1989, pp. 359-373.

[Hal89] Hall, Richard W. "Fast Parallel Thinning Algorithms: Parallel Speed and Connectivity Preservation", *Communications of the ACM*, Vol. 32, No. 1, January, 1989, pp. 124-131.

[Han89] Han, Min-Hong, Jang, Dongsig, and Foster, Joseph, "Inspection of 2-D Objects Using Pattern Matching Method", *Pattern Recognition*, Vol. 22, No. 5, 1989, pp. 567-575.

[Hat83] Hattich, W., "Recognition of Overlapping Workpieces by Model-Directed Construction of Object Contours", *Artificial Vision for Robots*, Chapman & Hall, New York, pp. 77-92.

[Hay86] Hayes, J. P., et al, "Architecture of a Hypercube Supercomputer", *Proceedings of International Conference on Parallel Processing*, August, 1986.

[Hol87] Holt, Christopher M., Stewart, Alan, Clint, Maurice, and Perrott, Ronald H., "An Improved Parallel Thinning Algorithm", *Communications of the ACM*, Vol. 30, No. 2, February 1987, pp. 156-160.

[Hop85] Hopfield, J. J. and Tank, D. W., "Neural Computations of Decisions in Optimization Problems", *Biological Cybernetics*, Vol. 52, July 1985, pp. 141-152.

[Hu62] Hu, Ming-Kuei, "Visual Pattern Recognition by Moment Invariants", *IRE Transactions on Information Theory*, Vol. 8, Feburary 1962, pp. 197-219.

[Int86] Intel Corporation, *iPSC Technical Description*, 1986.

[Jac80] Jackins, C. L. and Tanimoto, S. L., "Oct-trees and Their Use in Representing 3D Objects", *Computer Graphics and Image Processing*, Vol. 14, 1980, pp. 249-270.

[Jor73] Jordan, Bernard W., Lennon, William J., and Holm, Barry D., "An Improved Algorithm for the Generation of Nonparametric Curves", *IEEE Transactions on Computers*, Vol. C-22, No. 12, December 1973, pp. 1052-1060.

[Kel82] Kelley, R. B., Birk, J. R., Martins, H. A. S., and Tella, R., "A Robot System Which Acquires Cylindrical Workpieces from Bins", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 12, No. 2, March/April 1982, pp. 204-213.

[Kim87] Kim, Yeon C. and Aggarwal, J. K., "Positioning Three-Dimensional Objects Using Streo Images", *Joural of Robotics and Automation*, Vol. RA-3, No. 4, August 1987, pp. 361-373.

[Kno86] Knoll, Thomas F. and Jain, Ramesh C., "Recognizing Partially Visible Objects Using Feature Indexed Hypothesis", *IEEE Journal of Robotics and Automation*, Vol. RA-2, No. 1, March 1986, pp. 3-13.

[Koc85] Koch, Mark W. and Kashyap, R. L., "A Vision System to Identify Occluded Industrial Parts", *IEEE International Conference on Robotics and Automation*, 1985, pp. 55-60.

[Koc87] Koch, Mark W. and Kashyap, Rangasami L., "Using Polygons to Recognize and Locate Partially Occluded Objects", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-9, No. 4, July 1987, pp. 483-494.

[Kue85] Kuehn, James T., Fessler, Jeffrey A. and Siegel, Howard Jay, "Parallel Image Thinning and Vectorization on PASM", *Proceedings of CVPR '85: IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1985, pp. 368-374.

[Kul90] Kulkarni, A. D., Yap, Al. C., and Byars, P., "Neural Networks for Invariant Object Recognition", *Proceedings of the 1990 Symposium on Applied Computing*, 1990, pp. 28-32.

[Leu88] Leung, C. H., "Structural Matching Using Neural Networks", *Abstracts of the First Annual Internaltional Neural Network Society Meeting*, Vol. 1, 1988, pp. 31.

[Li89] Li, Wei and Nasrabadi, Nasser M., "Object Recognition Based on Graph Matching Implemented by a Hopfield-Style Neural Network", *Proceedings of International Joint Conference on Neural Network*, Vol. II, 1989, pp. 287-290.

[Low87] Lowe, David G., "Three-Dimensional Object Recognition from Single Two-Dimensional Images", *Artificial Intelligence*, Vol. 31, 1987, pp. 355-395.

[Lu86] Lu, H. E. and Wang, P. S. P., "A Comment on a Fast Parallel Algorithm for Thinning Digital Patterns", *Communications of the ACM*, Vol. 29, No. 3, March 1986, pp. 239-242.

[Lun86] Lunscher, W. H. H. J. and Beddoes, M. P., "Optimal Edge Detector Evaluation", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-16, No. 2, March/April 1986, pp. 304-312.

[Mar80] Marr, D. and Hildreth, E., "Theory of Edge Detection", *Proceedings of Royal Society of London*, B, Vol. 207, 1980, pp. 187-217.

[Mar87] Martinerz-Perez, M. Pilar, "A Thinning Algorithm Based on Contours", *Computer Vision, Graphics, and Image Processing*, Vol. 39, 1987, pp. 186-201.

[McI88] McIntosh, James H., "Matching Straight Lines", *Computer Vision, Graphics, and Image Processing*, Vol. 42, 1988, pp. 386-408.

[Mea81] Meagher, D. J., "Geometric Modeling Using Octree Encoding", *Computer Graphics and Image Processing*, Vol. 19, pp. 129-147.

[Meh90] Mehrotra, Rajiv, Kung, Fu K., and Grosky, William I., "Industrial Part Recognition Using a Component-Index", *Image and Vision Computing*, Vol. 8, No. 3, August 1990, pp. 225-232.

[Mud87] Mudge, T. N. and Abdel-Rahman, T. S., "Vision Algorithms for Hypercube Machines", *Journal of Parallel and Distributed Computing*, Vol. 4, 1987, pp.79-94.

[Nug88] Nugent, Steven F., "The iPSC/2 Direct-Connect Communications Technology", *Concurrent Supercomputing*, Intel Corporation, 1988, pp. 59-68.

[Off85] Offen, R. J. *VLSI Image Processing*, McGraw-Hill Book Company, New York, 1985.

[Per77] Persoon, Eric and Fu, King-Sun, "Shape Discrimination Using Fourier Descriptors", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-7, No. 3, March 1977, pp. 170-179.

[Per78] Perkins, W. A., "A Model-Based Vision System for Industrial Parts", *IEEE Transaction on Computers*, Vol. C-27, No. 2, February 1978, pp. 126-143.

[Pra84] Pratt, William K., *Digital Image Processing*, John Wiley & Sons, Inc., New York, 1984.

[Ros82] Rosenfeld, Azriel and Kak, Avinash C., *Digital Picture Processing*, Academic Press, New York, 1982.

[Ros89] Rosin, Paul L. and West, Geoff A. W., "Segmentation of edges into lines and arcs", *Image and Vision Computing*, Vol. 7, No. 2, May 1989, pp. 109-114.

[Rum84] Rummel, P. and Beutel, W., "Workpiece Recognition and Inspection by a Model-Based Scene Analysis System", *Pattern Recognition*, Vol. 17, No. 1, 1984, pp. 141-148.

[Shi87] Shirai, Yoshiaki, *Three-Dimensional Computer Vision*, Springer-Verlag, New York, 1987.

[Smi87] Smith, David and Myers, Donald, "Edge Detection in Tactile Images", *IEEE 1987 International Conference on Robotics & Automation*, Vol. 3, 1987, pp. 1500-1505.

[Sob72] Sobel, V. A., "Device for Analyzing the Structure of Complex Contour Images", *Automatic Remote Control*, Vol. 33, 1972, pp. 775-779.

[Sto82] Stockman, G. C., Kopstein, K., and Benett, S., "Matching Images to Models for Registration and Object Detection Via Clustering", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-4, No. 3, May 1982, pp. 229-241.

[Sue79] Suenaga, Yasuhito, Kamae, Takahiko, and Kobayashi, Tomonori, "A High-Speed Algorithm for the Generation of Straight Lines and Circular Arcs", *IEEE Transactions on Computers*, Vol. C-28, No. 10, October 1979, pp. 728-736.

[Tor86] Torre, Vincent and Poggio, Tomaso A., "On Edge Detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, No. 2, March 1986, pp. 147-163.

[Tou74] Tou, J. T. and Gonzalez, R. C., *Pattern Recognition Principles*, Addison-Wesley Publishing Company, Massachusetts, 1974.

[Tre84] Tremblay, Jean-Paul and Sorenson, Paul G., *An Introduction to Data Structures with Applications*, McGraw-Hill, Inc., New York, 1984.

[Tro80] Tropf, H., "Analysis-by-Synthesis Search for Semantic Segmentation, Applied to Workpiece Recognition", *Proceedings of the 5th International Conference on Pattern Recognition*, Miami, December 1980, pp. 241-244.

[Tur85] Turney, Jerry L., Mudge, Trevor N., and Volz, R. A., " Recognizing partially occluded parts", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-7, July 1985, pp. 410-421.

[Vit87] Vitter, Jeffrey Scott and Chen, Wen-Chin, *Design and Analysis of Coalesced Hashing*, Oxford University Press, New York, 1987.

[Vli89] Vliet, Lucas J. Van and Young, Ian T., "A Nonlinear Laplace Operator as Edge Detector in Noisy Image", *Computer Vision, Graphics, and Image Processing*, Vol. 45, 1989, pp. 167-195.

[Wal80] Wallace, T. P. and Wintz, P. A., "An Efficient Three-Dimensional Aircraft Recognition Algorithm Using Normalized Fourier Descriptors", *Computer Graphics and Image Processing*, Vol. 13, 1980, pp. 99-126.

[Wal87] Wallace, A. M., "Matching Segmented Scenes to Models Using Pairwise Relationships Between Features", *Image and Vision Computing*, 1987, pp. 114-120.

[Wec88] Wechsler, Harry and Zimmerman, George Lee, "2-D Invariant Object Recognition Using Distributed Associative Memory", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10, No. 6, November 1988, pp. 811-821.

[Wec89] Wechsler, Harry and Zimmerman, George Lee, "Distributed Associative Memory (DAM) for Bin-Picking", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 8, August 1989, pp. 814-822.

[Zha84] Zhang, T. Y. and Suen, C. Y., "A Fast Parallel Algorithms for Thinning Digital Patterns", *Communications of the ACM*, Vol. 27, No. 3, March 1984, pp. 236-239.

APPENDIXES

APPENDIX A

DERIVATION OF EQUATIONS (III.5), (III.6), AND (III.7)

The error function to be minimized is

$$J = \sum_{i=1}^{n} [(X_i - X_c)^2 + (Y_i - Y_c)^2 - R^2]^2 \tag{A.1}$$

From the condition

$$\frac{\partial J}{\partial X_c} = 0,$$

we can obtain

$$\frac{\partial J}{\partial X_c} = 2 \sum_{i=1}^{n} [(X_i - X_c)^2 + (Y_i - Y_c)^2 - R^2][-2(X_i - X_c)] = 0 \tag{A.2}$$

The above equation can be rewritten by

$$\sum_{i=1}^{n} [(X_i - X_c)^2 + (Y_i - Y_c)^2 - R^2](X_i - X_c) = 0 \tag{A.3}$$

By solving the above equation, we can derive Equation (III.5).

From the condition

$$\frac{\partial J}{\partial Y_c} = 0,$$

we can obtain

$$\frac{\partial J}{\partial Y_c} = 2\sum_{i=1}^{n}[(X_i - X_c)^2 + (Y_i - Y_c)^2 - R^2][-2(Y_i - Y_c)] = 0 \tag{A.4}$$

The above equation can be rewritten by

$$\sum_{i=1}^{n}[(X_i - X_c)^2 + (Y_i - Y_c)^2 - R^2](Y_i - Y_c) = 0 \tag{A.5}$$

Solving the above equation gives us Equation (III.6).

From the condition

$$\frac{\partial J}{\partial R} = 0,$$

we can obtain

$$\frac{\partial J}{\partial R} = 2\sum_{i=1}^{n}[(X_i - X_c)^2 + (Y_i - Y_c)^2 - R^2](-2R) = 0 \tag{A.6}$$

The above equation can be rewritten by

$$\sum_{i=1}^{n}[(X_i - X_c)^2 + (Y_i - Y_c)^2 - R^2] = 0 \tag{A.7}$$

or

$$\sum_{i=1}^{n}(X_i - X_c)^2 + \sum_{i=1}^{n}(Y_i - Y_c)^2 - \sum_{i=1}^{n}R^2 = 0 \tag{A.8}$$

Since

$$\sum_{i=1}^{n}R^2 = nR^2,$$

we can obtain Equation (III.7) from (A.8).

APPENDIX B

DERIVATION OF EQUATIONS (IV.20), (IV.21), AND (IV.22)

From Equation (IV.18),

$$Q = \sum_{i=1}^{n} (I_{xi} - M_{xi}\cos\theta + M_{yi}\sin\theta - t_x)^2 + \sum_{i=1}^{n} (I_{yi} - M_{xi}\sin\theta - M_{yi}\cos\theta - t_y)^2 \tag{B.1}$$

From the condition

$$\frac{\partial Q}{\partial t_x} = 0,$$

we can obtain

$$\frac{\partial Q}{\partial t_x} = -2\sum_{i=1}^{n} (I_{xi} - M_{xi}\cos\theta + M_{yi}\sin\theta - t_x) = 0 \tag{B.2}$$

Solving the above equation for t_x, we can drive Equation (IV.20).

From the condition

$$\frac{\partial Q}{\partial t_y} = 0,$$

we can get

$$\frac{\partial Q}{\partial t_y} = -2\sum_{i=1}^{n}(I_{yi} - M_{xi}\sin\theta - M_{yi}\cos\theta - t_y) = 0 \tag{B.3}$$

We can obtain Equation (IV.21) by solving the above equation for t_y.

From the condition

$$\frac{\partial Q}{\partial \theta} = 0,$$

we can get

$$\begin{aligned}\frac{\partial Q}{\partial \theta} &= 2\sum_{i=1}^{n}(I_{xi} - M_{xi}\cos\theta + M_{yi}\sin\theta - t_x)(M_{xi}\sin\theta + M_{yi}\cos\theta) + \\ &\quad 2\sum_{i=1}^{n}(I_{yi} - M_{xi}\sin\theta - M_{yi}\cos\theta - t_y)(-M_{xi}\cos\theta + M_{yi}\sin\theta) \\ &= 2\sum_{i=1}^{n}(I_{xi} - t_x)(M_{xi}\sin\theta + M_{yi}\cos\theta) + \\ &\quad 2\sum_{i=1}^{n}(I_{yi} - t_y)(-M_{xi}\cos\theta + M_{yi}\sin\theta) = 0\end{aligned} \tag{B.4}$$

Solving the above equation gives us

$$\begin{aligned}&n\sin\theta\left(\sum_{i=1}^{n} I_{xi}M_{xi} - t_x\sum_{i=1}^{n} M_{xi} + \sum_{i=1}^{n} I_{yi}M_{yi} - t_y\sum_{i=1}^{n} M_{yi}\right) + \\ &\quad n\cos\theta\left(\sum_{i=1}^{n} I_{xi}M_{yi} - t_x\sum_{i=1}^{n} M_{yi} - \sum_{i=1}^{n} I_{yi}M_{xi} + t_y\sum_{i=1}^{n} M_{xi}\right) = 0\end{aligned} \tag{B.5}$$

By substituting Equations (IV.20) and (IV.21) for t_x and t_y in the above equation, respectively, we can get Equation (IV.22).

APPENDIX C

RESULTING DATA OF EXPERIMENT 3

Synthetic Image	Object Name	Hypothesized	Matched	Matching Ratio (%)	Occlusion Ratio (%)
Image 1	knif	YES	YES	99.2	52.7
	lopl	YES	YES	98.2	47.8
	adwr	YES	YES	92.9	38.5
	cowr	YES	NO	0.0	100.0
Image 2	cutt	YES	YES	98.4	33.8
	smsc	YES	NO	0.0	10.8
	dipl	YES	YES	98.3	33.5
	bisc	YES	YES	96.3	41.1
Image 3	lopl	YES	YES	91.9	16.9
	dipl	YES	YES	92.6	16.4
	alwr	NO	NO	0.0	33.8
Image 4	lopl	NO	NO	0.0	28.9
	bisc	YES	NO	77.8	48.1
	alwr	YES	YES	86.0	45.3
	adwr	YES	YES	97.1	26.3
	dipl	YES	YES	92.7	32.1
Image 5	smsc	NO	NO	0.0	67.2
	lopl	YES	YES	95.7	18.4
	adwr	YES	YES	96.2	24.4
Image 6	adwr	NO	NO	0.0	41.6
	knif	YES	YES	95.1	34.3
	ofsc	YES	YES	93.0	13.0
Image 7	bisc	YES	YES	86.6	49.9
	cutt	YES	YES	99.4	46.3
	adwr	YES	YES	96.7	25.8
	ofsc	YES	YES	88.4	37.6
Image 8	bisc	YES	YES	96.8	36.5
	dipl	YES	YES	95.6	31.3
	knif	YES	YES	93.4	27.5

Synthetic Image	Object Name	Hypothesized	Matched	Matching Ratio (%)	Occlusion Ratio (%)
Image 9	lopl	YES	YES	94.2	39.4
	knif	NO	NO	0.0	36.9
	adwr	YES	YES	94.7	43.2
Image 10	knif	YES	YES	96.9	23.5
	smsc	NO	NO	63.0	27.3
	dipl	YES	YES	99.1	25.3
	alwr	NO	NO	0.0	38.7
Image 11	ofsc	YES	YES	88.1	16.5
	smsc	YES	YES	97.7	37.3
	alwr	YES	YES	96.3	23.5
	lopl	YES	YES	95.0	16.8
Image 12	lopl	NO	NO	56.0	52.7
	knif	YES	YES	99.3	46.4
	dipl	YES	YES	90.9	29.0
Image 13	lopl	YES	YES	96.7	16.2
	adwr	YES	YES	93.9	24.4
	smsc	NO	NO	0.0	61.3
Image 14	ofsc	YES	YES	90.8	41.8
	cowr	YES	YES	88.0	37.3
	dipl	NO	NO	72.8	41.7
	adwr	YES	YES	94.8	44.5
	lopl	YES	YES	93.8	24.4
Image 15	ofsc	YES	YES	96.6	8.7
	cowr	YES	YES	92.1	0.3
	cutt	YES	YES	98.4	2.4
Image 16	knif	YES	YES	99.3	15.5
	smsc	YES	YES	95.9	50.4
	alwr	YES	YES	98.1	18.7
Image 17	cutt	YES	YES	80.0	60.7
	cowr	YES	YES	96.1	73.4
	adwr	YES	YES	95.5	44.2
	lopl	YES	YES	93.9	48.3
	dipl	YES	YES	83.3	69.6
Image 18	dipl	YES	YES	93.9	54.2
	cowr	YES	YES	95.1	14.7
	bisc	YES	YES	97.1	32.8
	knif	YES	YES	99.7	35.3
Image 19	smsc	NO	NO	0.0	35.5
	ofsc	YES	YES	91.2	12.9
	bisc	YES	NO	0.0	18.3

Synthetic Image	Object Name	Hypothesized	Matched	Matching Ratio (%)	Occlusion Ratio (%)
Image 20	smsc	NO	NO	0.0	55.5
	cutt	YES	YES	98.9	47.1
	dipl	YES	YES	97.5	45.3
	adwr	YES	YES	88.5	47.9
	lopl	NO	NO	79.6	48.7

NOTE: alwr = Allen wrench; lopl = Long nose plier; adwr = adjustable wrench; dipl = Diagonal plier; smsc = Small screwdriver; ofsc = Offset screwdriver; cutt = Cutter; cowr = Combinational wrench; knif = Knife; bisc = Big screwdriver

VITA

Joong Hwan Baek

Candidate for the Degree of

Doctor of Philosophy

Thesis: PARALLEL OCCLUDED OBJECT RECOGNITION USING EXTENDED LOCAL FEATURES ON THE HYPERCUBE MULTIPROCESSOR COMPUTER

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Choong-Nam-Do, Korea, June 28, 1959, the son of Nam Soon Baek and In Sook Kim.

Education: Graduated from Bo-Moon High School, Dae-Jeon, Korea, in January, 1977; received the Bachelor of Science degree in Telecommunication Engineering from Hankuk Aviation College in February, 1981; received the Master of Science degree in Electrical Engineering from Oklahoma State University in July, 1987; completed the requirements for the Doctor of Philosophy degree at Oklahoma State University in July, 1991.

Professional Experience: Telecommunication Cable Network Designer, Dong-Ah Engineering Company Ltd., Seoul, Korea, June, 1981, to October, 1984; Graduate Research or Teaching Assistant, School of Electrical and Computer Engineering, Oklahoma State University, August, 1987, to July, 1991.