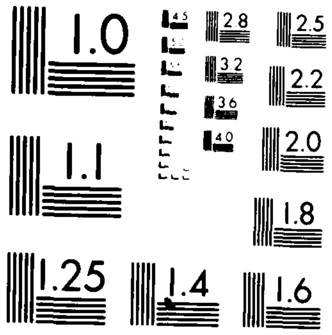


MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

02

A RULE-BASED PATTERN MATCHING SYSTEM FOR THE RECOGNITION OF THREE-DIMENSIONAL LINE DRAWN OBJECTS:- A FOUNDATION FOR VIDEO TRACKING

Submitted by

Y.S. Fong

Assistant Professor

Electrical and Computer Engineering

Clarkson University

Potsdam, NY

NRL Contract No: N00014-85-C-2421

ABSTRACT

Object recognition is an important subtask in image understanding, moving object tracking, and scene analysis. When identifying an object in a scene, it is essential that the same object is recognized as being so from different view angles. Also, in cases where the object is occluded or the image is noisy, the recognition is expected to function under the existence of uncertainty.

In this report, a system for object recognition, with emphasis on view angle independence, is studied. The system uses simple line drawn objects as the input image. An algorithm to extract important information from the image is developed. A rule-based pattern matching scheme is used to recognize the object in the image. In each decision made, a confidence factor is associated to indicate the system's certainty in making this decision.

This study shows that the rule-based pattern matching system is a useful and flexible framework for object recognition and scene analysis.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 General Description:

In video image tracking, the goal is to establish correspondence of objects in one frame of image with those in the next. To establish this correspondence, the same object in the two consecutive frames has to be recognized as being so. Since the relative position between the image sensor and the object changes as the object moves across the field of view, the recognition rules used have to be independent of the consequential changes in view angles.

This report concerns object recognition using image processing and pattern recognition methods. Specificially, it proposes the use of rule-based system as the recognition mechanism. The main interest is to achieve recognition capability independent of the viewing angle. Simple line drawn objects are used in the image initially. An algorithm to extract relevant information from the image was developed. A rule-based pattern matching scheme is used to recognize the object in the image. The system tries to recognize the object as being one of a fixed set of objects which are recognizable by it. A confidence factor is also associated with each decision.

1.2 Outline of Report:

A brief introduction to the topics associated with pattern recognition and a discussion of the various schemes that have been used is covered in Chapter 2. Chapter 2 also describes some of the research work done in this area and the objectives of this thesis. In Chapter 3, the scanning algorithm which scans the image to create a positional and linking information table of the object in the image is described. Also, some of the observations based on which this algorithm was developed are included. Chapter 4 covers the topics which are related to the task of recognition of three-dimensional objects. The feature extraction and matching methods are explained. Chapter 4 also covers the schemes used to recognize incomplete images or images of objects with uncertain identity. In Chapter 5, some results of the experiments performed on the system are presented. Finally, Chapter 6 discusses some modifications to improve the capabilities of the system and also includes the conclusions of this work.

CHAPTER II

REVIEW OF RELATED WORK AND PROBLEM DESCRIPTION

2.1 Introduction :

The advent of the digital computer has stimulated an ever-increasing effort to expand the domain of computer applications. The motivation for this effort comes mainly from the need to find more efficient ways of doing things that machines have never done before. The area of computer vision has evolved from this motive.

One goal of computer vision research is to give computers human-like vision capabilities so that machines can sense the environment in their field of view. The problems, thus, shift from one of sensing the data to a much more difficult one of understanding it. Though the process of vision is so obvious to us, it is something that no one really understands. Numerous studies in psychology and physiology have resulted in many interesting facts about perception, but not sufficient for us to duplicate the process on a machine.

Since the entire problem of vision is an extremely difficult one, most of the research work in this field has been in trying to solve more modest problems related in part to the process of vision. Many of these involve pattern classification or the assignment of a physical object or event to one of several prespecified categories. An excellent introduction to pattern recognition systems is provided in a paper by K.S. Fu [8] and in a text by Duda and Hart [6]. The following pages constitute an overview of pattern recognition systems presented in the above mentioned sources.

Pattern Recognition

A primary area of research which is of direct importance to computer vision is Pattern Recognition. A pattern recognition system would try to classify given data as belonging to one of the several classes in a defined set.

Some examples of the applications of pattern recognition systems are :

blood sample classification

fingerprint classification

classification of solid objects

scene analysis

machine part classification

target identification

medical diagnosis

speech and signal analysis

A general form of a pattern recognition system would be like the one shown in Figure 2.1. The transducer senses the input and converts it into a form more suitable for machine processing. The feature extractor

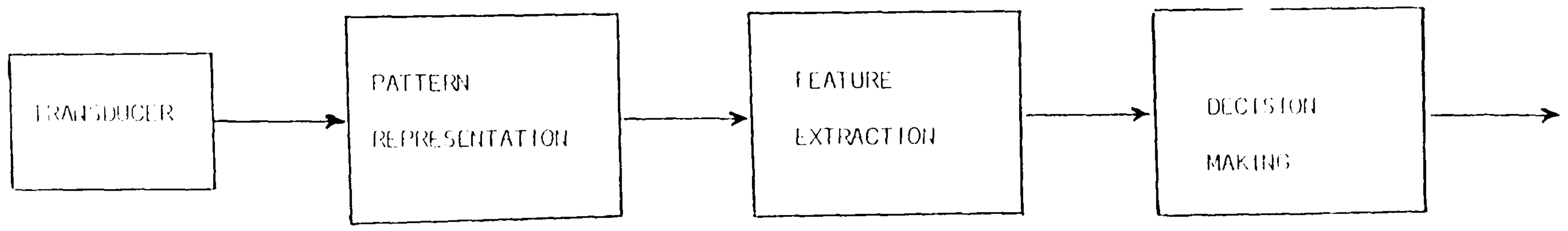


Figure 2.1 : Block Diagram of a general pattern recognition system

extracts revelant information from the data. This information is used by the classifier to assign the input data to one of a finite number of categories or classes.

In computer vision systems, the transducer is a camera which converts the scene into a digitized gray-scale intensity image. Intensity images are arrays of numbers that indicate brightness at points on a regularly spaced grid. This data contains no explicit information about depth, and though people can easily infer depth relationships between image regions, automatic inference of such relationships is difficult.

An alternative to intensity images has become available in recent years, which is digitized range data. Range data are in the form of arrays of numbers, where the numbers quantify the distances from the sensor focal plane to the object surfaces within the field of view along rays emanating from points on a regularly placed grid. Since correct depth information depends only on geometry and not on illumination and refectivity, intensity image problems with shadows and surface marking do not occur. Therefore, the process of recognizing objects by their shape should be less difficult in range images than in intensity images.

The next step after the image has been obtained from the transducer, is to try to transform this pixel image into something which is more meaningful for the analysis of the image. For this, the further processing of data is carried out in a number of ways. The many methods proposed can be grouped into three major categories :

Template matching approach

Decision-theoretic or discriminant approach

Syntactic and structural approach

Template Matching : In the template matching approach, a set of templates or prototypes, one for each pattern class, is stored in the machine. The input pattern is matched or compared with the template of each class, and the classification is based on a preselected matching criterion or similarity measure (e.g. correlation). In other words, if the input pattern matches the template of the ith pattern better than it matches any other template, then the input pattern is classified as being from the ith pattern class. For machine simplicity, input patterns and the templates are usually represented in their raw data form.

The template matching approach has been used in printed-character recognizers. The main difficulties with this approach lie in selecting a good template for each pattern class and in defining a good matching criterion, especially when large variations and distortions are expected in the patterns under study.

Decision theoretic : In this approach, a pattern is represented by a set of N features or an N-dimensional vector. The decision making process is based on a similarity measure which is expressed in terms of a distance measure or a discriminant function. (see Figure 2.2)

The task of feature extraction is very much problem dependent. The basic function of a feature extractor is to map each data point onto a point in the 'feature space'. To do this, it must compute for each data point the values for a number of features (e.g. intensity, gradient, etc.). The problem of classification is in essence one of partitioning the feature space into regions, one region for each category or class. Ideally, one would like these regions to be non-overlapping so that the decision made is never in doubt. But, if this is not possible, the objective should be to reduce the probability of error or uncertainty in the decision.

Applications of decision-theoretic pattern recognition include character recognition, biomedical data analysis, processing of seismic waves and target detection and identification.

Syntactic : In the structural and syntactic approach, a pattern is represented as a string, tree, or a graph of pattern primitives and their relations. The decision-making process is, generally, a syntax analysis or parsing procedure. Conventional parsing requires an exact match between the unknown input sentence and a sentence generated by the pattern grammar. This limits the applicability of the syntactic approach to noise-free or artificial patterns. To overcome this limitation, a parsing procedure called error-correcting parsing has been developed. (see Figure 2.3)

Applications of syntactic pattern recognition include character recognition, speech recognition, waveform analysis, target recognition, and geological data processing.

Recent Trends in Computer Vision :

Artificial Intelligence tools like expert systems can be employed in the final stage of a structural method for object classification in an image. These artificial intelligence techniques have been applied in a number of ways [17] : rule-based inference, prototype or model building, fuzzy pattern matching, etc. These methods are especially useful when no a priori probability laws for an object to belong to different classes are known (i.e. statistical methods cannot be used).

A rule-based system can be considered as a structural pattern recognition system. It usually implies two steps :

(1) Many parameters describing the shape and the position of the object are measured in the image.

(2) This representation of the object is matched with a model, known in terms of the same parameters.

A general form of a rule-based system is shown in Figure 2.4.

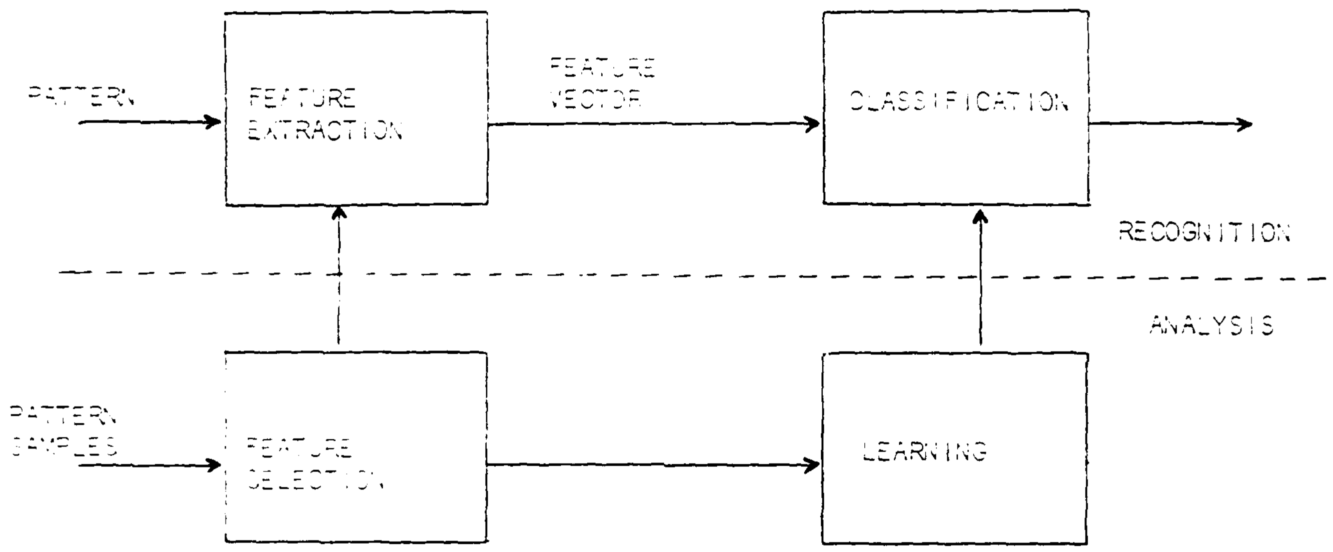


Figure 2.2 : Block Diagram of a decision-theoretic pattern recognition system

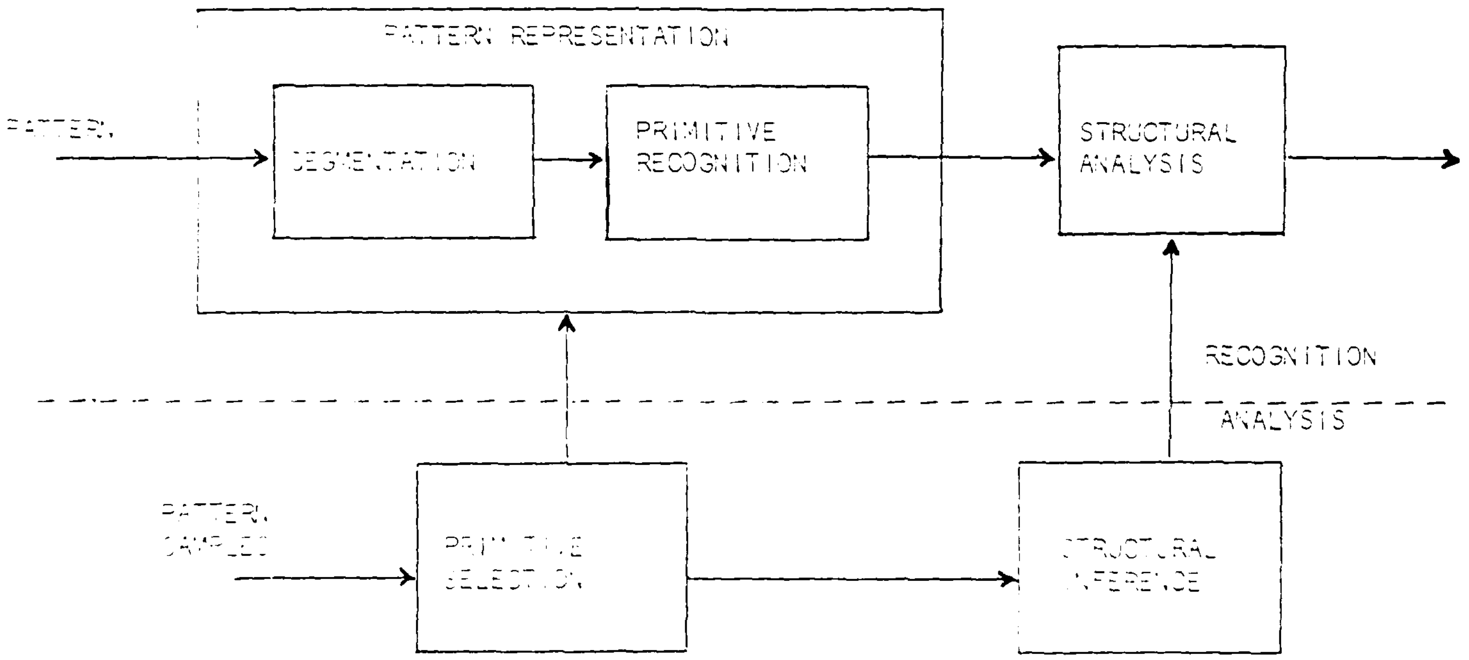


Figure 2.3 : Block Diagram of a [illegible] system

The long term memory (LTM) embodies the model representing the system knowledge. The short term memory (STM) is the storage area for the input image parameters. Primitive Recognition is responsible for the actual pattern matching process. Geometrical inference refers to a special set of rules used to analyze the structural and positional relationships between the various regions of the object. The system has to update the input parameters (in STM) occasionally to record how far it has proceeded in recognizing the object. The STM is manipulated by the Modification block. The primitive recognition module uses a set of rules to start the process of matching the input parameters to the stored models in a logical manner. Further rules are invoked, depending on the success or failure of the present rules. This continues until the object has been classified.

Such a system, thus, presents a complete separation of the knowledge from the control structure. The main advantage of this system is its flexibility. The system can be expanded to accommodate more 'recognizable objects' by the addition of new rules in the knowledge base and slight modifications in the control structure. The rules themselves can be modified to be more strict or relaxed about the constraints necessary for a pattern to be classified as an object.

Recently, several systems using these approaches have been used to perform various tasks from low-level image segmentation [14] to such complex problems as Interpretation of aerial images [12] and Scene Analysis [5].

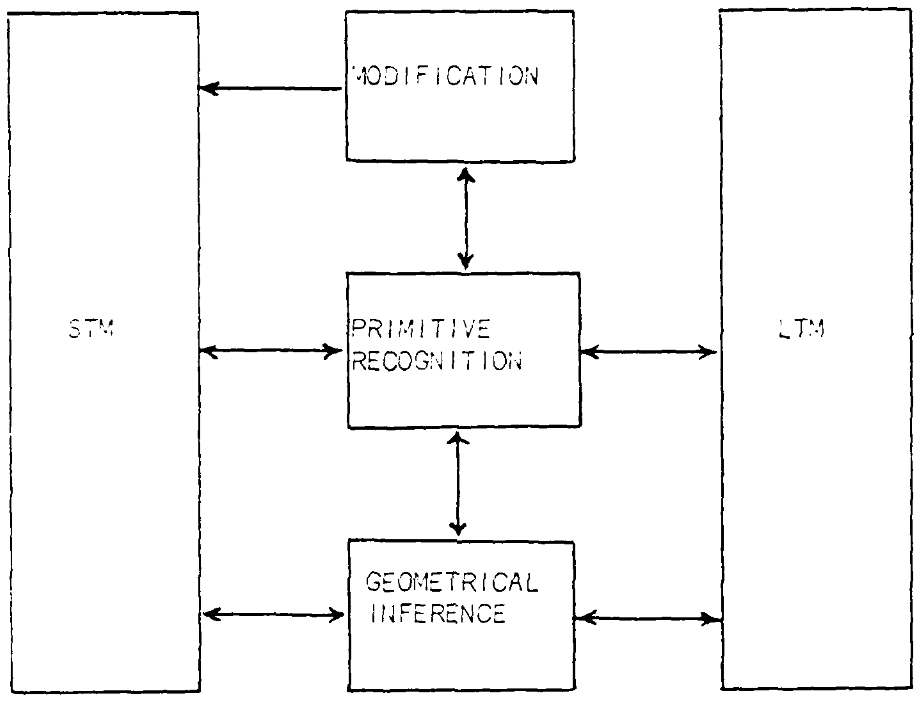


Figure 2.4 : Block Diagram of a Rule-based system

2.2 Three Dimensional Object Recognition :

The problem of three-dimensional object recognition is a non-trivial one. A lot of effort has been done in trying to analyze how human-beings process information about an object to finally recognize it. A plausible explanation is that we work with certain expectations about the scene and that we have a database of possible objects from which we choose the most likely candidate. This seems to be reasonable as it is quite likely that we cannot make a judgement from views of an object we have never seen before and are more likely to identify the unusual view as being caused by something we have seen before.

The process of three-dimensional object recognition involves several basic operations. First, we need to describe and characterize the three-dimensional objects (knowledge representation). Next, we must extract revelant information from the scene (this involves segmentation, edge detection, feature extraction). Finally, the extracted information must be processed and compared with the three-dimensional objects.

One of the first researchers to be concerned with three-dimensional objects was L.G. Roberts [15]. Roberts' approach involved describing the three-dimensional environment which generated the image rather than describing the picture itself. He represented each type of object in a three-dimensional coordinate system; this representation is called a model. By using a transformation matrix, each model was tried to transform into the scene object. The object was then classified according to the model that best fitted it.

Falk [7] used fixed models of the objects that could appear in the scene. With the models and a large set of heuristics, Falk's program followed a hypothesize-and-test strategy to identify objects.

In recent years, more interest has concentrated on developing better representation schemes for three-dimensional objects. There are currently three major representation schemes : volume, surface and skeleton representations.

A very famous system for three-dimensional interpretation of two-dimensional images is the model-based ACRONYM system (Brooks et al. [2]). It is flexible and modular in design, uses view-independent volumetric object models and has a complex, large-scale nature. The system is based on the prediction-hypothesis-verification paradigm.

There are many other 3-D object recognition schemes based on intensity images. Mulgaonkar et al [13] devised a scene analysis system that recognized 3-D objects using geometric and relational reasoning. The modeling scheme used is the 'generalized blob' model proposed by Shapiro [16].

Lee and Fu [10] proposed a design for a general computer vision system that would be capable of 3-D object recognition using a single image. Their aim was to create a system that allows for the proper interaction of the top-down (model-guided) analysis and bottom-up (data driven) analysis.

Chakravarty and Freeman [3] have developed a technique that uses characteristic views as a basis for intensity image 3-D object recognition. The set of all possible perspective projection views of an object is partitioned into a finite set of topological equivalence classes, which are represented by characteristic views. Matching is performed using line-junction labelling constraints on detected edges.

Some 3-D object recognition techniques are based purely on object silhouettes and cannot distinguish between objects that have the same set of silhouettes. McKee and Aggarwal [11] have worked on recognizing 3-D curved objects from a partial silhouette description.

Wallace and Wintz [18] have used global 2-D shape descriptors to recognize 3-D aircraft shapes by matching against a stored library of shape descriptors. One shape descriptor set is computed and compressed for each discrete viewing angle. This gives the system view independence at the cost of storing many descriptors. Given an arbitrary view of a known aircraft, two-dimensional shape descriptors are computed and matched against each precomputed view description in the library for each possible aircraft.

2.3 Problem Definition and Explanation :

Judging from the number of published works, there certainly has been a great amount of research in the area of object recognition. When identifying an object in a scene, it is essential that the same object be recognized as such from different view angles. Also, in cases where the image is noisy, the recognition system is expected to function correctly though it might incorporate an element of uncertainty in its decision. The use of a rule-based system for object recognition would be appropriate because of its flexibility and capability of incorporating contextual information into the recognition process.

A rule-based system for the recognition of line-drawn objects is suggested. Line drawing is selected in this study because lines and edges are the most important features extracted by the human vision system from a scene. Obtaining line drawings of an object from its image involves many processes like image segmentation and edge detection. There are numerous algorithms available to perform these tasks and they will not be discussed here.

A block diagram of the system is shown in Figure 2.5. There are two levels in the system. These two levels are responsible for two separate processing functions. The lower level has the line drawn image at its input. In this level, information about the object in the image is extracted using an algorithm which scans the image for corner points. When all the corner points have been located, information about the

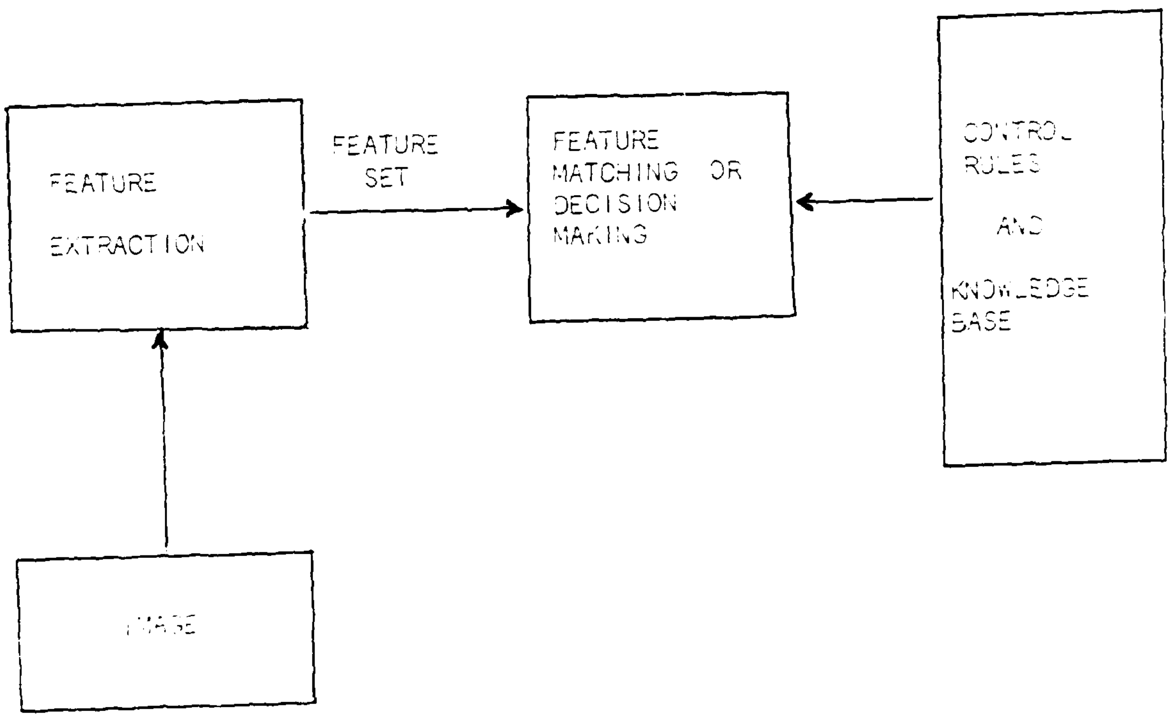

FEATURE EXTRACTION
FEATURE SET
FEATURE MATCHING OR DECISION MAKING
CONTROL RULES AND KNOWLEDGE BASE
IMAGE

Figure 2.5 : Block Diagram of proposed system

object will be stored in a table. The table contains information about the location of every corner point. It also contains the linking information which explains the connections between the corners and the direction of these connecting edges. Information about the type of each corner is also recorded; the type of a corner is decided by studying the number and direction of the lines meeting at that corner. Information in this table is then translated into a set of features. These features form the input to the upper level.

The upper level uses the features to recognize the object. The features are simple descriptions about the number of vertices detected in the object, their locations, junction types and some other information which is used to complete the line drawing in case the original image was incomplete. The upper level uses stored knowledge about the objects. By matching the features obtained from the image selectively with the stored features, a decision about the identity of the object is made. A perfect match indicates that the object has been identified correctly. If no match has been found and the extracted information indicates that there are some noisy regions in the image due to which some features could not be detected, the system tries to restore the line drawing by adding lines or vertices, starting with the most probable one. This transforms the original features into a new set of features and the system undergoes a search to identify the object. A confidence factor is associated with every decision to indicate the certainty with which the system has made the decision.

In the next two chapters, the two levels are described in detail. Chapter 3 explains all the key issues in the feature extraction level such as edge scanning and corner detection. Chapter 4 discusses the features used for the pattern matching process and also covers issues involved in recognition of noisy images.

CHAPTER III

DESCRIPTION OF SCANNING ALGORITHM

3.1 General Description :

This algorithm is used to extract information about the edges and corners in the line drawing of the object. The image is assumed to contain a bi-level line drawing of a polyhedral object. In other words, the image is assumed to be segmented and the edges detected to give this line drawing representation of the object. The algorithm searches for the first pixel in the line drawing and starting from this point, it scans the edge lines until a corner point is reached. This process is repeated until all the corners have been recorded.

In this discussion and in the next chapter, 'corner' and 'junction' have been used interchangeably. The description begins with the explanation of the assumptions made and the data structures involved in the algorithm. Scanning the edge lines and locating the corner points are the key elements of this algorithm and are described in separate sections. Examples are included to explain some points which need clarification. The output of the algorithm includes a combined coordinate and linking table. From this table, information about the position of every corner point and its connectivity to other corner points can be obtained.

The algorithm starts with scanning the image frame from the upper left corner until it detects the first non-zero pixel, which is assumed to belong to the line drawing. The search is then directed by the edges of the line drawing in one of eight basic directions in the order shown in Figure 3.1. The basic data structure is an array type in which each element of the array corresponds to one corner point. Each element of the array, hence, should contain all the positional and linking information associated with that corner point. To accomplish this, a data structure illustrated in Figure 3.2 is used. In this figure, CORNER is the array in which each element contains information about a corner point. **px** and **py** are the coordinates of the corner point; NEBOR is the array that contains information about the neighbouring corners of a corner point. **nx** and **ny** are the coordinates of a neighbouring corner to which the corner (px,py) is connected; the direction of the connecting line is **dir.** The maximum number of corners can be changed easily, but for simple polyhedral objects eight corners seem reasonable. The number of neighbouring corners is limited to four since it is unlikely to encounter an object with more than four planes meeting at a point. However, the limit on the number of neighbours can also be changed, if needed.

<u>The Scanning Process</u> : Initially, the table entries are all set to zero. When the first non-zero pixel is detected, its coordinates are entered in the table for the first corner point. Then, a procedure to detect if this point has any neighbouring points is invoked. These points form the starting location for a search for a possible corner

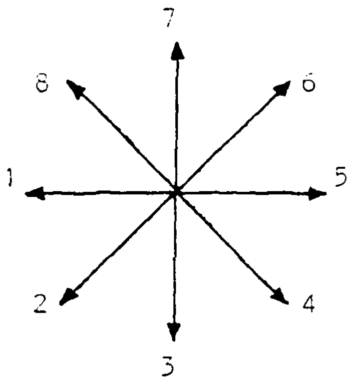


Figure 3.1 : Directions of scanning

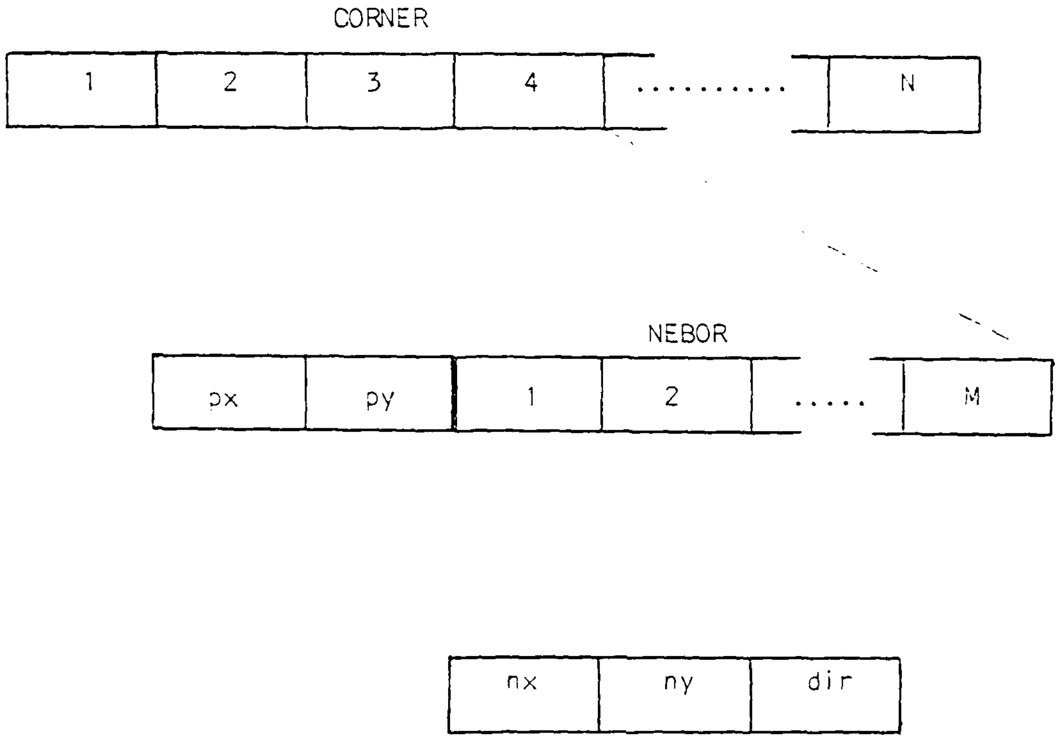


Figure 3.2 : Data structure used to record positional and linking information

point located in the direction along the neighbouring point. We have assumed that the first non-zero pixel will always be a point on the line drawing. This is true for all complete line drawings. If the image has some spurious points, then it is possible that the first non-zero pixel is in no way associated with the object of interest. To handle such situations, the algorithm assumes the point to be a corner and locates its neighbouring corner. If no neighbouring points are found or if the neighbouring corner is at a distance less than a set threshold (about 8 pixels), then the original point which was assumed to be a corner is discarded and the search proceeds to find the next pixel which could be on the line drawing. A corner is always defined to be those points in the image in which the direction of the edge changes by a large amount. This factor depends on the number of directions that can be assigned to a line. In this case, there are only eight directions. Hence, the change in slope at the corner points should be large enough to indicate a change in direction of the line being scanned.

3.2 Searching Edge Lines :

The selection of the neighbouring point is critical since it decides the direction along which the search is to be conducted. If we take the immediate neighbours of the corner point, the direction cannot be determined with any amount of certainty. This is because even though a neighbouring point may be in one particular direction, its neighbours along the same edge may end up going in a different direction. A good way to decide the direction in which a line is proceeding is to observe the sequence of the directions from a pixel to its neighbour in the general direction along the line. A perfectly vertical or horizontal line will indicate the same direction from one pixel to its neighbour, but, an inclined line may go something like two pixels horizontally and three pixels vertically. By observing the sequence for some length along the line, the direction of the line can be accurately determined.

The method used in this algorithm is not so strict in determining the direction, but is a general method implemented after observing many lines of different slopes. It was observed that there is always a cluster of image points around a corner point. These points in the local area around the junction corner group along several narrow strips radiating from the junction. If observed very close to the junction corner, the points form a meaningless cluster, but when seen from a distance they take the shape of lines moving out of the corner point. So, to determine the direction of links, a window is placed at the corner point and the points along the periphery of the window are taken to be the neighbouring points. The direction of the lines is then determined from the location of the neighbouring point with respect to the corner point itself. This is illustrated in Figure 3.3. The window size is chosen such that when the point on the window edge is scanned, there is only one neighbour in the direction that was determined earlier. In this algorithm, a 9 x 9 window was found to operate well. The scanning process then proceeds along the line until a new corner point is reached.

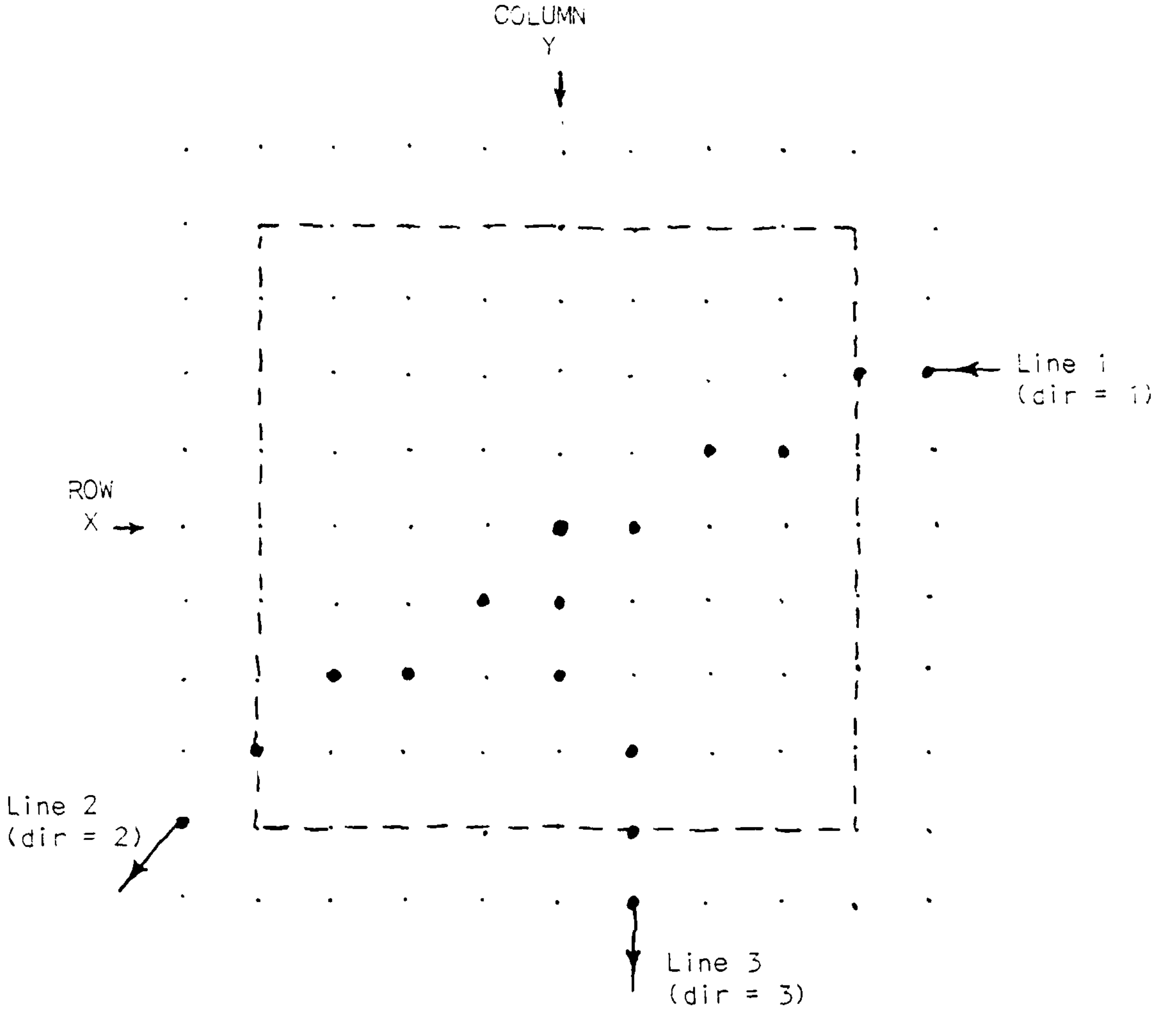


Figure 3.3 : Window used to scan edges around pixel (x,y)

3.3 Detection of Corners :

In the discussion below, neighbouring points refer to the immediate surrounding points (on a 3 x 3 window) and not the neighbours along the edge of the 9 x 9 window used to describe the neighbours of a corner in Section 3.2. Usually, there will be a lot of image points around a corner. Using this fact, by keeping count of the number of neighbours obtained during the scanning process (it is usually 1 during the middle portion of the edge lines), it is possible to determine the presence a corner. It should be noted that when scanning along an edge line, three neighbouring pixels are scanned; the one in the direction of the edge line and the ones on either side of it. This method will not lead to the exact location of the corner point since the clustering of points around the corner might extend to more than a couple of pixels away from it. To correct for this, a threshold is used which overflows when the direction of scanning changes from the direction that was set originally for more than a couple of pixel intervals. Since the direction of edge lines at corners are different, the threshold will also indicate the presence of a corner point. By combining both these methods, the corner points can be located to within two pixel positions.

Let us use the image in Figure 3.3 to illustrate how this method works. Assume that the scanning is along line 1 in the direction towards the corner at (x,y). Until the scanning reaches the non-zero pixel that is one pixel away from (x,y), there would be only one neighbour obtained each time. For the pixel at (x,y+1), there are two neighbours in the scanned direction (dir = 1). The most likely one is (x,y) since it is closest to the specified direction (dir = 1). Continuing from (x,y) along line 2, after one pixel the number of neighbours reduces to one. The algorithm decides that a corner point has been passed since it crossed the crowded area in which a corner is located. Thus, the corner can be located to within two pixels. In case line 2 were not present, the scanning would proceed along line 3. But the direction of scanning would now change to 3 from the original direction of 1. This would cause the scanning to stop within two pixels from (x,y) due to the overflow of a set threshold.

3.4 Table maintenance :

Each time a corner is located, the table is searched to see if this point occurs anywhere in it. If the point has never been recorded then it is recorded as a new corner and corresponding entries are made in the 'neighbour' field for both the new corner and the corner from which the scanning originated. If the corner has been recorded earlier, then only the necessary 'neighbour' field entries are made, if they were not present already. If the corner was recorded as a new corner, the algorithm takes it as the originating corner and goes through the scanning process in a recursive manner. When all the neighbours of all the corners have been searched the process stops. At this stage the table is completed and all the entries for each corner are correctly recorded.

3.5 Summary of Algorithm :

The Algorithm to create the position and linking information table can be summarized in the following steps :

(1) Seek the first corner point (or the first non-zero pixel scanned).

(2) Scan its neighbouring points at the window edge.

(3) For each neighbour, scan in the corresponding direction until a corner point is reached; if no more neighbours then END.

(4) Update the table to indicate the presence of the line scanned above.

(5) Scan neighbouring points of new corner; if no new neighbour then return to (3) for previous corner else make recursive call to (2).

A recursive call means that the current status of the corner point is stored on a stack and the programs loops back to execute the same code. The status information would include location of the point, location and direction of the corner at which the scan originated before reaching the current corner point, neighbours of the current point. If no new neighbour is found in step 5, the status is popped off the stack and the scan proceeds from that point as if it were uninterrupted.

A sample image frame and its corresponding information table are shown in Figure 3.4. The first non-zero pixel found is (30,10) and it is recorded as corner 1. It is found to have three neighbours. Starting with the leftmost one, the edge is scanned till corner 2 at (10,40) is reached. This corner has two neighbours, but the neighbour along the edge connecting corners 1 and 2 is not considered for scanning since it has already been done. Starting at corner 2, the scanning process reaches corner 3 at (40,50). Each time a new corner is reached, the table is updated to include the new links found. From corner 3 which has three neighbours, corner 4 is reached and corner 4 leads the scanning to corner 1. Since corner 1 has already been recorded in the table, the program returns to corner 4 and since there are no other unscanned corners from corner 4, corner 3 becomes corner under consideration. Since the link to corner 1 from corner 3 is the only one remaining to be scanned, this process is done and the next corner is corner 2. As there are no new neighbours at corner 2, the scanning process returns to corner 1. From here, the two edges to corners 3 and 4 are scanned separately and then the process stops. At this point, the table would look as shown in Figure 3.4b.

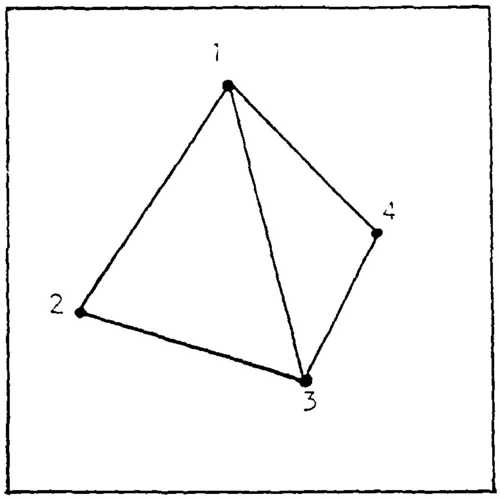


Figure 3.4a : Sample image frame

CORNER			NEBOR			
#	px	py	#	nx	ny	dir
1	30	10	1	10	40	2
			2	40	50	3
			3	50	30	4
			4	0	0	0
2	10	40	1	30	10	6
			2	40	50	5
			3	0	0	0
			4	0	0	0
3	40	50	1	10	40	1
			2	50	30	6
			3	30	10	7
			4	0	0	0
4	50	30	1	40	50	2
			2	30	10	8
			3	0	0	0
			4	0	0	0

Figure 3.4b : Positional and linking information table for sample object shown above

3.6 Special cases :

Modifications for objects with curved edges :

The algorithm discussed above was primarily written for objects with straight line edges. For objects with curved edges, like sphere, cylinder, or cones, some modifications need to be made.

For spheres, any view angle would produce a circle in the image frame. Theorotically, a circle should be traced as a single line. One would expect that the tracing of the circle would result in a single point and indicate no corner points. In practice, the digitized image of a circle is far from being a single line. It is in the form of a lot of short lines connected around the circumference. A test run of an image containing a circle resulted in about sixteen segments, each segment being connected to a segment at its two ends. Thus, by increasing the number of corner elements in the table, the presence of a sphere in the image can be easily detected.

In the case of cylinders the problem is slightly more complex. In addition to the elliptical outline, there are two straight edges and another curved edge. We would need more corner elements in the information table to represent a cylinder. Furthermore, there would be four corner points which form a junction of three edges. This fact can be used in addition to the large number of corner points to detect a cylindrical object.

The situation can be summarized as follows :

It is generally more difficult to handle objects with curved edges because of the limitations in the digitization of curved lines. Curved lines look like a set of connected short segments. So, it is difficult to distinguish a curved object from a polyhedral solid with many vertices.

CHAPTER IV

RECOGNITION OF THREE DIMENSIONAL OBJECTS

4.1 Description of three dimensional objects :

After the linking and position table has been created, the next step is to select certain features from it which form the elements of the matching process for recognition. These features must be chosen such that the object can be described and distinguished from the other objects. For this, it is essential that these features form answers to a number of questions concerning the elements and structure of the geometrical figure.

How many faces can be seen ?

What are the common edges ?

What 2-D figures do the separate faces represent ?

What are the position of the faces in relation to each other ?

What kind of geometrical object does the line drawing represent ?

A lot of structural information can be obtained by simply studying the corners or vertices of the objects. A limited set of junction types are used to describe the vertices. Junctions are classified depending on the geometrical configuring of their incident lines (lines meeting at the junction) [9]. The junction types considered here are illustrated in Figure 4.1 and described below :

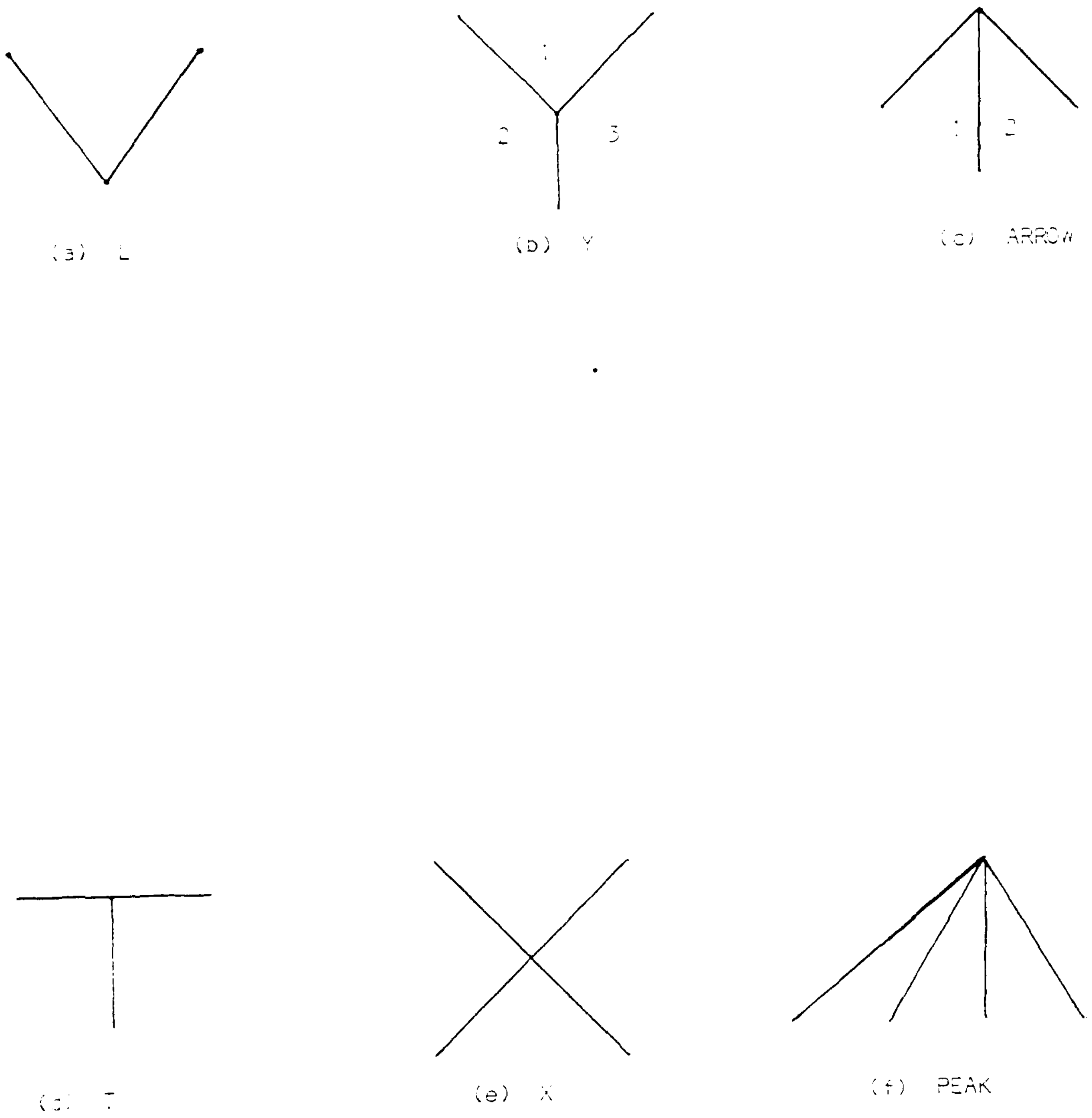


Figure 4.1 : Corner types used in classification

L : Forms the outline of a face and does not indicate any intersection of faces.

Y : Formed by the intersection of three faces; in Figure 4.1b, regions 1 and 2, 2 and 3, 1 and 3 are linked.

ARROW : Formed by the intersection of two faces at the junction; in Figure 4.1c, regions 1 and 2 are linked at the head of the arrow and the stem is the common edge of the two faces.

T : Indicates the meeting of two regions as above but the view angle is different (Figure 4.1d); this type of edge is important when a pair of them occur with their stems collinear since it indicates that the two junctions might have been formed by the same body.

X : Formed the intersection of four (or more) faces which meet at a common point (Figure 4.1e); this type of junction is found in pyramid-type of objects.

Peak : Indicates the meeting of several faces at a common point; in Figure 4.1f all the adjacent regions are linked; this type of junction is also found in pyramid-type objects.

The recognition phase has to be made independent of the view angle (location of the camera). Hence, the use of dimensions and angles becomes a problem since these measures will differ with the view. If dimensions are to be used in the matching process of recognition, then, we might have to use some kind of transformation on these measures based on the a priori knowledge of the position of the camera. Yet this method provides independence in a single dimension only. If the object is viewed from a different angle, it is very likely to produce a completely different view.

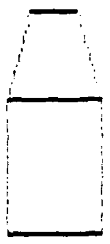
Thus, it is extremely difficult to have only one stored model of an object and try to use transformations on it to get a match with the object for all views. A simple solution, then, is to store a model for every possible view and try to match the object with every model. If the object matches with any one of the stored models then it is recognized. Still, features like length of edges, angles between edges, etc. will be very difficult to handle. So, the features must be chosen such that they will not be affected by small changes in viewing position. Using the corner points and their junction type is a very simple alternative.

Using the number of visible corners and their junction type as the features does not give a complete description of the object, but it may be sufficient to distinguish an object from the others.

4.2 Ambiguity in recognition :

However, it is still likely that an object when viewed from a particular angle looks like another object. This will lead to an ambiguity and the recognition phase might not be able to recognize the object as one particular object. Hence, it is necessary to include a measure of the system's certainty (or uncertainty) with the decision it makes. This is done by associating each model of an object with a confidence factor which is indicative of the certainty with which that model fits the description of the object. In other words, a precomputed confidence factor is also stored alongside the features of the object for each model. With this modification, what is achieved is that when an image that fits the possible description of two or more objects is encountered, the system indicates its uncertainty about the identity of the object by matching it to all the possible objects and associating a confidence measure alongside each object.

A sample session is shown in Figure 4.2. After scanning the image, the features extracted will indicate that there were six corners, of which two corners have three incident edges (type 3) and four corners have two incident edges (type 2). The system is instructed to recognize the object by the user input **object?**. Using the extracted information, the program searches the knowledge base for a matching set of features. The stored knowledge about the objects is arranged as shown in Figure 4.4. All views of all possible objects are searched to find a match.



```
Running vector f1.dat

Good UNSW - PROLOG
| ?- object.
object : cube , prob = 35
** yes
object : prism , prob = 15
** yes
object : lshape , prob = 25
** yes
object : lshape , prob = 25
** yes
|
```

Figure 4.2 : Sample output of the recognition program

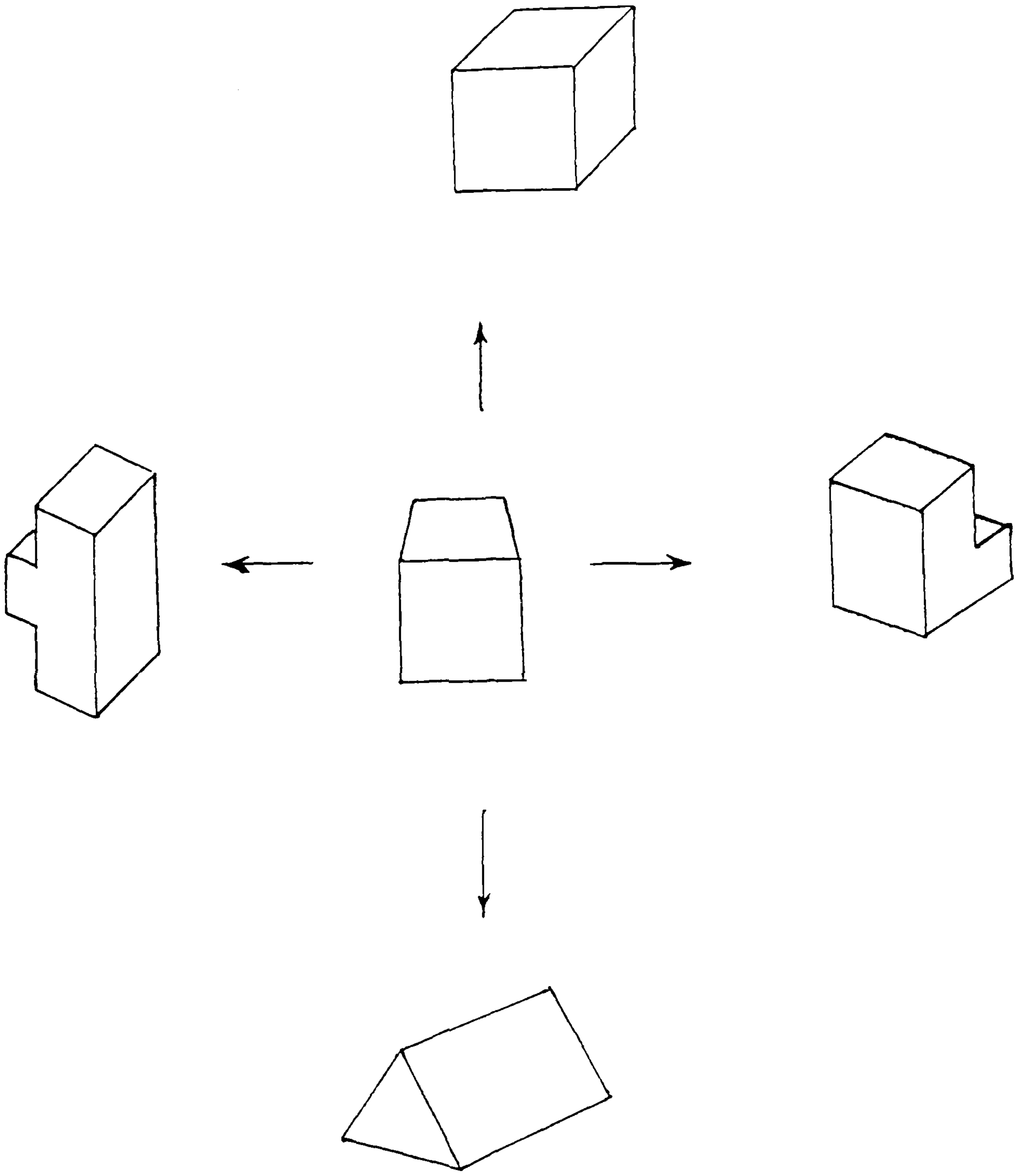


Figure 4.3 : Same image obtained from different objects viewed from different angles

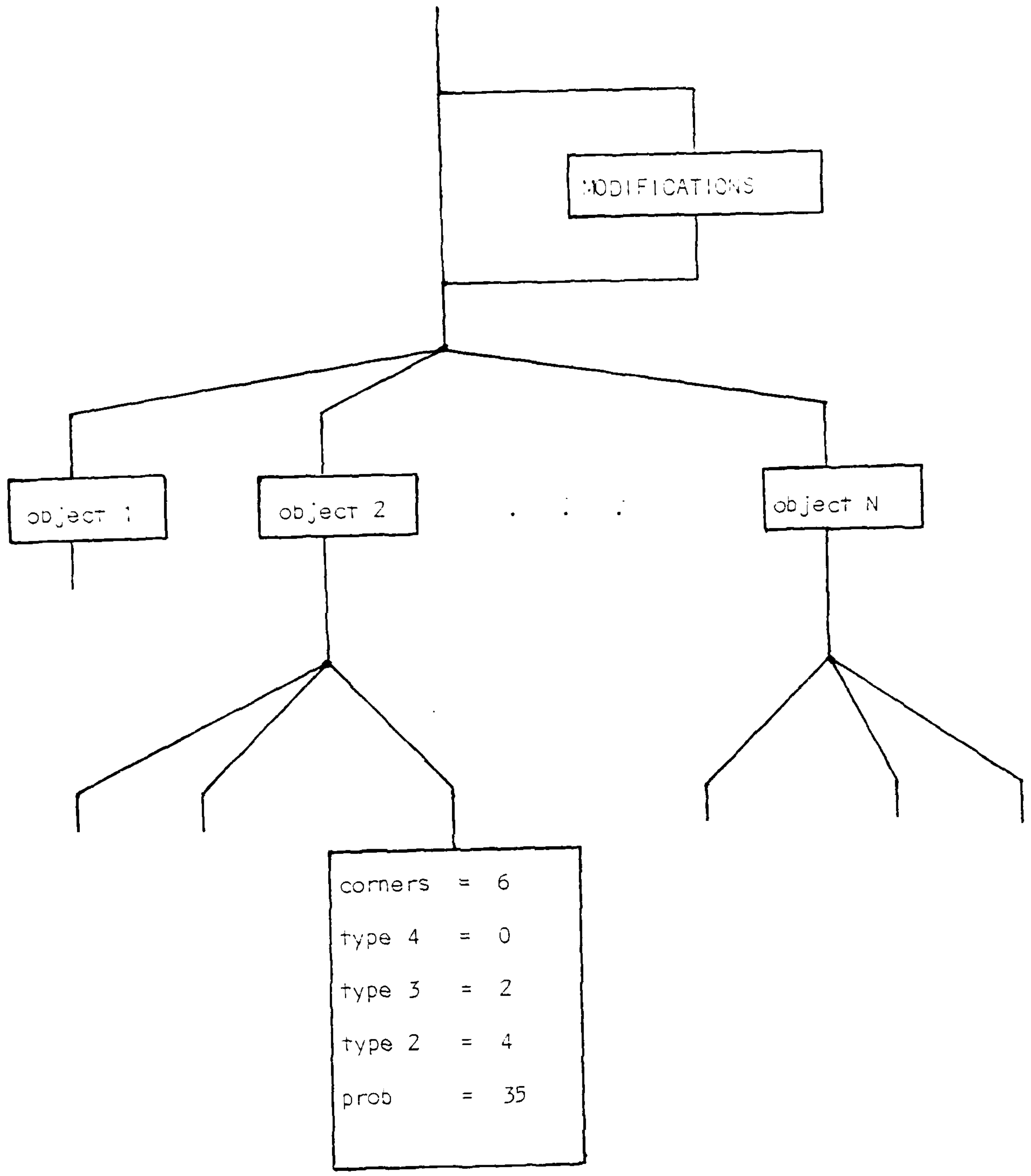


Figure 4.4 : Structure of the knowledge base

Whenever a particular feature set (corresponding to a particular view angle) of an object matches the extracted features, the program informs the user that a match is found and also the confidence factor which was associated with the stored features of the object (**prob**). The first object looked up is a cube. One set of features stored for a cube had the same values for number of corners and types of corners. A match is found between the extracted features and these stored features. For this view of a cube, the confidence factor stored was 35. Hence, the system responds with the identity of the object (*object : cube*) *and* the certainty of its decision (prob = 35).

The system first searches for a match with different feature sets *of the same object*. Whenever a match is found, the remaining sets are not considered. If no match w[illegible] found, the next object is considered for matching. In any case, the search covers all the objects. In this case, the object could have been a cubical one (prob = 35) or a prism (prob = 15) or a L-shaped object (prob = 25) or a T-shaped object (prob = 25). These confidence measures can be adjusted according to past experience about the objects. Figure 4.3 shows the viewing angle which would result in this image for each of the above mentioned objects.

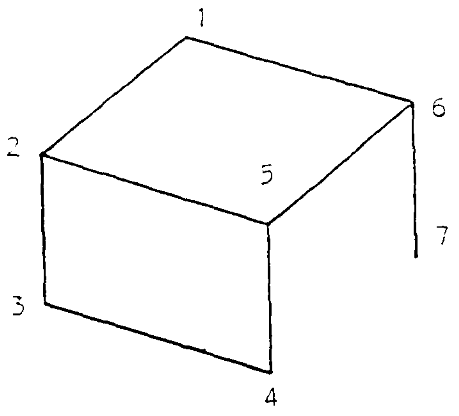
When an object cannot be completely identified using the elementary features of corner types, it may be necessary to invoke more rules to eliminate some of the choices made earlier. To do this, the system will have to study more detailed features like number of visible faces and the shapes of the individual faces.

4.3 Noisy or incomplete line drawings :

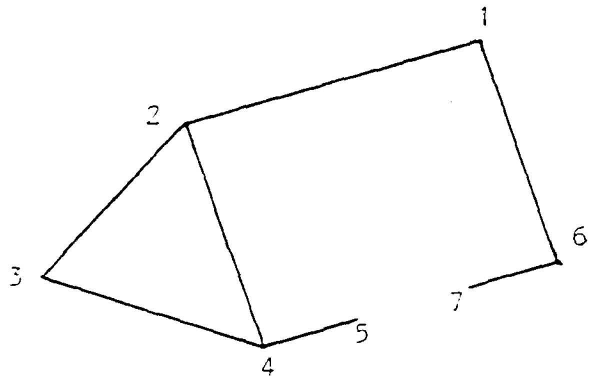
We have till now assumed that the line drawing that we started out with was complete, i.e. it had perfect, continuous edges and no part of the line drawing was faded or erased. This may not be true in a practical situation if the line drawing was obtained from a camera image of the object. Adverse lighting conditions or shadows can cause some edges of the object to be unidentifiable as edges since there isn't enough contrast. Figure 4.5 shows some disformations that can occur in the line drawn image.

Such incomplete line drawings cannot be handled by the system. In other words, the system will fail to find any match for the object and cannot make a decision. In such cases, it is desirable to be able to detect the errors in the input. To detect that the line drawing is flawed, a simple rule is invoked. It can be clearly observed that when the line drawing is incomplete, there will be some corners present in the image which will be connected to only one other corner (refer to Figure 4.5). Alternatively, it can be said that if a corner exists which has only one neighbouring corner then the line drawing has some discontinuity. Any solid object cannot have a corner or vertex that is connected to less than two other vertices. Using this rule, it is possible to determine that the features extracted from the object are not correct and some modifications have to be made. Modifications will involve alterations of the database itself. Features of the original object will have to be removed and features corresponding to the

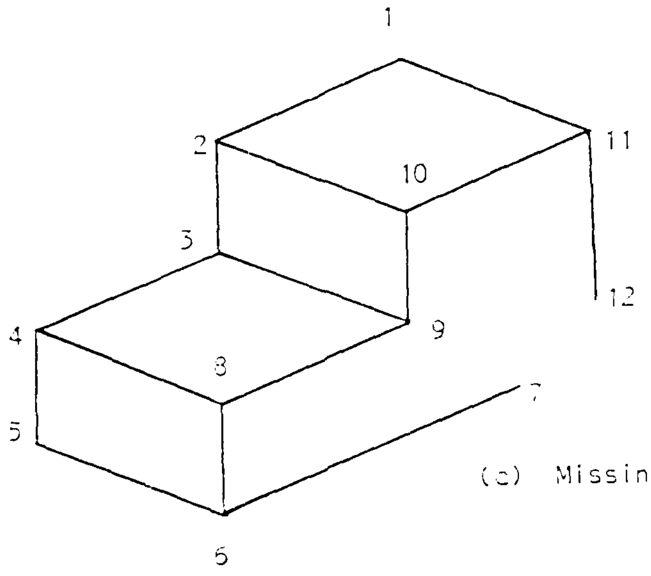
Figure 4.5 : Examples of noisy images



(a) Missing edge



(b) Incomplete edge



(c) Missing corner

modified object will have to be inserted into the database. When the alterations are completed or there are no other unconnected corners, the matching process can be started as before.

Modifications are done based upon the number of defective corners seen in the object. If there is only one corner which is not completely connected, then it is linked to the closest corner under the condition that this linking is possible, i.e. this new link should form a new face and not intersect any other links. To determine the number of faces, the linking information is used to trace the edges to find all possible sets of edges which close on themselves. Intersection of links are approximated by using the positional information to determine whether any of the other corners lie within a zone; the zone is the box which has the two corners being linked as diagonal points. The system also uses its knowledge about the corner type to finally decide to which corner the unconnected corner should be linked (higher preference is given to the corner with less number of incident lines). Under these conditions in Figure 4.5a, corner 7 can only be linked to corner 4.

If there are two defective corners, then there are two possibilities.

(1) The two defective corners are not actual corners, but are part of the same edge. In Figure 4.5b, corners 5 and 7 should be linked.

(2) The two defective corners are part of the same corner which is missing from the image. In Figure 4.5c, corners 7 and 12 should be linked to form a corner.

In the first case, the only modification is to reduce the number of corners by two since no new corners are supposed to exist. In the second case, the number of corners are reduced by one and its type is also included in the database.

Similar rules are framed for cases where there are more defective corners to come up with a possible model of the original object. Whenever a modification is made on the original image, it means that there is more uncertainty about the actual identity of the object. So, the system reduces the confidence factor associated with its decisions for every modification made. Actually, the modification is done before the matching process. Depending on the type of modification, the system starts out with a negative confidence factor. So, when a matching set of features is found for the object, the original confidence factor is automatically reduced.

CHAPTER V

RESULTS

5.1 PROGRAMMING ENVIRONMENT :

The rule-based system was tested by using different types of regular polyhedral solids. A Z-100 was used to perform all the tests. The feature extraction program was coded in Pascal and the rule-based recognition program was coded in Prolog. A database containing eight object descriptions was used for testing. The set of objects consisted of prisms, pyramids, composite objects like L-shaped and T-shaped solids and also spheres and cylinders (see Figure 5.1). The input is in the form of a line drawing of one of these objects in a 64 x 64 image frame in which the lines are indicated by a higher intensity than the background of uniform intensity. The line drawing was generated by using simple graphic routines. All the pixels in a 64 x 64 frame enclosing the line drawing were stored in a data file. All the corners in the line drawing were required to be more than a certain threshold (8 pixel positions) apart. This restriction was imposed to avoid error due to some spurious points in the data file.

The data file formed the input to the scanning program which generates the table containing the coordinate positions of all the vertices in the drawing and also their connections to the other

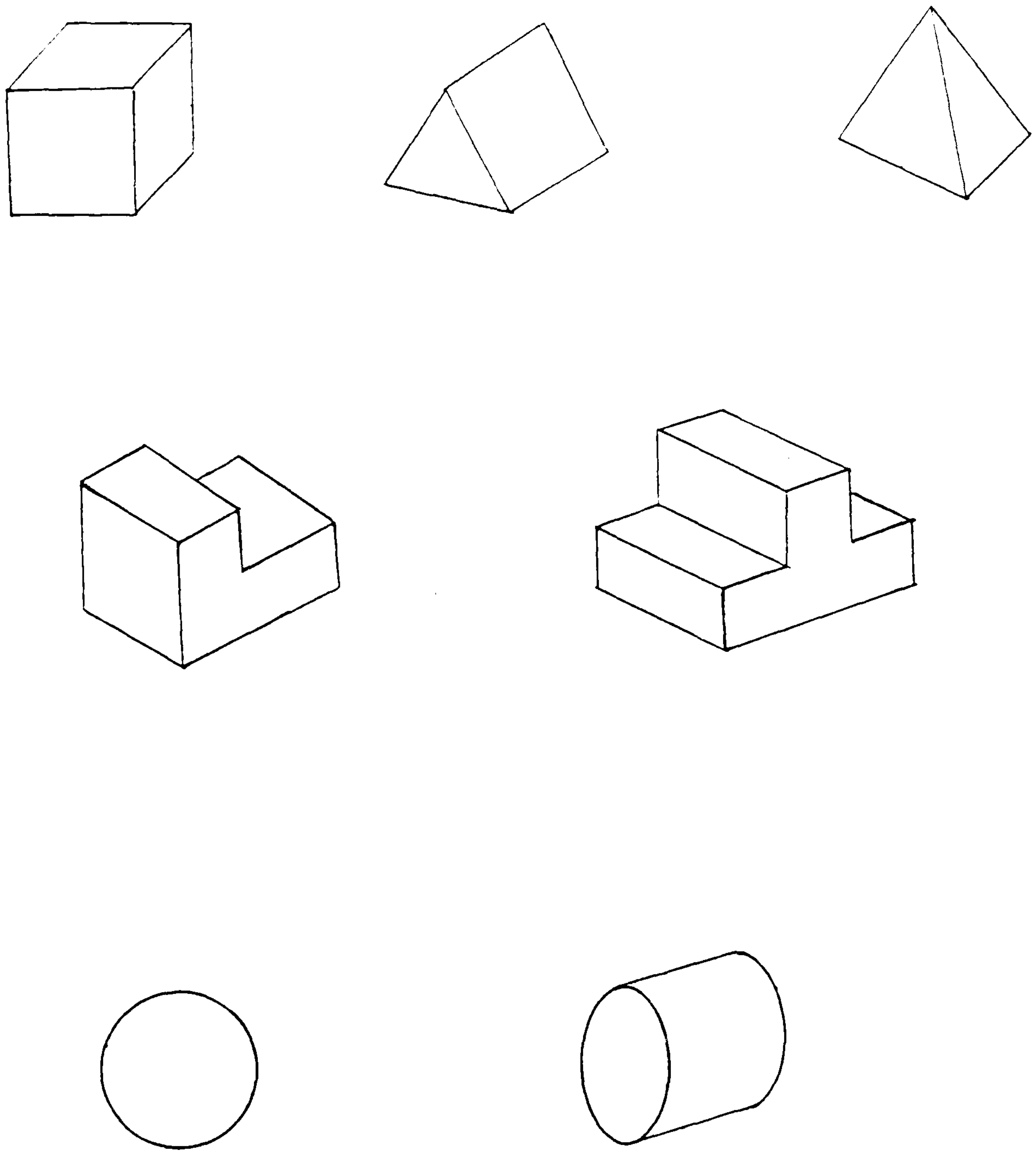


Figure 5.1 : Objects used in the test set

vertices. The feature extraction part of the program, then, extracts certain predefined features by going through all the entries in the table. The features consist of position of each vertex, information about which vertices are linked by the edges, description of the vertices (junction type). These features were stored in another data file. The features should be in a specific format that can be read by the Prolog program which does the task of identifying the object.

5.2 RESULTS :

Several data files of different objects were created for testing. The view angles for the objects were also changed. The program was successful in deciding the possible objects that the image could be representing. The output consists of the choice of the system about the identity of the object and an associated confidence factor. The confidence factor is a measure of the system's certainty in that decision. The program was also tested for incomplete line drawings. The program first guesses the lines to be added to complete the drawing and then goes through tne usual process of identification.

A sample output corresponding to the input image in Figure 5.2a is shown in Figure 5.2b. The input image is a complete line drawing of a cube. The system guesses that the object is cube and the confidence factor is 100.

When the same image but with one edge missing (Figure 5.3a) is input, the system still decides that the object is a cube. The confidence factor, however, is reduced to 75.

For an input image as shown in Figure 5.4a, the object is identified as either a prism or a pyramid, both being equally likely (see Figure 5.4b).

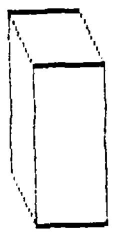
The same drawing as before, but with an incomplete edge is shown in Figure 5.5a and the corresponding output is shown in Figure 5.5b. The result is the same except that there is more uncertainty in the second case with the incomplete edge.

Another sample run for a different object are shown in Figures 5.6 and 5.7. The system was able to identify the object when there was a missing corner in the original image.

Figure 5.8a shows an image with some random noise. The object in the image is recognized as a cube in spite of the noise. In this case, the program that scans the image takes care of the noise as explained in Chapter 3.

Figure 5.2 : Test result 1 :

(a) Input :



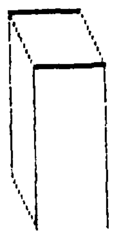
(b) Output :

```
B>prolog rec.pr f.dat

MSdos UNSW - PROLOG
:
: object?
object : cube > prob = 100
** yes
:
```

Figure 5.3 : Test result 2 :

(a) Input :



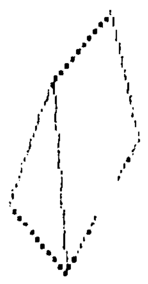
(b) Output :

```
B>prolog rec.pr f.dat

MSdos UNSW - PROLOG
:
:object?
object : cube > prob = 75
** yes
:
```

Figure 5.5 : Test result 4 :

(a) Input :



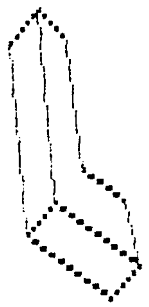
(b) Output :

```
A>prolog rec.pr f2.dat

MSdos UNSW - PROLOG
: object?
object : pyramid > prob = 40
** yes
object : prism > prob = 40
** yes
:
```

Figure 5.6 : Test result 5 :

(a) Input :

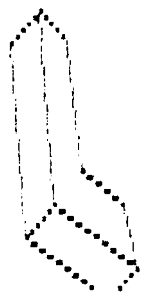


(b) Output :

```
A)prolog rec5.pr f1.dat

msdos UNSW - PROLOG
:
:
:
:
: object?
object ( lshape ) prob = 100
** yes
:
:
:
```

Figure 5.7 : Test result 6 :



(a) Input

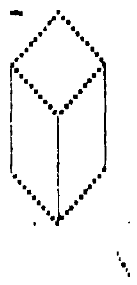
(b) Output :

```
% prolog rec5.pr f2.dat

NCode: UNSW - PROLOG
:
:
:
:
: object?
object ( lshape ) prob = 80
** yes
:
:
:
```

Figure 5.8 : Test result 7 :

(a) Input :



(b) Output :

```
A>prolog rec.pr f0.dat

MSdos UNSW - PROLOG
:
: object?
object : cube / prob = 100
** yes
:
```

CHAPTER VI

DISCUSSION AND CONCLUSIONS

6.1 Limitations of the present system :

The system discussed so far can be considered as a small portion of a large computer vision system. It does not contain any high level geometrical concepts of 3-D objects, but only some basic properties of polyhedra concerning edges and vertices. This limitation comes as a result of emphasizing on generality. The decisions are not based on particular shapes or topologies, but only on general laws.

The capabilities of the present system can be enhanced by the addition of more tasks and distributing these tasks between several levels. The present system can be thought of as having only two levels other than the user. The first level is the one which accepts the segmented binary image with the edges of the object enhanced and produces a table containing all the relevant information about the object. Then, a few preselected features are extracted and made available to the second level. The second level is a rule based interpreter which tries to classify the feature set as that belonging to one of several prespecified objects. This process is guided by a set of rules. In this case, the first level tries to provide the second level with as much information as it can gather from the image. The second level, then, uses as much information it needs to classify the object. This scheme, evidently, does not have any interaction between the two levels.

6.2 Suggested Modifications :

A better approach would be to have a separate feature extraction level which is controlled by the classifier or interpreter. Initially, only a simple set of features is extracted and then more features can be requested from the feature extractor according to the needs developed by the classifier. e.g. if the classifier has reached a point where it seems like two objects could have developed the present set of features, then it can ask for some more information like the types of faces and angles between the edges. In this way, the classifier gets whatever information it needs without the feature extractor having to provide more features than is required. For simple objects, it could come to a decision faster using fewer rules and features and for more complicated objects it would fire more rules which would call for more features. This would make the system faster and more efficient.

This can be accomplished by employing a rule-based strategy for feature extraction in which the rules for getting new features are fired from the classifier. Since, some of the features involve a lot of searching, sorting and calculations to be done, the process of feature extraction is quite intensive computationally. At present, the image scanning program is written in Pascal and the classifier is coded in

Prolog. The version of Prolog used is not very powerful for numerical computations and hence the feature extraction process is also done by a Pascal program. With the help of a more powerful language, it might be possible to integrate these two separate process within the same program.

A robust method for feature extraction would be to include some general rules of feature selection in the rule database. Using these rules and a few test data, the details of which are provided to the system by the user, the system should be able to 'learn' something about the sample objects. When actual data is encountered, this knowledge is used to select the proper features and proceed with the feature matching process for recognition.

Another interesting modification is to enhance the user interaction. For some problems, the system might not be able to conclude anything from the given input data. In such instances, if the system had the capability to request and receive more information from the user, rather than trying to search for it on its own, it would become more efficient. This problem falls more into the area of expert systems. The major problem of this enhancement is the user-program interface. The interface should be such that the user as well as the interactive process should be able to understand the information being exchanged. In all simplicity, this would require the system to have natural language processing capabilities so that it can communicate with the user in a language that can be understood by the user, like

English. Language processing is a very big problem in itself. It could still be possible to maintain communication by a simple 'yes-no' type of session. For the simple objects that this system was originally expected to recognize, a user interface was not deemed necessary, but it is certainly a useful addition for future work.

Finally, to make this system of practical use, it sh ld be able to recognize objects in a scene rather than when they are presented individually. This needs the introduction of more levels into the system. A block diagram of such a system with its functional units is shown in Figure 6.1. At the lowest levels, the image will have to be segmented to separate the regions in the image frame where the individual objects are likely to be found. Then the next level of processing is to detect the edges of the object in each region. When this is complete, each object has to be identified as before by going through the subsequent upper levels. All these processes have to be controlled by a control unit. There are rules for every kind of processing that needs to be done. The control unit has to fire the correct rules depending on the level of recognition that has been established and try to identify the object with the minimum amount of uncertainty.

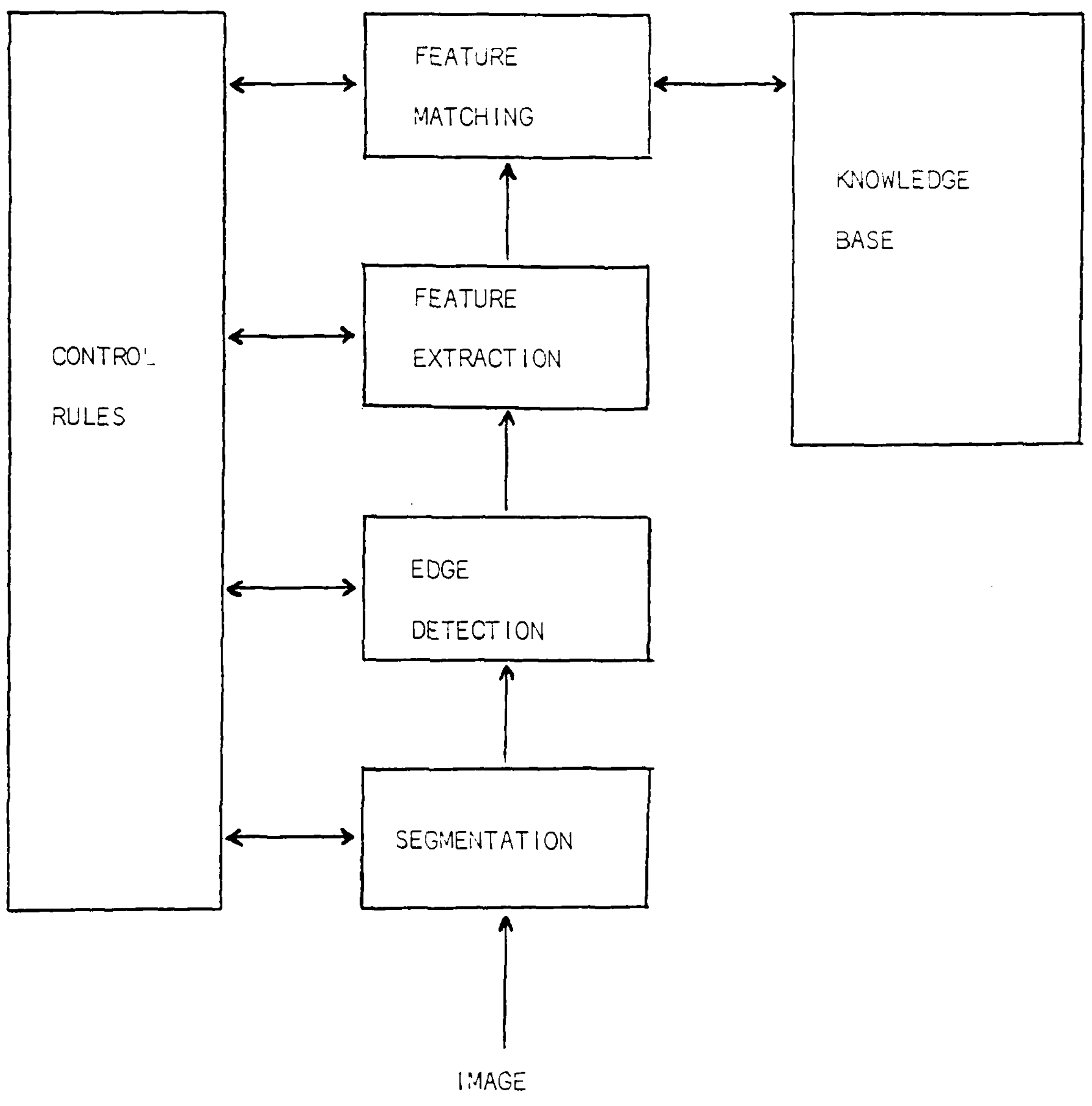


Figure 6.1 : Block Diagram of Modified system

6.3 CONCLUSIONS :

In this thesis, an attempt was made to develop a simple and flexible system to recognize objects. The emphasis is on generality rather than specifics about the shape, dimensions and other 3-D descriptions of the object. An attempt was made to make the decision process as independent of the viewing position as possible. This was tried out on simple polyhedral objects. Since only basic shapes of objects were considered, simple features were used to describe the object. For more complicated objects it is necessary to increase the number and type of features. But since the aim is to only identify the object, it is sufficient to distinguish the object from the others rather than obtaining a complete description of it. The present system performs this task satisfactorily. It identified objects even when the input drawing had some irregularities or some parts were missing. Compared to the conventional methods of statistical pattern recognition or direct pattern matching, the use of rules to guide the recognition process is definitely more efficient and powerful. Its main strength lies in its non-rigid structure. It can be modified or enhanced with simple changes in the rules. Some possible modifications have been suggested in chapter 5. Overall, the results showed that reasonably accurate identification is possible in most cases.

REFERENCES :

(1) Brooks R.A., Model-based three-dimensional interpretations of two-dimensional images, IEEE trans. on Pattern Analysis and Machine Intelligence, Pami-5, No.2, 1983.

(2) Brooks R.A., Greiner R., Binford T.O., The ACRONYM model-based vision system, Proc. of International Joint Conference on Artificial Intelligence, IJCAI 1979.

(3) Chakravarty I., Freeman H., Charateristic views as a basis for reccognition of three-dimensional objects, IPL-TR-034, Renneselear Polytechnic Institute, 1982.

(4) Clocksin W.F., Mellish C.S., Programming in Prolog, Springer-Verlag, 1981,1984.

(5) Douglass R.J., Interpreting three-dimensional scenes : A model building approach, Computer Graphics Image Processing, vol.17, 1981.

(6) Duda R.O., Hart P.E., Pattern Classification and Scene Analysis, John Wiley and Sons, 1973.

(7) Falk G., Interpretation of imperfect line data as a three-dimensional scene, Artificial Intelligence, No.3, 1972.

(8) Fu K.S., Pattern Recognition for Automatic Visual Inspection, IEEE Computer, Dec. 1982.

(9) Guzman A., Decomposition of a visual scene into three-dimensional bodies, AFIPS Fall Joint Conference, 1968.

(10) Lee H.C., Fu K.S., A computer vision system for generating object descriptions, Proc. of Pattern Recognition and Image Processing Conf., 1982.

(11) McKee J.W., Aggarwal J.K., Computer Recognition of partial views of

three-dimensional curved objects, Comp Sci Tech Rep 17, Univ of Texas, Austin, 1975.

(12) McKeown D.M., Harvey W.A., McDermott J., Rule-Based Interpretation of Aerial Imagery, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol.PAMI-7, no.5, 1985.

(13) Mulgaonkar P.G., Shapiro L.G., Haralick R.M., Recognizing three-dimensional objects single from single perspective views using geometric and relational reasoning, Proc. of Pattern Recognition and Image Processing Conf., 1982.

(14) Nazif A.M., Levine M.D., Low Level Image Segmentation : An expert system, IEEE trans. on Pattern Analysis and Machine Intelligence, vol.PAMI-6, no.5, 1984.

(15) Roberts L.G., Machine perception of three-dimensional solids, Optical and Electro-Optical Information Processing, MIT press, 1965.

(16) Shapiro L.G., Mulgaonkar P.G., Moriarty J.D., Haralick R.M., A Generalized Blob Model for Three-Dimensional Object Description, Second IEEE Workshop on Picture Description and Management, August 1980.

(17) Thonnat M., Granger C., Berthod M., Design of an expert system for object classifying through an application to the classification of galaxies, Proceedings, IEEE Computer Vision and Pattern Recognition Conference, June 1985.

(18) Wallace T.P., Wintz P.A., An efficient three-dimensional aircraft recognition algorithm using normalized Fourier descriptors, Computer Graphics Image Processing, vol.13, 1980.

APPENDICES

APPENDIX I

Listing of the program to scan the image :

```
PROGRAM LINESERCH(output);

{ This program is used to extract positional and linking
  information from a 64 x 64 image frame containing
  a line drawing of regular solid polyhedron.
  The image is stored in a two-dimensional array 'pix'.
  The information is stored in another array 'corner'.     }

{$R+,W4}
const maxcor=8;maxnebor=4;minlen=48;

type map=record
               xn,yn : integer;
               npdir : Ø..8;
          end;
     point=record
                 xp,yp : integer;
                 nebor : array[1..maxnebor] of map;
            end;
     narray=array[1..32] of integer;

var  pix : array[1..64,1..64] of integer;
     corner : array[1..maxcor] of point;
     i,j,line,col,dat,indir : integer;
     start,restart : boolean;
     data: text;


 PROCEDURE CHECK(px,py,nx,ny,dir:integer;var found:boolean);

{ This procedure is used to update the entries in the
  information table.  The entry parameters are the
  co-ordinates of the starting and ending corner points
  of the edge most recently scanned.
  At exit,  the table is updated and 'found' is set to
  indicate whether the last corner was a new one.      }

 var ncor,nnebor,cor1,cor2:integer;
     exit:boolean;
```

```
begin {check}
     found:= false; exit:=false;cor1:=0;cor2:=0;
     ncor:=1;
     while((ncor<maxcor)and not(exit)) do
     begin
          with corner[ncor] do
          begin
               if((abs(xp-px)<3)and(abs(yp-py)<3)) then cor1:=ncor;
               if((abs(xp-nx)<3)and(abs(yp-ny)<3)) then
               begin
                    cor2:=ncor;found:=true;
               end;
               if((xp=0)and(yp=0)and not(found)) then
               begin
                    xp:=nx;yp:=ny;cor2:=ncor;exit:=true;
               end;
          end;{with}
          ncor:=ncor+1;
     end;{while}
     with corner[cor1] do
     begin
          exit:=false;nnebor:=1;
          while((nnebor<=maxnebor)and not(exit)) do
          begin
               with nebor[nnebor] do
               begin
                    if((abs(xn-nx)<3)and(abs(yn-ny)<3)) then
                    exit:=true;
                    if((xn=0)and(yn=0)) then
                    begin
                         xn:=nx;yn:=ny;npdir:=dir;exit:=true;
                         end;
                    end;
                    nnebor:=nnebor+1;
               end;
     end;

    with corner[cor2] do
    begin
         nnebor:=1;exit:=false;
         while((nnebor<=maxnebor)and not(exit)) do
         begin
              with nebor[nnebor] do
              begin
                   if((abs(xn-px)<3)and(abs(yn-py)<3)) then
                   exit:=true;
                   if((xn=0)and(yn=0)) then
                   begin
                        xn:=px;yn:=py;
                        npdir:=(dir+4)mod 8;
                        if npdir=0 then npdir:=8;
                        exit:=true;
                        end;
                    end;
                nnebor:=nnebor+1;
                end;
            end;

     end;{of procedure CHECK}



     PROCEDURE SCANCORNER(px,py,dir:integer;var nx,ny:integer);

     { This procedure starts scanning a possible edge in the
       specified direction starting at the corner points whose
       co-ordinates are specified.  It uses a threshold 'thresh'
       and 'maxcount' to determine whether the other corner of
       edge is obtained.  If so,  it returns the co-ordinates of
       the new corner point.                                    }

     var x,y,count,maxcount,minval,minpos,thresh,i: integer;
         n : array[1..8] of integer;
         finish : boolean;

     begin {procedure}
          nx:=px;ny:=py;count:=0;maxcount:=0;finish:=false;thresh:=0;
          while not(finish) do
          begin
               x:=px;y:=py;count:=0;
               for i:=1 to 8 do n[i]:=0;
               if((dir=1)or(dir=2)or(dir=8))then
               if pix[x,y-1]=1 then
               begin
                    n[1]:=1;count:=count+1;
                    if dir<>1 then thresh:=thresh+1 else thresh:=0;
               end;
               if((dir=1)or(dir=2)or(dir=3)) then
               if pix[x+1,y-1]=1 then
               begin
                    n[2]:=1;count:=count+1;
                    if dir<>2 then thresh:=thresh+1 else thresh:=0;
               end;
               if((dir=2)or(dir=3)or(dir=4)) then
               if pix[x+1,y]=1 then
               begin
                    n[3]:=1;count:=count+1;
                    if dir<>3 then thresh:=thresh+1 else thresh:=0;
               end;
               if((dir=3)or(dir=4)or(dir=5)) then
               if pix[x+1,y+1]=1 then
               begin
                    n[4]:=1;count:=count+1;
     if dir<>4 then thresh:=thresh+1 else thresh:=0;
end;
if((dir=4)or(dir=5)or(dir=6)) then
if pix[x,y+1]=1 then
begin
     n[5]:=1;count:=count+1;
     if dir<>5 then thresh:=thresh+1 else thresh:=0;
end;
if((dir=5)or(dir=6)or(dir=7)) then
if pix[x-1,y+1]=1 then
begin
     n[6]:=1;count:=count+1;
     if dir<>6 then thresh:=thresh+1 else thresh:=0;
end;
if((dir=6)or(dir=7)or(dir=8)) then
if pix[x-1,y]=1 then
begin
     n[7]:=1;count:=count+1;
     if dir<>7 then thresh:=thresh+1 else thresh:=0;
end;
if((dir=7)or(dir=8)or(dir=1))then
if pix[x-1,y-1]=1 then
begin
     n[8]:=1;count:= count+1;
     if dir<>8 then thresh:=thresh+1 else thresh:=0;
end;

if((maxcount<count)and(thresh<2)) then
begin
     if count=1 then
     begin
          for i:= 1 to 8 do
          if n[i]=1 then
          begin
               case i of
               1,5:px:=x;
               2,3,4:px:=x+1;
               6,7,8:px:=x-1;
               end;
               case i of
               1,2,8:py:=y-1;
               3,7:py:=y;
               4,5,6:py:=y+1;
               end; {of case}
               nx:=px;ny:=py;
          end;
end
else
begin
     for i:=1 to 8 do
     begin
          if n[i]=1 then n[i]:=abs(i-dir)  else n[i]:=25;
```

```
                    end;

                    i:=2;minval:=n[1];minpos:=1;
                    while i<=8 do
                    begin
                          if n[i]<minval then
                          begin
                               minval:=n[i];minpos:=i;
                          end;
                    i:=i+1;
                    end;
                    case minpos of
                    1,5:px:=x;
                    2,3,4:px:=x+1;
                    6,7,8:px:=x-1;
                    end;
                    case minpos of
                    1,2,8:py:=y-1;
                    3,7:py:=y;
                    4,5,6:py:=y+1;
                    end; {of case}
               nx:=px;ny:=py;
               maxcount:=count;
               end;
          end
          else finish:=true;
          end;{of while not finish}
    end;{of procedure SCANCORNER}


    PROCEDURE NEBOR(x,y:integer;var n:narray);

   { This procedure scans the pixels along the edge of a
     4x4 window surrounding the specified points and returns
     a 33 element array 'narray' which indicates the status
     of every pixel that was scanned.                        }

    var i,tx,ty :integer;

    begin
         tx:=x;ty:=y-4;
         for i:=0 to 4 do
         begin
              if pix[tx+i,ty]=1 then n[i+1]:=1 else n[i+1]:=0;
              if i>0 then
              begin
                   if pix[tx-i,ty]=1 then n[33-i]:=1 else n[33-i]:=0;
              end;
         end;
         tx:=x+4;ty:=y;
         for i:=-3 to 4 do
```

```
      if pix[tx,ty+i]=1 then n[9+i]:=1 else n[9+i]:=0;
      tx:=x-4;ty:=y;
      for i:=-3 to 3 do
      if pix[tx,ty+i]=1 then n[25-i]:=1 else n[25-i]:=0;
      tx:=x;ty:=y+4;
      for i:=0 to 3 do
      begin
           if pix[tx+i,ty]=1 then n[17-i]:=1 else n[17-i]:=0;
           if pix[tx-i-1,ty]=1 then n[18+i]:=1 else n[18+i]:=0;
      end;
 end; { of procedure NEBOR }



 PROCEDURE SCANNEBOR(x,y,from:integer);

{ This procedure searches for corners connected to the specified
  corner in all directions except the one which led the scanning
  to this point,  specified by 'from'.  If it finds a neighbouring
  corner,  it updates the information table and then scans for its
  neighbours and continues this recursive operation.  If a
  corner was found to be entered in the table,  then it searches
  for the other neighbours until all are exhausted.              }

 var i,px,py,nx,ny,pdir  : integer;
     dir : 1..8;
     n : narray;
     found : boolean;

 begin
      nebor(x,y,n);
      if from=4 then pdir:=8 else pdir:= (from+4)mod 8;
      for i:=1 to 32 do
      begin
           if n[i]=1 then
           begin
                case i of
                1,2,32: dir:=1;
                3,4,5,6,7 : dir:=2;
                8,9,10: dir:=3;
                11,12,13,14,15  : dir:=4;
                16,17,18: dir:=5;
                19,20,21,22,23: dir:=6;
                24,25,26: dir:=7;
                27,28,29,30,31: dir:=8;
                end; {of case}
                if (i = 3) then
                   if pix[x+2,y-5] = 1 then  dir := 1;
                if (i = 7)  then
                   if pix[x+5,y-2] = 1 then  dir := 3;
                if (i = 11) then
```

```
                        if pix[x+5,y+2] = 1 then  dir := 3;
                     if (i = 15) then
                        if pix[x+2,y+5] = 1 then  dir := 5;
                     if (i = 19) then
                        if pix[x-2,y+5] = 1 then  dir := 5;
                     if (i = 23) then
                        if pix[x-5,y+2] = 1 then  dir := 7;
                     if (i = 27) then
                        if pix[x-5,y-2] = 1 then  dir := 7;
                     if (i = 31) then
                        if pix[x-2,y-5] = 1 then  dir := 1;

               if dir<>pdir then
               begin
                    if i<=5 then
                    begin
                         px:=x+i-1;py:=y-4;
                    end
               else if((i>5)and(i<=13)) then
               begin
                    px:=x+4;py:=y+i-9;
               end
               else if((i>13)and(i<=21)) then
               begin
                    px:=x+17-i;py:=y+4;
               end
               else if((i>21)and(i<=29)) then
               begin
                    px:=x-4;py:=y+25-i;
               end
               else
               begin
                    px:=x+i-33;py:=y-4;
               end; {of if}

               scancorner(px,py,dir,nx,ny);
               if((sqr(x-nx) + sqr(y-ny))>minlen) then
               begin
                    check(x,y,nx,ny,dir,found);
                    if not(found) then scannebor(nx,ny,dir);
               end;
          end; {of if not previos dir}
          end;
    end;
    end; { of recursive procedure SCANNEBOR }


  begin {main}

  { read the image frame from file 'datafile.dat' }
```

```
        assign(data,'datafile.dat');
        reset(data);
        for i:=1 to 64 do
        begin
             for j:=1 to 64 do
             begin
                  read(data,dat);
                  pix[i,j]:=dat
             end;
        end;
        close(data);

  { initialise data table 'corner' }

        for i:=1 to maxcor do
        begin
             with corner[i] do
             begin
                  xp:=0;yp:=0;
                  for j:=1 to maxnebor do
                  begin
                       with nebor[j] do
                       begin
                            xn:=0;yn:=0;npdir:=0;
                       end; {with}
                  end; {for}
             end; {with}
        end; {for}


   {inital part of scanning process;  seek first pixel}

        line:=2;col:=2;start:=false;
        while((line<=63)and(not(start))) do
        begin
             if col=63 then
             begin
                  line:=line + 1;
                  col:=2;
             end
             else col:=col+1;
          if(pix[line,col]=1) then
          begin
               start:=true;
               corner[1].xp:=line;corner[1].yp:=col;indir:=-4;
               scannebor(line,col,indir);
               restart:=false; i:=1
               while((i<=maxcor)and not(restart)) do
               begin
                    if((corner[i].xp=0)and(corner[i].yp=0)) then
                    restart := true;
                    i := i+1;
```

```
                end;
                if (i<=2) then start := false;
           end;
        end; {of while}

{ output to screen the co-ordinates of all the detected corners }

        writeln('corner# x co-ord y co-ord ');
        for i:=1 to maxcor do
        begin
             writeln(i:20,corner[i].xp:20,corner[i].yp:20);
        end;


end.
```

APPENDIX II

Listing of the program for recognition :

This program uses a rule-based matching strategy to recognize an object. The features of the object are assumed to be stored separately and this feature file should be included in the Prolog database when the interpreter is run. The program expects the user to type in the question 'object?' when the prompt occurs to begin the process of pattern. The program informs the user of the identity of the object and the confidence factor when a match has been found.

Listing of Program RECOGNIZ :

```
phh([]) :- nl.
phh([H|T]) :- write(H),tab(1),phh(T).

eq(Dim1,Dim2) :- Dim1 == Dim2.

linear(X,Y) :-   X == Y + 4;
                 Y == X + 4;
                 X == Y.


initialise(A,B,C,D) :-  numcor(A),
                        cor(4,B),
                        cor(3,C),
                        cor(2,D).


init(P) :- nummis(M),
           initt(M,P).
initt(M,P) :-   M == Ø,
                P is Ø.

initt(M,P) :-   M == 1,
                Ml is 1,
                adjustl(Ml,P).

initt(M,P) :-   M == 2,
                Ml is 1, M2 is 2,
                adjust2(Ml,M2,P).


adjustl(M,P) :- miscor(M,Pl,D), near(Pl,N),
                cor(2,Q), Ql is Q + 1,
                retract(cor(2,Q)), assert(cor(2,Ql)),
                type(N,T), cor(T,Y), Yl is Y - 1,
                retract(cor(T,Y)), assert(cor(T,Yl)),
                Tl is T + 1,
                cor(Tl,Z), Zl is Z + 1,
                retract(cor(Tl,Z)), assert(cor(Tl,Zl)),
                retract(nummis(1)), assert(nummis(Ø)),
                P is -25.

adjust2(Ml,M2,P) :- miscor(Ml,Pl,Dl),
                    miscor(M2,P2,D2),
                    linear(Dl,D2),!,
                    numcor(X), Xl is X - 2,
                    retract(numcor(X)), assert(numcor(Xl)),
                    retract(nummis(2)), assert(nummis(Ø)),
                    P is -1Ø.


adjust2(Ml,M2,P) :- miscor(Ml,Pl,Dl), miscor(M2,P2,D2),
                    not(linear(Dl,D2)),!,
                    numcor(X), Xl is X - 1,
                    retract(numcor(X)), assert(numcor(Xl)),
                    cor(2,Y), Yl is Y + 1,
                    retract(cor(2,Y)), assert(cor(2,Yl)),
                    retract(nummis(2)), assert(nummis(Ø)),
                    P is -2Ø.


cube(N,T3,T2,P,Pl) :- eq(N,6), eq(T3,2), eq(T2,4),Pl is P + 35.
cube(N,T3,T2,P,Pl) :- eq(N,7), eq(T3,4), eq(T2,3),Pl is P + 1ØØ.
cube(N,T3,T2,P,Pl) :- eq(N,4),eq(T2,4),Pl is P + 15.

pyrmd(N,T4,T3,T2,P,Pl) :- eq(N,5),eq(T4,1),eq(T3,2),eq(T2,2),Pl is P + 1ØØ.
pyrmd(N,T4,T3,T2,P,Pl) :- eq(N,5),eq(T4,1),eq(T3,4),Pl is P + 1ØØ.
pyrmd(N,T4,T3,T2,P,Pl) :- eq(N,5),eq(T3,4),eq(T2,1),Pl is P + 1ØØ.
pyrmd(N,T4,T3,T2,P,Pl) :- eq(N,5),eq(T3,2),eq(T2,3),Pl is P + 5Ø.
pyrmd(N,T4,T3,T2,P,Pl) :- eq(N,4),eq(T2,4),Pl is P + 15.
pyrmd(N,T4,T3,T2,P,P1) :- eq(N,4),eq(T3,2),eq(T2,2),P1 is P + 100.
pyrmd(N,T4,T3,T2,P,P1) :- eq(N,3),eq(T2,3),P1 is P + 50.

prism(N,T3,T2,P,P1) :- eq(N,6), eq(T3,2), eq(T2,4),P1 is P + 15.
prism(N,T3,T2,P,P1) :- eq(N,6), eq(T3,4), eq(T2,2),P1 is P + 100.
prism(N,T3,T2,P,P1) :- eq(N,5), eq(T3,2),eq(T2,3),P1 is P + 50.
prism(N,T3,T2,P,P1) :- eq(N,4), eq(T2,4),P1 is P + 15.
prism(N,T3,T2,P,P1) :- eq(N,3), eq(T2,3),P1 is P + 50.


lshape(N,T3,T2,P,P1) :- eq(N,6), eq(T3,2), eq(T2,4),P1 is P + 25.
lshape(N,T3,T2,P,P1) :- eq(N,6), eq(T2,6),P1 is P + 100.
lshape(N,T3,T2,P,P1) :- eq(N,4), eq(T2,4),P1 is P + 15.
lshape(N,T3,T2,P,P1) :- eq(N,9), eq(T3,4), eq(T2,5),P1 is P + 100.
lshape(N,T3,T2,P,P1) :- eq(N,10), eq(T3,6), eq(T2,4),P1 is P + 50.
lshape(N,T3,T2,P,P1) :- eq(N,11), eq(T3,8), eq(T2,3),P1 is P + 100.
lshape(N,T3,T2,P,P1) :- eq(N,11), eq(T3,6), eq(T2,5),P1 is P + 100.

tshape(N,T3,T2,P,P1) :- eq(N,6), eq(T3,2), eq(T2,4),P1 is P + 25.
tshape(N,T3,T2,P,P1) :- eq(N,8), eq(T3,4), eq(T2,4),P1 is P + 100.
tshape(N,T3,T2,P,P1) :- eq(N,4), eq(T2,4),P1 is P + 15.
tshape(N,T3,T2,P,P1) :- eq(N,8), eq(T2,8),P1 is P + 100.
tshape(N,T3,T2,P,P1) :- eq(N,10), eq(T3,6), eq(T2,4),P1 is P + 50.
tshape(N,T3,T2,P,P1) :- eq(N,10), eq(T3,2), eq(T2,8),P1 is P + 100.
tshape(N,T3,T2,P,P1) :- eq(N,12), eq(T3,6), eq(T2,6),P1 is P + 100.
tshape(N,T3,T2,P,P1) :- eq(N,13), eq(T3,6), eq(T2,7),P1 is P + 100.
tshape(N,T3,T2,P,P1) :- eq(N,14), eq(T3,6), eq(T2,8),P1 is P + 100.
tshape(N,T3,T2,P,P1) :- eq(N,14), eq(T3,10), eq(T2,4),P1 is P + 100.
tshape(N,T3,T2,P,P1) :- eq(N,15), eq(T3,10), eq(T2,5),P1 is P + 100.


obj1(P) :-
        initialise(N,T4,T3,T2),
        cube(N,T3,T2,P,P1),
        phh([object,:,cube,>,prob,=,P1]).

obj2(P) :-
        initialise(N,T4,T3,T2),
        pyrmd(N,T4,T3,T2,P,P1),
        phh([object,:,pyramid,>,prob,=,P1]).

obj3(P) :-
        initialise(N,T4,T3,T2),
        prism(N,T3,T2,P,P1),
        phh([object,:,prism,>,prob,=,P1]).


obj4(P) :-
        initialise(N,T4,T3,T2),
        lshape(N,T3,T2,P,P1),
        phh([object,:,lshape,>,prob,=,P1]).
```

```
obj5(P) :-
        initialise(N,T4,T3,T2),
        tshape(N,T3,T2,P,P1),
        phh([object,:,tshape,>,prob,=,P1]).



obj(P) :- obj1(P).
obj(P) :- obj2(P).
obj(P) :- obj3(P).
obj(P) :- obj4(P).
obj(P) :- obj5(P).

object :- init(P),obj(P).
```

END

DTIC

9—86