NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AFFINE INVARIANT MATCHING
OF NOISY OBJECTS

by

Chang-Lung Kao

December 1989

Thesis Advisor Chin-Hwa Lee

Unclassified

security classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification: Unclassified			1b Restrictive Markings
2a Security Classification Authority			3 Distribution Availability of Report: Approved for public release; distribution is unlimited.
2b Declassification Downgrading Schedule			
4 Performing Organization Report Number(s)			5 Monitoring Organization Report Number(s)
6a Name of Performing Organization: Naval Postgraduate School	6b Office Symbol (*if applicable*): 62		7a Name of Monitoring Organization: Naval Postgraduate School
6c Address (*city, state, and ZIP code*): Monterey, CA 93943-5000			7b Address (*city, state, and ZIP code*): Monterey, CA 93943-5000
8a Name of Funding Sponsoring Organization	8b Office Symbol (*if applicable*)		9 Procurement Instrument Identification Number
8c Address (*city, state, and ZIP code*)			10 Source of Funding Numbers: Program Element No / Project No / Task No / Work Unit Accession No

11 Title (*include security classification*): AFFINE INVARIANT MATCHING OF NOISY OBJECTS

12 Personal Author(s): Chang-Lung Kao

13a Type of Report	13b Time Covered	14 Date of Report (*year, month, day*)	15 Page Count
Master's Thesis	From To	December 1989	98

16 Supplementary Notation: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes			18 Subject Terms (*continue on reverse if necessary and identify by block number*)
Field	Group	Subgroup	Affine Transformation; Affine Invariant Matching; Hashing

19 Abstract (*continue on reverse if necessary and identify by block number*)

In computer vision many techniques have been developed for object recognition. The affine invariant matching algorithm proposed by Hummel and Wolfson (1988) is a new and interesting method. Under affine invariant transformation, objects with translation, rotation, scale changes, and or even partial occlusion will have the same or similar coefficients. However, some serious problems exist in the original algorithm.

This thesis begins with the discussion of the affine transformation. The shortcomings that can occur in this method such as the basis instability, the collision of hash table, and the noise sensitivity will be discussed. Among them the noise sensitivity is a serious problem. This can always cause the recognition procedure to fail. In this thesis an improved affine invariant matching algorithm was developed to overcome the noise problem and other disadvantages of the original algorithm.

The area test criteria were adopted to avoid the numerical instability problem. The modified hashing structure using a special hash function was implemented to achieve faster accessing and voting. In the recognition procedure, the partial voting technique with the consideration of false peaks from the voting array highly enhanced the noise tolerance of the algorithm. Finally, the results obtained from the improved algorithm clearly showed better performance than those of the original algorithm.

20 Distribution Availability of Abstract	21 Abstract Security Classification	
☒ unclassified unlimited ☐ same as report ☐ DTIC users	Unclassified	
22a Name of Responsible Individual: Chin-Hwa Lee	22b Telephone (*include Area code*): (408) 646-2190	22c Office Symbol: 62Le

DD FORM 1473,84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

security classification of this page

Unclassified

Approved for public release; distribution is unlimited.

Affine Invariant Matching
of Noisy Objects

by

Chang-Lung Kao
LCDR, Taiwan Republic of China Navy
B.S., Chinese Naval Academy 1980

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 1989

Author: Chang-Lung Kao

Chang-Lung Kao

Approved by:

Chin-Hwa Lee, Thesis Advisor

Murali Tummala, Second Reader

John P. Powers, Chairman,
Department of Electrical and Computer Engineering

ABSTRACT

In computer vision many techniques have been developed for object recognition. The affine invariant matching algorithm proposed by Hummel and Wolfson (1988) is a new and interesting method. Under affine invariant transformation, objects with translation, rotation, scale changes, and or even partial occlusion will have the same or similar coefficients. However, some serious problems exist in the original algorithm.

This thesis begins with the discussion of the affine transformation. The shortcomings that can occur in this method such as the basis instability, the collision of hash table, and the noise sensitivity will be discussed. Among them the noise sensitivity is a serious problem. This can always cause the recognition procedure to fail. In this thesis an improved affine invariant matching algorithm was developed to overcome the noise problem and other disadvantages of the original algorithm.

The area test criteria were adopted to avoid the numerical instability problem. The modified hashing structure using a special hash function was implemented to achieve faster accessing and voting. In the recognition procedure, the partial voting technique with the consideration of false peaks from the voting array highly enhanced the noise tolerance of the algorithm. Finally, the results obtained from the improved algorithm clearly showed better performance than those of the original algorithm.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I wish to express my thanks to my Thesis Advisor, Dr. Chin-Hwa Lee, for his understanding, infinite patience, and guidance. I would also like to thank Dr. Murali Tummala who provided a lot of help and enthusiasm when it was greatly needed. Last, but by no means least, I would like to extend deep appreciation to my family and friends who gave their total support throughout the entire project

I. INTRODUCTION

Recognition of objects in natural scenes is a major subject of study in computer vision. It is important in many applications including robotics, automated inspection, and aerial photo-interpretation. A number of practical recognition systems are model-based systems where the task is to match known models against the test objects in the scene.

Generally, according to the matching scheme the techniques for 2-D object recognition can be classified into three categories. They are template matching, feature matching, and transform matching. Template matching is the oldest technique known. It is basically a technique of 2-D cross-correlation between the model object and the test object. In feature matching a structural model of the object in terms of features or object primitives is constructed. Recognition is based on the structure. These features can be served as a compact description of the object and can be easily extracted from it. The above two matching techniques are performed in the original 2-D spatial domain. The last approach is to transform the original data into a different domain with the matching done in this new domain. [Ref. 1: pp. 14-20]

The representation method of an object determines the complexity of the matching algorithm and also places a limit on the capability of the matching algorithm. For example, a representation based on the Fourier Descriptors can not handle the partially occluded objects because of its global Fourier feature. Therefore, an ideal object representation shall have the following desirable characteristics [Ref. 2] :

- It should be local. This means that the coding of an object is not dependent on the entire boundary, nor is it dependent on an external reference point such as a centroid or a starting point.

- It should be independent of the orientation, scale, and even partial occlusion of the object.
- It should be bounded. This means that a small change to part of the boundary should produce only a small local change in the representation.
- It should allow for efficient and robust matching in the presence of noise.
- It should uniquely specify a single object.
- It should contain information about the object at varying levels of details so that the matching process can be performed at different levels of coarseness.
- It should be computationally efficient.

These characteristics of object representation are ideal, and it is by no means obvious from the outset that a representation with such characteristics can be found.

Many techniques were developed for object representation and matching in the above three categories, but none of them are simultaneously orientation, scale, and occlusion invariant. The affine invariant matching proposed by Hummel and Wolfson (1988) is a new and efficient transform matching technique which can handle objects with changes in scale, orientation, and partial occlusion. However, this technique has some shortcomings.

The purpose of this thesis is to discuss these shortcomings and derive the solutions to overcome the corresponding problems. There are five chapters included in this thesis. Chapter I is a brief description of 2-D object recognition and the ideal representation characteristics. Chapter II presents the principle of affine invariant transformation and its original implementation on point matching. In Chapter III the emphasis is on the

discussion of the shortcomings of the original matching algorithm such as the basis instability, the collision of hash table, and the noise problem. Then, solutions were derived to overcome these problems. Among these shortcomings, the main weakness of the original matching algorithm is its sensitivity to noise perturbation. This will always cause the recognition procedure to fail. One side effect caused by the solution to reduce noise perturbation is that the matching algorithm suffers from false peaks in the accumulator array due to the partial voting technique. All the problems and solutions will be discussed in detail in this chapter. In Chapter IV, these solutions were implemented in the improved affine matching algorithm in order to enhance some of the ideal characteristics of the object representation and matching algorithm. Hence, the new algorithm can work better in a real situation. In other words, it can handle noisy test objects and still preserve scale, orientation, and occlusion invariance. Chapter V shows the results obtained from the computer experiments. By comparing the results reported here with the results of the original algorithm, the new algorithm clearly shows better performance than the original method. Finally, Chapter VI presents the conclusions and the recommendations for future study.

II. DISCUSSION ON AFFINE INVARIANT MATCHING

A. AFFINE TRANSFORMATION OF PLANAR POINTS

1. General Statement

From a theoretical as well as a practical point of view, feature transformation is an important topic in object recognition. The affine transformation is a new method to characterize objects with invariance in respect to translation, rotation, scale change, and partial occlusion. But the most important characteristic of all is the simplicity of this method.

The study is concentrated on the recognition of 2-D flat rigid objects. Under this restriction, two different images of the same flat object are in affine correspondence, i.e., there is a non singular 2×2 matrix A and a 2-D (translation) vector $\mathbf{b}$, such that each point $\mathbf{x}$ in the first image can be translated to the corresponding point $A\mathbf{x} + \mathbf{b}$ in the second image. Our problem is to find the correspondence of the objects in a scene and a stored model in the data base. [Ref. 3: pp. 3-4]

In order to identify the occluded or overlapped objects, a local feature must be used. Local features depend only on the shape or structure of the object restricted to a small area. In the study here the local point feature is used.

2. Transform Representation

It is well known that any three non-collinear points (called a *triplet* or a *basis triplet*) can uniquely specify a plane. Consequently, affine transformation of a plane is associated with the transformation of three non-collinear points. Moreover, the affine transformation is unique because it maps any non-collinear triplet in one plane to another non-collinear triplet in the other plane.

Suppose a set of points (called *interesting points*) of an object is extracted from an image. Any three non-collinear points of the set form a triplet. The *affine coefficients* (ξ, η) of the rest of the points of the object can be calculated based on this particular triplet. The most important characteristic of this transform is that for each non-collinear triplet the affine coefficients of all other points of the object are affine invariant. In other words, the same point of the object transformed on different basis will have the same affine coefficients.

Let **a**, **b**, **c** denote the vectors of a selected triplet of the plane, and **T** denotes the transformation to a different plane. As shown in Figure 1, one can express the affine coefficients (ξ, η) of an arbitrary point **p** by the following equation :

$$\begin{pmatrix} \xi \\ \eta \end{pmatrix} = (\mathbf{a}-\mathbf{c} \quad \mathbf{b}-\mathbf{c})^{-1}(\mathbf{p}-\mathbf{c}) \tag{2.1}$$

where $\mathbf{a} = \begin{pmatrix} a_x \\ a_y \end{pmatrix}$, $\mathbf{b} = \begin{pmatrix} b_x \\ b_y \end{pmatrix}$, $\mathbf{c} = \begin{pmatrix} c_x \\ c_y \end{pmatrix}$, and $\mathbf{p} = \begin{pmatrix} p_x \\ p_y \end{pmatrix}$.

Hence, given the affine coefficients and the triplet, the corresponding point in an image can be computed by

$$\mathbf{p} = \xi(\mathbf{a}-\mathbf{c}) + \eta(\mathbf{b}-\mathbf{c}) + \mathbf{c}. \tag{2.2}$$

If both the point **p** and its triplet are transformed by **T** to a new image plane, its new affine coefficients are

$$\begin{aligned} \begin{pmatrix} \xi' \\ \eta' \end{pmatrix} &= (\mathbf{T}(\mathbf{a}-\mathbf{c}) \quad \mathbf{T}(\mathbf{b}-\mathbf{c}))^{-1}\mathbf{T}(\mathbf{p}-\mathbf{c}) \\ &= (\mathbf{T}(\mathbf{a}-\mathbf{c} \quad \mathbf{b}-\mathbf{c}))^{-1}\mathbf{T}(\mathbf{p}-\mathbf{c}) \\ &= (\mathbf{a}-\mathbf{c} \quad \mathbf{b}-\mathbf{c})^{-1}\mathbf{T}^{-1}\mathbf{T}(\mathbf{p}-\mathbf{c}) \\ &= (\mathbf{a}-\mathbf{c} \quad \mathbf{b}-\mathbf{c})^{-1}(\mathbf{p}-\mathbf{c}). \end{aligned} \tag{2.3}$$

From equations (2.1) and (2.3), it is clearly proven that the affine coefficients are transform invariant [Ref 4: pp. 3-4]. Figure 1 shows the invariant property graphically.

B. ORIGINAL MATCHING ALGORITHM

As was mentioned in the previous section, given a triplet, each point of the plane can be represented as a coefficient pair (ξ, η) on this basis. Since there is no preference for any specific triplet, the set of model interesting points from the model can yield a large number of non-collinear triplets. Given m interesting points of the model at least $C_3^m P_3^3 - N$ cases should be considered, where N is the number of undesired collinear triplets. For object recognition considered here, interesting points of a test object is going to be used to match those of the stored models. If there are n points in the test object, the worst case complexity for recognition using this method is $(m \times n)^3 \times t$, where t is the complexity of verifying one point of the model against that of the test object. Again if n and t are in the same order of magnitude as m, the worst case complexity could be of order m^7. [Ref. 3: pp. 6] Such a complexity is quite unfavorable. Hence, it is necessary to divide the algorithm into two different steps. The first one is an off-line step called *preprocessing*, and the second one is an on-line step called *recognition*. The objective here is to identify the test object in a scene as one of the possible models stored. The two steps are explained briefly as follows.

1. Preprocessing (Off-line Step)

a) Represent the model objects by a set of interesting points.

b) For each non-collinear triplet, calculate the coefficients (ξ, η) of all the other model interesting points on this basis triplet.

c) Use these coefficients by hashing them into an entry of a table (called *hash table*). The entry stores all the corresponding pairs (modelno, tripletno) for those affine co-

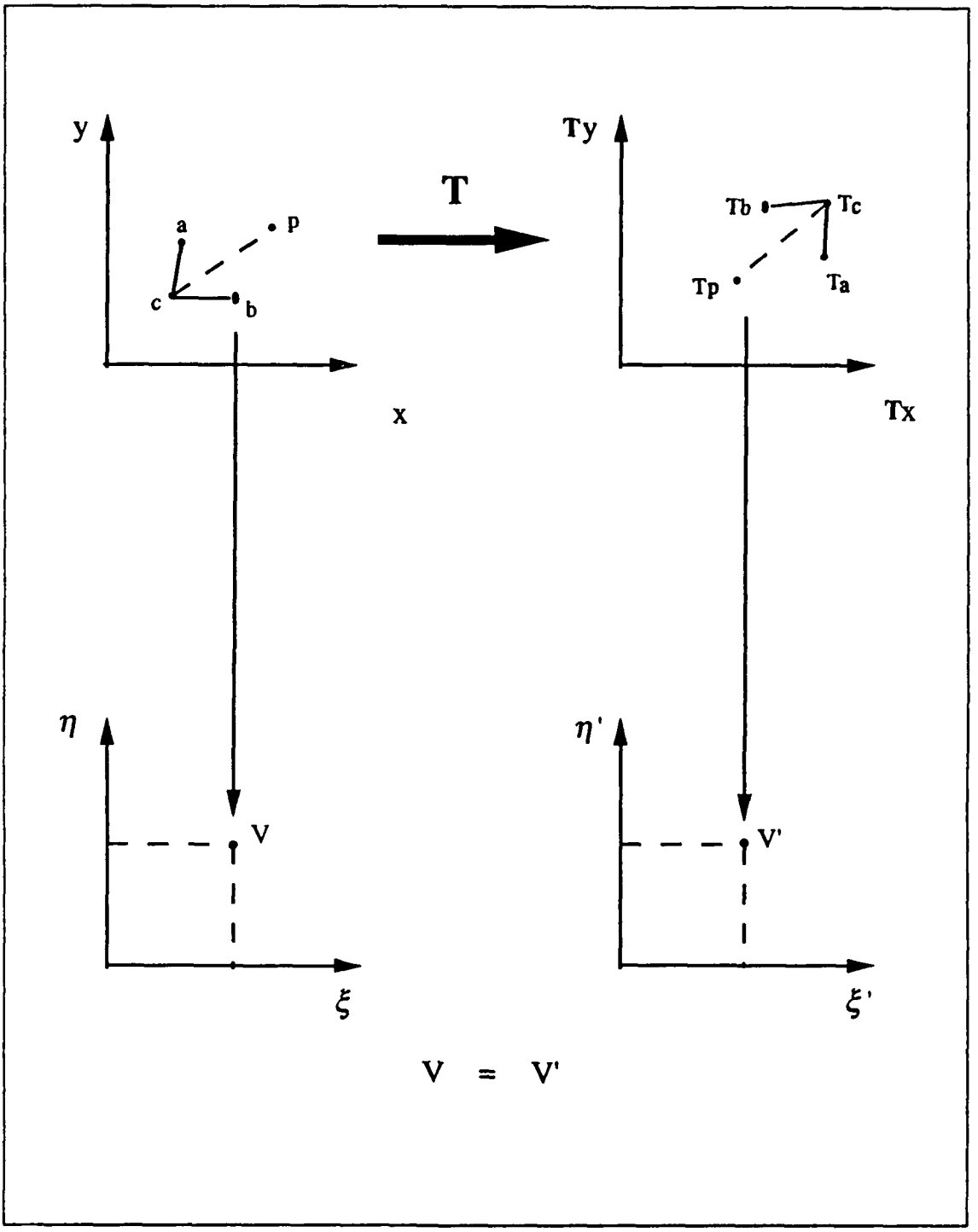

Figure 1. Affine invariant characteristic

efficients (ξ, η). Within the hashing structure all entries with the same coefficients are linked together by pointers.

d) This is done for as many models as necessary, i.e., for each different model repeat procedures a) to c).

2. Recognition (On-line Step)

a) Input a set of interesting points of the test object in an image.

b) Choose a non-collinear triplet as a basis, compute the coefficients (ξ, η) of the other points based on this triplet.

c) For each coefficient pair (ξ, η), the hash table is accessed and a vote for the appropriate entries is tallied.

d) If a certain entry in the hash table obtains the largest number of votes, the data related to this entry can be reconstructed and shown on the display.

e) Check another model as necessary, i.e., repeat procedures a) to d).

This two-step algorithm is illustrared in Figure 2.

C. SHORTCOMINGS OF AFFINE INVARIANT MATCHING

Besides the disadvantage of complexity, there are other shortcomings that limit the accuracy and efficiency of the scheme. These shortcomings are discussed in the following. The solutions to these problems will be the subject of the next section.

1. Basis Instability

In order to understand the basis instability problem it is advantageous to interpret the affine transform from the geometrical point of view. It is possible to rewrite equation (2.1) in terms of the components of the coordinates of vectors **a**, **b**, **c**, and **p** as follows

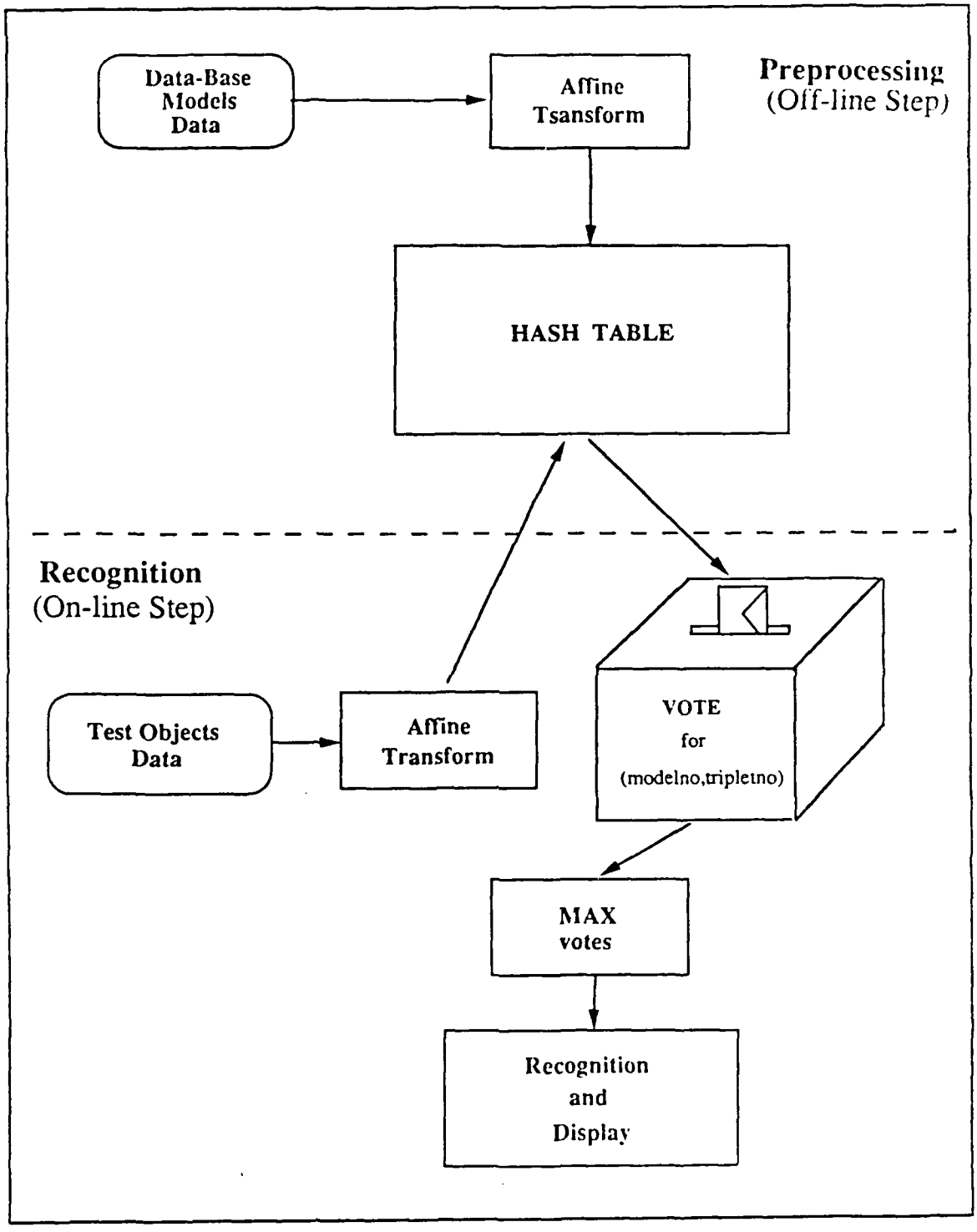

Figure 2. Flow chart of the affine invariant matching algorithm

$$\begin{pmatrix} \xi \\ \eta \end{pmatrix} = \frac{\begin{pmatrix} (p_x - c_x)(b_y - c_y) - (b_x - c_x)(p_y - c_y) \\ (a_x - c_x)(p_y - c_y) - (p_x - c_x)(a_y - c_y) \end{pmatrix}}{(a_x - c_x)(b_y - c_y) - (b_x - c_x)(a_y - c_y)} . \tag{2.4}$$

More explicity the affine coefficients are given by

$$\xi = \frac{(p_x - c_x)(b_y - c_y) - (b_x - c_x)(p_y - c_y)}{(a_x - c_x)(b_y - c_y) - (b_x - c_x)(a_y - c_y)} , \tag{2.5}$$

and

$$\eta = \frac{(a_x - c_x)(p_y - c_y) - (p_x - c_x)(a_y - c_y)}{(a_x - c_x)(b_y - c_y) - (b_x - c_x)(a_y - c_y)} . \tag{2.6}$$

If equations (2.5) and (2.6) are interpreted from the geometrical point of view, the affine coefficients ξ and η are both ratios of areas. The area of the basis triplet triangle is given by

$$A_{abc} = \left| \frac{1}{2} \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} \right| . \tag{2.7}$$

The determinant can be simplified to

$$A_{abc} = \left| \frac{1}{2} \begin{vmatrix} a_x - c_x & a_y - c_y \\ b_x - c_x & b_y - c_y \end{vmatrix} \right| \tag{2.8}$$

or

$$A_{abc} = \left| \frac{1}{2} [(a_x - c_x)(b_y - c_y) - (b_x - c_x)(a_y - c_y)] \right| \tag{2.9}$$

which is identical to the magnitude of the denominator of equations (2.5) and (2.6), except for the 1/2 factor.

In other words, the denominator of equation (2.5) or equation (2.6) is proportional to the area of the triangle for which the vertices are the three non-collinear basis points **a**, **b** and **c**. Similary, the numerator of ξ is proportional to area A_{bcp} which is the area of the triangle that consists of points **b**, **c**, and **p**. The numerator of η is proportional to A_{acp} which is the area of the triangle that consists of points **a**, **c**, and **p**. Figure 3 shows the geometrical interpretation of the affine transformation.

If the area of A_{abc} is too small compared to the area of A_{bpc} or A_{apc}, the magnitude of the affine coefficient will be a very large number. This effect will cause the numerical instability in the affine plane. [Ref. 4: pp. 6-7]

2. Collision of Hash Table

Even if the unstable triplet points are ruled out, the values of ξ and η tend to concentrate highly around zero, i.e., these values are closing toward the origin of the affine plane, as shown in Figure 4. These small values of ξ and η will generate many long linked lists in the hash table, which causes serious collisions of entries in the table. Under this circumstance, the hash table accessing time increases, and the situation becomes impractical due to the numerous collisions. Thus, a special function which can distribute these small values to a large region is needed.

3. Noise Perturbation

When the test object appears in a real scene, usually the extraction of local feature (interesting point) is inaccurate to a certain extent in the presence of random fluctuations or "noise". In this scheme the local feature used is a set of fixed-value coordinates of the interesting points. If these points are perturbated in the presence of

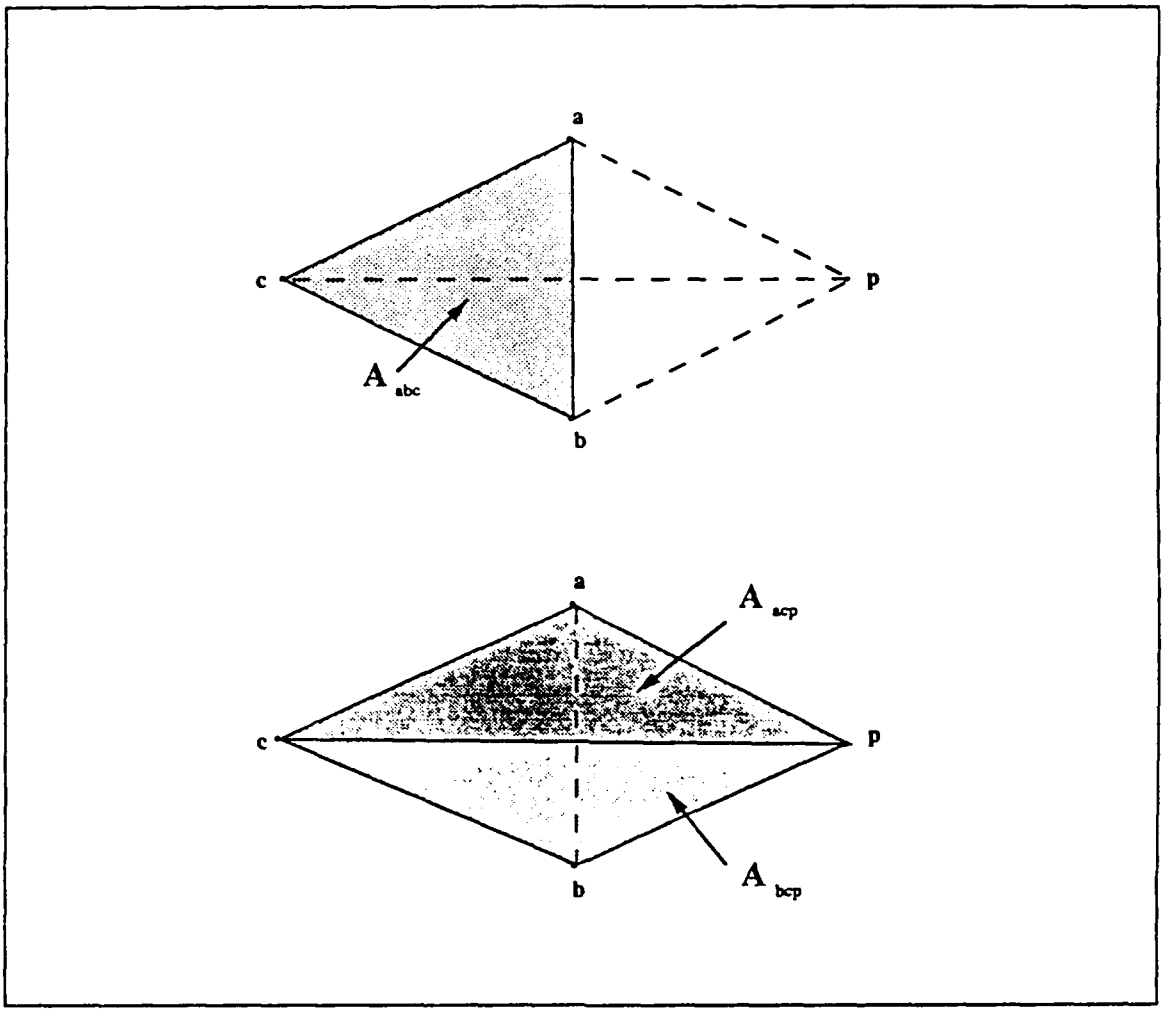

Figure 3. Geometrical interpretation of affine transformation

noise, the corresponding affine coefficients ξ and η will deviate from the accurate values in the affine plane.

In order to show how the affine coefficients are affected by the noise, it is beneficial to take a look at a simple example as shown in the following. If the coordinates of a test triplet $<\mathbf{a}, \mathbf{b}, \mathbf{c}>$ and an arbitrary point **p** are given by

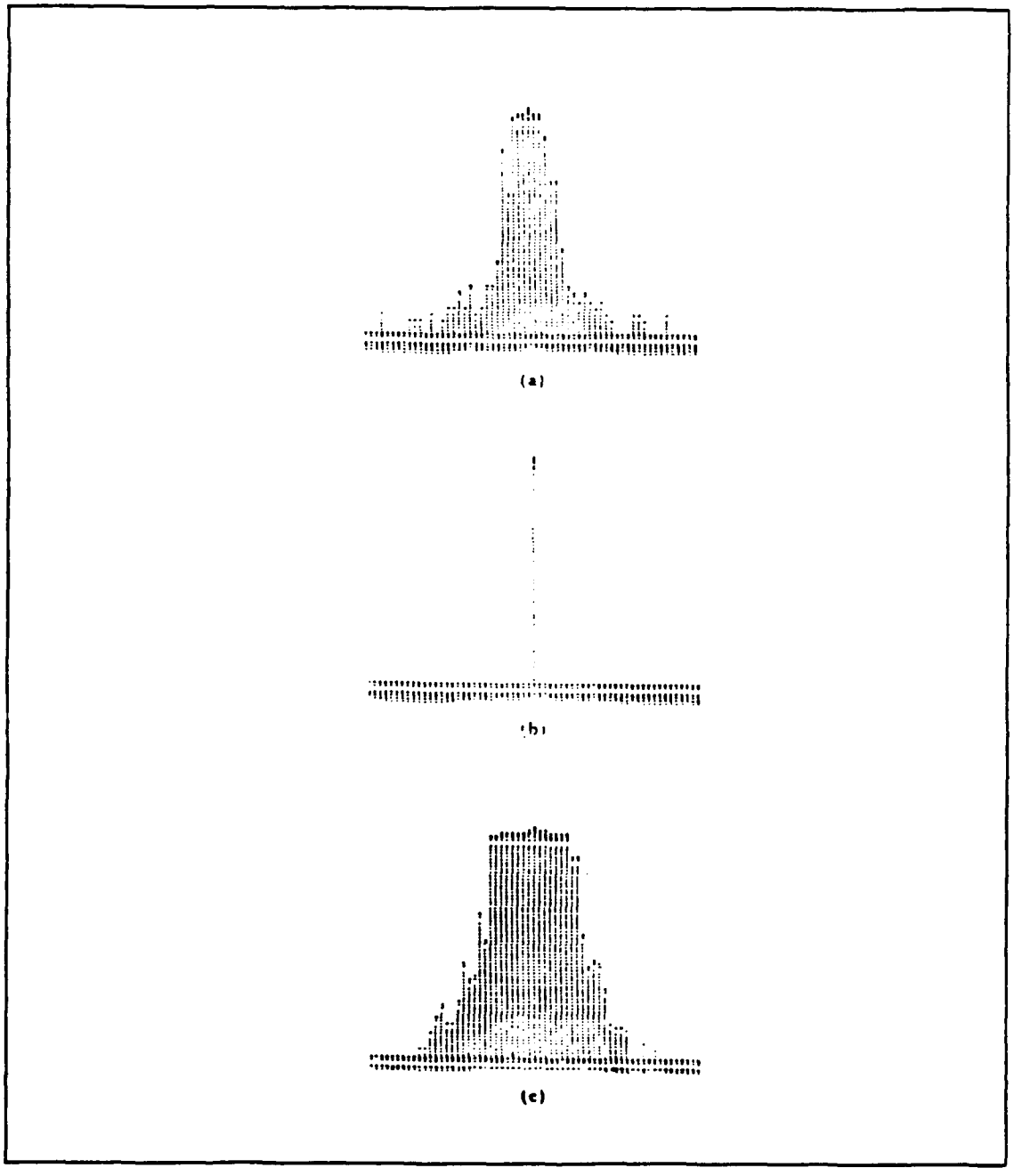

Figure 4. **Histograms of the affine coefficients:** (a) original; (b) ruling out unstable triplets; and (c) same as (b) with x axis scaled to smaller range of values. [From Ref. 3: pp. 8]

$$\mathbf{a} = \begin{bmatrix} 20 \\ 30 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 30 \\ 40 \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} 40 \\ 30 \end{bmatrix}, \quad \mathbf{p} = \begin{bmatrix} 40 \\ 10 \end{bmatrix}.$$

Using equations (2.5) and (2.6), we have

$$\xi = 1, \quad \eta = -2.$$

Assume that the point vectors **b** and **p** are perturbated by the presence of noise and that their coordinates change to

$$\mathbf{b} = \begin{bmatrix} 31 \\ 39.5 \end{bmatrix}, \quad \mathbf{p} = \begin{bmatrix} 40.5 \\ 11 \end{bmatrix}.$$

Again, by using equations (2.5) and (2.6)

$$\xi' = 0.875, \quad \eta' = -2,$$

so that the deviations

$$\Delta\xi = \xi - \xi' = 0.875, \quad \Delta\eta = \eta - \eta' = 0.$$

Since the affine coefficients of the model interesting points are integer values, the above small deviation will cause an erroneous or a failed match in the hash table.

Figure 5 shows the voting results of the accumulation array in the hash table. From this plot it can be seen that the votes of most triplets decrease due to the noise perturbation. This means that potential erroneous matching can occur in the hash table.

Obviously, the affine transformation of a point is very sensitive to noise perturbation. This will always cause the recognition procedure to fail. For real images the background noise is inevitable. Hence, the noise sensitivity of affine transform matching is the main problem to be solved in this study.

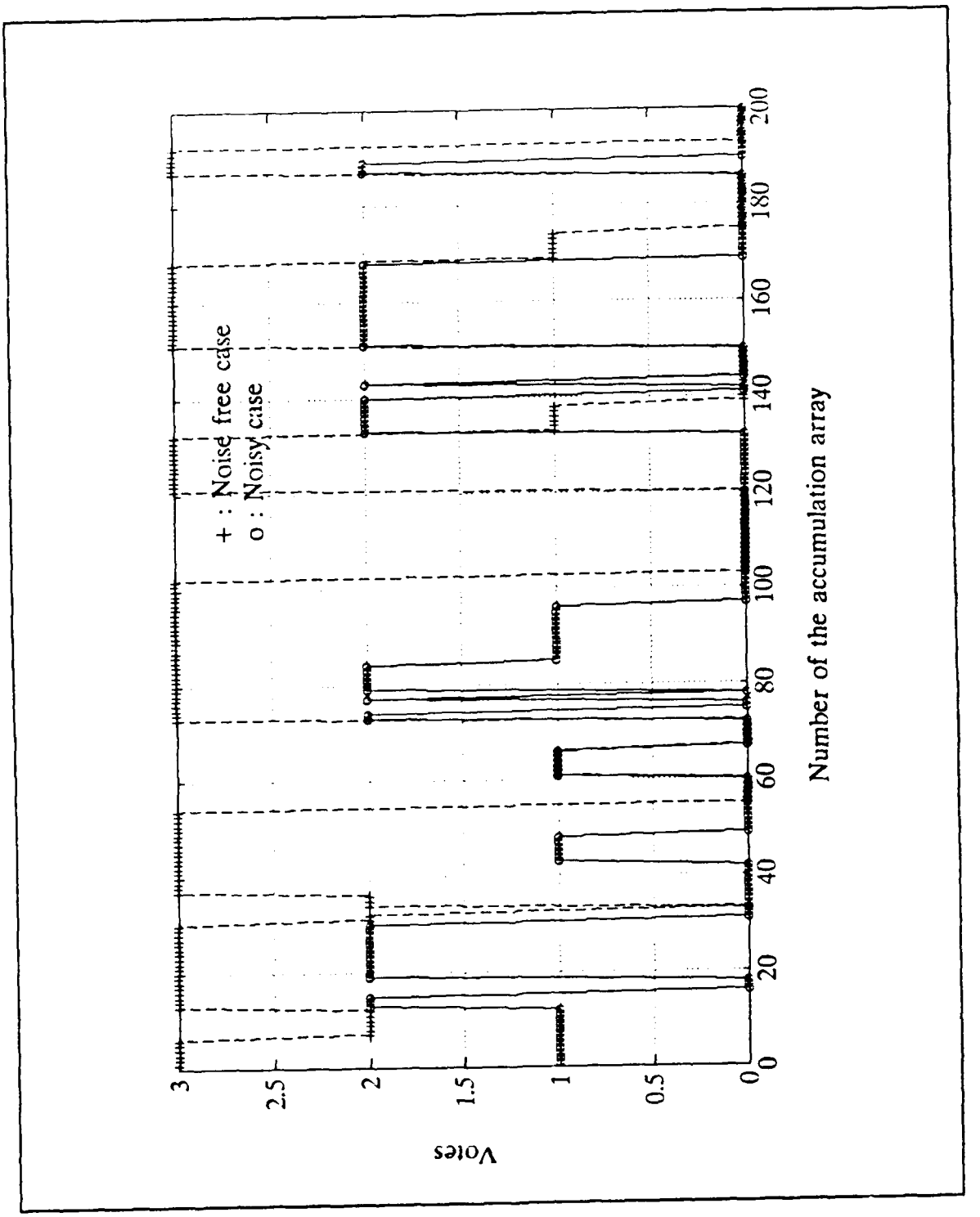

Figure 5. **Voting results of the accumulation array in the hash table**

D. SOLUTIONS TO THE PROBLEMS

This section discusses the solutions to the shortcomings of the affine matching algorithm.

1. Screening to Reduce Numerical Instability

The selection of triplets is an important step in either the preprocessing or the recognition procedures. If a triplet yielding a relatively small area is used as a basis, it will cause large amplification of noise in the affine plane, i.e., the error of ξ and η will become very large. Therefore, the solution to the instability problem is to rule out this kind of undesirable triplets.

The concept of "relatively small area" is an ambiguous idea. It can vary from object to object. It is necessary to explain this in detail. In order to get a reflexible criterion, the triplet's area is compared with the total area of the object. The following restrictions are required for both the preprocessing and the recognition [Ref. 4: pp. 8].

- In the preprocessing step, only the non-collinear triplet with an area of at least 1/20 of the total area of the object will be used.
- In the recognition step, the non-collinear triplet used should have at least 1/10 of the total area of the object. Here, adoption of the factor 1/10 instead of 1/20 is due to the consideration of noise perturbation in the recognition step. Because the input image is always affected by the background noise, a less stringent tolerance is adopted.

By ruling out these undesirable triplets, not only can the numerical instability problem of this scheme be solved, but also the bounds of coordinates in the affine plane can be determined. This results in a smaller and efficient hash table.

2. Hashing Function

From the observation of the affine coefficients it can be seen that the higher concentration occurs at small values, i.e., values of ξ and η close to the origin of the 2-D affine plane. The phenomenon of the concentration of small values of affine coefficients induces many long linked lists in the hashing table. Consequetly, the accessing time of the hash table in this algorithm deteriorates. Thus, it is intuitively beneficial to use a special hash function to scatter those small values as far apart as possible in the affine plane.

Besides the linear hash functions with bad performance, there are some non-linear hash functions with a good performance available, such as atan, asinh and tanh. Figure 6 shows these functions. From the experimental results, atan seems to be the best one as it can scatter the small affine coefficients over a wide range of the affine plane [Ref. 4: pp. 9-10]. Hence, the size of the linked list can be reduced through appropriate quantization in the affine plane. The hashing concept and the quantization problem will be discussed in the next section.

3. Noise Problem

The affine invariant matching algorithm is very rigid in calculating the coefficients ξ and η. If the coefficient pairs are based on noisy interesting points, the recognition will fail due to the error matching in the hash table. Consequently, the noise sensitivity of this matching system is a serious problem. In order to solve the disadvantage of noise sensitivity, some techniques to improve this main problem are proposed.

a. Quantization in the Affine Plane

In the original algorithm the hash table can be considered as a two dimentional array of pointers, each pointer points to the entry with the same affine coefficients. Hence, the implementation of the hash table is essentially quantization of the

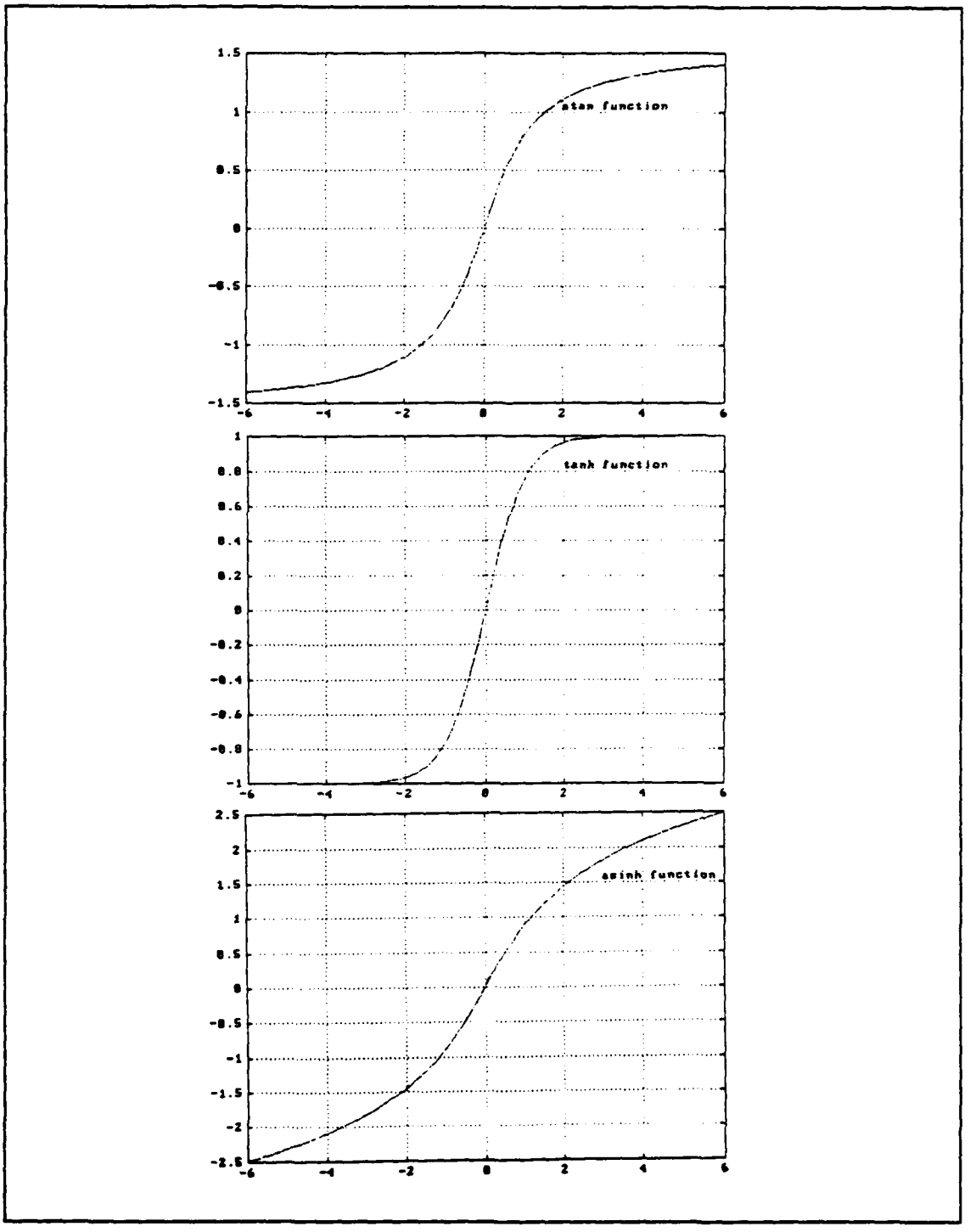

Figure 6. Hashing functions of atan, asinh, and tanh

affine plane with infinite number of bins (or buckets). Each bucket in the hash table is indexed by the quantized coefficients (ξ, η). But, the original approach has a lot of shortcomings as mentioned before. In order to tolerate the noise perturbation on the test image, a quantization with finite number of bins is necessary.

Generally speaking, the bigger the table size, the better the spatial resolution of the quantization. However, under the consideration of noise perturbation there is actually a trade-off between the total number of bins and the bucket size. Although using a smaller total number of bins can tolerate more noise perturbation, longer linked lists will increase the accessing time of the hash table. If the quantization bin size is reduced, it will be difficult to survive in the presence of noise. After trying many bucket sizes and total number of bins in different cases, the same conclusion as Costa, Haralick, and Shapirp [Ref. 4: pp. 10-11] was reached, i.e., there is no optimal hash table size that can work well for all data. The table size used here for all experiments was 90×90.

Before spatial sampling the affine plane, the bounds for both the coefficients ξ and η should be known. According to the assumptions made in the first section of this chapter, the maximun and minimum values of the affine coefficients can be calculated. Referring to Figure 7, the bounds of ξ and η can be derived as follows.

The image plane has a spatial resolution of 512×512. Points **a**, **b**, **c** make up a set of non-collinear triplet, and **p** is an arbitrary point on the same plane. In order to find the maximun and minimun bounds on ξ and η, the worst cases shown in Figure 7 are considered. Assume A is the area of the input object. The triplet is at the lower left hand corner, and the arbitrary point is at the upper right hand corner of the image plane as shown in case (a). In case (b) the position of the triplet and the arbitrary point are reversed. Let the magnitude of both the vector $\vec{ca}$ and the vector $\vec{cb}$ equal to Δ , which equals $\sqrt{\frac{A}{10}}$ due to the previous restriction. Using equation (2.5) and equation (2.6) for case (a) yields

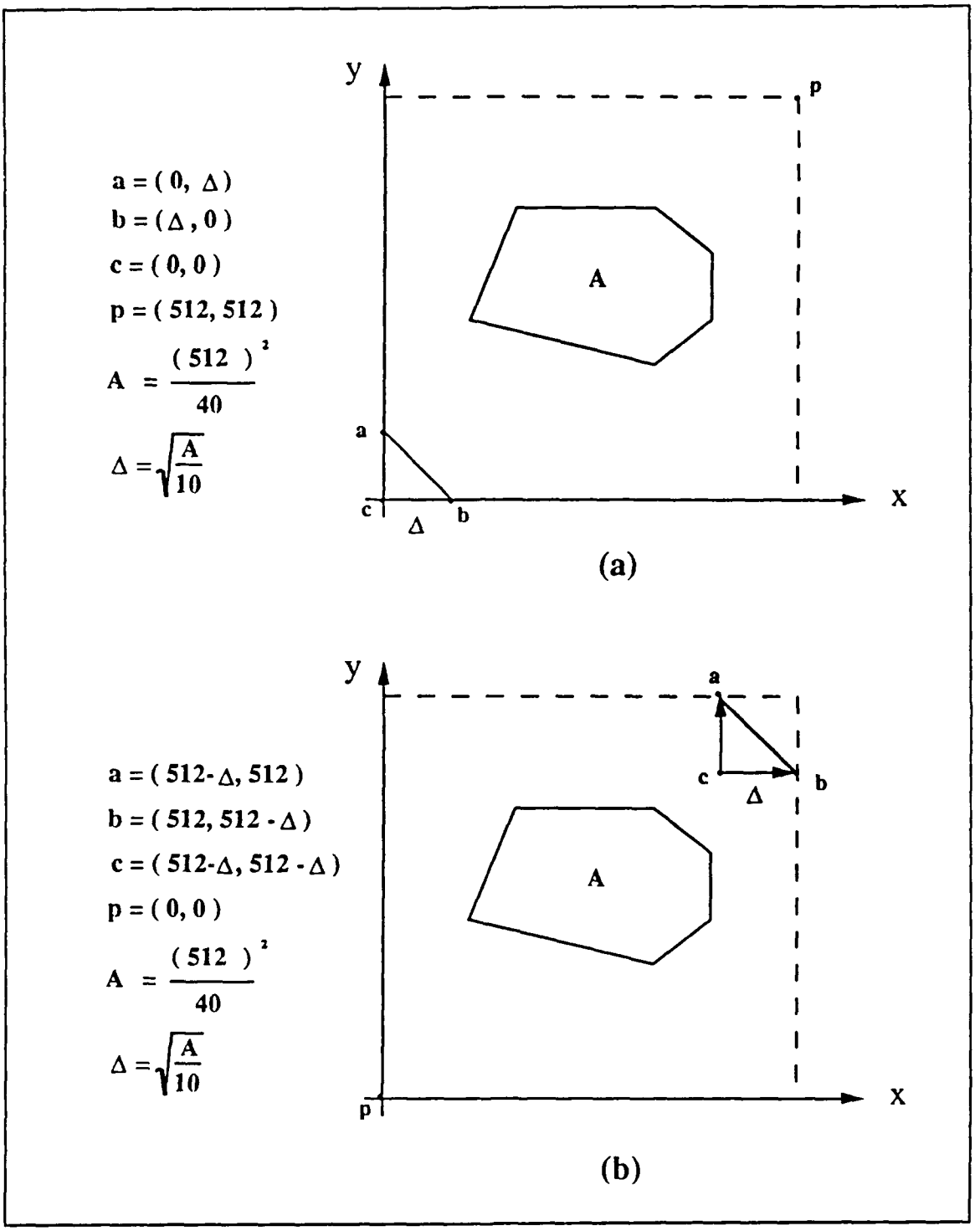

Figure 7. Calculation of the bounds of affine coefficient

$$\begin{aligned}\xi_{max} = \eta_{max} &= \frac{(512-0)(0-0)-(\Delta-0)(512-0)}{(0-0)(0-0)-(\Delta-0)(\Delta-0)} \\ &= \frac{512}{\Delta} \\ &= 512\sqrt{\frac{10}{A}} \\ &\doteq \frac{1619}{\sqrt{A}}\end{aligned} \tag{2.10}$$

and

$$\begin{aligned}\xi_{min} = \eta_{min} &= \frac{(0-0)(512-0)-(512-0)(\Delta-0)}{(0-0)(0-0)-(\Delta-0)(\Delta-0)} \\ &= \frac{-512}{\Delta} \\ &= -512\sqrt{\frac{10}{A}} \\ &\doteq \frac{-1619}{\sqrt{A}}\,.\end{aligned} \tag{2.11}$$

Similarly results for case (b) can be obtained. Hence, the bounds

$$\xi_{min} = \frac{-1619}{\sqrt{A}} \le \xi \le \frac{1619}{\sqrt{A}} = \xi_{max} \tag{2.12}$$

$$\eta_{min} = \frac{-1619}{\sqrt{A}} \le \eta \le \frac{1619}{\sqrt{A}} = \eta_{max}. \tag{2.13}$$

Having the bounds on ξ, η and the 90 × 90 table size, it is possible to compute the bin width $\Delta\xi$ and $\Delta\eta$ as follows

$$\Delta\xi = \frac{2\ atan(\frac{1619}{\sqrt{A}})}{90} \tag{2.14}$$

$$\Delta\eta = \frac{2\ atan(\frac{1619}{\sqrt{A}})}{90}\,. \tag{2.15}$$

b. Improved Hash Structure

Hashing is an efficient technique for address calculation. Sometimes, it is also called a *randomizing* or *scrambling* method. In this technique the item's feature is converted into a near-random number, and this number is used to determine where the item is going to be stored. Within the hash table every item is associated with an affine coefficient pair. By using the atan hashing function and the 90×90 spatial quantization the original entry index (ξ, η) is replaced by the pair (i, j).

In the remainder we discuss how to use the index (i, j) to generate the entry key and how to convert the key into a more or less random number uniformly distributed in the hash table. In the following discussion the entry key is called an *inkey*, and the random number is called a *hashkey*.

(1) Generation of inkey. The hash table is a two dimentional structure indexed by (i, j), and an inkey is a one dimentional variable. Conversion of (i, j) to an inkey will cause collision of entries. In order to reduce this kind of collision, two different one-to-one functions with appropriate operations are used. For simplicity, the inkey can be computed in the following equation with tan 30° and tan 20° as the two factors,

$$inkey' = i \times \tan 30^{\circ} + j \times \tan 20^{\circ}. \tag{2.16}$$

The two terms in the above equation are two different one-to-one functions so that i, j in the equation can not commute with each other and keep the same value of *inkey'*. Due to this characteristic the number of collisions in the hash table can be reduced. On the other hand, the value from the above equation is a real number, and to avoid the trunction error this number is multiplied by 100 before the trunction. Hence, the equation for inkey becomes

$$inkey = trunc(inkey' \times 100). \tag{2.17}$$

(2) Generation of hashkey. Given the inkey, it can be converted to a hashkey. There are many key-to-address conversion algorithms, such as mid-square method, dividing, shifting, folding, and digit analysis ... etc. From the experimental conclusion of many researchers, the best choice is the simple division method. [Ref. 5] Using this method the inkey is divided by a prime number which is approxmately equal to or less than the number of the total hash table entries, and the remainder is taken as the relative bucket number, i.e., the hashkey. The generation of inkey and hashkey is illustrated in Figure 8.

c. Partial Voting

The noise sensitivity problem can be lessened by using a hash table where the bucket size is not very fine. The table size chosen was 90×90. In addition, it is still possible to improve the matching algorithm by using the partial voting technique. In this technique eight-neighbor mesh as shown in Figure 9 is used where each bin has the corresponding voting weight. For each interesting point whose quantized affine index is (i, j), one vote is counted for the entries (modelno, tripletno) associated with the bucket (i, j) and all the eight neighbors of that particular bucket will also receive partial votes. For example, entries associated with bucket (i-1, j-1), (i-1, j+1), (i+1, j-1) and (i+1, j+1) will each receive a 0.25 vote; entries for bucket (i-1, j), (i, j-1), (i, j+1) and (i+1, j) will each receive a 0.5 vote [Ref. 4: pp. 11]. As a result of partial voting there will be some false peaks appearing in the two dimentional accumulation voting array. The entry which received the maximun votes may not be the one of interest. In order to get rid of these false peaks an additional verification procedure must be considered. The verification procedure will be discussed in the implementation of the algorithm.

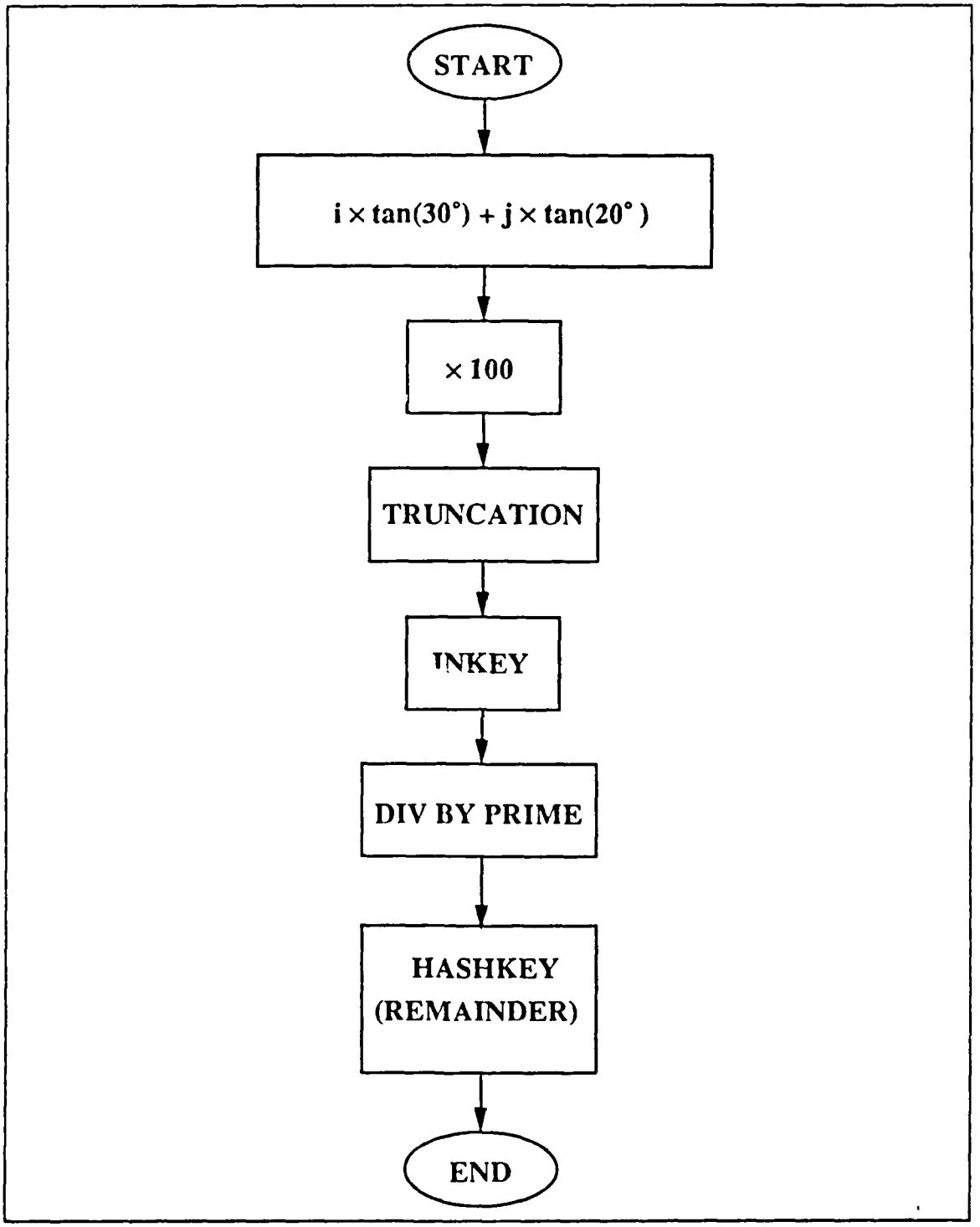

Figure 8. Generation of inkey and hashkey

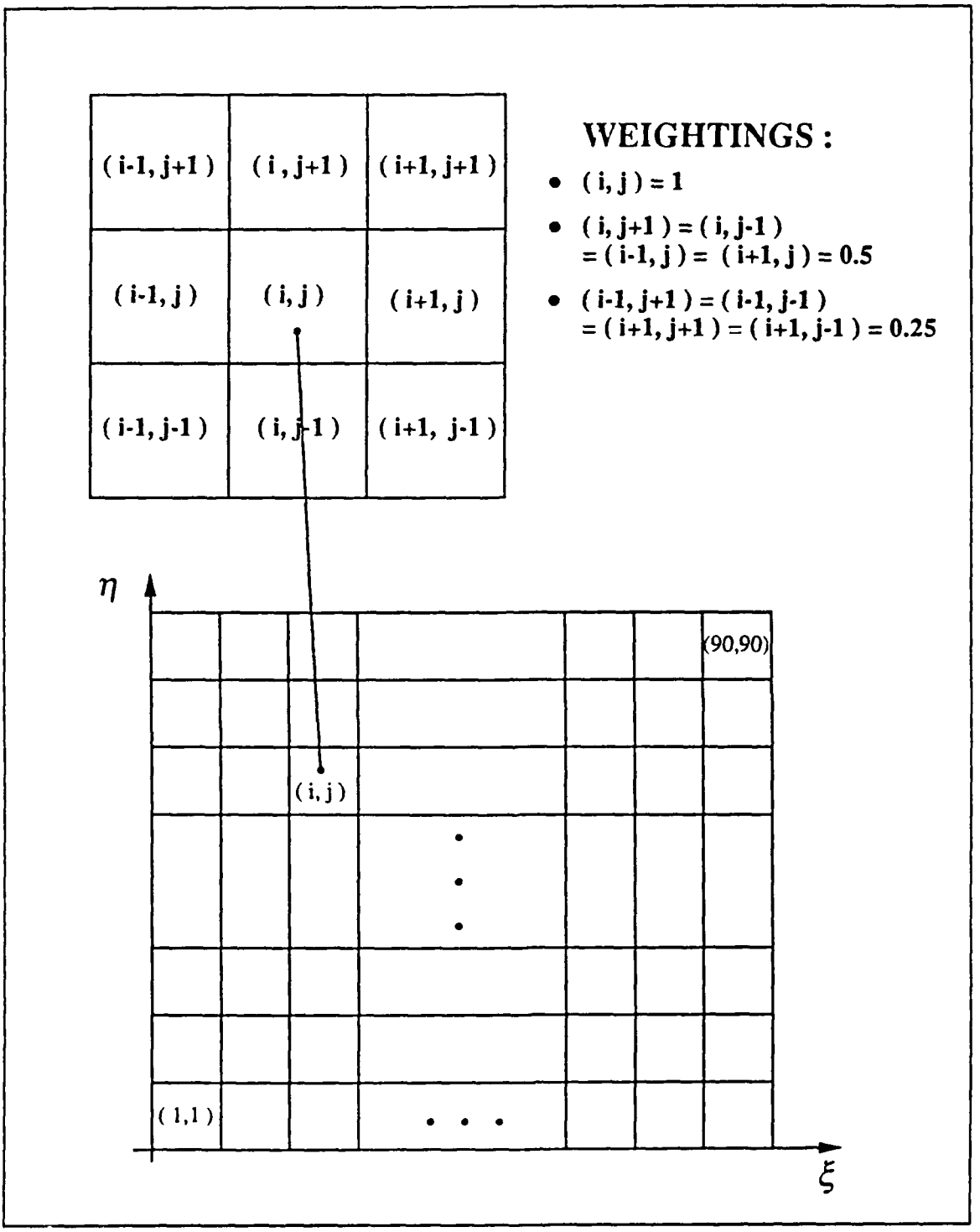

Figure 9. Partial voting technique

III. ALGORITHM IMPLEMENTATION

From the discussion of the previous chapter the affine transformation is known to be a very effective method for object detection at different views, although a great deal of computation is needed in the process. This makes the affine transformation technique impractical unless the complexity can be reduced. Hence, the algorithm was divided into two steps. The first step is the preprocessing at which data-base representation of models is built. This step does not involve any information from the test scene. It is executed off-line before the recognition. The second step is the recognition at which the data-base is used to match the models against the test objects. With the suggested two-step scheme, it is possible to keep the complexity of on-line computations low. As discussed in the previous chapter it is necessary to solve some major problems of the algorithm, such as the basis instability, the collision of the hash table, and the noise sensitivity. The solutions to these problems were proposed. The implementation of this improved two-step algorithm is discussed in the following. The Pascal source code of all modules is listed in Appendix A.

A. PREPROCESSING (DATA-BASE SETUP)

This off-line step has four modules which are explained separately. The operations of the preprocessing step are shown in Figure 10.

1. Rule Out the Undesired Triplets

This module solves the numerical instability problem by ruling out the trouble-some triplets. The following procedures are included.

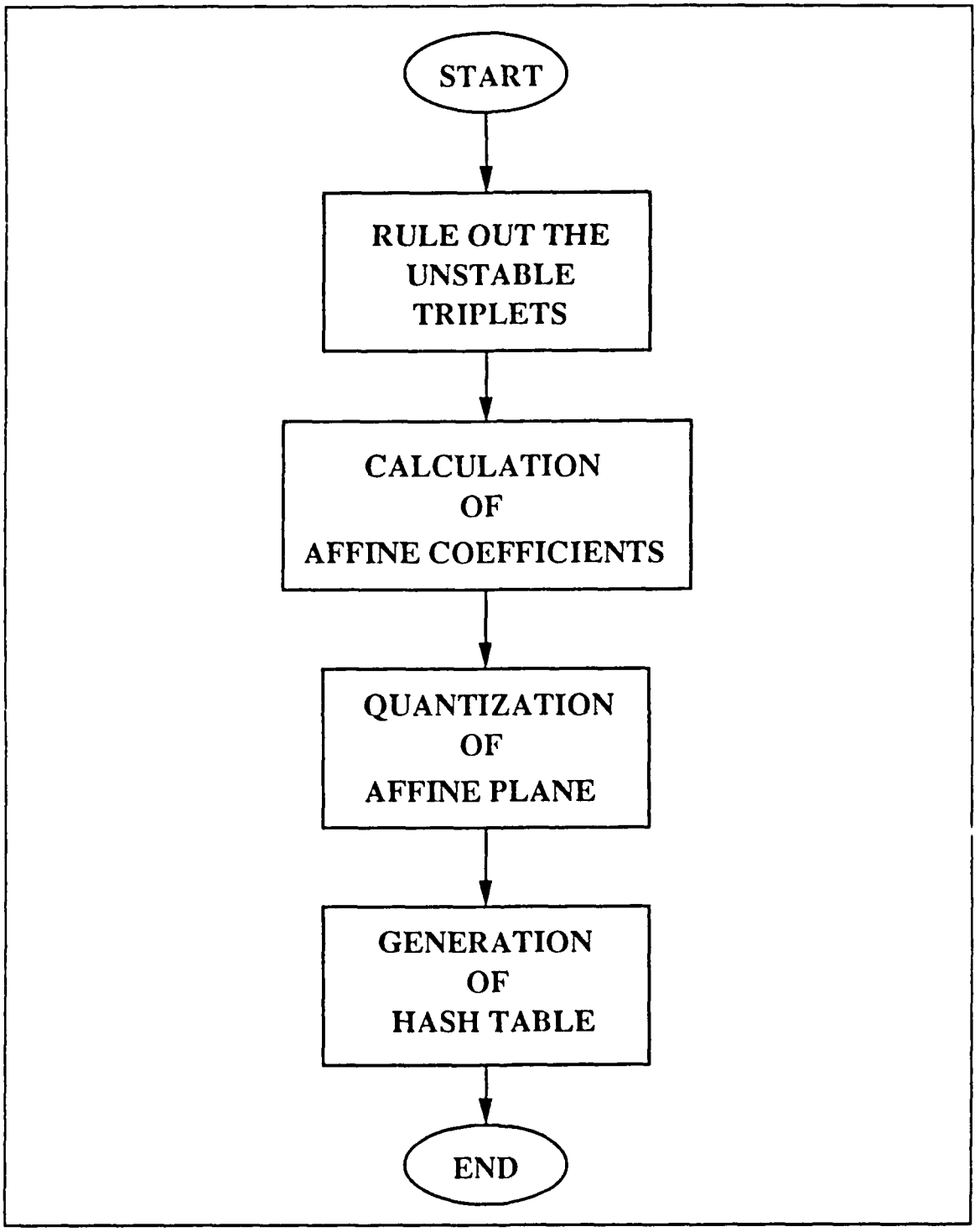

Figure 10. Flowchart of the preprocessing step

a. COMBINATION

This is a recursive procedure which generates all possible combinations of triplets from a set of input interesting points. If there are m interesting points, the total number of combinations will be

$$C_3^m = \frac{m(m-1)(m-2)}{3!}. \tag{3.1}$$

For each combination generated this procedure will call the SELECTION procedure.

b. SELECTION

This procedure screens the triplet passed from the COMBINATION procedure. If the area of this triplet is less than one twentieth of the corresponding model area, it will be bypassed and the control returned to the COMBINATION procedure for a new triplet. If the triplet area is greater than or equal to this value, then the control will go to the PERMUTATION procedure. Since a triplet passing the area test is also a non-collinear set, no additional collinearity test is necessary in this procedure.

c. PERMUTATION

The order of points of the selected triplet is important, because a different order of the same triplet can generate different affine coefficients. Hence, this procedure considers all the permutations of the selected triplet, i.e., every triplet has $P_3^3 = 6$ permutation cases.

2. Calculation of Affine Coefficients

For a given triplet $<\mathbf{a}, \mathbf{b}, \mathbf{c}>$, this module calculates the affine coefficients (ξ, η) of all other points $\mathbf{p}_i$ based on this triplet, that is

$$\begin{pmatrix} \xi_i \\ \eta_i \end{pmatrix} = \begin{pmatrix} (a_x - b_x)(b_x - c_x) \\ (a_y - b_y)(b_y - c_y) \end{pmatrix}^{-1} \begin{pmatrix} p_{x_i} - c_x \\ p_{y_i} - c_y \end{pmatrix} \tag{3.2}$$

where $i = 1, 2, 3, ..., m - 3$.

According to the above equation this module includes several matrix operation modules. The INVERSE computes the inverse matrix in the equation, and the MATRIX-MULT multiplies this inverse matrix with the other matrix to get the affine coefficients ξ and η.

3. Quantization of the Affine Plane

In order to tolerate the noise perturbation, a 90 × 90 table-size was used based on the previous discussion. This means that the affine plane can be divided into 8100 different bins, where each bin is indexed by (i, j). Two procedures are called in this module, namely, SCATTER and QUANTIZATION.

a. SCATTER

Before quantization, an atan was used as the hashing function. This special function can scatter the affine coefficients ξ and η over a wide range in the affine plane to avoid the collision in hash table due to the concentration of affine coefficients. So, the scattered affine coefficients are

$$\tilde{\xi} = atan(\xi) \tag{3.3}$$

$$\tilde{\eta} = atan(\eta). \tag{3.4}$$

b. QUANTIZATION

From equations (2.12), (2.13), and the assumed values of A, it is possible to calculate the bounds on the affine coefficients. Since a 90 × 90 table-size was used, the quantization index (i, j) for all affine coefficient pairs (ξ, η) can be calculated as in the following :

$$k1 = \frac{\tilde{\xi}}{\Delta_{\xi}} \tag{3.5}$$

$$k2 = \frac{\tilde{\eta}}{\Delta\eta} \tag{3.6}$$

where $\Delta\xi = \Delta\eta = \dfrac{2\ atan(\frac{1619}{\sqrt{A}})}{90}$,

for $k1 \geq 0$

$$i = trunc(k1) + 46 \tag{3.7}$$

for $k1 < 0$

$$i = trunc(k1) + 45 \tag{3.8}$$

for $k2 \geq 0$

$$j = trunc(k2) + 46 \tag{3.9}$$

for $k2 < 0$

$$j = trunc(k2) + 45. \tag{3.10}$$

4. Hash Table Generation

The purpose of this module is to create a hash table which can be stored in the data-base to identify the test objects from the scene. There are three procedures in this module as detailed below.

a. KEY-CONVERSION

Since the two-dimensional index (i, j) is not a numeric key, it can be converted into a numeric form for the convenience of implementation. The conversion should be done with the least collision of entry and without losing information in the key. Equation (3.11) was selected to meet the above requirements. This procedure uses it to generate the inkey.

$$inkey = trunc[(i \times \tan 30° + j \times \tan 20°) \times 100]. \tag{3.11}$$

In a good hash table, the inkey should be distributed as evenly as possible over the range of the table size. The KEY-CONVERSION procedure uses the simple method of division to convert the inkey to a hashkey, i.e.,

$$hashkey = inkey \ \ rem \ \ N. \tag{3.12}$$

where rem is the operation of reminder and N is the prime number that is approximately equal to or less than the size of the hash table.

b. HASHING

To setup the hash table, all the records of interesting points have to be put into the table as entries and each record with the same inkey will be linked by pointers. The HASHING procedure checks to see whether the entry in the hash table accessed by the hashkey is empty. If it is, then insertion of this input record into the hash table will be done right away. Otherwise, it will invoke the COLLISION procedure.

c. COLLISION

The link field of every entry in the hash table indicates whether the entry is occupied or not. If the entry is empty, its link field was set to -1. If the accessed entry in the hash table is not empty, this procedure will find the next available slot for the input record. Then, it will put the relative address (hashkey) of the new slot into the link

field of the previous one which has the same inkey. After all the records are entered, the hash table linked by pointers is established. Figure 11 illustrates the structure of the hash table.

B. RECOGNITION (IDENTIFICATION OF TEST OBJECTS)

After the preprocessing step is accomplished, a set of interesting points of the test objects can be processed to identify the unknown objects in the recognition step. There are three modules involved in this off-line step. The operation of the recognition step is shown in Figure 12. The following is a discussion of these steps.

1. Calculation of Input Data

This step is similar to part of the preprocessing step except that there is a difference in the hash table. Here, only input affine coefficients based on stable basis triplets need to be calculated. Since the noise perturbation is inevitable in the input image, sensitivity is an important issue here. In order to ensure the tolerance to a noisy environment, the criterion factor 1/20 of the area in the SELECTION procedure of the preprocessing step was replaced by the factor 1/10. The triplet area should be at least 1/10 of the total area of the test object, otherwise the triplet will be rejected. The output of this module is a data file called *tally*. Figure 13 shows the structure of the tally file.

2. Affine Matching

This matching module is an important part in the recognition procedure. Partial voting technique was implemented to generate all the candidate data. The following is a discussion of the procedures included in this module.

a. PARTIAL VOTING

For a given interesting point, this procedure checks all the nearby affine coefficients in the tally file. After quantization, every affine coefficient pair (ξ, η) belongs to a corresponding bucket indexed by (i, j). Therefore, the matching procedure is nothing but a mapping between the tally file and the hash table. If there is a match,

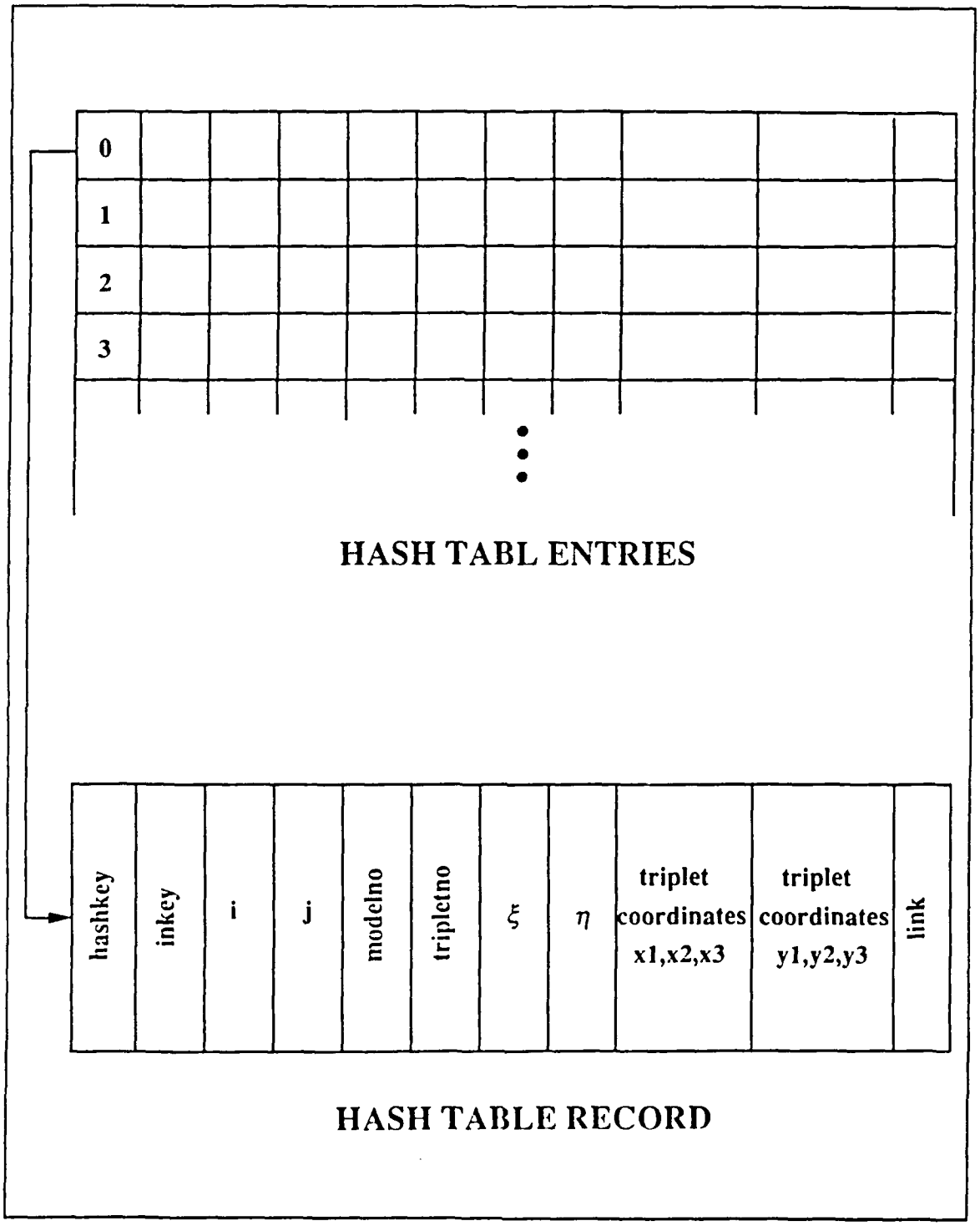

Figure 11. Structure of the hash table

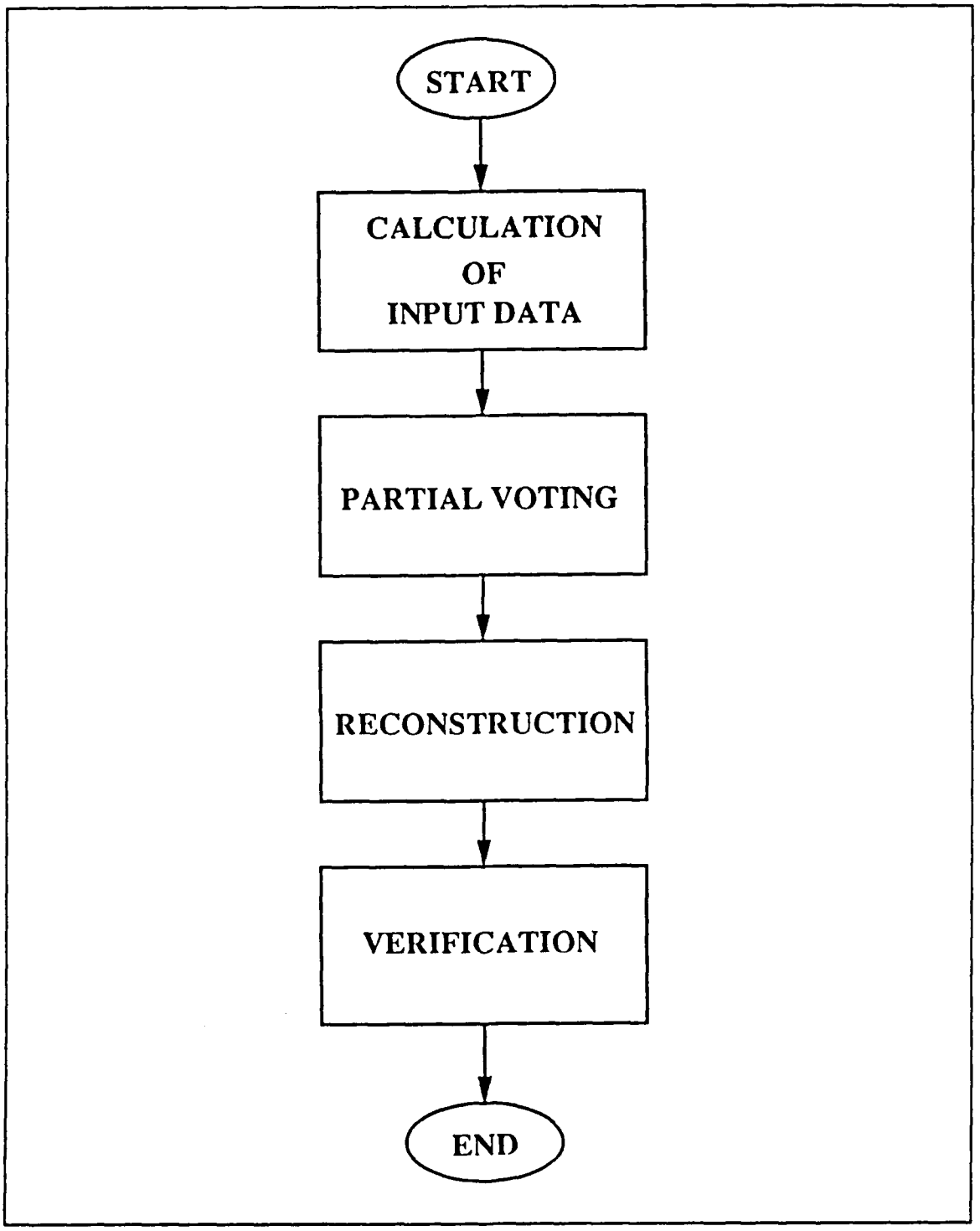

Figure 12. Flowchart of the recogniton step

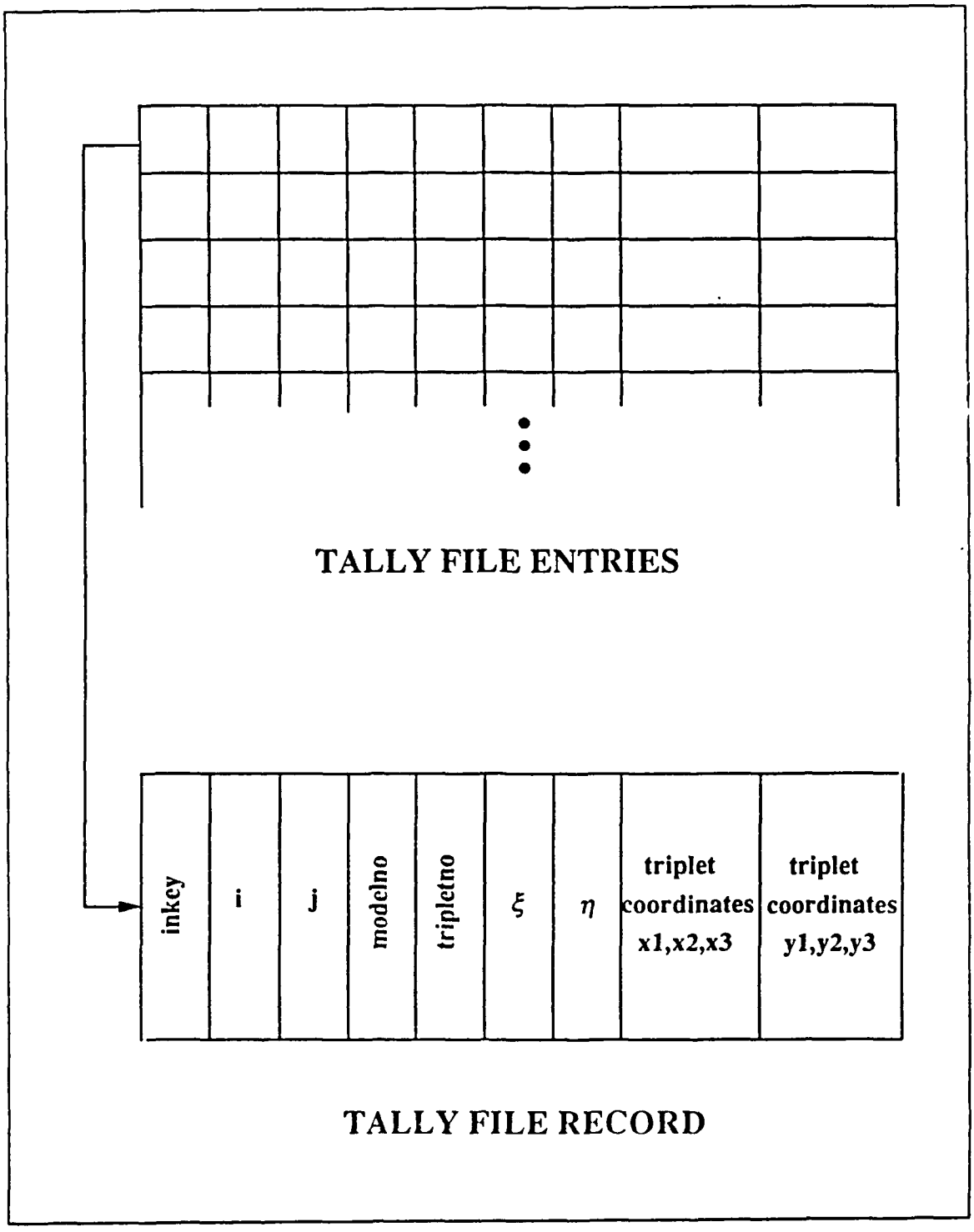

Figure 13. Structure of the tally data file

every entry associated with the bucket (i, j) will receive one vote. Since the partial voting technique was used to tolerate the noise perturbation, the eight neighbors of the bucket (i , j) will receive partial votes, i.e., entries associated with buckets (i-1, j-1), (i-1, j+1), (i+1, j-1) and (i+1, j+1) will receive a 0.25 vote; entries associated with buckets (i-1, j), (i, j-1), (i, j+1) and (i+1, j) will receive a 0.5 vote.

When all the entries in the tally file are matched in the procedure, the corresponding voting array of the affine coefficient pair will accumulate all the votes it has gotten during the campaign. In ideal noise-free case, the entry that received the maximum votes indicates an exact matching between the test object and the data-base model. But in the noisy case, the result is different. In the latter case partial voting technique was applied to reduce the noise perturbation problem. It is evident that some voting array bins will receive extra votes due to the multiple voting. Some false peaks may appear in the voting array as the by-product of the partial voting technique. Therefore, the entry which received the maximum votes may not be the desired candidate. This kind of problem will be treated in the following procedures.

b. TOP-SIX-MAX

One way to solve the false peak problem is to conduct additional tests on all the entries after partial voting. But, it is impractical because of the long processing time. Another approach is proposed based on the fact that the false entries may sometimes receive maximum vote, but the true ones should have votes close to the false peak. Based on the experimental experience, the first six highest number of votes are chosen to select the suited entries in the hash table. This procedure essentially finds the top six maximum number of votes.

c. ALL-MAX-SELECTION

In this procedure all the entries which satisfy the above criterion will be selected. Since only a small amount of entries will be checked instead of all the ones in the next procedure, this method can save a lot of operation time in the recognition step.

d. CHECK

This CHECK procedure will test all the entries sent from the MAX-SELECTION procedure. After all the redundant entries are deleted, the appropriate ones are picked by going through a TEST sub-procedure included in this module. The qualified entries can be considered as the candidate data.

3. RECONSTRUCTION

This module does the inverse affine transformation to calculate all the interesting point sets of the qualified candidate triplets.

4. VERIFICATION

Since not all the candidate interesting point sets are correct, the purpose of this module is to check all those interesting point sets until finding the correct one. Finally, the identified images will be displayed on the screen.

IV. EXPERIMENTAL RESULTS AND PERFORMANCE

Presented in this chapter are the computer simulation results and the experimental performance of the improved affine invariant matching algorithm. These experimental tests reconfirmed the affine invariance of this algorithm. It also showed the improved noise characteristic which is the main problem of the original algorithm. Since our study is concentrated on the algorithm improvement, models and test objects were generated artificially in the computer. There are two different sets of data used in the experiments. For the convenience of explanation, the first set is referred to as SET1 which consists of two simple objects, and the second one is referred to as SET2 which consists of two complicated objects. Figure 14 shows these two sets of models.

A. NOISE-FREE TESTS

First, an ideal noise-free situation is considered. The model objects in SET1 and SET2 are undergone similarity transformation. Occlusion is also evident in the test objects. The main objective is to check the affine invariant characteristics of the algorithm, i.e., rotation, translation, and scaling. The ability to detect occluded object will also be tested.

1. Similarity Transformation Test

The unknown test object of SET1 is used as the input of the algorithm. The test object and the recognized object of SET1 are shown in Figure 15. Figure 16 shows the same kind of test for SET2. This is a case where the recognition of models from the test objects with rotation, translation, and scaling are demonstrated.

2. Occlusion Test

A composite overlapping scene consists of two different test objects can be identified through the affine invariant matching algorithm. Furthermore, one vertex

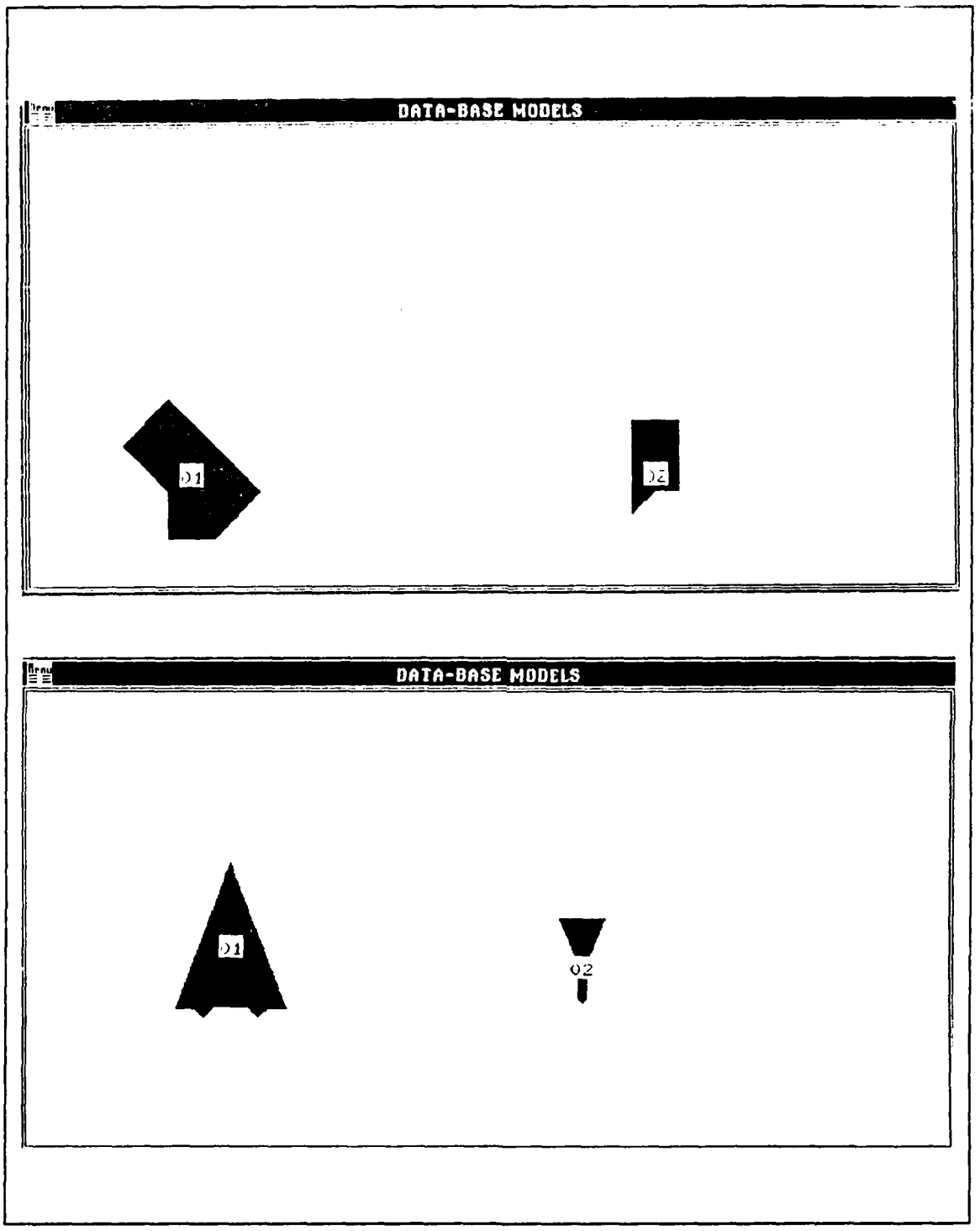

Figure 14. Data-base models of SET1(above) and SET2(below)

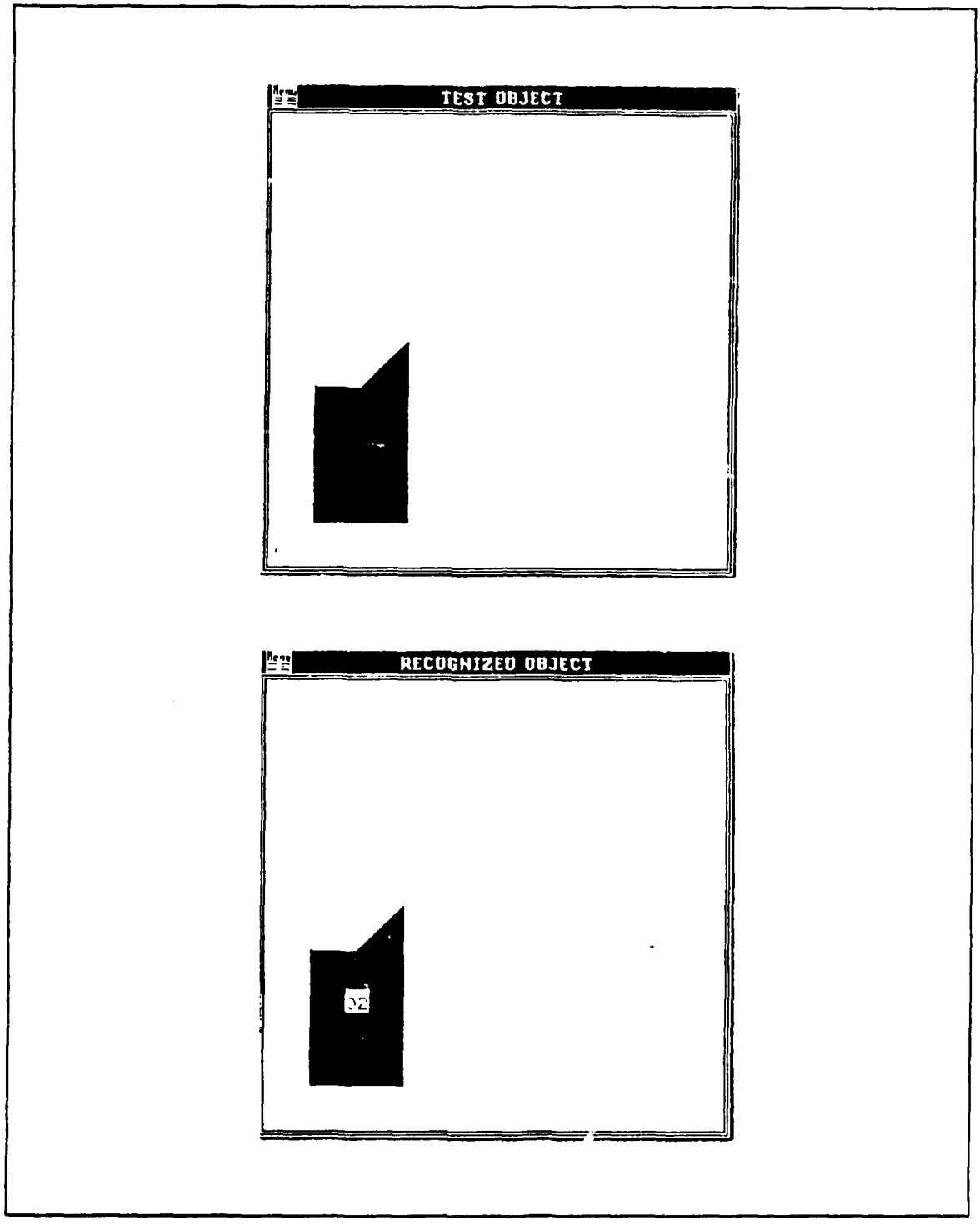

Figure 15. Similarity transform test of SET1

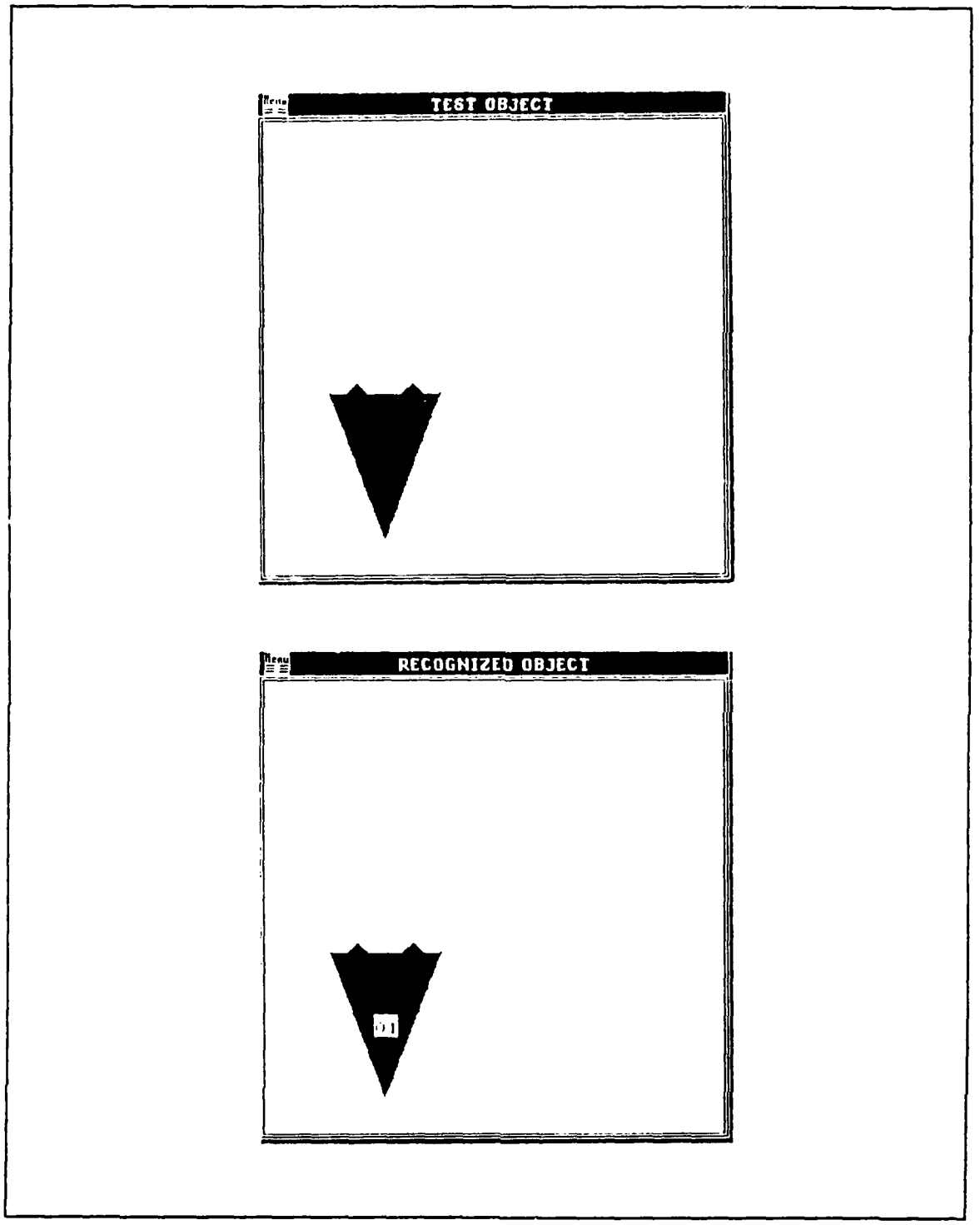

Figure 16. Similarity transform test of SET2

point of the polygonal shape is missing for the test objects of SET1 shown in Figure 17. Two polygonal vertices for the test objects of SET2 are also missing in Figure 18. The test results of SET1 and SET2 are also shown at the bottom in Figure 17 and Figure 18, respectively. From the reconstructed results of SET1 and SET2, it is obvious that the hidden vertices of the test objects in both cases can still be recognized because of the special characteristic of the algorithm.

B. NOISE PERTURBATION TESTS

In this section the test objects will be identified in scenes with noise perturbations. In usual situations a filter can be applied to remove most of the noise. But in reality, some vertex points will still have small errors after the extraction procedure. It was noted in the previous chapters that the affine invariant matching algorithm was very rigid in calculating the coefficients ξ and η. If the coordinates of some vertex points changed a little in the presence of noise, the affine coefficients based on these noisy coordinates would deviate from the true positions in the affine plane. Even if the error of the extraction is very small, it can cause the original recognition step to fail. This is due to the erroneous matching occurred in the hash table.

The tests conducted in this section showed the noise tolerance of the improved algorithm. According to the limits of the the affine plane resolution and the mask size of the partial voting, the allowable maximum error in the vertex extraction is about $\sqrt{2}$. This means that the error region of the interesting point is a circle with a center on the point and a radius equals to $\sqrt{2}$ as shown in Figure 19. The variation of the vertex may fall inside or on the circle. In the following tests only the worst cases are considered, i.e., the test objects are occluded and some of its interesting points are disturbed by the noise.

Figure 20 shows the plot of the test objects of SET1 with four interesting points disturbed. Figure 21 shows the recognition of SET1 with four interesting points dis-

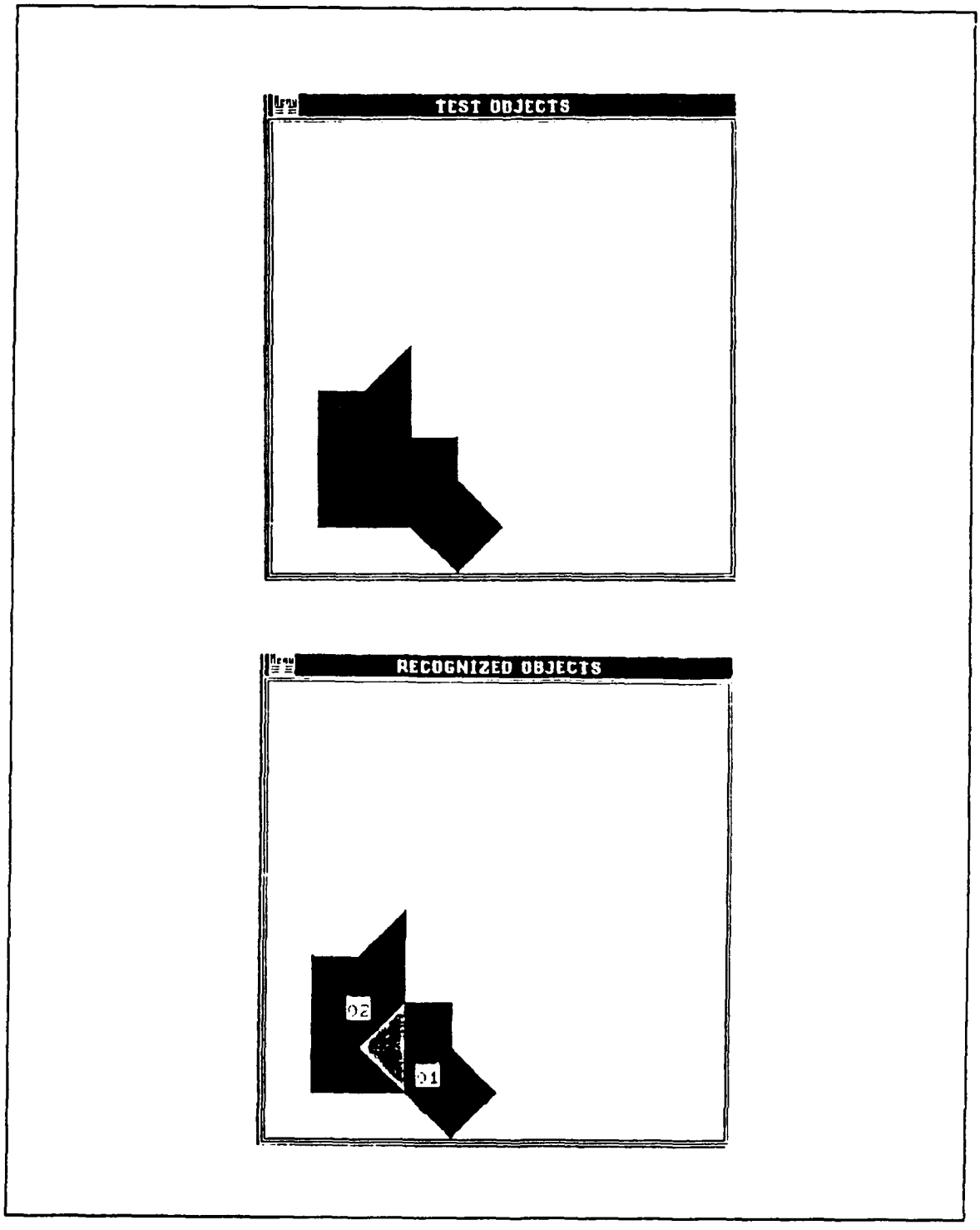

Figure 17. Occluded objects test of SET1

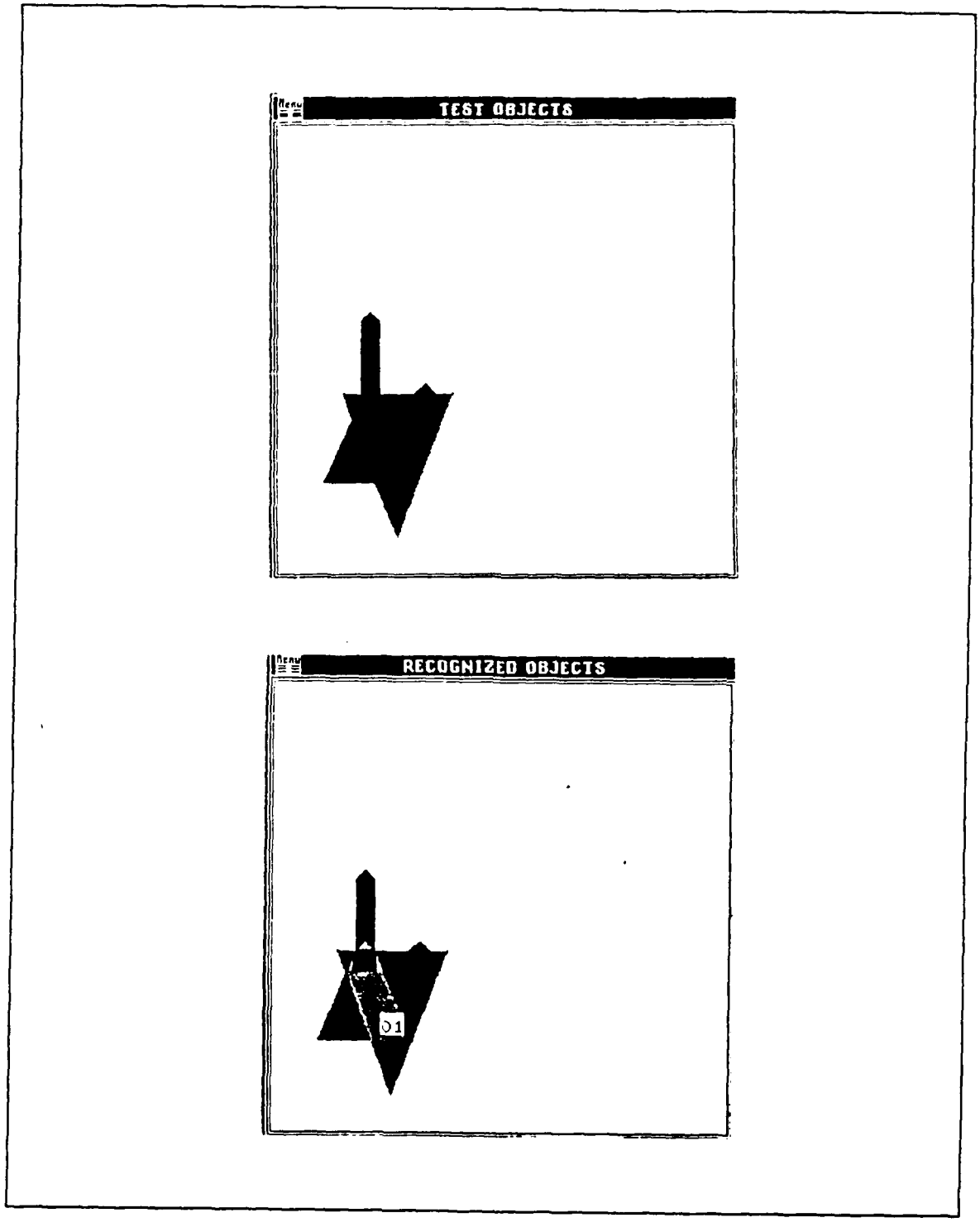

Figure 18. Occluded objects test of SET2

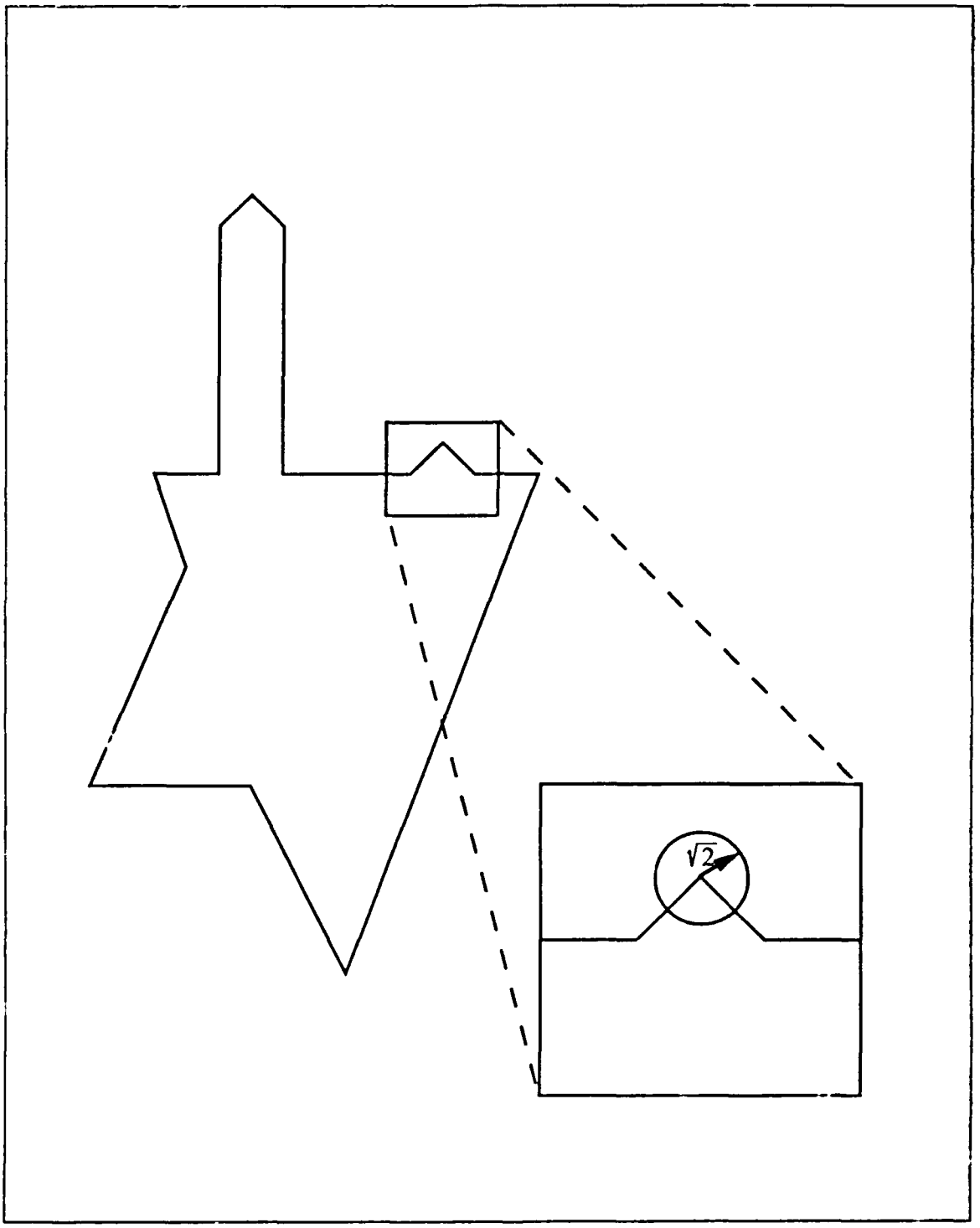

Figure 19. Error region of the disturbed interesting point

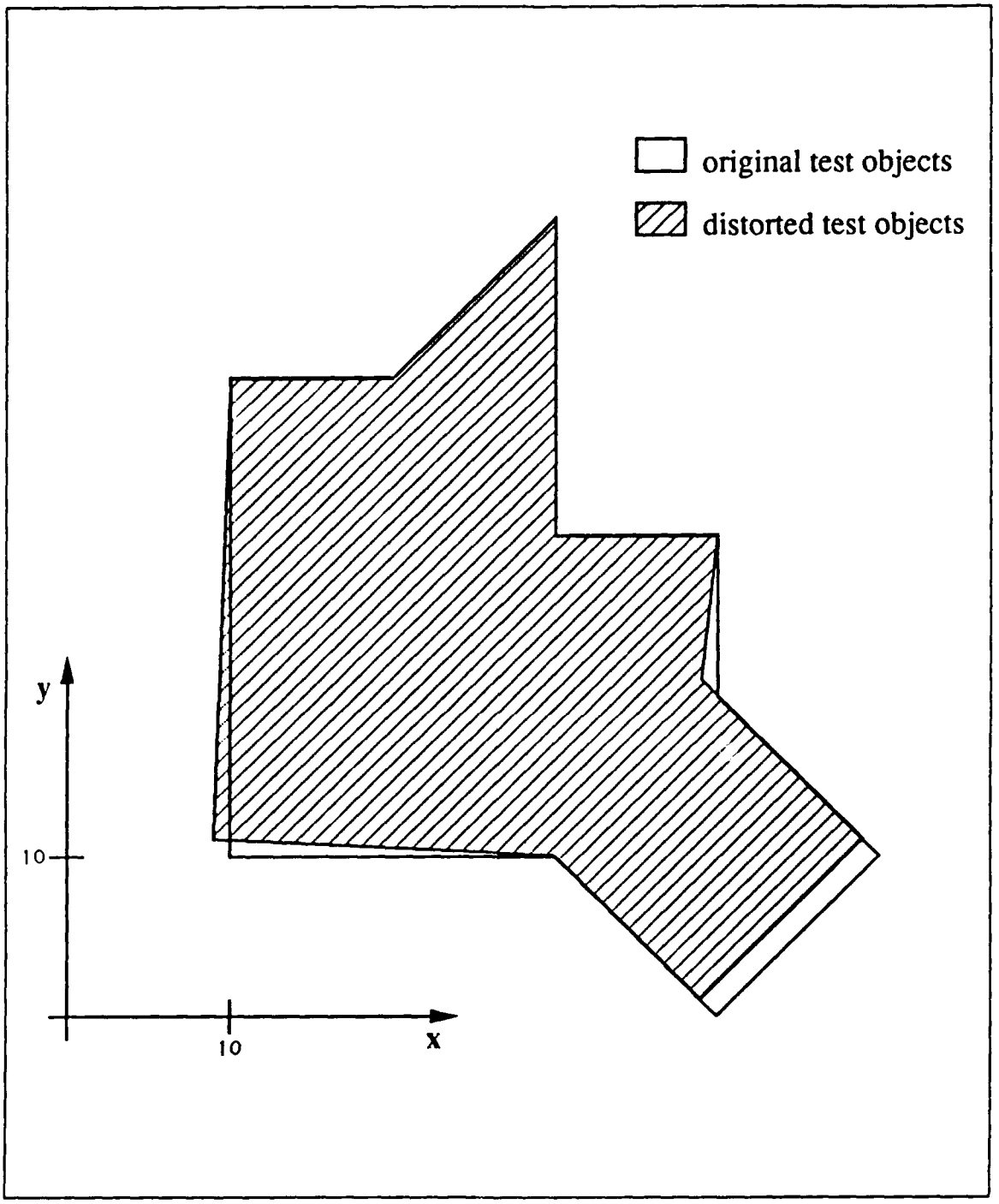

Figure 20. Plot of the test objects of SET1 with four interesting points disturbed

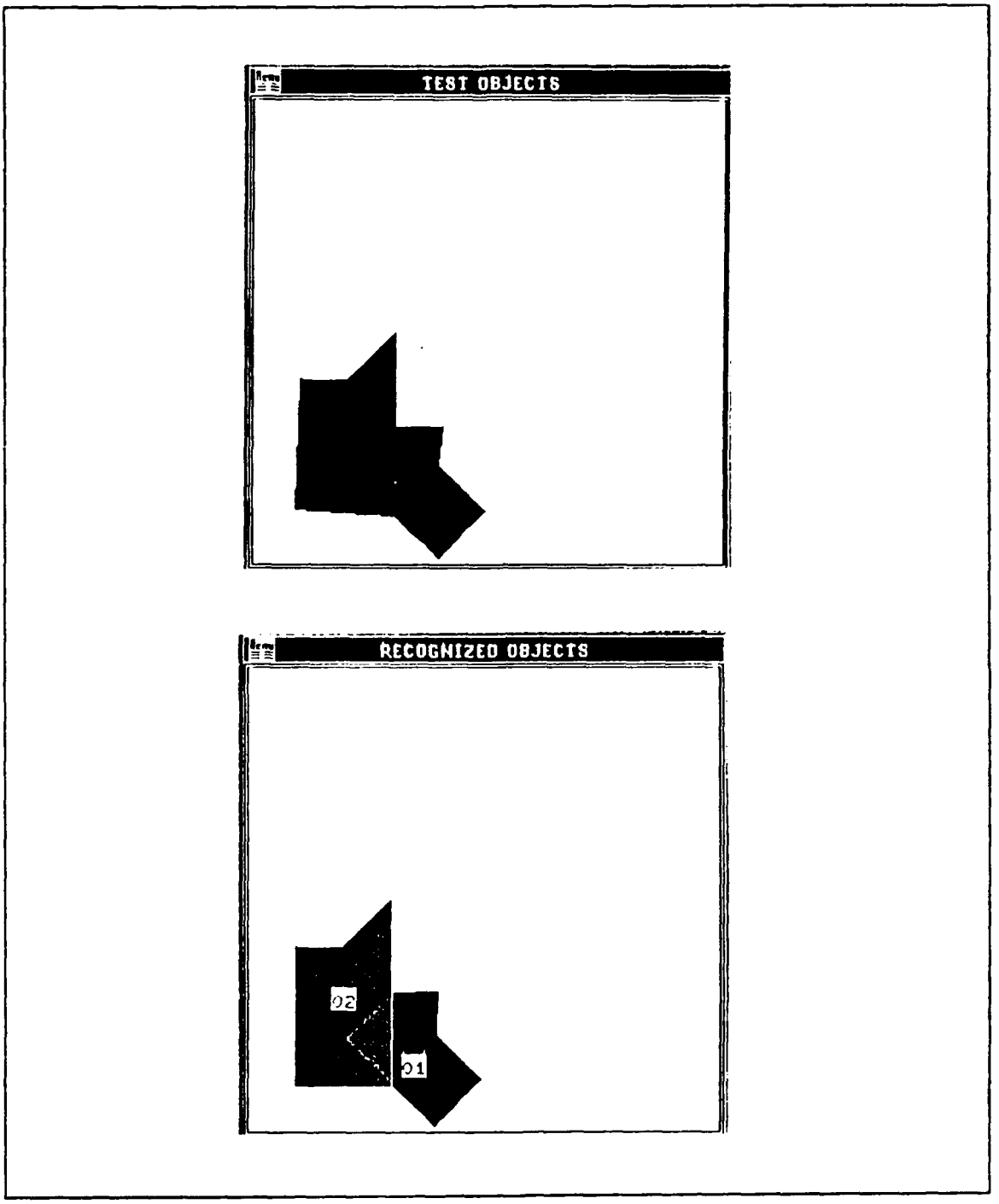

Figure 21. Recognition of SET1 with four interesting points disturbed

turbed. Figure 22 shows the plot of the test objects of SET2 with six interesting points disturbed. Figure 23 shows the recognition of SET2 with six interesting points disturbed. Figure 24 shows the plot of the test objects of SET1 with seven interesting points disturbed. Figure 25 shows the recognition of SET1 with seven interesting points disturbed. Figure 26 shows the plot of the test objects of SET2 with ten interesting points disturbed. Figure 27 shows the recognition of SET2 with ten interesting points disturbed.

C. PERFORMANCE

By inspecting the results of our experimental tests, it is evident that almost all of the limitations of the original affine matching algorithm have been improved. The old problems include the numerical instability, the collision of hash table, and the noise sensitivity. Since the modified hash structure was used, the problem of hash table collision due to nonuniform hashkeys was greatly reduced. Consequently, about 1300 ms of CPU time was saved in the recognition step of the noise free tests. Initially, the original algorithm is very sensitive to noise and fails in noisy tests. In order to increase the tolerance to noise perturbation, a quantized affine plane with 90×90 resolute cells was chosen. After the quantization, the partial voting technique was also used to select all the qualified candidate triplets. Finally, in the verification procedure the identified object(s) was displayed on the screen. For the improved algorithm the noise tolerance is enhanced by a factor of 70%, i.e., even when seventy percent of the interesting points of the test object are disturbed, the improved algorithm can still identify the test objects.

There is a trade-off between the processing speed and the noise tolerance. To have more noise tolerance, the partial voting technique is used. This will decrease the speed and increase the complexity of the algorithm. Therefore, the processing speed of the improved algorithm is still highly dependent on the number of interesting points in the test object. Another existing limitation of the algorithm is that only objects with small

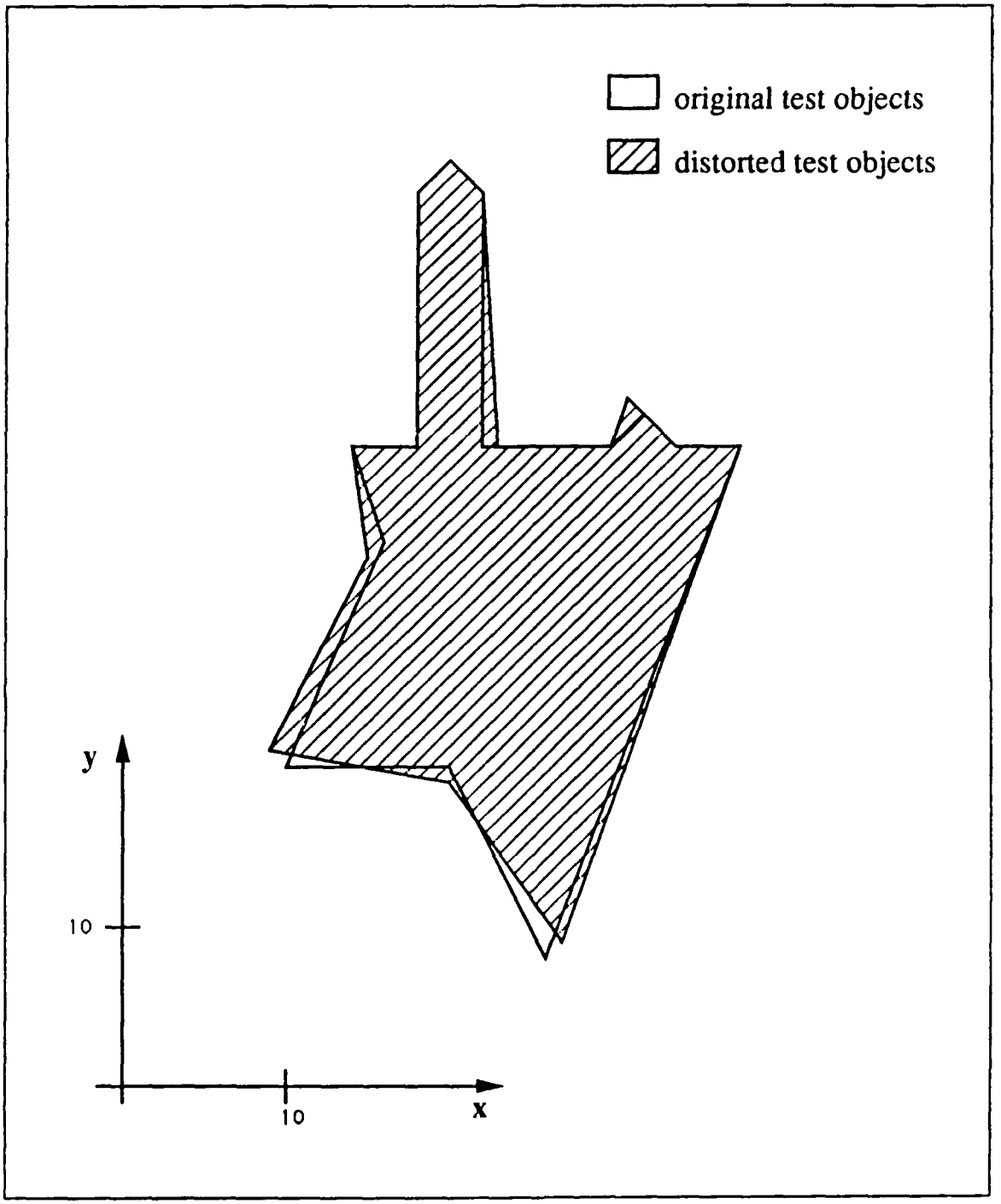

Figure 22. Plot of the test objects of SET2 with six interesting points disturbed

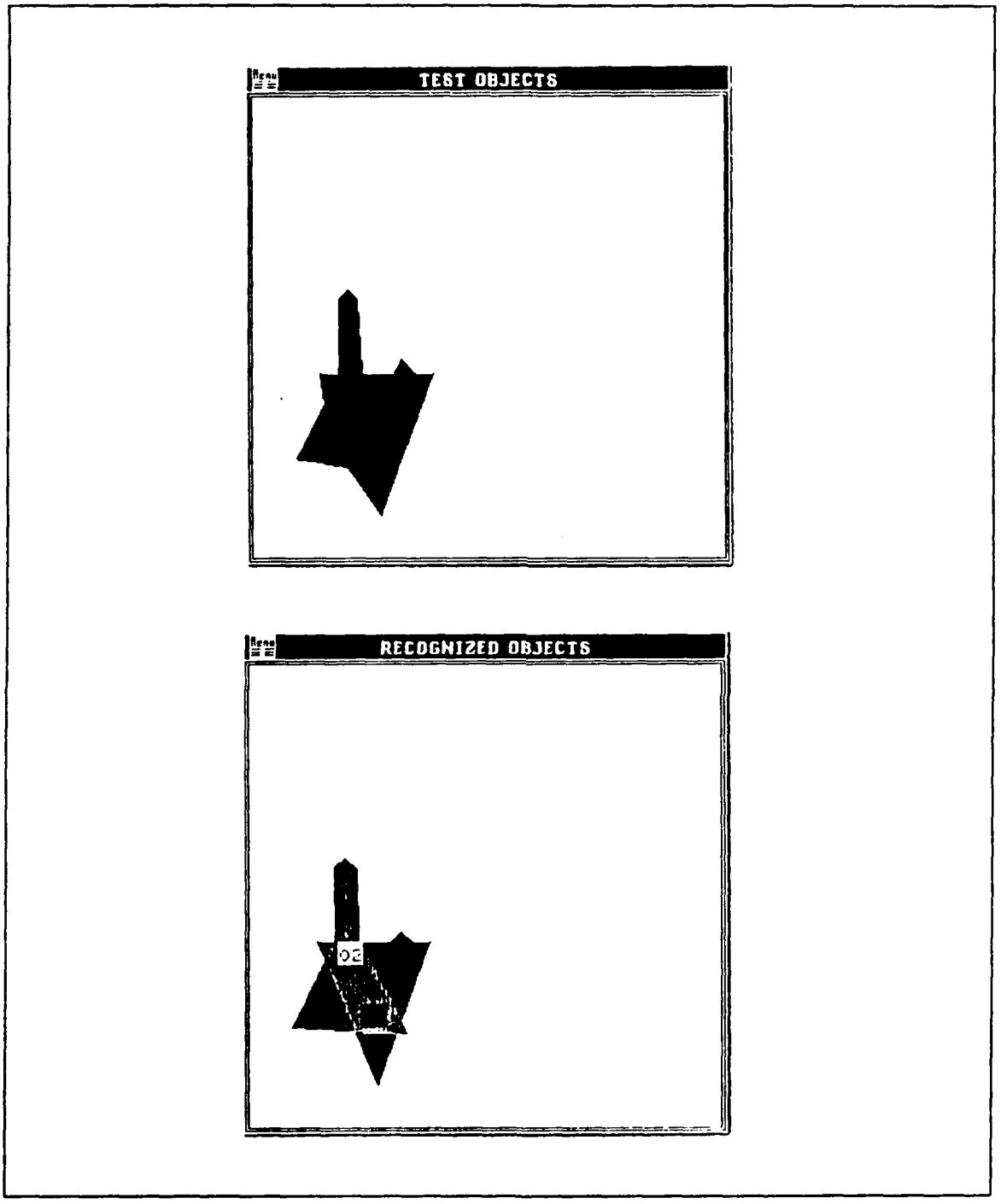

Figure 23. Recognition of SET2 with six interesting points disturbed

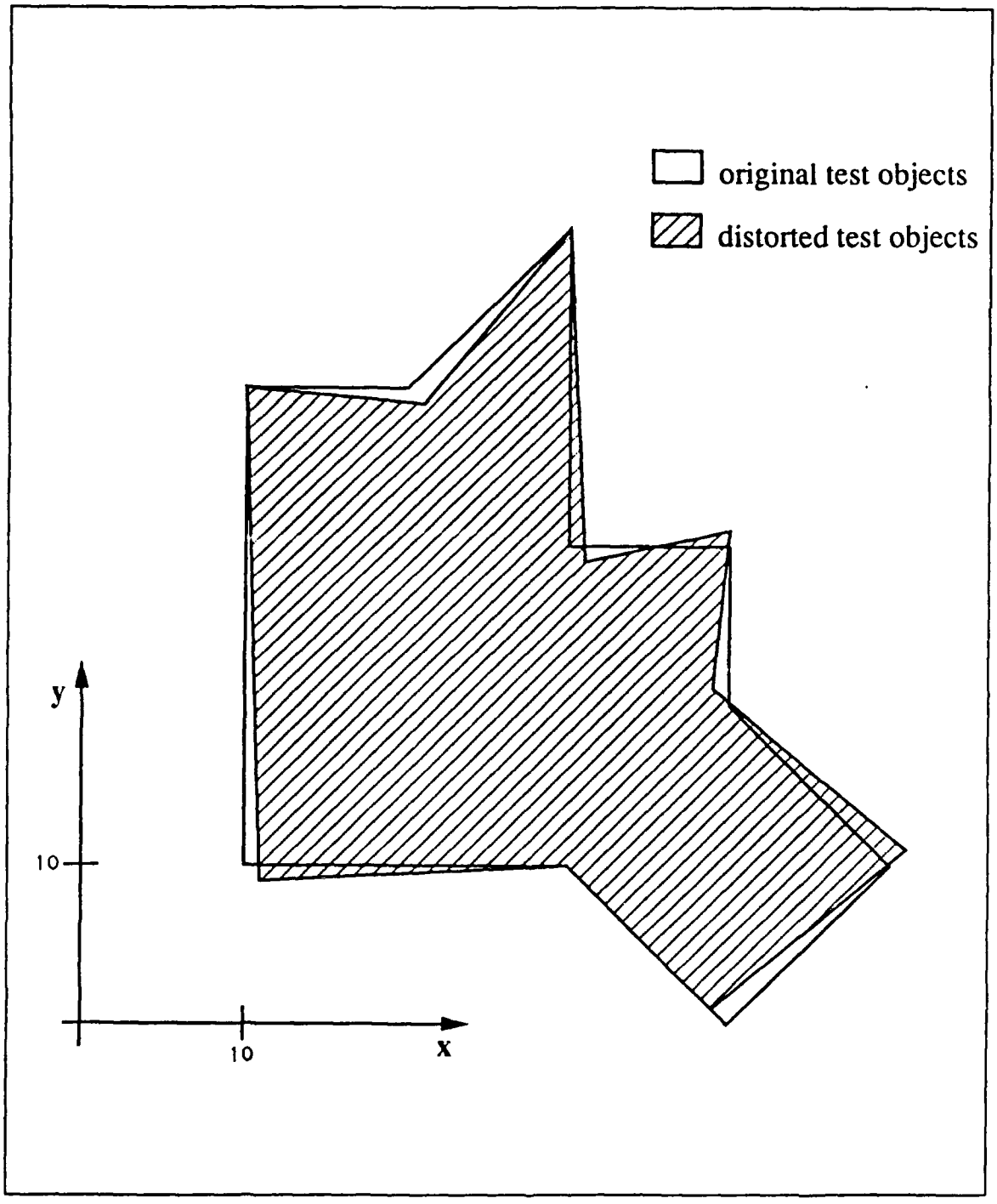

Figure 24. Plot of the test objects of SET1 with seven interesting points disturbed

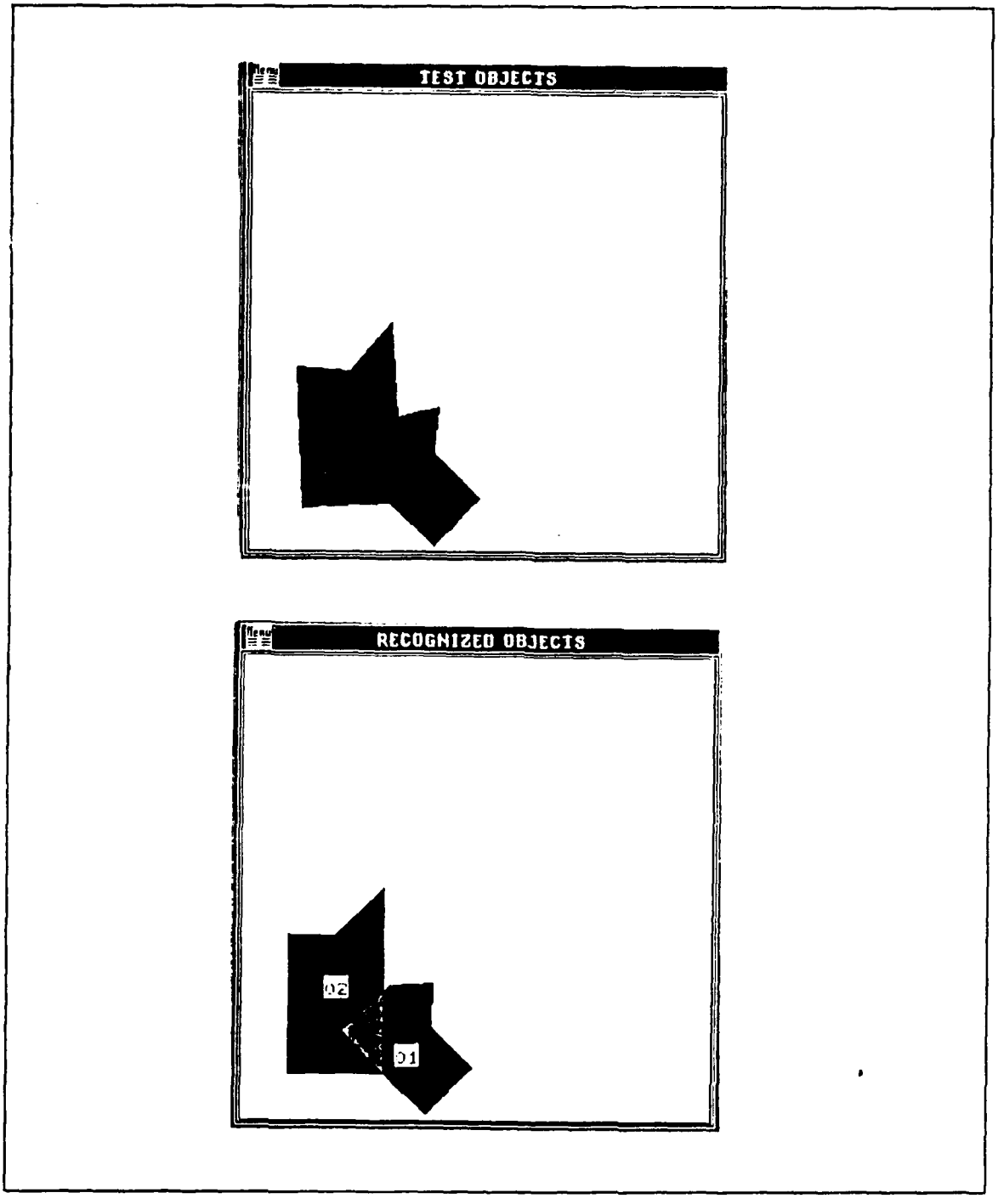

Figure 25. Recognition of SET1 with seven interesting points disturbed

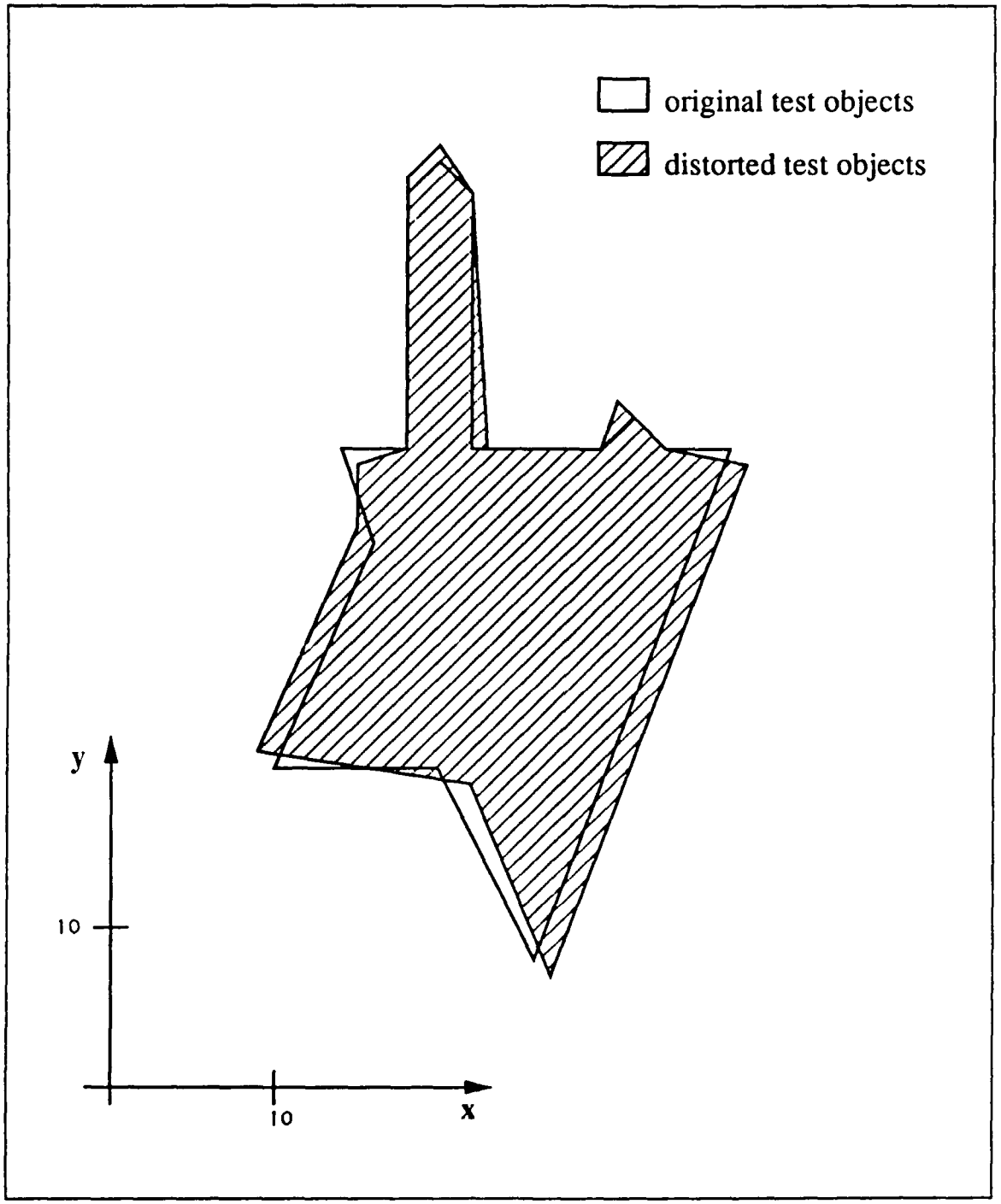

Figure 26. Plot of the test objects of SET2 with ten interesting points disturbed

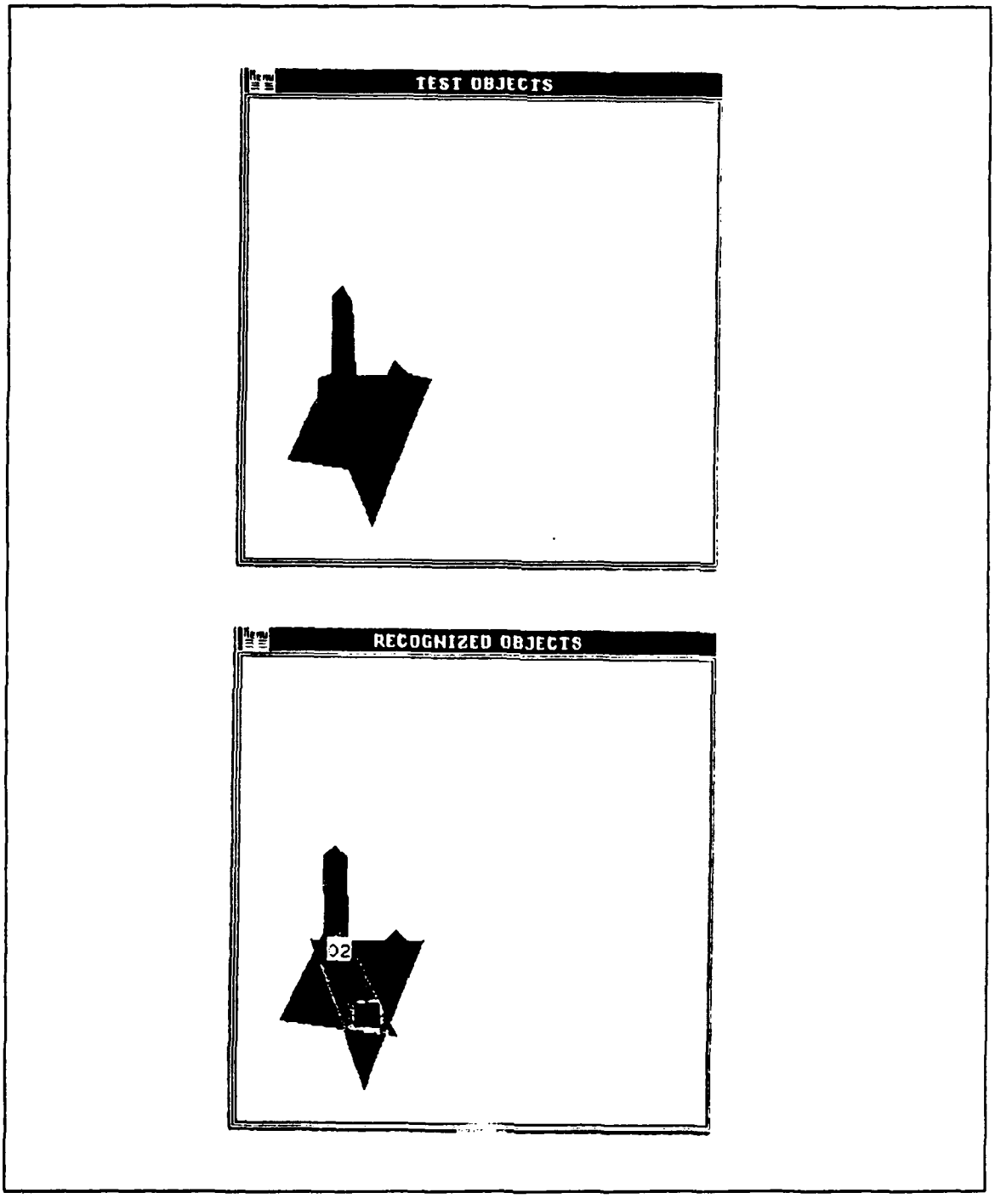

Figure 27. Recognition of SET2 with ten interesting points disturbed

number of vertices are handled here. Additionally, the system developed here can not be applied to patterns with concave boundary curves. However, the algorithm can be extended in these cases by using the footprint method proposed by Hummel and Wolfson [Ref. 3].

D. EXPERIENCE GAINED

The experience gained in this study can be summarized as follows.

1. Numerical Instability

When the interesting points of a triplet are too close to each other, the affine coefficients calculated based on this triplet tend to be unstable. This undesirable situation can be avoided by applying the area test criteria in both the Preprocessing and the Recognition steps.

2. Hashing Function

The values of ξ and η tend to concentrate highly around the origin of the affine plane because they are all small numbers. These small-valued coefficients will generate many long linked lists in the hash table. Hence, the accessing time will increase and become impractically long. Both the linear and non-linear hashing functions were tested in the experiments. The results show that the non-linear atan function is the best among many hashing functions. This function can scatter the small affine coefficients evenly over a wide range of the affine plane. Consequently, the size of the linked list can be reduced and a desirable processing speed can be achieved.

3. Quantization of Affine Plane

In order to tolerate the noise perturbation on the test objects, coarser quantization of affine plane is desirable. After calculating the bounds on ξ and η, an appropriate table size was chosen to implement the hash table. But, there is no optimal hash table size that can work well for all kinds of data. In our experimental tests a 90 $\times$ 90 table size was selected.

4. Modified Hash Structure

After quantization, every entry in the hash table is indexed by (i, j). This two-dimentional index is not a numeric key, it needs to be converted into a numeric value to implement the one-dimentional hash table. However, this kind of conversion will induce multiple collisions in the hash table. In order to reduce these collisions, two different one-to-one functions were adopted to generate the numerical inkey.

5. Partial Voting Technique

The noise problem can be lessened by using a coarser quantization of the affine plane. In addition, it is possible to improve the original matching algorithm by applying the partial voting technique. For every quantized affine index (i, j), there is a corresponding eight-neighbor mesh where each bin has a different voting weight. This technique can increase the noise tolerance in the recognition step.

6. False Peaks of Voting Array

As a result of the partial voting, the affine invariant matching algorithm suffers from the false peaks in the accumulation array. The entry which received the maximum votes may not be the correct one. To overcome this problem, the top six maximum votes are chosen as the candidates. By going through the verification procedure the correct candidate can be selected.

7. Noise Perturbation

The original algorithm is very rigid in calculating the affine coefficients. Hence, it is very sensitive to noise perturbation. From the experimental results of the improved algorithm, the noise tolerance shows an increase of 70% with respect to the original method.

8. Speed and Complexity

In noise free cases, the improved matching algorithm can achieve a higher speed than the original one when dealing with more complicated images. In noisy cases, the

speed deteriorates due to the execuation of partial voting. But, in both cases the execuation time of the improved algorithm is still highly dependent on the number of interesting points involved in the test.

V. CONCLUSIONS AND FUTURE RESEARCH

The object recognition is a major and difficult task in computer vision. The affine invariant matching technique proposed by Hummel and Wolfson is a new and interesting method for recognizing flat objects in a perspective projection image. Since the recognition of partially occluded objects is the main concern, the local point feature was used in the affine invariant matching algorithm. In order to reduce the complexity of the algorithm, a two-step scheme was applied. The algorithm was divided into two different steps, the off-line preprocessing step and the on-line recognition step.

This thesis discusses the shortcomings that occurred in the original algorithm. They are the basis instability, the collision of hash table, and the noise problem. These shortcomings make the original matching algorithm impractical to deal with real images. The original algorithm was improved by introducing several modifications. The area test criteria were used to get rid of unsuitable triplets to avoid numerical errors. The modified hash structure clearly improved the collision problem of the hash table. The partial voting technique with the consideration of false peaks in the voting array increased the noise tolerance of the original algorithm. Finally, it was shown that the results obtained from the improved algorithm are much better than the results from the original method.

Possible subjects for future study related to affine invariant matching could be :

- From the probability point of view, set up an optimal model to improve the recognition performance for noisy objects.
- Use the footprint method to implement the affine invariant curve matching.
- Extend the affine invariant matching to handle 3-D object recognition.

- Design special purpose hardware to achieve parallel searching of multiple objects in real time.

APPENDIX A. PASCAL SOURCE CODE

This appendix contains the Pascal source code used in the improved affine matching system. Explanation of each of the modules are discussed in Chapter III. Some procedures in these programs are adapted from the original implementation [Ref. 6].

```
****************************** PREPROCESSING ***********************************
****************************** (Off-line Step) *********************************

program model_gen(output);
(* This module generates the interesting points of data-base models,
   SET1 is used as the example *)
(* o/p : model_interesting_point data file *)
type
  ary = array [1..2,1..50] of integer;
var
  i,j,modelno,interestingno : integer;
  output : text;
  x,y : ary;
begin
  open(output,'model_interesting_point.dat',history := new);
  rewrite(output);
(* model 1 *)
  modelno := 1; interestingno := 9;
  x[1,1] := 32; x[1,2] := 36; x[1,3] := 38; x[1,4] := 40;
  x[1,5] := 48; x[1,6] := 50; x[1,7] := 52; x[1,8] := 56;
  x[1,9] := 44;
  y[1,1] := 30; y[1,2] := 30; y[1,3] := 28; y[1,4] := 30;
  y[1,5] := 30; y[1,6] := 28; y[1,7] := 30; y[1,8] := 30;
  y[1,9] := 62;
  writeln(output,modelno,interestingno);
  for i := 1 to interestingno do
    writeln(output,x[1,i],y[1,i]);
(* model 2 *)
  modelno := 2; interestingno := 7;
  x[2,1] := 20; x[2,2] := 21; x[2,3] := 21; x[2,4] := 25;
  x[2,5] := 15; x[2,6] := 19; x[2,7] := 19;
  y[2,1] := 31; y[2,2] := 32; y[2,3] := 40; y[2,4] := 50;
  y[2,5] := 50; y[2,6] := 40; y[2,7] := 32;
  writeln(output,modelno,interestingno);
  for i := 1 to interestingno do
    writeln(output,x[2,i],y[2,i]);
  close(output);
end.

*******************************************************************************

program model_affine_calculation(input,output);
(* This module rules out the undesirable triplets and finds all the
```

```
   affine coefficient pair of data-base models *)
(* i/p : model_interesting_point data file
   o/p : model_affine_coefficient data file *)
const
  total = 9999;
  area1 = 392.0;
  area2 = 76.0;
type
  ary = array [1..total] of integer;
  ary1 = array [1..2,1..2] of real;
  ary2 = array [1..3] of integer;
  ary3 = array [1..2,1..1] of real;
  ary4 = array [1..2,1..2] of integer;
var
  count,i,j,k,index : integer;
  outmodelno,outinterestingno : array [1..10] of integer;
  modelno,tripletno,interestingno,table_size : integer;
  x,y,rest_basex,rest_basey : ary;
  inverse_matrix : ary1;
  basex,basey,comx,comy : ary2;
  diff,coord : ary3;
  base_matrix : ary4;

procedure matrix_mult(inverse_matrix : ary1;
                      var diff,coord : ary3);
const
  k = 2;
  l = 2;
  m = 1;
var
  i,i1,n : integer;
begin
  for i := 1 to l do
    for i1 := 1 to m do
      coord[i,i1] := 0;
  for i := 1 to l do
    for i1 := 1 to m do
      for n := 1 to l do
        coord[i,i1] := coord[i,i1] + inverse_matrix[i,n] *
                       diff[n,i1];
end;

procedure inverse(base_matrix : ary4
                  var inverse_matrix : ary1);
var
  det : real;
begin
  for i := 1 to 2 do
    det := (base_matrix[1,1] * base_matrix[2,2]) -
           (base_matrix[1,2] * base_matrix[2,1]);
  if (det <> 0.0) then
    begin
      inverse_matrix[1,1] := base_matrix[2,2]/det;
      inverse_matrix[1,2] := base_matrix[1,2] * (-1) /det;
      inverse_matrix[2,1] := base_matrix[2,1] * (-1) /det;
      inverse_matrix[2,2] := base_matrix[1,1] /det;
```

```
    end;
end;

procedure calculate_coord(basex,basey : ary2);
var
  i,m,i1,i2,i3 : integer;
begin
  for i := 1 to 2 do
    begin
      base_matrix[1,i] := basex[i] - basex[i+1];
      base_matrix[2,i] := basey[i] - basey[i+1];
    end;
  inverse(base_matrix,inverse_matrix);
  i2 := 1;
  for i := 1 to interestingno do
    begin
      if ((basex[1] <> x[i]) or (basey[1] <> y[i])) and
         ((basex[2] <> x[i]) or (basey[2] <> y[i])) and
         ((basex[3] <> x[i]) or (basey[3] <> y[i])) then
        begin
          rest_basex[i2] := x[i];
          rest_basey[i2] := y[i];
          i2 := i2 + 1;
        end;
    end;
  for i := 1 to (interestingno[3) do
    begin
      diff[1,1] := rest_basex[i] - basex[3];
      diff[2,1] := rest_basey[i] - basey[3];
      matrix_mult(inverse_matrix,diff,coord);
      count := count +1;
      i3 := count mod (interestingno-3);
      if i3 = 1 then tripletno := tripletno + 1;
        write(output,modelno,tripletno,coord[1,1],coord[2,1]);
      for i1 := 1 to 3 do
        write(output,basex[i1],basey[i1]);
        writeln(output);
    end;
end;

procedure perm(basex,basey : ary2;
               k : integer);
var
  i,tx,ty : integer;
begin
  if k = 3 then
    calculate_coord(basex,basey)
  else
    for i := k to 3 do
      begin
        tx := basex[i];
        ty := basey[i];
        basex[i] := basex[k];
        basey[i] := basey[k];
        basex[k] := tx;
        basey[k] := ty;
```

```
        perm(basex,basey,k+1);
      end;
end;

procedure area_test(comx,comy : ary2;i : integer);
var
  s,a,b,c,sarea,area : real;
begin
  a := sqr(comx[2][comx[1])+sqr(comy[2]]comy[1]);
  b := sqr(comx[3][comx[2])+sqr(comy[3]]comy[2]);
  c := sqr(comx[1][comx[3])+sqr(comy[1]]comy[3]);
  a := sqrt(a);
  b := sqrt(b);
  c := sqrt(c);
  s := (a + b + c )/2.0;
  if i = 1 then area := area1;
  if i <> 1 then area := area2;
  sarea := s*(s-a)*(s-b)*(s-c);
  sarea := sqrt(sarea);
  area := area/20.0;
  if (sarea > area) then
    perm(comx,comy,1);
end;


procedure comb(k,index,j : integer);
var
  i : integer;
begin
  if index > 3 then area_test(comx,comy,j)
  else
    for i := k to interestingno do
      begin
        comx[index] := x[i];
        comy[index] := y[i];
        j := j;
        comb(i+1,index+1,j);
      end;
end;

begin(* main *)
  open(input,'model_interesting_point.dat',history := old);
  reset(intput);
  open(output,'model_affine_coefficient.dat',history := new);
  rewrite(output);
  table_size := 0;j:= 1;
  while not eof(input) do
    begin
      readln(input,outmodelno[j],outinterestingno[j]);
      modelno := outmodelno[j];
      interestingno := outinterestingno[j];
      for i := 1 to interestingno do
        readln(input,x[i],y[i]);
      count :=0;tripletno := 0;j := j+1;
      comb(1,1,modelno);
      table_size := table_size + count;
```

```
    end;
  writeln(output,table_size,outmodelno[1],outinterestingno[1],
          outmodelno[2],outinterestingno[2]);
  close(input);
  close(output);
end.

*******************************************************************************

program model_inkey(input,output);
(* This module quantizes the affine plane into 8100 different bins and
   converts each bin index (i, j) to an inkey *)
(* i/p : model_affine_coefficient data file
   o/p : inkey data file *)
type
  inrec = record
    modelno,tripletno : integer;
    key1,key2 : real;
    basex1,basey1,basex2,basey2,basex3,basey3 : integer
    end;
var
  b : inrec;
  infile,outfile : text;
  k1,k2,dk,inkey1 : real;
  i,j,i1,j1,size,prim,inkey : integer;

procedure prime(var a : integer);
var
  i,b : integer;
begin
  b := trunc(a/2);
  for i := 2 to b do
    begin
      if (a mod i) = 0 then
        begin
          a := a-1;
          prime(a);
        end;
    end;
end;

begin(* main *)
  open (input,'model_affine_coefficient.dat',history:=old);
  reset(input);
  open (output,'inkey.dat',history:=new);
  rewrite(output);
  j1 :=0;
  while not eof(input) do
    begin
      readln(input);
      j1 := j1+1;
    end;
  close(input);
  open (input,'model_affine_coefficient.dat',history:=old);
  reset(input);
  i1 := 1;
```

```
  dk := 2.0*arctan(20.0)/90.0;
  size := j1 - 1;
  prim := size;
  prime(prim);
  writeln(output,size,prim);
  while i1 <> j1 do
    with b do
      begin
        readln(input,modelno,tripletno,key1,key2,
               basex1,basey1,basex2,basey2,basex3,basey3);
        k1 := arctan(key1)/dk;
        k2 := arctan(key2)/dk;
        if (k1 >= 0.0) then
          begin
            i := trunc(k1) + 46;
            if (i > 90) then i := 90;
          end;
        if (k1 < 0.0) then
          begin
            i := trunc(k1) + 45;
            if (i < 1) then i := 1;
          end;
        if (k2 >= 0.0) then
          begin
            j := trunc(k2) + 46;
            if (j > 90) then j := 90;
          end;
        if (k2 < 0.0) then
          begin
            j := trunc(k2) + 45;
            if (j < 1) then j := 1;
          end;
        inkey1 := (0.839*i + 0.364*j)*100;
        inkey := trunc(inkey1);
        writeln(output,inkey,' ':2,i:2,' ':2,j:2,modelno:2,
                tripletno,key1,key2,basex1,basey1,basex2,
                basey2,basex3,basey3);
        i1 := i1+1;
      end;
    close(input);
    close(output);
end.

*****************************************************************************

program hashing(input,output);
(* This module uses the inkey to generate the hashkey and sets up
   the hash table *)
(* i/p : inkey data file
   o/p : hash_table data file *)
const
  database_modelno = 2;
type
  rec = record
    inkey,i,j,modelno,tripletno,link : integer;
    key1,key2 : real;
```

```
    basex1,basey1,basex2,basey2,basex3,basey3 : integer;
    end;
  outrec = record
    hashkey,inkey,i,j,modelno,tripletno,link : integer;
    key1,key2 : real;
    basex1,basey1,basex2,basey2,basex3,basey3 : integer;
    end;
var
  a : array[0..99999] of outrec;
  b : rec;
  i1,i2,j1,j2,k2,table_size,prim : integer;
  outmodelno,outinterestingno : array [1..database_modelno] of integer;
  c : array[0..99999] of integer;

procedure print;
var
  i2 : integer;
begin
  for i2 := 0 to table_size - 1  do
    begin
    if a[i2].hashkey <> - 1 then
      with a[i2] do
        writeln(output,hashkey,inkey,' ':2,i:2,' ':2,j:2,' ':2,
                modelno:2,tripletno:4,key1,key2,basex1:4,
                basey1:4,basex2:4,basey2:4,basex3:4,basey3:4,link)
    end;
end;

procedure collision(var m,n : integer);
begin
  while a[m].link <> - 1 do
    m := a[m].link;
  n := m + 1 ;
  while a[n].hashkey <> - 1 do
    begin
      n := n+1;
      if n > table_size - 1  then n := 0
    end;
  a[m].link := n;
  a[n].hashkey := n;
  a[n].inkey := b.inkey;
  a[n].i := b.i;
  a[n].j := b.j;
  a[n].modelno := b.modelno;
  a[n].tripletno := b.tripletno;
  a[n].key1 := b.key1;
  a[n].key2 := b.key2;
  a[n].basex1 := b.basex1;
  a[n].basey1 := b.basey1;
  a[n].basex2 := b.basex2;
  a[n].basey2 := b.basey2;
  a[n].basex3 := b.basex3;
  a[n].basey3 := b.basey3;
end;
```

```
procedure initialize;
var
  i1 : integer;
begin
  for i1 := 0 to table_size - 1  do
    begin
      a[i1].hashkey := - 1;
      a[i1].link := - 1;
      c[i1] := 0;
    end;
end;

procedure insert(var m,n :integer);
begin
if (a[m].hashkey = - 1)  then
  begin
    a[m].hashkey :=m;
    a[m].inkey := b.inkey;
    a[m].i := b.i;
    a[m].j := b.j;
    a[m].modelno := b.modelno;
    a[m].tripletno := b.tripletno;
    a[m].key1 := b.key1;
    a[m].key2 := b.key2;
    a[m].basex1 := b.basex1;
    a[m].basey1 := b.basey1;
    a[m].basex2 := b.basex2;
    a[m].basey2 := b.basey2;
    a[m].basex3 := b.basex3;
    a[m].basey3 := b.basey3;
    c[n] := 1;
  end;
end;

begin (* main *)
  open (input,'inkey.dat',history := old);
  reset(input);
  open (output,'hash_table.dat',history := new);
  rewrite(output);
  readln(input,table_size,prim);
  writeln(output,table_size,prim);
  initialize;
  i1 :=0;j1 := - 1;
  while not eof(input) do
    begin
      with b do
        begin
          readln(input,inkey,i,j,modelno,tripletno,key1,key2,
                 basex1,basey1,basex2,basey2,basex3,basey3);
          i1 := inkey rem prim;
        end;
      j1 := j1 + 1;
      insert(i1,j1);
    end;
  close(input);
  open (input,'inkey.dat',history := old);
```

```
  reset(input);
  i2 := 0;j2 :=- 1;k2 := 0;
  readln(input,table_size,prim);
  while not eof(input) do
    begin
      begin
        with b do
          begin
            readln(input,inkey,i,j,modelno,tripletno,key1,key2,
                   basex1,basey1,basex2,basey2,basex3,basey3);
            i2 := inkey rem prim;
          end;
        j2 := j2 + 1;
        if c[j2] = 0 then
          collision(i2,k2);
      end;
  close(input);
  print;
  close(output);
end.
```

********** RECOGNITION **********
********** (On-line Step) **********

```
program test_gen(output);
(* This module generates the interesting points of test objects *)
(* o/p : test_interesting_point data file *)
type
  ary = array [1..2,1..50] of integer;
var
  i,j,modelno,interestingno : integer;
  x,y : ary;
begin
  open(output,'test_interesting_point.dat',history := new);
  rewrite(output);
(* test object *)
  modelno:=1;interestingno := 10;
  x[1,1] := 50; x[1,2] := 40; x[1,3] := 30;
  x[1,4] := 10; x[1,5] := 10;
  x[1,6] := 20; x[1,7] := 30; x[1,8] := 30; x[1,9] := 40;
  x[1,10] := 40;
  y[1,1] := 10; y[1,2] := 0; y[1,3] := 10;
  y[1,4] := 10; y[1,5] := 40;
  y[1,6] := 40; y[1,7] := 50; y[1,8] := 30; y[1,9] := 30;
  y[1,10] := 20;
  writeln(output,modelno,interestingno);
  for i := 1 to interestingno do
  writeln(interesting,x[1,i],y[1,i]);
  close(output);
end.
```

****

```
program test_affine_calculation(input,output);
(* This module rules out the undesirable triplets and finds all the
   affine coefficient pair of test objects *)
(* i/p : test_interesting_point data file
```

```
   o/p : test_affine_coefficient data file *)
const
  total = 9999;
  tarea = 533.0;
type
  ary = array [1..total] of integer;
  ary1 = array [1..2,1..2] of real;
  ary2 = array [1..3] of integer;
  ary3 = array [1..2,1..1] of real;
  ary4 = array [1..2,1..2] of integer;
var
  count,i,k,index : integer;
  modelno,interestingno,table_size : integer;
  x,y,rest_basex,rest_basey : ary;
  inverse_matrix : ary1;
  basex,basey,comx,comy : ary2;
  diff,coord : ary3;
  base_matrix : ary4;

procedure test_matrix_mult(inverse_matrix : ary1;
                           var diff,coord : ary3);
const
  k = 2;
  l = 2;
  m = 1;
var
  i,j,n : integer;
begin
  for i := 1 to l do
    for j := 1 to m do
      cord[i,j] := 0;
  for i := 1 to l do
    for j := 1 to m do
      for n := 1 to l do
        coord[i,j] := coord[i,j] + inverse_matrix[i,n] *
                      diff[n,j];
end;

procedure test_inverse(base_matrix : ary4;
                       var inverse_matrix : ary1);
var
  det : real;
begin
  for i := 1 to 2 do
    det := (base_matrix[1,1] * base_matrix[2,2]) -
           (base_matrix[1,2] * base_matrix[2,1]);
    if (det <> 0.0) then
      begin
        inverse_matrix[1,1] := base_matrix[2,2]/det;
        inverse_matrix[1,2] := base_matrix[1,2] * (-1) /det;
        inverse_matrix[2,1] := base_matrix[2,1] * (-1) /det;
        inverse_matrix[2,2] := base_matrix[1,1] /det;
      end;
end;
```

```
procedure test_calculate_coord(basex,basey : ary2);
var
  i,j,i1,i2 : integer;
begin
  for i := 1 to 2 do
    begin
      base_matrix[1,i] := basex[i] - basex[i+1];
      base_matrix[2,i] := basey[i] - basey[i+1];
    end;
  test_inverse(base_matrix,inverse_matrix);
  i2 := 1;
  for i := 1 to interestingno do
  begin
    if ((basex[1] <> x[i]) or (basey[1] <> y[i])) and
       ((basex[2] <> x[i]) or (basey[2] <> y[i])) and
       ((basex[3] <> x[i]) or (basey[3] <> y[i])) then
      begin
        rest_basex[i2] := x[i];
        rest_basey[i2] := y[i];
        i2 := i2 +1;
      end;
  end;
  for i := 1 to interestingno[3 do
    begin
      diff[1,1] := rest_basex[i] - basex[3];
      diff[2,1] := rest_basey[i] - basey[3];
      test_matrix_mult(inverse_matrix,diff,coord);
      count := count +1;
      write(coordinate,modelno,count,coord[1,1],coord[2,1]);
      for i1 := 1 to 3 do
        write(output,basex[i1],basey[i1]);
      writeln(output);
    end;
end;

procedure test_perm(basex,basey : ary2;
                    k : integer);
var
  i,tx,ty : integer;
begin
  if k = 3 then
    test_calculate_coord(basex,basey)
  else
    for i := k to 3 do
      begin
        tx := basex[i];
        ty := basey[i];
        basex[i] := basex[k];
        basey[i] := basey[k];
        basex[k] := tx;
        basey[k] := ty;
        test_perm(basex,basey,k+1);
      end;
end;
```

```
procedure area_test(comx,comy : ary2);
var
  s,a,b,c,sarea,area : real;
begin
  a := sqr(comx[2][comx[1])+sqr(comy[2]]comy[1]);
  b := sqr(comx[3][comx[2])+sqr(comy[3]]comy[2]);
  c := sqr(comx[1][comx[3])+sqr(comy[1]]comy[3]);
  a := sqrt(a);
  b := sqrt(b);
  c := sqrt(c);
  s := (a + b + c)/2.0;
  sarea := s*(s[a)*(s]b)*(s[c);
  sarea := sqrt(sarea);
  area := tarea/10.0;
  if (sarea > area) then
    test_perm(comx,comy,1);
end;

procedure comb(k,index : integer);
var
  i : integer;
begin
  if index > 3 then area_test(comx,comy)
  else
    for i := k to interestingno do
      begin
        comx[index] := x[i];
        comy[index] := y[i];
        comb(i+1,index+1);
      end;
end;

begin(* main *)
  open(input,'test_interesting_point.dat',history := old);
  reset(input);
  open(output,'test_affine_coefficient.dat',history := new);
  rewrite(output);
  table_size := 0;
  while not eof(input) do
    begin
      readln(input,modelno,interestingno);
      writeln(output,interestingno);
      for i := 1 to interestingno do
        readln(input,x[i],y[i]);
      count :=0;
      comb(1,1);
      table_size := table_size + count;
    end;
    close(input);
    close(output);
end.

*****************************************************************************

program test_inkey(input,output);
(* This module converts every bin index (i, j) to an inkey and generates
```

```
   the tally data file *)
(* i/p : test_affine_coefficient data file
   o/p : tally data file *)
type
  inrec = record
    modelno,tripletno : integer;
    key1,key2 : real;
    basex1,basey1,basex2,basey2,basex3,basey3 : integer
    end;
var
  b : inrec;
  infile,outfile : text;
  dk,k1,k2,inkey1 : real;
  i,i1,j,interestingno,inkey : integer;
begin
  open (input,'test_affine_coefficient.dat',history:=old);
  reset(input);
  open (output,'tally.dat',history:=new);
  rewrite(output);
  i1 := 0;
  readln(input,interestingno);
  writeln(output,interestingno);
  dk := 2.0*arctan(20.0)/90.0;
  while not eof(infile) do
    with b do
      begin
        readln(infile,modelno,tripletno,key1,key2,
               basex1,basey1,basex2,basey2,basex3,basey3);
        k1 := arctan(key1)/dk;
        k2 := arctan(key2)/dk;
        if (k1 >= 0.0) then
          begin
            i := trunc(k1) + 46;
            if (i > 90) then i := 90;
          end;
        if (k1 < 0.0) then
          begin
            i := trunc(k1) + 45;
            if (i < 1) then i := 1;
          end;
        if (k2 >= 0.0) then
          begin
            j := trunc(k2) + 46;
            if (j > 90) then j := 90;
          end;
        if (k2 < 0.0) then
          begin
            j := trunc(k2) + 45;
            if (j < 1) then j := 1;
          end;
        inkey1 := (0.839*i + 0.364*j)*100;
        inkey := trunc(inkey1);
        writeln(output,inkey,i,j,key1,key2,
                basex1,basey1,basex2,basey2,basex3,basey3);
        i1 := i1+1;
      end;
```

```
  close(infile);
  close(outfile);
end.

*****************************************************************************

program affine_matching(input,candfile,output);
(* This module applies partial voting technique to select all the
   entries which satisfy the top[six]max criteria *)
(* i/p : hash_table and tally data files
   o/p : vote data file *)
const
  database_modelno = 2;
type
  candrec = record
    inkey,i,j : integer;
    key1,key2 : real;
    basex1,basey1,basex2,basey2,basex3,basey3 : integer;
    end;
  modelrec = record
    hashkey,inkey,i,j,modelno,tripletno,link : integer;
    key1,key2 : real;
    basex1,basey1,basex2,basey2,basex3,basey3,a : integer;
    end;
  collected_modelrec = record
    hashkey,inkey,i,j,modelno,tripletno,link : integer;
    key1,key2 : real;
    basex1,basey1,basex2,basey2,basex3,basey3,a : integer;
    vote1 : real;
    end;
var
  temporary_model1,temp : array[0..9999] of modelrec;
  model : array [0..23000] of modelrec;
  collected_model : array[1..9999] of collected_modelrec;
  tmodel,smodel,gmodel,pmodel : array[1..1000] of collected_modelrec;
  cand : array[0..19999] of candrec;
  cand1,cand2 : array [1..9999] of candrec;
  tcand,scand,gcand,pcand : array[1..1000] of candrec;
  i1,j1,k,k1,m,index,count,n,prim,f,h,g,hkey,ix,iy,jx,jy,c1,c2 : integer;
  max11,max12,max13,max14,max15,max16
  max21,max22,max23,max24,max25,max26
  th1,th2 : real;
  table_size,interestingno,test_interestingno_3,
  b1,b2,s1,s2,w,t1,t2 : integer;
  candfile : text;
  vote : array[0..database_modelno,0..27999] of real;
  outmodelno,outinterestingno : array [1..database_modelno] of integer;

procedure initialize1;
var
  i,j : integer;
begin
  for i := 0 to database_modelno do
    for j := 0 to table_size do
      vote[i,j] := 0.0;
end;
```

```
procedure print(i6,k :integer);
begin
  with gmodel[i6] do
    begin
      writeln(output,hashkey,inkey,modelno:2,tripletno:4,
              key1,key2,basex1:4,basey1:4,basex2:4,basey2:4,
              basex3:4,basey3:4,vote1,k);
      gcand[i6].key1 := key1;
      gcand[i6].key2 := key2;
    end;
  with gcand[i6] do
    writeln(output,' ':10,inkey,' ':6,key1,key2,basex1:4,
            basey1:4,basex2:4,basey2:4,basex3:4,basey3:4);
end;

procedure delet(h : integer);
var
  k,i,j,j1,q,h1,l,w : integer;
begin
  k := 1;h1 := 0;w := 0;
  while (k <= h) do
    begin
      if (smodel[k].a = 0) then
        begin
          j := 0;
          q := smodel[k].inkey;
          for i := 1 to h do
             begin
               if (smodel[i].inkey = q) and (j = 0) then
                 begin
                   j := j + 1;
                   h1 := h1 + 1;
                   gmodel[h1] := smodel[i];
                   gcand[h1] := scand[i];
                 end
               else
                 if (smodel[i].inkey = q) and (j <> 0) then
                     smodel[i].a := 1;
             end;
        end;
        k := k +1;
    end;
  if (gmodel[1].modelno = 1) and (h1 >= t1) then
    begin
      for l := 1 to h1 do
        print(l,h1);
    end;
  if (gmodel[1].modelno = 2) and (h1 >= t2) then
    begin
      for l := 1 to h1 do
        print(l,h1);
    end;
end;
```

```
procedure select1(p1 : integer);
var
  k,q,h,i : integer;
begin
  k := 1;
  while(k <= p1) and (s1 = 0) do
    begin
      if (tmodel[k].a = 0) then
        begin
          h := 0;
          q := tmodel[k].tripletno;
          for i := 1 to p1 do
            begin
              if (tmodel[i].tripletno = q) then
                begin
                  h := h + 1;
                  smodel[h] := tmodel[i];
                  scand[h] := tcand[i];
                  tmodel[i].a := 1;
                end;
            end;
          if (h >= t1) then
            delet(h);
        end;
      k := k + 1;
    end;
end;

procedure select2(p1 : integer);
var
  k,q,h,i : integer;
begin
  k := 1;
  while(k <= p1) and (s2 = 0) do
    begin
      if (tmodel[k].a = 0) then
        begin
          h := 0;
          q := tmodel[k].tripletno;
          for i := 1 to p1 do
            begin
              if (tmodel[i].tripletno = q) then
                begin
                  h := h + 1;
                  smodel[h] := tmodel[i];
                  scand[h] := tcand[i];
                  tmodel[i].a := 1;
                end;
            end;
          if (h >= t2) then
          delet(h);
        end;
      k := k + 1;
    end;
end;
```

```
procedure all_maximum_selected(n : integer);
var
  i5,p1 : integer;
begin
  if (s1 =0) then
    begin
      p1 := 0;
      for i5 := 1 to n do
        with collected_model[i5] do
          if (modelno = 1) and
             ((cand2[i5].inkey <> 0) and (vote1 = max11)) then
            begin
              p1 := p1 + 1;
              tmodel[p1] := collected_model[i5];
              tcand[p1] := cand2[i5];
            end;
      select1(p1);
    end;
  if (s1 =0) then
    begin
      p1 := 0;
      for i5 := 1 to n do
        with collected_model[i5] do
          if (modelno = 1) and
             ((cand2[i5].inkey <> 0) and (vote1 = max12)) then
            begin
              p1 := p1 + 1;
              tmodel[p1] := collected_model[i5];
              tcand[p1] := cand2[i5];
            end;
      select1(p1);
    end;
  if (s1 =0) then
    begin
      p1 := 0;
      for i5 := 1 to n do
        with collected_model[i5] do
          if (modelno = 1) and
             ((cand2[i5].inkey <> 0) and (vote1 = max13)) then
            begin
              p1 := p1 + 1;
              tmodel[p1] := collected_model[i5];
              tcand[p1] := cand2[i5];
            end;
      select1(p1);
    end;
  if (s1 =0) then
    begin
      p1 := 0;
      for i5 := 1 to n do
        with collected_model[i5] do
          if (modelno = 1) and
             ((cand2[i5].inkey <> 0) and (vote1 = max14)) then
            begin
              p1 := p1 + 1;
              tmodel[p1] := collected_model[i5];
```

```
                tcand[p1] := cand2[i5];
              end;
        select1(p1);
      end;
    if (s1 =0) then
      begin
        p1 := 0;
        for i5 := 1 to n do
           with collected_model[i5] do
           if (modelno = 1) and
              ((cand2[i5].inkey <> 0) and (vote1 = max15)) then
            begin
              p1 := p1 + 1;
              tmodel[p1] := collected_model[i5];
              tcand[p1] := cand2[i5];
            end;
        select1(p1);
      end;
    if (s1 =0) then
      begin
      p1 := 0;
      for i5 := 1 to n do
        with collected_model[i5] do
          if (modelno = 1) and
             ((cand2[i5].inkey <> 0) and (vote1 = max16)) then
           begin
             p1 := p1 + 1;
             tmodel[p1] := collected_model[i5];
             tcand[p1] := cand2[i5];
           end;
        select1(p1);
      end;
    if (s2 =0) then
      begin
        p1 := 0;
        for i5 := 1 to n do
          with collected_model[i5] do
            if (modelno = 2) and
               ((cand2[i5].inkey <> 0) and (vote1 = max21)) then
              begin
                p1 := p1 + 1;
                tmodel[p1] := collected_model[i5];
                tcand[p1] := cand2[i5];
              end;
        select1(p1);
      end;
    if (s2 =0) then
      begin
        p1 := 0;
        for i5 := 1 to n do
          with collected_model[i5] do
            if (modelno = 2) and
               ((cand2[i5].inkey <> 0) and (vote1 = max22)) then
              begin
                p1 := p1 + 1;
                tmodel[p1] := collected_model[i5];
```

```
            tcand[p1] := cand2[i5];
          end;
    select1(p1);
  end;
if (s2 =0) then
  begin
    p1 := 0;
    for i5 := 1 to n do
      with collected_model[i5] do
        if (modelno = 2) and
           ((cand2[i5].inkey <> 0) and (vote1 = max23)) then
          begin
            p1 := p1 + 1;
            tmodel[p1] := collected_model[i5];
            tcand[p1] := cand2[i5];
          end;
    select1(p1);
  end;
if (s2 =0) then
  begin
    p1 := 0;
    for i5 := 1 to n do
      with collected_model[i5] do
        if (modelno = 2) and
           ((cand2[i5].inkey <> 0) and (vote1 = max24)) then
          begin
            p1 := p1 + 1;
            tmodel[p1] := collected_model[i5];
            tcand[p1] := cand2[i5];
          end;
    select1(p1);
  end;
if (s2 =0) then
  begin
    p1 := 0;
    for i5 := 1 to n do
      with collected_model[i5] do
        if (modelno = 2) and
           ((cand2[i5].inkey <> 0) and (vote1 = max25)) then
          begin
            p1 := p1 + 1;
            tmodel[p1] := collected_model[i5];
            tcand[p1] := cand2[i5];
          end;
    select1(p1);
  end;
if (s2 =0) then
  begin
  p1 := 0;
  for i5 := 1 to n do
    with collected_model[i5] do
      if (modelno = 2) and
         ((cand2[i5].inkey <> 0) and (vote1 = max26)) then
        begin
          p1 := p1 + 1;
          tmodel[p1] := collected_model[i5];
```

```
            tcand[p1] := cand2[i5];
          end;
        select1(p1);
      end;
    end;

procedure verify(j1,k,m : integer;var n : integer);
var
  i4,q : integer;
begin
  for i4 := 1 to m do
    with temporary_model[i4] do
      if (modelno = temporary_model[k].modelno) and
         (tripletno = temporary_model[k].tripletno) then
        begin
          n := n + 1;
          collected_model[n].hashkey := hashkey;
          collected_model[n].inkey := inkey;
          collected_model[n].modelno := modelno;
          collected_model[n].tripletno := tripletno;
          collected_model[n].key1 := key1;
          collected_model[n].key2 := key2;
          collected_model[n].basex1 := basex1;
          collected_model[n].basey1 := basey1;
          collected_model[n].basex2 := basex2;
          collected_model[n].basey2 := basey2;
          collected_model[n].basex3 := basex3;
          collected_model[n].basey3 := basey3;
          collected_model[n].vote1 := vote[modelno,tripletno];
          for q := (j1 - test_interestingno_3) to j1 - 1 do
            if (abs(cand[q].key1 - key1) < 0.04) and
               (abs(cand[q].key2 - key2) < 0.04) then
              cand2[n] := cand[q];
        end;
end;

procedure top_six_maximun(m : integer;var k : integer);
var
  i3 : integer;
  max : real;
begin
  max :=0.0;
  for i3 := 1 to m do
    with temporary_model[i3] do
      if (vote[modelno,tripletno] > max) then
        begin
          k :=i3;
          max := vote[modelno,tripletno];
          if (modelno = 1) then
          begin
          if (max > max11) then
            max11 := max
          else
          if (max <> max11) and (max > max12) then
            max12 := max
          else
```

```
            if ((max <> max11) and (max <> max12)) and (max > max13) then
              max13 := max
            else
            if ((max <> max11) and (max <> max12)) and ((max <> max13) and
                (max > max14)) then
              max14 := max
            else
            if (((max <> max11) and (max <> max12)) and ((max <> max13) and
                 (max <> max14))) and (max > max15) then
              max15 := max
            else
            if ((max <> max11) and (max <> max12)) and ((max <> max13) and
                (max <> max14)) and ((max <> max15) and (max > max16)) then
              max16 := max;
            end;
            if (modelno = 2) then
            begin
            if (max > max21) then
              max21 := max
            else
            if (max <> max21) and (max > max22) then
              max22 := max
            else
            if ((max <> max21) and (max <> max22)) and (max > max23) then
                 max23 := max
            else
            if ((max <> max21) and (max <> max22)) and ((max <> max23) and
                (max > max24)) then
              max24 := max
            else
            if (((max <> max21) and (max <> max22)) and ((max <> max23) and
                 (max <> max24))) and (max > max25) then
              max25 := max
            else
            if ((max <> max21) and (max <> max22)) and ((max <> max23) and
                (max <> max24)) and ((max <> max25) and (max > max26)) then
              max26 := max;
            end;
          end;
end;

procedure search(hkey,i,j,f : integer;var m : integer);
var
  i2 : integer;
begin
  i2 := hkey;
  while (i2 <> [1)  do
    begin
      if (i = model[i2].i) and
         (j = model[i2].j) then
        begin
          m := m + 1;
          temporary_model[m] := model[i2];
          with temporary_model[m] do
            begin
            if f = 1 then
```

```
                vote[modelno,tripletno] := vote[modelno,tripletno] + 1.0;
                if f = 2 then
                vote[modelno,tripletno] := vote[modelno,tripletno] + 0.5;
                if f = 3 then
                vote[modelno,tripletno] := vote[modelno,tripletno] + 0.25;
                end;
            end;
          i2 := model[i2].link;
        end;
    end;

procedure partial(prim,i,j,f : integer;var m : integer);
var
  inkey,hkey : integer;
  inkey1 : real;
begin
  inkey1 := (0.839*i + 0.364*j)*100;
  inkey := trunc(inkey1);
  hkey :=  inkey rem prim;
  search(hkey,i,j,f,m);
end;

begin(* main *)
  open (input,'hash_table.dat',history := old);
  reset(input);
  open (candfile,'tally.dat',history := old);
  reset(candfile);
  open (ref,'ref.dat',history := old);
  reset(ref);
  open (output,'vote.dat',history := new);
  rewrite(output);
  readln(input,table_size,prim);
  readln(candfile,interestingno);
  readln(ref,th1,th2);
  test_interestingno_3 := interestingno - 3;
  initialize1;
  t1 := trunc(th1);
  t2 := trunc(th2);
  i1 :=0;j1 := 0;k1 := 1;c1 := 1;c2 := 1;
  while not eof(input) do
    begin
      with model[i1] do
        readln(input,hashkey,inkey,i,j,modelno,tripletno,key1,key2,
               basex1,basey1,basex2,basey2,basex3,basey3,link);
        i1 := i1 + 1;
    end;
  m := 0;k :=1;n := 0;f := 1;g := 2;h := 3;
(* partial voting *)
  while not eof(candfile) do
  begin
    with cand[j1] do
    begin
    readln(candfile,inkey,i,j,key1,key2,basex1,basey1,
           basex2,basey2,basex3,basey3);
    hkey := inkey rem prim;
    search(hkey,i,j,f,m);
```

```
      ix := i - 1;
      iy := i + 1;
      jx := j - 1;
      jy := j + 1;
      if ((i > 0) and (i < 91)) and ((jx > 0) and (jx < 91)) then
        partial(prim,i,jx,g,m);
      if ((i > 0) and (i < 91)) and ((jy > 0) and (jy < 91)) then
        partial(prim,i,jy,g,m);
      if ((ix > 0) and (ix < 91)) and ((j > 0) and (j < 91)) then
        partial(prim,ix,j,g,m);
      if ((iy > 0) and (iy < 91)) and ((j > 0) and (j < 91)) then
        partial(prim,iy,j,g,m);
      if ((iy > 0) and (iy < 91)) and ((jx > 0) and (jx < 91)) then
        partial(prim,iy,jx,h,m);
      if ((iy > 0) and (iy < 91)) and ((jy > 0) and (jy < 91)) then
        partial(prim,iy,jy,h,m);
      if ((ix > 0) and (ix < 91)) and ((jy > 0) and (jy < 91)) then
        partial(prim,ix,jy,h,m);
      if ((ix > 0) and (ix < 91)) and ((jx > 0) and (jx < 91)) then
        partial(prim,ix,jx,h,m);
      end;
    j1 := j1 + 1;
    count := j1 mod test_interestingno_3;
    if count = 0 then
      begin
        top_six_maximun(m,k);
        verify(j1,k,m,n);
        m := 0;
        initialize1;
      end;
    end;
    all_maximun_selected(n);
    close(input);
    close(candfile);
    close(ref);
    close(output);
end.

*****************************************************************************************

program reconstruct(input,vote,ploting);
(* This module does the inverse affine transformation to calculate
   the interesting point sets of the qualified candidate triplets *)
(* i/p : model_interesting_point and vote data files
   o/p : ploting data file *)
const
  totalmodelno = 2;
type
  ary1 = array [1..10] of integer;
  ary2 = array [1..999] of integer;
  ary3 = array [1..99] of real;
  ary4 = array [1..10,1..99] of integer;
  ary5 = array [1..3] of integer;
  ary6 = array [1..2,1..2] of integer;
  ary7 = array [1..2,1..2] of real;
```

```
  ary8 = array [1..2,1..1] of real;
var
  orimodelno : ary1;
  select_basex,select_basey,test_basex,test_basey,rest_basex,
  rest_basey : ary2;
  hashkey,inkey,modelno,tripletno : integer;
  recogmodelno,recointeresting,vot,a : integer;
  collected_key1,collected_key2,vot1 : real;
  test_recogx,test_recogy,ordered_test_inter_coordx,
  ordered_test_inter_coordy,
  ordered_test_basex,ordered_test_basey : ary3;
  x,y,oriinterestno : ary4;
  oritx,ority : ary5;
  base_matrix : ary6;
  inverse_matrix : ary7;
  diff,coord : ary8;
  deletemodelno :  array [0..totalmodelno] of integer;
  vote,ploting : text;
  korder,jr : integer;
  i : integer;

procedure matrix_mult(inverse_matrix : ary7;
                      var diff,coord : ary8);
const
  k = 2;
  l = 2;
  m = 1;
var
  i,i1,n : integer;
begin
  for i := 1 to l do
    for i1 := 1 to m do
      coord[i,i1] := 0;
    for i := 1 to l do
      for i1 := 1 to m do
        for n := 1 to l do
          coord[i,i1] := coord[i,i1] + inverse_matrix[i,n] *
                         diff[n,i1];
end;

procedure inverse(base_matrix : ary6;
var
  inverse_matrix : ary7);
var
  det : real;
begin
  det := (base_matrix[1,1] * base_matrix[2,2]) -
  (base_matrix[1,2] * base_matrix[2,1]);
  if (det <> 0.0) then
    begin
      inverse_matrix[1,1] := base_matrix[2,2]/det;
      inverse_matrix[1,2] := base_matrix[1,2] * (-1) /det;
      inverse_matrix[2,1] := base_matrix[2,1] * (-1) /det;
      inverse_matrix[2,2] := base_matrix[1,1] /det;
    end;
end;
```

```
procedure calculate_coord(recogmodelno : integer;
                          cbasex,cbasey : ary5);
var
  i,m,i1,i2,i3 : integer;
begin
  for i := 1 to 2 do
    begin
      base_matrix[1,i] := cbasex[i] - cbasex[i+1];
      base_matrix[2,i] := cbasey[i] - cbasey[i+1];
    end;
  inverse(base_matrix,inverse_matrix);
  i2 := 1;
  for i := 1 to oriinterestno[orimodelno[recogmodelno],
                              recogmodelno] do
    begin
      if ((cbasex[1] <> x[recogmodelno,i])
         or (cbasey[1] <> y[recogmodelno,i])) and
         ((cbasex[2] <> x[recogmodelno,i])
         or (cbasey[2] <> y[recogmodelno,i])) and
         ((cbasex[3] <> x[recogmodelno,i])
         or (cbasey[3] <> y[recogmodelno,i])) then
        begin
          rest_basex[i] := x[recogmodelno,i];
          rest_basey[i] := y[recogmodelno,i];
          diff[1,1] := rest_basex[i] - cbasex[3];
          diff[2,1] := rest_basey[i] - cbasey[3];
          matrix_mult(inverse_matrix,diff,coord);
          ordered_test_inter_coordx[i]:=coord[1,1]*
          (ordered_test_basex[1][ordered_test_basex]2[)+
           coord[2,1]*(ordered_test_basex[2] -
                       ordered_test_basex[3])+
                       ordered_test_basex[3];
           ordered_test_inter_coordy[i]:=coord[1,1]*
          (ordered_test_basey[1][ordered_test_basey]2[)+
           coord[2,1]*(ordered_test_basey[2] -
                       ordered_test_basey[3])+
           ordered_test_basey[3];
        end;
    end;
  for i := 1 to oriinterestno[orimodelno]recogmodelno[,
                              recogmodelno] do
  writeln(ploting,ordered_test_inter_coordx[i],
          ordered_test_inter_coordy[i]);
end;

rocedure colinear(var comx,comy : ary5;
                  var t1,t2 : real);
var
  colinearx1,colinearx2,colineary1,colineary2 : integer;
begin
  colinearx1 := comx[2] - comx[1];
  colinearx2 := comx[3] - comx[2];
  colineary1 := comy[2] - comy[1];
  colineary2 := comy[3] - comy[2];
  if (colinearx1 <> 0) then
```

```
    t1 := arctan(colineary1 / colinearx1)
  else t1 := 1.5707;
  if (colinearx2 <> 0) then
    t2 := arctan(colineary2 / colinearx2)
  else t2 := 1.5707;
end;

procedure pretest(i,j : integer;
                  var t1,t2 : real);
var
  count : integer;
begin
  count := 1;
  while ((test_recogx[j]<>0) or (test_recogy[j]<>0)) and
        (count < 4) do
    begin
      oritx[count]:=x[i,j];
      ority[count]:=y[i,j];
      ordered_test_inter_coordx[j] := test_recogx[j];
      ordered_test_inter_coordy[j] := test_recogy[j];
      ordered_test_basex[count] := test_recogx[j];
      ordered_test_basey[count] := test_recogy[j];
      j := j+1;
      count := count+1;
    end;
  colinear(oritx,ority,t1,t2);
end;

procedure mapping(bx,by : real;
                  var ordering : integer;
                  i : integer);
var
  j : integer;
begin
  for j := 1 to oriinterestno[orimodelno[i],i] do
     begin
       if (abs(bx - x[i,j]) < 0.00001) and
          (abs(by - y[i,j]) < 0.00001) then
         ordering := j;
     end;
end;

procedure interest;
var
  i,j : integer;
begin
  open(input,'model_interesting_point.dat',history := old);
  reset(input);
  open(ploting,'ploting.dat',history := new);
  rewrite(ploting);
  i:=1;
  while not eof(interesting) do
    begin
      read(interesting,orimodelno[i]);
      readln(interesting,oriinterestno[orimodelno[i],i]);
      for j := 1 to oriinterestno[orimodelno[i],i] do
```

```
      readln(interesting,x[orimodelno[i],j],
             y[orimodelno[i],j]);
      i := i+1;
    end;
end;

procedure recognized;
var
  i,i1,i2,j,j1 : integer;
  select_model_tempx,select_model_tempy,t1,t2 :real;
begin
  open (vote,'vote.dat',history := old);
  reset(vote);
  i :=1;
  while (not eof(vote)) do
  begin
    readln(vote,hashkey,inkey,modelno,tripletno,collected_key1,
           collected_key2,
           select_basex[1],select_basey[1],select_basex[2],
           select_basey[2],select_basex[3],select_basey[3],vot1,a);
    readln(vote,inkey,collected_key1,collected_key2,
           test_basex[1],test_basey[1],test_basex[2],
           test_basey[2],test_basex[3],test_basey[3]);
    recogmodelno := modelno;
    if (vot1/(oriinterestno[orimodelno[recogmodelno],recogmodelno]-3) >= 0.5) then
      begin
      for J:=1 to 3 do
        begin
          mapping(select_basex[j],select_basey[j],jr,
                  recogmodelno);
          test_recogx[jr] := test_basex[j];
          test_recogy[jr] := test_basey[j];
        end;
      select_model_tempx := collected_key1*(select_basex[1]-
      select_basex[2])+ collected_key2*(select_basex[2]-
      select_basex[3])+select_basex[3];
      select_model_tempy := collected_key1*(select_basey[1]-
      select_basey[2])+ collected_key2*(select_basey[2]-
      select_basey[3])+select_basey[3];
      mapping(select_model_tempx,select_model_tempy,jr,
      recogmodelno);
      test_recogx[jr] := collected_key1*(test_basex[1]-
      test_basex[2])+
      collected_key2*(test_basex[2]-test_basex[3])
      +test_basex[3];
      test_recogy[jr] := collected_key1*(test_basey[1]-
      test_basey[2])+
      collected_key2*(test_basey[2]-test_basey[3])
      +test_basey[3];
      for j1 := 1 to a[1 do
      begin
      readln(vote,hashkey,inkey,modelno,tripletno,
             collected_key1,collected_key2,
             select_basex[1],select_basey[1],select_basex[2],
             select_basey[2],select_basex[3],select_basey[3]);
             readln(vote,inkey,collected_key1,collected_key2,
```

```
            test_basex[1],test_basey[1],test_basex[2],
            test_basey[2],test_basex[3],test_basey[3]);
        select_model_tempx := collected_key1*(select_basex[1]-
        select_basex[2])+ collected_key2*(select_basex[2]-
        select_basex[3])+select_basex[3];
        select_model_tempy := collected_key1*(select_basey[1]-
        select_basey[2])+ collected_key2*(select_basey[2]-
        select_basey[3])+select_basey[3];
        mapping(select_model_tempx,select_model_tempy,jr,
                recogmodelno);
        test_recogx[jr] := collected_key1*(test_basex[1]-
        test_basex[2])+collected_key2*
        (test_basex[2]-test_basex[3])+test_basex]3[;
        test_recogy[jr] := collected_key1*(test_basey[1]-
        test_basey[2])+collected_key2*
        (test_basey[2]-test_basey[3])+test_basey]3[;
        end;
        writeln(ploting,recogmodelno,oriinterestno[
                orimodelno[recogmodelno],recogmodelno]);
        i1:=1;t1:=1;t2:=1;
        pretest(recogmodelno,i1,t1,t2);
        if t1<>t2 then calculate_coord(recogmodelno,
                                       oritx,ority)
        else
        pretest(recogmodelno,i1+1,t1,t2);
        end;
      end;
end;

egin(* main *)
 interest;
 recognized;
end.
```

LIST OF REFERENCES

1. Quek Gim Pew, "Recognition of Two-Dimentional Shapes," Master's thesis, Naval Postgraduate School, Monterey, California, 1986.

2. Mokhtarian, F., and Mackworth, A., "Scale-Based Description and Recognition of Planar Curves and Two-Dimentional Shapes," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, pp. 34-43, 1986.

3. Lamdan, Y., Schwartz, J.T., and Wolfson, H.J., "Object Recognition by Affine Invariant Matching," Presented at *IEEE Conference on Computer Vision and Pattern Recognition*, June, 1988.

4. Costa, M., Haralick, R.M., and Shapiro, L.G., "Optimal Affine-Invariant Point Matching," Private communication from Washington University, 1989.

5. Martin, J.T., "Computer Data-Base Organization," 2nd ed, Prentice-Hall, Inc. Englewood Cliffs, N.J. 1977.

6. Hsu, Tao-I, "Affine Invariant Object Recognition by Voting Techniques," Master's thesis, Naval Postgraduate School, Monterey, California, 1988.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Department Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Professor Chin-Hwa Lee, Code 62Le Naval Postgraduate School Monterey, CA 93943-5000	10
5.	Professor Murali Tummala, Code 62Tu Naval Postgraduate School Monterey, CA 93943-5000	1
6.	Ta Fu Proving Groung Library P.O. Box 90502-1, Ta-Fu, I-Lan Taiwan, R.O.C.	2
7.	Chung Cheng Institute of Technology, Main Library P.O. Box 8243, Ta-Hsi, 33500, Tao-Yuan Taiwan, R.O.C.	1
8.	Kao, Chang-Lung 7, Lane 72, Tzu-Lin, Chin-Tan, 23180 Sin-Tien County, Taipei, Taiwan, R.O.C.	6
9.	Wang, Chen-Shan SMC 1596, NPS Monterey, CA 93943-5000	5