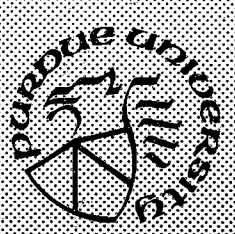
12-1-1988

3D-POLY: A Robot Vision System for Recognizing Objects in Occluded Environments

C. H. Chen
Purdue University

A. C. Kak
Purdue University



3D-POLY: A Robot Vision System for Recognizing Objects in Occluded Environments

C. H. Chen
A. C. Kak

TR-EE 88-48
December 1988

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

3D-POLY: A ROBOT VISION SYSTEM FOR RECOGNIZING OBJECTS IN OCCLUDED ENVIRONMENTS

C. H. Chen and A. C. Kak

Robot Vision Lab
School of Electrical Engineering
Purdue University
W. Lafayette, IN 47907

Technical Report TR-EE 88-48
December 1988

ABSTRACT

The two factors that determine the time complexity associated with model-driven interpretation of range maps are: 1) the particular strategy used for the generation of object hypotheses; and 2) the manner in which both the model and the sensed data are organized, data organization being a primary determinant of the efficiency of verification of a given hypothesis. In this report, we present 3D-POLY, a working system for recognizing objects in the presence of occlusion and against cluttered backgrounds. The time complexity of this system is only $O(n^2)$ for single object recognition, where n is the number of features on the object. The most novel aspect of this system is the manner in which the feature data are organized for the models. We use a data structure called the *feature sphere* for the purpose. We will present efficient algorithms for assigning a feature to its proper place on a feature sphere, and for extracting the neighbors of a given feature from the feature sphere representation. For hypothesis generation, we use *local feature sets*, a notion similar to those used before us by Bolles, Shirai and others. The combination of the feature sphere idea for streamlining verification and the local feature sets for hypothesis generation results in a system whose time complexity has a polynomial bound.

In addition to recognizing objects in occluded environments, 3D-POLY also possesses model learning capability. Model learning consists of looking at a model object from different views and integrating the resulting information. The 3D-POLY system also contains utilities for range image segmentation and classification of scene surfaces.

TABLE OF CONTENTS

Page

Page

INTRODUCTION

The goal of this research is to develop a robot vision system that can recognize and locate objects randomly positioned and oriented in 3-D space, possibly occurring in heaps. Idealy, the capability of such a vision system should approximate that of the human vision system [*Ma*−82], which can perform recognition of a wide variety of objects in real time even under poor lighting condition. While computer vision research in the last 25 years or so has only proved that such a real-time general vision system remains a distant goal, by using active sensors it is possible today to design systems that can indeed identify objects and compute their poses and do so with a measure of robustness in occluded environments.

In general, one must address the following issues when designing a robot vision system:

- **Image acquisition:**
  The question here is what types of images one should use and how they should be acquired.
- **Feature extraction:**
  What features should be extracted from an image in order to describe the shapes and geometrical relation of the object surfaces seen in the image.
- **Model representation:**
  How one should represent object knowledge that would allow efficient retrieval of model data.
- **Matching algorithm:**
  Here the issue is how image features should be matched with object features.

It should be emphasized that these four main issues are highly interrelated, in the sense that the representations and methods used for one have a bearing on the representations and methods used for others. The reason for this interdependence is the fact that in most cases the overall flow of control in the recognition process corresponds to what is shown in Fig. 0.1. Clearly, how features are extracted depends to some extent on the type of acquired data.

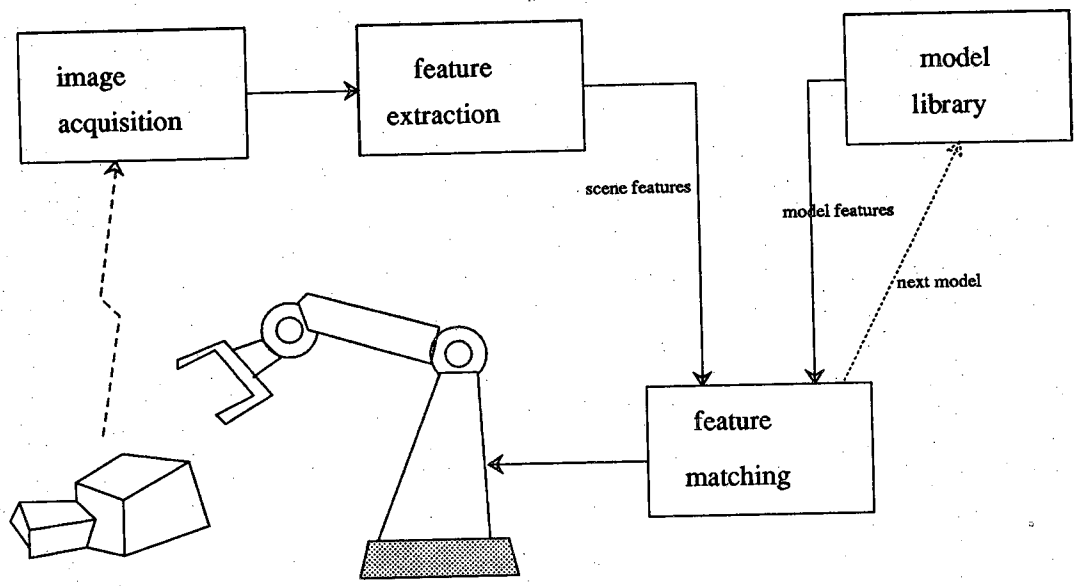


Figure 0.1. A system diagram of the four basic components of an object recognition system.

Clearly, object recognition calls for the extraction of scene features, measurement of their attributes, such as shape and other geometrical characteristics, and the relationships of the features to one another. Since features play a pivotal role in recognition, we must choose those features that can be detected reliably from images and that possess sufficient discriminatory power for distinguishing between objects. In addition, since we also need to locate the objects, i.e. to determine their positions and orientations, the set of features should also provide spatial information about the objects. For these reasons, 3D-POLY uses geometric features that constitute the shapes of objects. Geometric features include surfaces, edges and points, and each of them is specified by a set of attributes such as shape, relation and position/orientation.

It stands to reason that the data acquired about a scene must allow us to extract these geometric features. For various reasons, 2-D reflectance images can not be used and one must take recourse to range maps, where each data element in the image is a quantized representation of distance to an object point from a reference plane or point. An extensive survey of various techniques for acquiring range images can be found in [*Ka*−85, *Be*−87]. In 3D-POLY, range images are generated by using structured light scanning.

A structured light range sensor consists of a light projector and a camera; the projector casts a stripe of light onto object surfaces and the camera detects the illuminated stripes. The range to any illuminated surface point is computed by using triangulation formulas. In the first chapter of this report, we show how perspective geometry can be used to derive a 4×3 calibration matrix that directly converts an image point into its corresponding 3-D coordinates. We also present a simple to use experimental procedure that yields this calibration matrix. We will use the phrase 'range map' to also refer to the (x,y,z) data obtained for all the illuminated points in a scene.

It is necessary to go through several processing and detection steps before geometric features can be extracted from a range map. These are presented in Chapter 2. We first describe a procedure for the computation of surface normals from range images. Over smooth surfaces this procedure works like the traditional ones in that a surface normal is computed by fitting a planar patch to the range map over a small window. However, in the vicinity of edges between different surfaces, our procedure has the virtue of placing the windows adaptively in such a manner that the surface normal computation does not get corrupted by attempting to fit a planar patch across an edge between two different surfaces. In Chapter 2, we also discuss the detection and the classification of three primitive surface types, namely, planar, cylindrical and conical.

Chapter 3 presents the heart of 3D-POLY. There we have discussed how the system generates and verifies hypotheses about object identities and poses. The hypothesis

generation and verification strategies presented there reduce the otherwise exponential time complexity to a low-order polynomial bound. For hypothesis generation surface features are grouped around vertices into local feature sets. For verification, 3D-POLY uses a special feature called principal direction that posses a separate definition for each different type of feature. Principal directions are used to organize the model data into a data structure called the feature sphere. It will be shown that using local feature sets for hypothesis formation and feature spheres for verification allows 3D-POLY to recognize objects with very low time complexity.

To be complete, an object recognition system must have the means to learn object models. 3D-POLY posses such capability, which is described in Chapter 4. We describe there a multi-view integration procedure for synthesizing the shape of an object. The major issue in model multi-view integration for shape synthesis is the detection of feature common to different views, a problem that is made especially difficult by the fact that the attribute values for the same feature may be different in different views.

CHAPTER 1
MODELING AND CALIBRATION OF STRUCTURED LIGHT SCANNERS

In this Chapter we have used projectivity theory to model the process of structured light scanning for 3D robot vision. The projectivity formalism is used to derive a 4×3 transformation matrix that converts points in the image plane into their corresponding 3D world coordinates. Calibration of the scanner consists of computing the coefficient of this matrix by showing to the system a set of lines generated by suitable object edges. We end this paper by showing how the matrix can be used to convert image pixel locations into the world coordinates of the corresponding object points using two different scanning strategies.

1.1. Introduction

Structured light scanning is a rugged approach to range mapping a scene for 3D robot vision. In order to take full advantage of the flexibility for viewpoint selection made possible by a six-degree-of-freedom robot, we use a portable structured light unit that can be picked up by the robot when it wants to gather 3D vision data (Fig. 1.1). Within the constraints imposed by manipulator kinematics, the unit can then be oriented in any direction deemed desirable by the robot for the task at hand, and scanned either in a translational or a rotational mode for data collection.

A structured light unit consists basically of a light projector and a camera. The light projector throws a plane of light in the direction of the scene. The intersection of this plane with an object creates a stripe of illuminated points on the object surface, the stripe being recorded in the camera image plane. If the unit is properly calibrated, the world coordinates of the illuminated points can be calculated by using triangulation formulas, as has been done by Agin [*Ag* –82]. Agin used a 4×3 collineation matrix to write down a geometric relationship between the illuminated pixel coordinates and the world coordinates of the corresponding object points. The coefficients of this matrix are explicit functions of the camera and projector parameters. Calibration of the system implies determination of the coefficients of this matrix, which requires that the camera and the projector parameters be precisely known -- these parameters being positions and orientations of the camera and the projector, and the internal magnifications of the camera

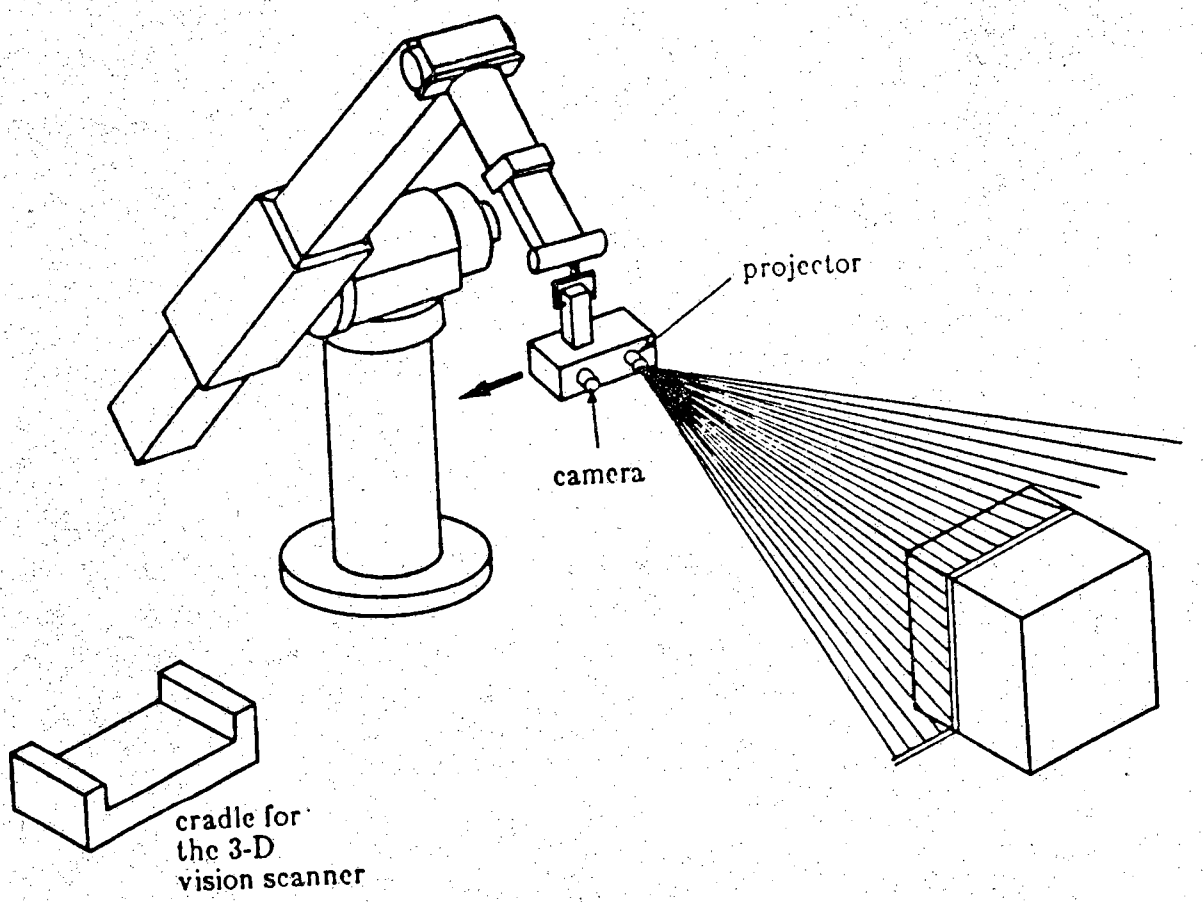


Figure 1.1. Robot engaged in scanning a scene with a detachable structured-light scanner.

lens system. Because of the explicit dependence of the matrix coefficients on such parameters, Again had to first calibrate the robot joints so that the required positions could pinned down precisely, and then he had to individually calibrate the camera aim, camera scale and the projector aim.

In this report, we look at the calibration problem from a different point of view. The basic goal of structured-light calibration is to find a formula that converts the 2-D coordinates of a recorded pixel in the image plane to the world coordinates of the corresponding object point. Our position is that it should be possible to obtain this relationship for a structured light system without having to worry about such low-level details as the precise locations and aiming vectors for the camera and the projector. However, we do not believe that it is possible to do away with the requirement that the robot itself be mechanically calibrated before it can be used in conjunction with a structured light system. In fact, the accuracy of the methods to be proposed in this report will be no better than the absolute accuracy of the robot.

Note that the problem of deriving formulas that take us from 3-D world coordinates to 2-D image coordinates and vice versa also arises in straightforward camera imaging. As is well known [*D&H*–73], it is possible to write down a 3×4 homogeneous transformation matrix that for a given object point yields uniquely its corresponding image point; but, if we desire a transformation in the reverse direction, viz, from the image to the world, it is only possible to calculate the direction to the object point -- and not its location -- by using a similar matrix.

In Section 2, we will show that for the case of structured light imaging if we apply the theory of projectivity to relate the points in the light plane with the corresponding points in the image plane, it is indeed possible to derive a 4×3 homogeneous transformation matrix that is reversible. This implies that for each object point of *a priori* known location, we can uniquely determine its camera image plane coordinates; and for each image point we can uniquely determine the world coordinates of the corresponding object point.

As we will show, the transformation matrix derived from the projectivity theory makes unnecessary the precise calculations of the locations of the camera and the projector and their aiming angles. Therefore, it is no longer critical that the robot joints be calibrated precisely, at least from the standpoint of enhancing the accuracy of range mapping.

We will also show that although from a purely theoretical standpoint only four object points at known locations are required for calibration -- meaning the computation of the elements of the transformation matrix -- the practical difficulty consisting of

knowing where exactly the object points are located has caused us to seek other approaches. We will describe our procedure which consists of showing to the robot at least six lines generated by suitable object edges in the scene. In this procedure, it is not necessary to know the exact locations of the beginnings and the ends of the lines, as long as their relative separations are known. Section 3 presents a procedure for computing the optimum values of the calibration matrix when more than six lines are shown to the robot.

Once a structured light system is calibrated, the process of scanning for the purpose of range mapping a scene can take various forms. We will talk about two methods: rotational scanning and linear scanning. In Section 4, we will formulate coordinate transformations for both methods.

Finally, in Section 5, we will show some calibration results and compare our technique with the two-plane calibration method.

1.2. Projective Geometry

First, we will define the notation used in this report.

$X, U, P, A, \cdots$
: An italic upper case letter refers to a point which may be on a line, on a plane, or in 3D space. Usually, X, Y, Z are points in space, and U, V are points in the image plane.

$X_b, X_s, \cdots$
: An italic upper case letter with a subscript also refers to a point, but in this case the homogeneous coordinates of the point are also specified. The subscript denotes the coordinate frame in which the point is defined.

$\boldsymbol{X}_b, \boldsymbol{X}_s, \cdots$
: Bold italic upper case letters with subscripts are used to denote the regular coordinates of a point.

$\boldsymbol{r}, \boldsymbol{s}, \boldsymbol{t}, \cdots$
: A bold italic lower case letter is used to denote a line or a plane.

$F_{2s}, F_b, \cdots$
: Letter F with a subscript is used for representing a coordinate frame. The subscript 2 specifies a two dimensional coordinate frame.

$T_{cb}, \cdots$
: Letter T with a subscript represents a transformation from one coordinate system to another. The first letter of the subscript denotes the original coordinates system,

while the second letter denotes the destination coordinate system.

1.2.1. One Dimensional Projectivity

On a plane, given a center of projection P and any two lines $\boldsymbol{s}$ and $\boldsymbol{r}$ not passing through P, as shown in Fig. 1.2, a one-dimensional projectivity is defined as follows: Let X be a point on line $\boldsymbol{s}$, its projective image X' on line $\boldsymbol{r}$ is the intersection of line PX with $\boldsymbol{r}$. Let A,B,C,D be any four distinct points on line $\boldsymbol{s}$, the *cross ratio* of A, B with respect to C, D is

$$(A,B;C,D) = \frac{AC}{BC} \cdot \frac{BD}{AD}$$

Let A',B',C',D' on line $\boldsymbol{r}$ be, respectively, the image points of A, B, C, D under the projectivity shown. An important property that follows from projectivity is the invariance of the cross-ratio. This invariance can be expressed as

$$(A,B;C,D) = (A',B';C',D') \tag{1.1-a}$$

or

$$\frac{AC}{BC}\,\frac{BD}{AD} = \frac{A'C'}{B'C'}\,\frac{B'D'}{A'D'} \tag{1.1-b}$$

With this relation established between $\boldsymbol{s}$ and $\boldsymbol{r}$, we can find the image point X' of X under this projectivity by substituting X for D, and X' for D':

$$(A,B;C,X) = (A',B';C',X') \tag{1.2}$$

It is obvious that the two corresponding sets of triplets, $\{A, B, C\}$ and $\{A', B', C'\}$, completely describe the projectivity on $\boldsymbol{s}$ and $\boldsymbol{r}$ from the projection center P. One may raise the following questions at this point: Can we always find a projectivity on a plane which converts a set of points on one line to a set of points on another line ? Is this projectivity unique? Answers to these questions, which are crucial to the main theme of this paper, are provided by the following theorem [*Ay*–67]:

The Fundamental Theorem of One Dimensional Projectivity

Given three distinct points on a line and another three points on a second line, there is one and only one projectivity which carries the first three points respectively into the second three points.

To illustrate the theorem, we first locate three points A, B, C at arbitrary places on a line $\boldsymbol{s}$ and another three points A', B',C' on a line $\boldsymbol{r}$ (Fig 1.3-a). For finding the unique projectivity, we will fix the line $\boldsymbol{s}$ in the plane and move around the line $\boldsymbol{r}$ on the plane

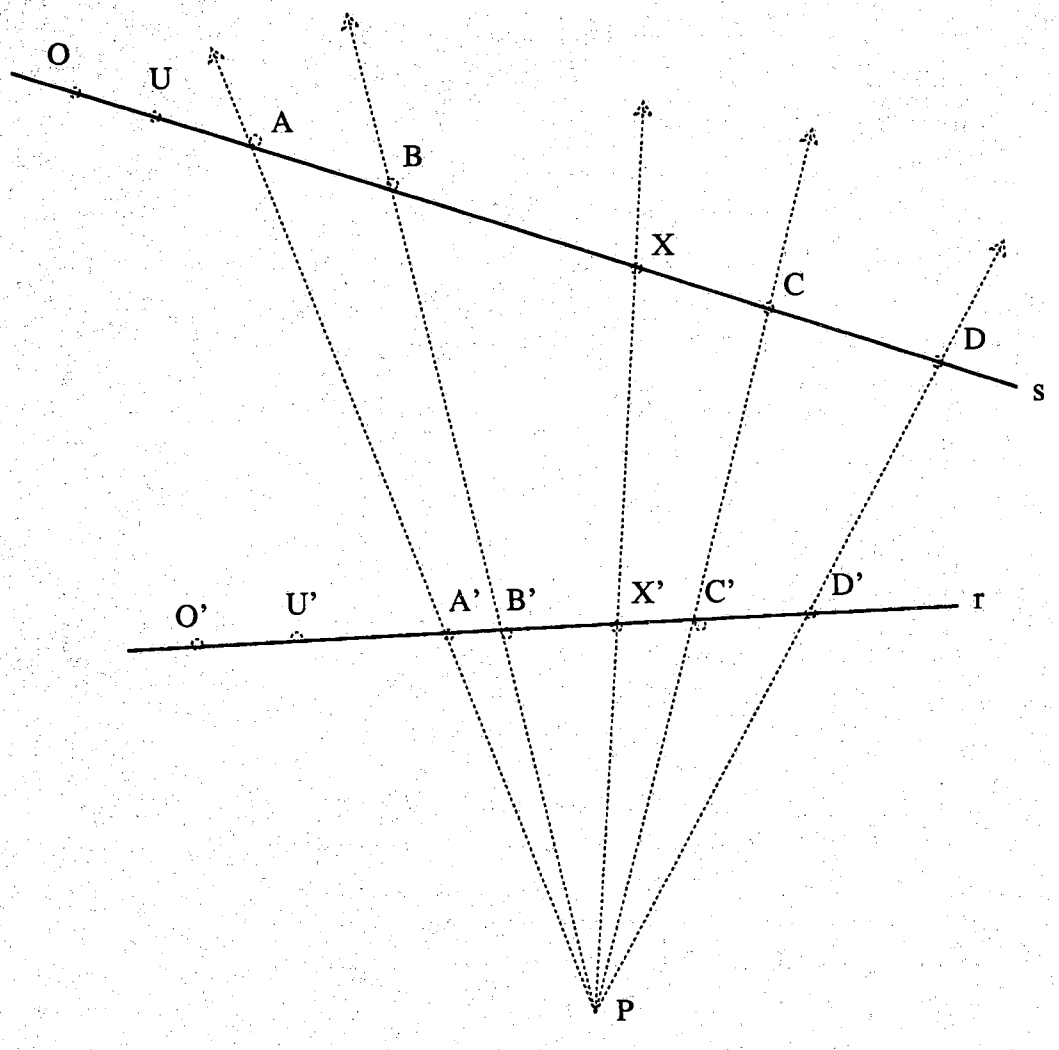


Figure 1.2.
One dimensional projectivity is illustrated here.

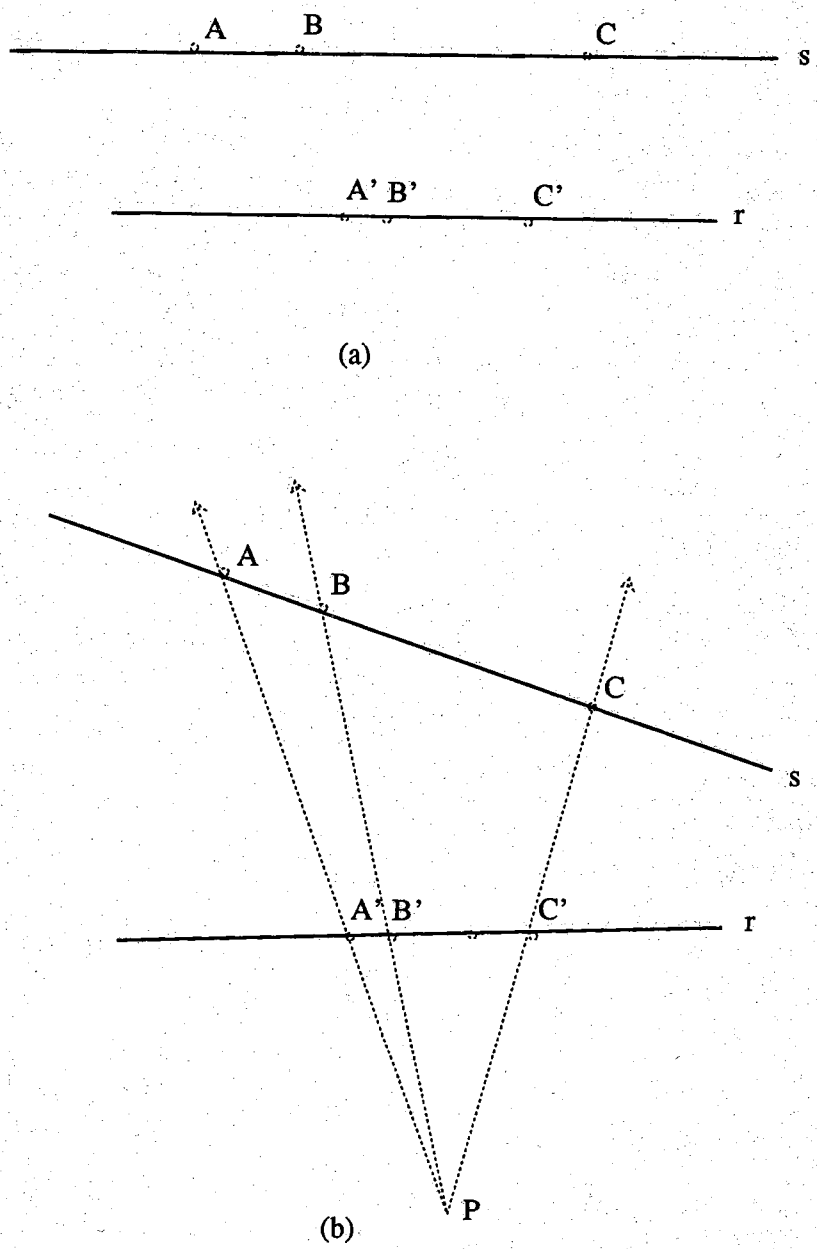


Figure 1.3.

a) Three points defined on each of the two lines that will be used for demonstrating 1-D projectivity. b) If we fix line s of (a) and move around line r shown there, there will exist only one projectivity for which AA', BB' and CC' will meet at a point.

until the three lines AA', BB', CC' meet at one point (Fig 1.3-b); this common point of intersection is the projection center of the projectivity. A more difficult case is shown in Fig. 1.4-a, in which the corresponding points on lines s and r are ordered differently. The projectivity for this case is shown in Figure 1.4-b.

For representing a point on a line, we need to define a coordinate system to express its position on the line. The familiar coordinate system on a line is established by selecting on the line a point O from which all measurements along the line are made, a unit of measure, and a sense of direction. Essentially, this consists of selecting a point O, called the origin, and U, called the unit point; to these two points we assign the coordinate values 0 and 1 respectively. The coordinate x of a point X on the line is then the directed distance of X from O. If on the other hand, a homogeneous coordinate system is desired, that can be done by assigning coordinates $(0,1)$ to O, $(1,1)$ to U, and (x_1, x_2) to any point X such that $x_1 / x_2 = x$. It is obvious that a point does not have a unique representation in a homogeneous coordinate system.

Let's say that we have chosen an origin O and a unit point U to define a coordinate system for a line s. Also, let O' and U' define a coordinate system for another line r. We do not require that the unit length OU on line s be equal to the unit length $O'U'$ on line r. We also do not require that the points O' and U' be the images of the points O and U under any projectivity. In fact, equation (1.2) is independent of the coordinate systems defined on either lines in the projectivity; this is a consequence of the following theorem that we present without proof:

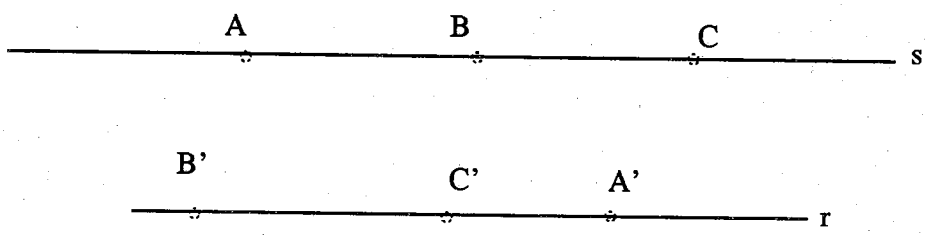
Theorem of Cross Ratio

The cross ratio of any four points on a line is independent of the coordinate system established on the line.

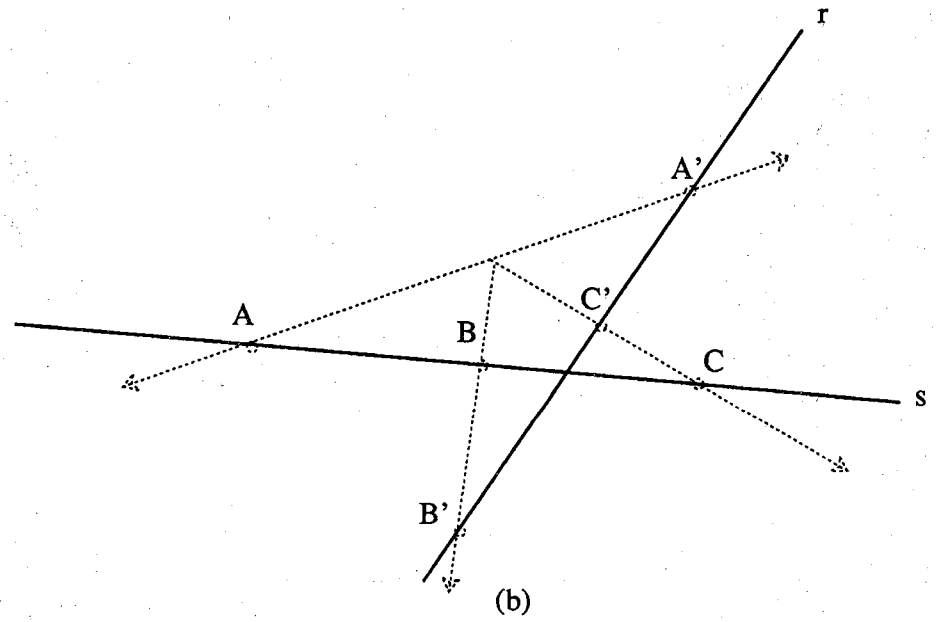
Given a point x on, say, the line s, it is a simple matter to derive a formula for the corresponding point on line r. With respect to the coordinate system on line s, let the points A, B, C, X have coordinates a, b, c, x respectively. Similarly on line r, let the points A', B', C', X' have coordinates a', b', c', x' respectively. Then equation (1.2) can be rewritten as:

$$\frac{(a-c)\,(b-x)}{(b-c)\,(a-x)} = \frac{(a'-c')\,(b'-x')}{(b'-c')\,(a'-x')} \tag{1.3}$$

We now solve (1.2) for x' in terms of x. Setting $\frac{(a-c)}{(b-c)} = \alpha$ and $\frac{(a'-c')}{(b'-c')} = \beta$, we have



(a)



(b)

Figure 1.4.

a) An example similar to that of Figure 3a except that the order of the three points on line r is opposite to the order on line s. b) The unique projectivity that corresponds to the case shown in (a).

$$x' = \frac{a_{11}\,x + a_{12}}{a_{21}\,x + a_{22}}$$

where $a_{11} = \alpha a' - \beta b'$, $a_{12} = ab'\beta - a'b\alpha$, $a_{21} = \alpha - \beta$, $a_{22} = a\beta - b\alpha$. In terms of homogeneous coordinates, we have, by setting $x = x_1 / x_2$, and $x' = x'_1 / x'_2$,

$$\frac{x'_1}{x'_2} = \frac{a_{11}x_1 + a_{12}x_2}{a_{21}x_1 + a_{22}x_2}$$

or

$$\begin{cases} \rho x'_1 = a_{11}x_1 + a_{12}x_2 \\ \rho x'_2 = a_{21}x_1 + a_{22}x_2 \end{cases}, \quad \rho \neq 0$$

In matrix form, we have

$$\rho \begin{bmatrix} x'_1 \\ x'_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{1.4}$$

Note that the existence of the free variable ρ. Since a point in homogeneous coordinates does not have a unique expression, that is, $x' = x_1 / x_2 = \rho x_1 / \rho x_2$. With the help of this free variable, we are ensured that regardless of the homogeneous coordinates chosen, the above expression for the projectivity solution will always satisfy equation (1.4). Also note that the roles of X and X' are exchangeable. We could consider X as the image of X', and we will get the same form of matrix equation as (1.3).

1.2.2. Two Dimensional Projectivity

We can establish a formalism for two dimensional projectivity in 3D space that is similar to the one dimensional projectivity in a plane. Let s and r be two planes in space and let there be a point P, which is neither on s nor on r, to be used as the center of projection (Fig. 1.5). For each point X on s, its image point X' on r is the intersection of line PX with plane r. It is obvious that the invariance of the cross-ratio is still valid for any four collinear points on s and their images point on r. Also, for any collinear points on s, their image points are also collinear. Extending the fundamental theorem of one dimensional projectivity, we have:

The Fundamental Theorem of Two Dimensional Projectivity

Given four distinct non-collinear points on a plane and another four distinct non-collinear points on the other plane, there is one and only one projectivity which carries the first four points respectively into the second four points.

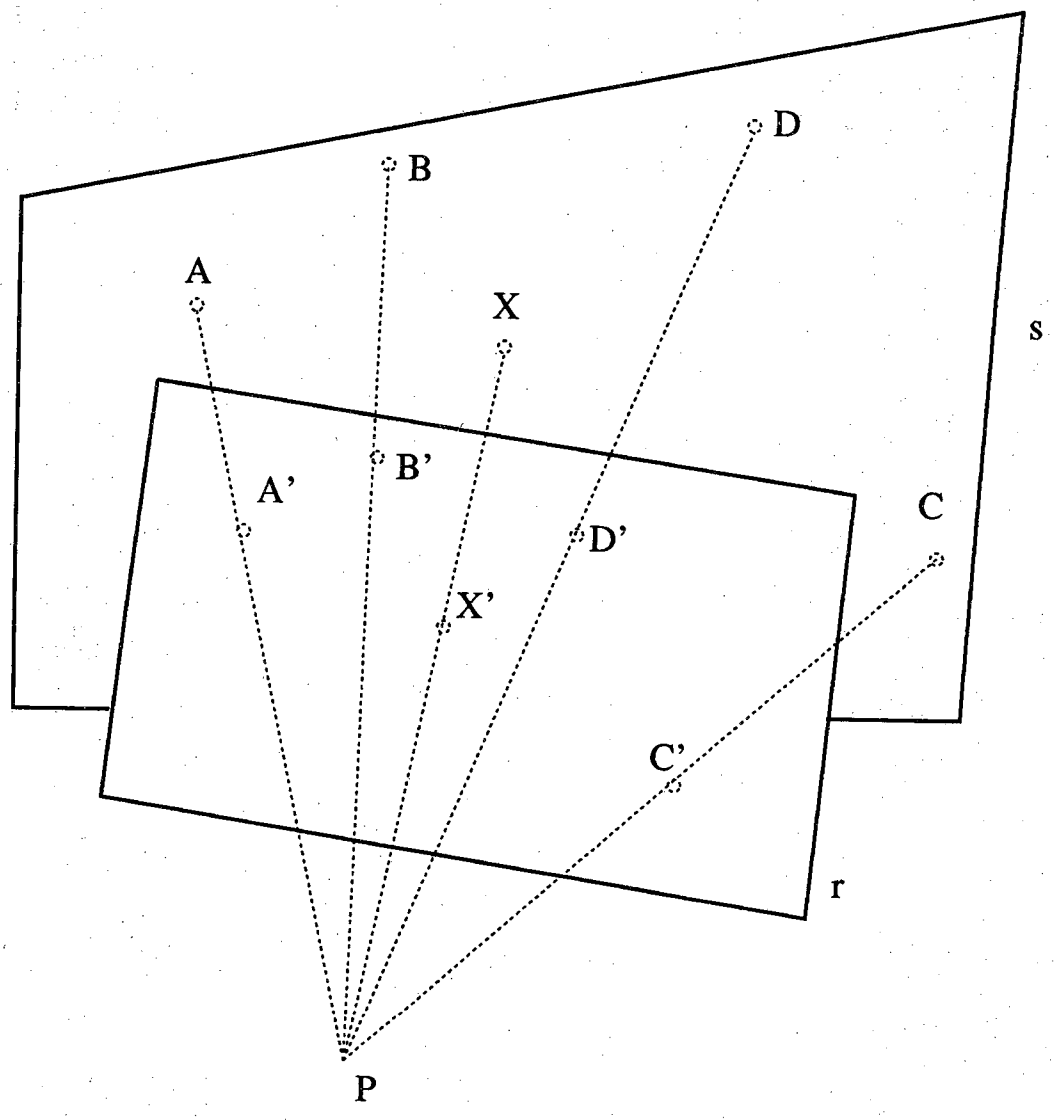


Figure 1.5.
Elements of two dimensional projectivity.

A homogeneous coordinate system can also be established on a plane by a simple extension of what was done for line projectivity. Suppose we choose a point (0, 0, 1) as the origin in a plane and use two orthogonal unit points, (1, 0, 1) and (0, 1, 1), to lay out a coordinate frame in the plane. The homogeneous coordinates of any point in the plane are given by $(x_1\ x_2\ x_3)$ with $x_3 \neq 0$; $(x_1/x_3,\ x_2/x_3)$ are the regular coordinates of the point. Analogous to the derivation of equation (1.4), we can get a 3×3 conversion matrix which converts a point X on plane $\boldsymbol{s}$ to its image point X' on plane $\boldsymbol{r}$, both points being expressed using homogeneous coordinates:

$$\rho \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{21} & a_{22} & a_{33} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

It is easy to verify that this equation preserves collinearity and invariance of the cross-ratio. Again, if we switch the roles of X and X', the above generic equation is still valid, i.e.,

$$\rho \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{21} & e_{22} & e_{33} \end{bmatrix} \cdot \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} \tag{1.5}$$

A structured light scanner can be modeled by using 2D projectivity as follows. We use the camera-focus as the center of projection P, and treat the light stripe plane as plane $\boldsymbol{s}$ and the camera image plane as plane $\boldsymbol{r}$. [This model is only valid under the condition that it be possible to use the pin-hole model for the camera (Fig. 1.6).] Although the coordinate system on the image plane can be arbitrary, a convenient definition consists of using the row index u and column index v of the digitized image as its two coordinates, and choosing the center of image plane as the origin. We will denote this coordinate frame on the image plane by F_{2c}. A point U in the image plane then has coordinates $(u,\ v)$ or, in a homogeneous coordinates system, $(u,\ v,\ 1)$ with respect to F_{2c}.

We also need to define a coordinate system on the light stripe plane. By virtue of the previous theorem, which says that the cross-ratios are independent of the choice of the coordinates system, we have considerable latitude in how we go about setting up this coordinate frame. We therefore choose one that can be easily related to the three dimensional base coordinate frame F_b for the robot. We will use $x,\ y, z$ to represent the three orthogonal axes in F_b. Then a point X_b defined in the frame F_b will have homogeneous coordinates $w(x,\ y,\ z,\ 1) = (wx,\ wy,\ wz,\ w)$. Imagine a translation and a rotation that brings F_b to a coordinate frame F_s whose center is on the plane $\boldsymbol{s}$ and whose xy plane is aligned with the plane $\boldsymbol{s}$. Since F_s is defined with respect to the base

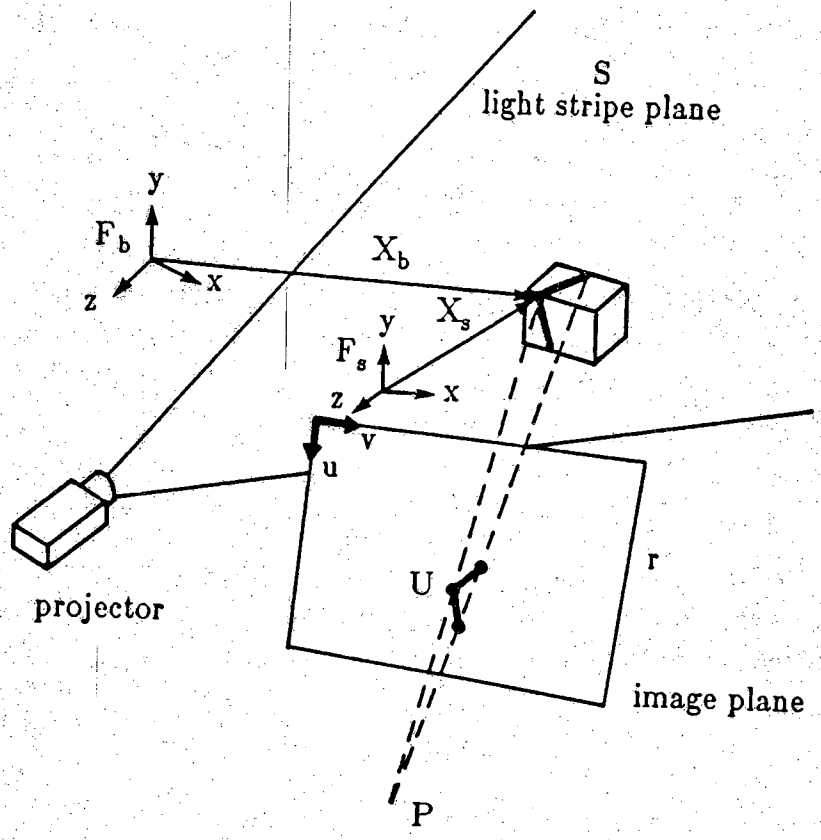


Figure 1.6. This figure shows that the structured light imaging process can be fit precisely into 2-D projectivity. We can consider the light stripe plane as plane **s** and the camera image plane as plane **r** in drawing correspondence with Figure 5. The camera focus center becomes the center of projection.

coordinate frame F_b, F_s contains all the information regarding the translation and rotation. Inheriting the coordinate system defined on the xy plane of the frame F_s, we can define a two dimensional coordinate frame F_{2s} on the plane s. Suppose a point X on plane s is assigned homogeneous coordinates (x_1, x_2, x_3) with respect to F_{2s}, where $x_3 \neq 0$. With respect to the frame F_s, which is three dimensional, the homogeneous coordinates of the same point are $(x_1, x_2, 0, x_3)$. The conversion of X from its two dimensional homogeneous coordinates in F_{2s} to its three dimensional homogeneous coordinates in F_s can then be written as

$$\begin{bmatrix} x_1 \\ x_2 \\ 0 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{1.6}$$

Now let X_s be the homogeneous coordinates of X with respect to the frame F_s. We can convert X_s to the homogeneous coordinates representation X_b with respect to the base frame F_b by multiplying X_s with F_s, that is

$$X_b = F_s \cdot X_s \tag{1.7}$$

Here F_s is a 4×4 matrix.

Substituting $(u,v,1)$ for (x'_1,x'_2,x'_3) in equation (1.5) and combining equations (1.6) and (1.7), we get a 4×3 conversion matrix T_{cb} that converts a point U in camera image plane to a light stripe point X_b in the robot base coordinate frame.

$$X_b = T_{cb} \cdot U$$

or

$$\rho \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \\ t_{41} & t_{42} & t_{43} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{1.8}$$

Note that we use subscript b to denote that X_b is in homogeneous coordinates with respect to the base coordinate frame F_b. Again, we use the free variable ρ to account for the non-uniqueness of homogeneous coordinate expressions.

1.3. Solving for the Conversion Matrix

We have shown that eq. (1.5) captures the general essence of two dimensional projectivity. For our particular case of transformations between the camera image plane and the light plane, the relationship represented by eq. (1.8) is however more suitable.

Obviously, the conversion matrix T_{cb} in eq. (1.8) depends upon both the positions and the orientations of the camera and the light plane projector. The purpose of calibration is to find this matrix without recourse to actually measuring these positions and orientations. Note that because of the free variable ρ in equation (1.8), we can set t_{43} in T_{cb} equal to 1 and the equation still holds. Our calibration is to determine the eleven unknown coefficients in T_{cb}.

We carry out our calibration by finding the 2-D projectivity that exists between the camera image plane and the light plane. By the fundamental theorem presented in Section 2.2, we can find this projectivity -- in principle at least -- by using four coplanar but non-collinear points in the light plane and their corresponding points in the image plane. By choosing four illuminated object points as calibration points, assuming that their 3-D coordinates and their cooresponding image coordinates can be measured correctly, we should be able to solve for the matrix T_{cb}. We will now show how one might set up equations for this purpose. Rewriting equation (1.8) as

$$\rho \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} \cdot U$$

and eliminating the free variable ρ, we have

$$\begin{aligned} x &= T_1 \cdot U \;/\; T_4 \cdot U \\ y &= T_2 \cdot U \;/\; T_4 \cdot U \\ z &= T_3 \cdot U \;/\; T_4 \cdot U \end{aligned} \tag{1.9}$$

or equivalently,

$$\begin{aligned} T_1 \cdot U - x\, T_4 \cdot U &= 0 \\ T_2 \cdot U - y\, T_4 \cdot U &= 0 \\ T_3 \cdot U - z\, T_4 \cdot U &= 0 \end{aligned} \tag{1.10}$$

Thus each calibration point produces a set of three linear equations in terms of the eleven coefficients of T_{cb}. Four calibration points would therefore lead to a set of twelve equations for the eleven unknowns. This number is one more than what we need. Since we could pick any eleven equations out of the twelve and get a solution for T_{cb}, we could ostensibly get different T_{cb}'s depending on the choice of the eleven equations; this would evidently be in contradiction to the uniqueness implied by the fundamental theorem of projectivity. However, we should note that the fundamental theorem

requires the four calibration points to be coplanar. Therefore, the twelve 3-D coordinate values of the four points are not independent of one another, and, in fact, they obey the constraint of the co-plane equation:

$$\det [X_b^1 \; X_b^2 \; X_b^3 \; X_b^4 \;] = 0$$

That is, one of the twelve coordinates is determined by the other eleven values. Since the above co-plane constraint is in fact implicit in equation (1.8), one of the twelve equations generated by the four calibration points is redundant. As a consequence, we can use any eleven equations and arrive at the same unique solution for T_{cb}.

1.4. A Procedure for Automatic Calibration

In practice, using four object points at *a priori* known locations for computing the matrix T_{cb} is beset with difficulties for the following reasons:

1) There are always some errors associated with the measurement of locations of the four calibration points in the robot base frame. On account of such errors, their coplanarity can not be completely guaranteed.

2) It is unrealistic to assume that the camera can be modeled perfectly by a pin-hole. A pin-hole model is of questionable validity, especially when zoom lenses are used. When the pin-hole approximation breaks down, there may be no unique center of projection.

3) Because of the non-zero thickness of the illumination stripe and other digitization aspects of camera imaging, there will always be some non-zero error associated with the location of the image point corresponding to an object point.

Since for these reasons T_{cb} can not be found exactly, our best hope is to estimate it by minimizing some error criterion in an over-determined system of linear equations. In other words, given more than 4 calibration points, we want to find the T_{cb} which best fits those calibration points. The T_{cb} that best fits equation (1.10) can be found by solving a linear least square problem, similar to the solution of camera calibration in [*B&B*–82].*

At this point the reader probably has the impression that, in order to calibrate a structured light system, one must first install in the robot work area a set of object points at *a a priori* known locations. However, that is not the case in practice. Since the robot is programmed to move the structured-light unit in discrete steps, it is possible that the light planes emitted from any of the allowed positions of the scanner will not

* Alternatively, one can use the squared sum of the error distances of the calibration points in world coordinates for the error criterion.

illuminate the object points. One way to get around this difficulty is to use extended objects in the work area, the objects being of such a shape that at least four non-collinear points are illuminated by the light plane emitted from the projector. After the vision data is collected, the world coordinates of these object points are measured by moving to their locations the robot end-effector. Clearly, this method would only work if the mechanical calibration of the robot is accurate. This method is hard to automate. By automating a vision calibration procedure we mean the following: We want to place certain objects at strategic locations in the robot work area; then by simply having the robot record structured-light data on these objects at any time a calibration is desired, it should be possible for the associated computer to figure out the calibration parameters.

We will now propose a procedure that is easier to automate. A flat trapezoidal object is located permanently in the work area. The object is shaped in such a manner that no two edges of the top-surface are parallel to each other. The end coordinates of the top edges of these objects are known to the robot; therefore, one might say that the equations that define the lines corresponding to these edges are known. Consider one such line: Since a line can be defined as the intersection of two planes, it is described by the following two equations corresponding to the two planes.

$$\begin{cases} a_1 x + b_1 y + c_1 z = d_1 \\ a_2 x + b_2 y + c_2 z = d_2 \end{cases} \tag{1.11}$$

When the scanner projects a stripe intersecting this calibration line, it generates an illuminated point whose image coordinates are given by, say, U. While, of course, we can record the image coordinates of U, its world coordinates are unknown. In the procedure being described, we have no need for the world coordinates of the illuminated object point on the line. By substituting the right hand side of equations (1.9) for the x,y,z in (1.11), we have

$$\begin{cases} a_1\, T_1 \cdot U + b_1\, T_2 \cdot U + c_1\, T_3 \cdot U = d_1\, T_4 \cdot U \\ a_2\, T_1 \cdot U + b_2\, T_2 \cdot U + c_2\, T_3 \cdot U = d_2\, T_4 \cdot U \end{cases}$$

It shows that each calibration line is capable of producing a set of two equations in terms of the 11 coefficients of T_{cb}. Therefore, if we use at least six calibration lines, we will have a system of over-determined linear equation to estimate the conversion matrix. As we will describe below, it is not necessary to use six different calibration lines, although one could certainly do so.

In our current implementation of this procedure, we use only two distinct object edges, which are not parallel, for generating two calibration lines from any single viewpoint. By moving the structured light unit to different heights above the table, we can record the image coordinates of the same two edges for generating as many

equations as we like. We will now describe a step-by-step description of the procedure. First note though that mounted in the robot work area is a flat object whose top surface is not parallel to the light plane of the scanner. After this initial setup, each time a calibration is carried out by the robot, it automatically carries out the following steps:

1) The robot moves the scanner to an initial position. The coordinate frame of the robot tool center is recorded.

2) The scanner makes projects a light plane onto the calibration block. This generates on the block a segment of the light stripe, whose two end points must lie on the two calibration lines respectively.

3) From the digitized image, record the image coordinates of the illuminated points corresponding to the two calibration lines. Substitute these image coordinates for U in the two line equations; this gives us four linear equations.

4) To acquire more calibration lines, use the robot to move the scanner by (d_x,d_y,d_z) to a new position (Fig. 1.7). Now the line equations will become

$$\begin{cases} a_1\, T_1 \cdot U + b_1\, T_2 \cdot U + c_1\, T_3 \cdot U = \\ \quad (d_1 - a_1 d_x - b_1 d_y - c_1 d_z)\, T_4 \cdot U \\ a_2\, T_1 \cdot U + b_2\, T_2 \cdot U + c_2\, T_3 \cdot U = \\ \quad (d_2 - a_2 d_x - b_2 d_y - c_2 d_z)\, T_4 \cdot U \end{cases}$$

Go back to step 2).

5) Minimize certain error criterion to find the best estimate of T_{cb}.

Note that the estimated conversion matrix is with respect to the scanner at the initial position only. We will remove this constraint in the next section.

1.5. Linear and Rotational Scanning

1.5.1. Formulation

If the range map of a scene is desired, the scene must be scanned in some manner with the structured-light unit. Linear scanning and rotational scanning are the two schemes used in our lab. In linear scanning, the orientation of the scanner is fixed, only its position is changed equally between successive light stripe projections, as shown in Figure 1.8. In rotational scanning, the robot holds the scanner at a fixed position, but rotates the scanner in equal angular increments about the axis of the wrist joint. The movement of the scanner is specified by the position and orientation of its end effector on which the tool-center is defined. Let us define the coordinate frame of the tool-center as F_t such that the z axis of F_t aligns with the axis of the robot's wrist joint. For

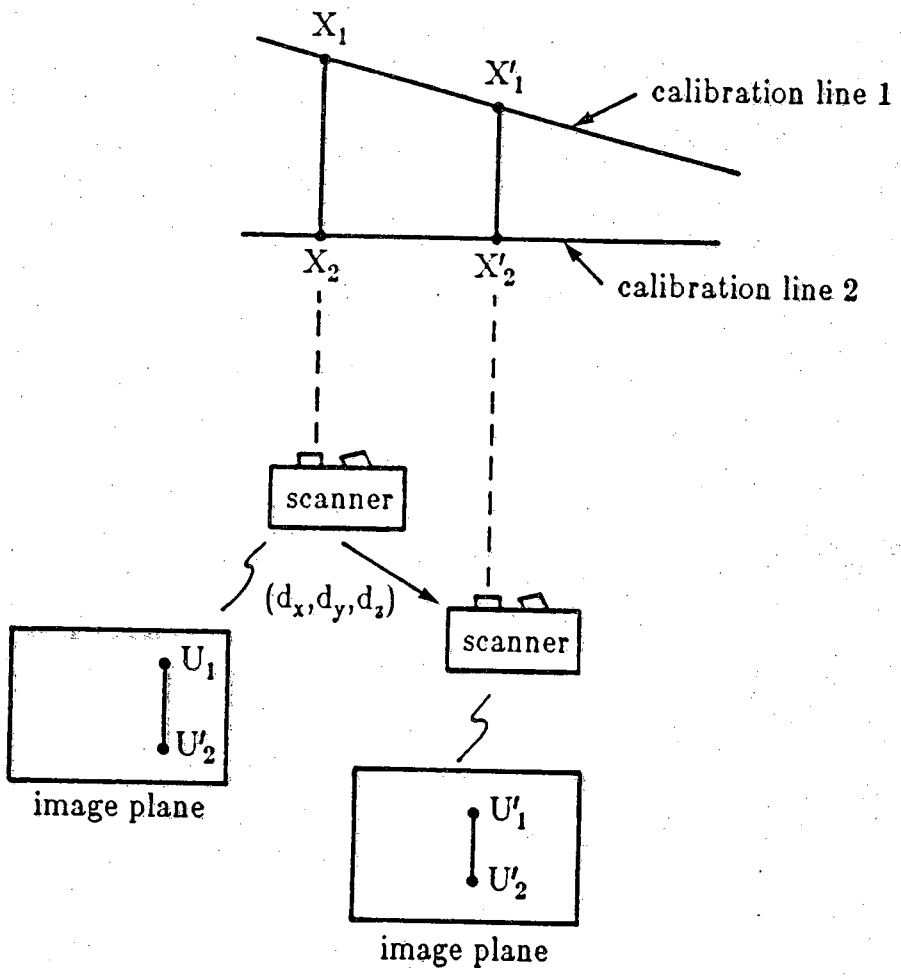


Figure 1.7. To acquire more calibration lines, the robot moves the scanner by (d_x, d_y, d_z) to a new position and makes projection.

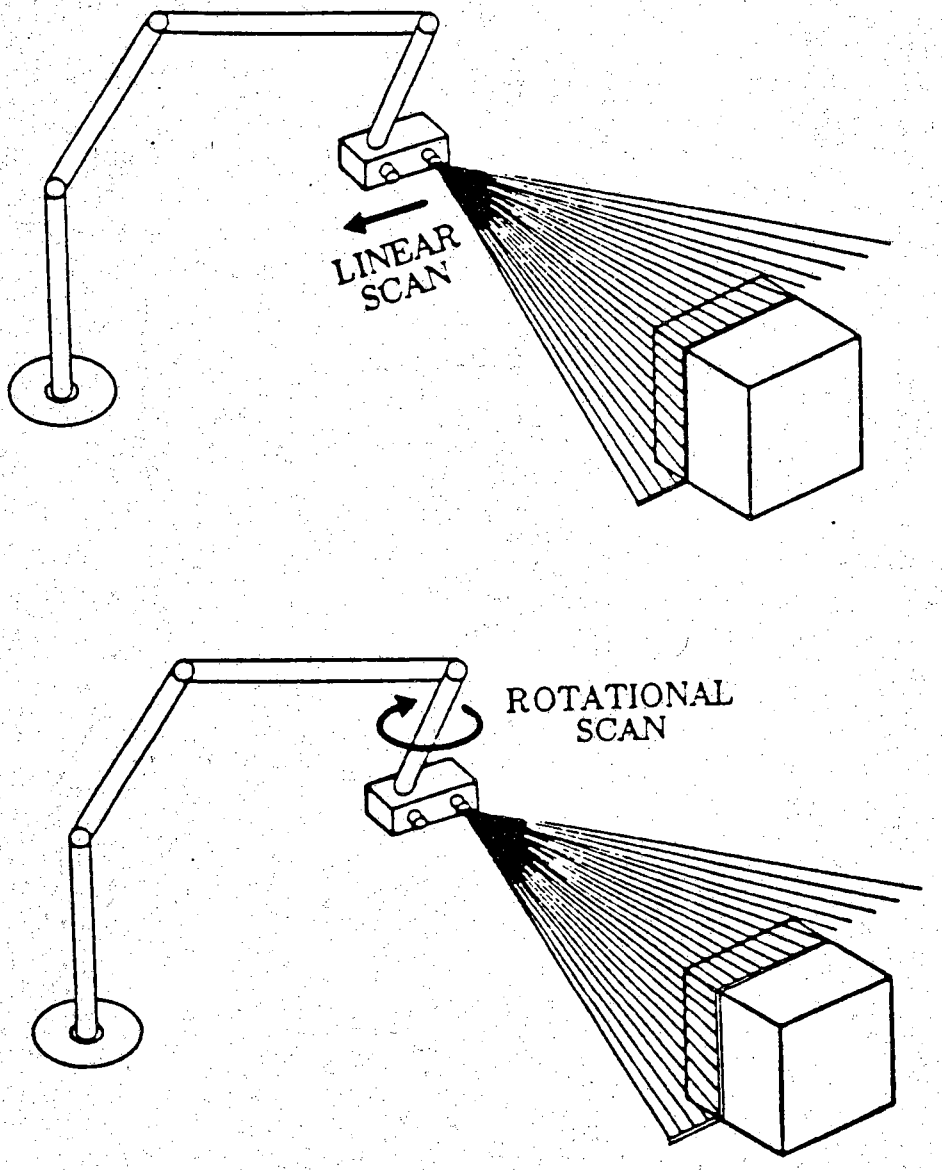


Figure 1.8. a) In linear scanning shown here, the orientation of the scanner is kept fixed while the scanner is translated along a line. b) In rotational scanning, while holding the scanner at a fixed position the robot rotates the scanner in equal angular increments about the axis of the wrist joint.

the case of linear scanning, we will express the translational movement from projection to projection by $\boldsymbol{D} = (d_x, d_y, d_z)$. This movement can be written as a translation transformation matrix:

$$H_d = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Similarly, for rotational scanning, if the angular increment between successive rotational positions of the scanner is δ, we can write down the following for a rotational transformation matrix

$$R_\delta = \begin{bmatrix} \cos\delta & -sin\delta & 0 & 0 \\ \sin\delta & \cos\delta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The conversion matrix T_{cb}, as obtained from the calibration process, is defined in the base coordinate frame F_b with the scanner at a specific position. Let the tool-center coordinate frame used for calibration be F_{t_c}. When scanning a scene, the position and orientation of the scanner will differ from those used during calibration. Therefore, during scanning, the tool-center coordinate frame, as represented by F_t, will be different from F_{t_c}. As a result, the T_{cb} matrix obtained from calibration can not be plugged directly in equation (1.8) for the purpose of computing the range map of a scene.

To get over this problem, we can convert the matrix T_{cb} to a matrix T_{ct}, which is defined in the tool-center coordinate frame F_{t_c}. This is done by

$$T_{ct} = (F_{t_c})^{-1} \cdot T_{cb} \tag{1.12}$$

This relation is depicted in Fig. 1.9. Thus T_{ct} converts an image point U into the corresponding object point in homogeneous coordinates with respect to frame F_t. Let F_{t_0} be the tool-center coordinate frame at the beginning of a scan and let j denote the j^{th} projection in a scan. In linear scanning, we have

$$F_{t_j} = F_{t_0} \cdot (H_d)^j$$

Therefore, we get

$$\begin{aligned} X_b &= F_{t_j} \cdot T_{ct} \cdot U \\ &= F_{t_0} \cdot (H_d)^j \cdot T_{ct} \cdot U \end{aligned} \tag{1.13}$$

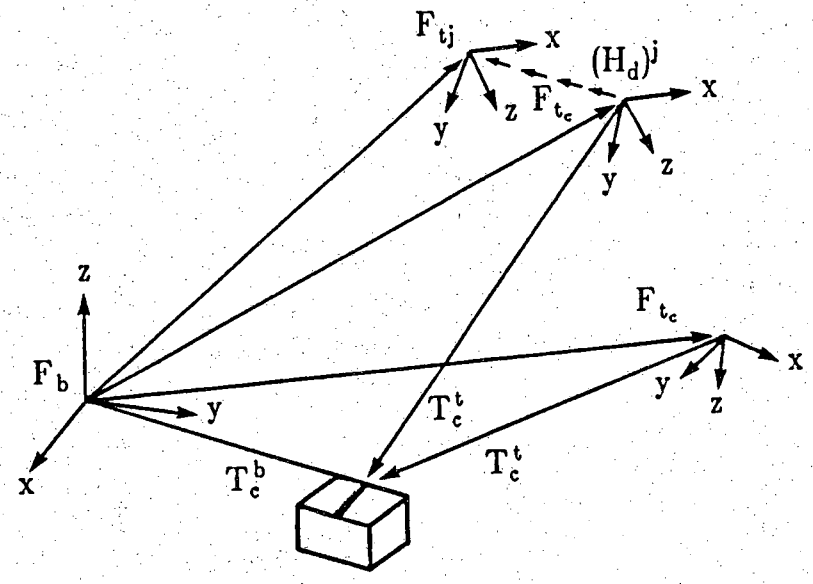


Figure 1.9. Relation among coordinate frames for linear scanning.

Similarly, for rotational scanning, we have

$$X_b = F_{t_0} \cdot (R_\delta)^j \cdot T_{ct} \cdot U \tag{1.14}$$

1.5.2. Analysis of Range Maps

Equations (1.13) and (1.14) provide us with formulas for computing the range map of a scene. For each light stripe projection during scanning, we record the column index v of the sampled illuminated object point in each row of the camera image. By applying equation (1.13) or (1.14), for each row indexed by u we have the 3-D coordinates $[x(u), y(u), z(u)]$ of the object point. These 3-D coordinates are then collected into a range map.

At this time, a few comments about the parametrization of the object surface are in order. Let row index of the scene range map be the same as the row index u of camera image plane; and let its column index be the index j associated with successive projections of the light stripes. Thus the range map can be expressed as $[x(u,j), y(u,j), z(u,j)]$. For example, if the camera image plane is of 480×512 resolution, and there are 80 projections in a scan, we will have a range map of size 480×80. Now consider the range map of a scene as the sampling of a visible surface, and assume that the surface is expressed as $f = [f_x, f_y, f_z]$. Its range map $f(u,j) = [f_x(u, j), f_y(u, j), f_z(u, j)]$ is the quantized parametrization of this visible surface. Note that the direction represented by the j index is directly related to the movement of the scanner from projection to projection. We want this "movement direction" to be perpendicular to the column direction of the camera image plane so that (u,j) will form an orthogonal parametrization of the surface. This can be important for later processing of the range map. For example, most 3-D edge detection operators are derived with the assumption of orthogonal parametrization.

1.6. Experimental Results and Conclusion

The structured light scanner used in our experiment consists of a Sony DC-37 CCD camera and an infrared projector. For conducting a calibration experiment, the calibration block is placed on the table and the scanner is moved to its initial position, which is about 20 inches above the table. A scan is then conducted along a line that is horizontal with respect to the work table; during the the calibration block is illuminated by three stripes. This process is repeated at four different heights, -- 20, 14, 8 and 2 inches -- above the work table, leading to range data on a total of 12 stripes. This data leads to 48 linear equations for the computation of the conversion matrix. The total time expended in the collection of calibration data is about a minute and the computer time for processing this information is about 3 seconds.

Although ultimately the evaluation of a calibration procedure must be carried out by determining the absolute accuracy of the system, for many purposes it is sufficient to compute the relative accuracy. By absolute accuracy we mean the precision with which the system locates a point with respect to the origin in the robot base coordinate system; and by relative accuracy we mean the precision with which the system makes a dimensional measurement of an object feature located in the robot area. In our experimental evaluation of the procedure described in this paper, we will only use relative accuracies. This is primarily owing to the fact that absolute accuracy tends to be a function of the accuracies of both the vision calibration and the robot arm calibration, meaning that a measurement of absolute accuracy may or may not tell us about the performance of a vision calibration technique.

After calibration, the relative accuracy of the procedure is evaluated by the computing the dimensions W and H of a block, like the one shown in Fig. 1.10. As expected, our experimental results show that the accuracies with which these two measurements can be made depend upon the distance of the block from the structured-light unit and the orientation of the block with respect to the scan direction. For the results reported here, the long axis of the block was kept approximately parallel to the scan direction. The results are shown in Table 1.

The reader might note that we have not taken into account any nonlinear lens distortions in our development of the calibration procedure. We have seen that these distortions become important for object points that are far away from the camera lens, usually farther than two feet. Lens nonlinearities may be taken into account by a variety of techniques presented by Tsai [*Ts*–86].

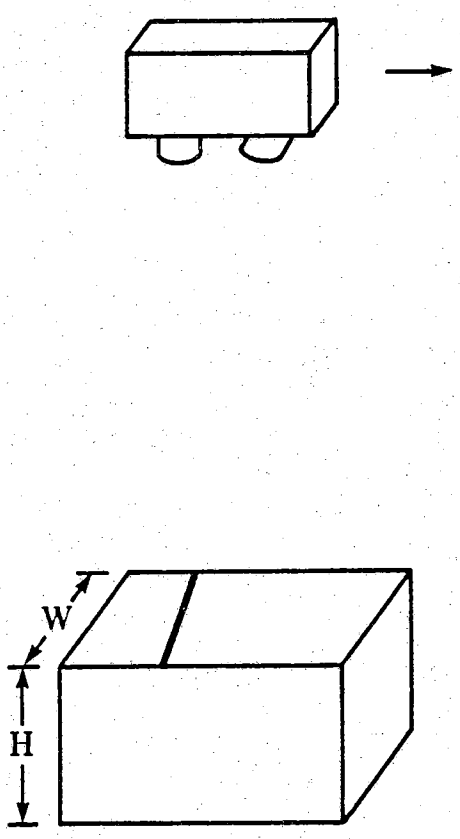


Figure 1.10. The width and the height of the block are computed from the range data in order to test the relative accuracy of calibration.

Table 1.1 Relative accuracy test results

d	8 *inch*	14 *inch*	20 *inch*
W_δ	--	<0.04 *inch*	<0.05 *inch*
H_δ	<0.03 *inch*	<0.14 *inch*	<0.30 *inch*

d: distance form the scanner to the block top surface
W_δ : difference between the computed width and the real width
H_δ : difference between the computed height and the real height

$W = 5.66\ inch$ $H = 6\ inch$

CHAPTER 2
EXTRACTION OF PRIMITIVE FEATURES FROM RANGE IMAGES

In this chapter we describe the processing steps that are invoked for extracting features from range images. As will be clear from the more precise definition in Chapter 3, a feature in our system is an analytically continuous surface, a straight or a curved edge, or a vertex. Each feature is characterized by a set of attribute-value of pairs. Feature extraction is basic to the recognition of objects and estimation of their poses; it is also basic to the "learning by showing" approach to the construction of object models, as discussed in Chapter 4.

There are three main processing steps described in this chapter. The first uses an adaptive window technique for accurate surface normal computation; our approach here is particularly accurate in the vicinity of boundaries between different surfaces. In the adaptive window technique, a window, used to compute the best local surface normal by fitting a plane to the local range points, is located adaptively depending upon a weighted planar-patch fitting error. Our second step describes how to segment a range image into smooth surface regions based on range and surface normal discontinuities. Finally, we present a scheme to classify segmented regions into three types of primitive surfaces, planar, cylindrical and conical, by fitting planes to the mappings of the surface normals onto Gaussian spheres.

2.1. Introduction

Our system uses the following three steps for feature extraction:

(a) Preprocessing: compute surface normals. The surface normal associated with a range point is computed from the equation of a best fit tangent plane to a small cluster of range points in the vicinity of the point in question.

(b) Segmentation: segment a range image into regions, each representing a smooth surface. Segmentation is accomplished by first extracting range and surface normal discontinuity points and then performing a connectivity analysis on the rest of the range map.

(c) Classification: classify segmented surfaces into the three primitive types. The classification of a surface is accomplished by first projecting the surface normals corresponding to a segmented surface onto a Gaussian sphere and then fitting a plane to the distribution of points on the sphere so obtained. The location of the plane from the center of the sphere is used for characterizing the object surface.

Since in our system both segmentation and surface classification rely on surface normals, accurate surface normal computation is crucial to the performance of feature extraction. By definition, the surface normal at a point is the unit vector normal to the tangent plane at the point. This surface normal can be computed from the cross product of two independent tangent vectors at the point; this straightforward approach has been used by Parvin and Medioni [*P&M*–86] and Besl and Jain [*B&J*–86]. The major drawback of this approach is its sensitivity to noise due to the differentiation involved in the computation. Another method for computing surface normals consists of fitting planar patches to small clusters of points within a window and using for the surface normal the normal to the planar patch; see, for example, [*M&B*–80], [*H&J*–87], [*Y&K*–89]. However, this approach, too, suffers from undesirable distortions, such as the "smoothing" of surface normals in the vicinity of boundaries between surfaces, because at such locations the window for computing the planar patch usually straddles the two surfaces. To overcome this shortcoming, we have modified this technique by using a notion first proposed by Nagao and Matsuyama [*N&M*–79] in the context of adaptive smoothing in 2-D image processing. They showed how the placement of a smoothing window near a region boundary should be made to depend upon extent of smoothness accomplished withing the window. In Section 2, we will apply this idea to surface normal computation.

A number of contributions have been reported on the subject of extracting primitive surfaces from a range map. Milgrim and Bjorklund [*M&B*–80], Bhanu[*Bh*–84], Boyter[*Bo*–84], Parvin and Medioni [*P&M*–86], and Yang and Kak [*Y&K*–86, *Y&K*–89] extract planar surfaces on the basis of the similarity of surface normals. Detection of cylindrical surfaces in range maps has been reported in [*A&B*–73], [*N&B*–77] and [*B&F*–81]. Faugeraus *et al.* [*F&et*–83] and Besl and Jain [*B&J*–88] show how analytically continuous surfaces of rather arbitrary shape can be extracted from range maps by fitting quadric and higher order surface functions to range data. Another method, similar in spirit to the approach presented in this chapter, is reported by Sethi and Jayaramamurthy [*S&J*–84]; in their scheme characteristic contours are used to distinguish between spheres, cylinders and cones (a characteristic contour being the locus of constant dot products between surface normals and any fixed vector). The characteristic contours of a sphere, a cylinder and a cone are a set of concentric circles, a set of parallel lines, and a set of intersecting lines, respectively. In the method of

Sethi and Jayaramamurthy, a decision tree is used to recognize the pattern of the characteristic contours in a Hough space.

The EGI (Extended Gaussian Image), which in the past has been used by [*Ik*–83], and [*Ho*–84] for object representation in 3-D vision, can also be used for classifying surfaces. It is rather well known that the EGI of planar, cylinderical and conical surfaces form special patterns on the Gaussian sphere; in the planar case, it is a small patch on the surface of the sphere, for a cylindrical surface, the points on the sphere lie on a great circle and, finally, for a conical surface, the points on the sphere lie on a minor circle. Printz [*Pr*–87] has shown how by analyzing the EGI pattern for approximate symmetries one can estimate the axis of a cylindrical or a conical surface. He expands the EGI distribution by expressing it as a sum of spherical harmonics and then estimates the symmetry axis by computing the enginvectors of a matrix whose elements are functions of the coefficients of the spherical harmonics representation. This method, though theoretically elegant, requires a large amount of computation and tends to be inaccurate because only finite terms of spherical harmonics can be used. [Printz did not show that the neglection of higher order terms in a spherical harmonic expansion of an EGI distribution did not degrade the accuracy of calculations.] Hebert and Ponce [*H&P*–82] propose using Hough transform to detect the three EGI patterns corresponding to the three primitive surfaces. Their Hough space has two parameters which are the two spherical angles of surface orientation (a surface orientation is defined for planar surfaces to be the direction of the normal to the planes, and for cylindrical and conical surfaces to be the direction of the axes of such surfaces). It is not clear how they can distinguish all the three EGI patterns on a 2-D Hough space since according to their formulation a cone needs three parameters, two for the axis and one for the angle of the cone. Moreover, the accuracy of the computed surface orientation will be limited by the resolution of the Hough space, let alone the overhead of constructing the Hough space.

While our aim is also to detect and classify the three EGI patterns corresponding to the three primitive surface types, we have avoided the use of Hough transforms. The method is based on the observation that planar and cylindrical surfaces are actually two special cases of a conical surface, especially from the standpoint of the EGI patterns they produce. There are two ways to look at this observation. A planar surface produces a small patch on the Gaussian sphere, whereas the EGI points corresponding to a general conical surface lie on a minor circle and, finally, for a cylindrical surface on a major circle. Therefore, if we fit a plane to the EGI points for all three cases, for the case of a planar object surface the fitted plane would have to be tangential to the Gaussian sphere and hence its perpendicular distance from the center of the Gaussian sphere would equal 1. On the other hand, the fitted plane for a conical object surface would be located at a distance between 0 and 1. And, finally, for a cylindrical object surface, the

fitted plane would must pass through the center of the Gaussian sphere. This observation allows us to construct a unified approach to the classification of primitive surface types. This unified approach will be presented in Section 4 of this chapter.

2.2. Computing Surface Normals via Adaptively Located Window

In general, a range map of a scene, generated by a structured light scanner, can be represented by the parametric form $\boldsymbol{p}_{i,j} = \boldsymbol{p}(i,j)$, where $\boldsymbol{p}$ stands for the $[x,y,z]$ coordinates of the object point illuminated by the *ith* stripe, the index j standing for sampling index along that stripe.

Since the surface normal at a surface point is a unit vector normal to the tangent plane at the point, the surface normal may be computed by

$$\boldsymbol{n} = \frac{\frac{\partial \boldsymbol{p}}{\partial i} \times \frac{\partial \boldsymbol{p}}{\partial j}}{\left| \frac{\partial \boldsymbol{p}}{\partial i} \times \frac{\partial \boldsymbol{p}}{\partial j} \right|}.$$

Finite difference operators can be used to compute the two partial derivatives $\frac{\partial \boldsymbol{p}}{\partial i}$ and $\frac{\partial \boldsymbol{p}}{\partial j}$. The major difficulty with this method is that it is very sensitive to noise due to the differentiation operation. In order to overcome the noise problem, the range data usually has to be smoothed extensively [*B&J*–86], which in many cases tends to distort the range data, especially in the vicinity of edges.

An alternative method, which is less noise-sensitive, is to estimate the tangent plane at each surface point by fitting a plane equation to the surface points in a neighborhood of the point in question. The equation of the fitting plane can be expressed as

$$f(x,y,z) = ax + by + cz - d = 0.$$

with $a^2+b^2+c^2 = 1$. The normal vector to the plane, denoted by $\boldsymbol{n}$ is $[a,b,c]$. The neighborhood over which the plane is fitted by this method is usually an $N \times N$ window centered at $\boldsymbol{p}_{i,j}$. We will denote the window by $W_{i,j}$. The best fit plane is found by minimizing the fitting error:

$$\begin{aligned} \varepsilon(W_{i,j}) &= \sum_{k,l \in W_{i,j}} f(x_{k,l}, y_{k,l}, z_{k,l})^2 \\ &= \sum_{k,l \in W_{i,j}} (\boldsymbol{n}\, \boldsymbol{p}_{k,l}^t - d)^2 \end{aligned} \tag{2.1}$$

with the constraint $a^2+b^2+c^2 = 1$, where $\boldsymbol{p}_{k,l} = [x_{k,l}, y_{k,l}, z_{k,l}]$. This constrained minimization can be accomplished by taking the partial derivatives of the following

Lagrange function:

$$L = \sum_{k,l \in W_{i,j}} (\boldsymbol{n}\, \boldsymbol{p}_{k,l}^t - d)^2 + \lambda (1 - \boldsymbol{n}\, \boldsymbol{n}^t)$$

with respect to $\boldsymbol{n}$, d and λ and setting them to zero. Expanding out the expression for L, we obtain

$$L = \sum_{k,l \in W_{i,j}} [\boldsymbol{n}\, \boldsymbol{p}_{k,l}^t\, \boldsymbol{p}_{k,l}\, \boldsymbol{n}^t - 2\, d\, \boldsymbol{p}_{k,l}\, \boldsymbol{n}^t] + N^2\, d^2 + \lambda (1 - \boldsymbol{n}\, \boldsymbol{n}^t)$$

$$= \boldsymbol{n} \Big(\sum_{k,l \in W_{i,j}} \boldsymbol{p}_{k,l}^t\, \boldsymbol{p}_{k,l} \Big) \boldsymbol{n}^t - 2\, d \Big(\sum_{k,l \in W_{i,j}} \boldsymbol{p}_{k,l} \Big) \boldsymbol{n}^t + N^2\, d^2 + \lambda (1 - \boldsymbol{n}\, \boldsymbol{n}^t)$$

Define a new matrix and a new vector as

$$\boldsymbol{Q} = \sum_{k,l \in W_{i,j}} \boldsymbol{p}_{k,l}^t\, \boldsymbol{p}_{k,l}$$

$$\boldsymbol{q} = \sum_{k,l \in W_{i,j}} \boldsymbol{p}_{k,l}$$

L can therefore be written as

$$L = \boldsymbol{n}\, \boldsymbol{Q}\, \boldsymbol{n}^t - 2\, d\, \boldsymbol{q}\, \boldsymbol{n}^t + N^2\, d^2 + \lambda (1 - \boldsymbol{n}\, \boldsymbol{n}^t)$$

We now set the following partial derivative to zero:

$$\frac{\partial L}{\partial \boldsymbol{n}} = 2\, \boldsymbol{n}\, \boldsymbol{Q} - 2\, d\, \boldsymbol{q} - 2\, \boldsymbol{n}\, \lambda = \boldsymbol{0} \tag{2.2}$$

$$\frac{\partial L}{\partial d} = -2\, \boldsymbol{n}\, \boldsymbol{q}^t + 2 N^2\, d = 0 \tag{2.3}$$

$$\frac{\partial L}{\partial \lambda} = 1 - \boldsymbol{n}\, \boldsymbol{n}^t = 0 \tag{2.4}$$

From (2.3), we have

$$d = \frac{1}{N^2} \boldsymbol{n}\, \boldsymbol{q}^t \tag{2.5}$$

Substituting this result in (2.2), we obtain

$$2\, \boldsymbol{n}\, \boldsymbol{Q} - \frac{2}{N^2} (\boldsymbol{n}\, \boldsymbol{q}^t)\, \boldsymbol{q} - 2\, \boldsymbol{n}\, \lambda = 0$$

or, equivalently,

$$\boldsymbol{n}\, \boldsymbol{R} = \boldsymbol{n}\, \lambda \tag{2.6}$$

where

$$\boldsymbol{R} = \boldsymbol{Q} - \frac{1}{N^2} \boldsymbol{q}^t\, \boldsymbol{q} = \sum_{k,l \in W_{i,j}} \boldsymbol{p}_{k,l}^t\, \boldsymbol{p}_{k,l} - \frac{1}{N^2} \sum_{k,l \in W_{i,j}} \boldsymbol{p}_{k,l}^t \sum_{k,l \in W_{i,j}} \boldsymbol{p}_{k,l}$$

Equation (2.6) implies that the normal vector $\boldsymbol{n}$ to the best fit plane is an eigenvector of the matrix $\boldsymbol{R}$, and λ is the corresponding eigenvalue. Since $\boldsymbol{R}$ is a 3×3 matrix and, therefore, in general, will possess 3 eigenvectors, the question arises is which of these corresponds to the desired $\boldsymbol{n}^t$? It is easy to verify that the plane fitting error associated with a particular eigenvector $\hat{\boldsymbol{n}}$ is equal to the corresponding eigenvalue by substituting d expressed by equation (2.5) in equation (2.1). Since we wish to minimize the fit error $\varepsilon(W_{i,j})$, the eigenvector to choose corresponds to the smallest eigenvalue.

Let $[a',b',c']$ be the eigenvector so computed (assume it has been normalized too). Since the estimated surface normal can take the value either $[a',b',c']^t$ or $[-a',-b',-c']^t$, in order to resolve the ambiguity, we assign

$$\begin{cases} \boldsymbol{n}_{i,j} = [a',b',c'] & \text{if } \boldsymbol{v}\cdot[a',b',c'] \le 0 \\ \boldsymbol{n}_{i,j} = [-a',-b',-c'] & \textit{else} \end{cases}$$

where $\boldsymbol{v}$ is the viewing direction of the range sensor.

The performance of this plane fitting method depends on the window size $N\times N$ to some extent. If the size is too small, say 3×3, the computed normal will be susceptible to noise and quantization error associated with range values. On the other hand, large sized windows cause smoothing distortions in the computation of surface normals, especially when the windows straddle boundaries between two smooth surfaces; the regions where such distortions occur are of size proportionate to the size of windows used for computation. Fig. 2.1 illustrates three possible placements for a 5×5 window, with the placements W_a and W_c located entirely within the smooth surfaces of the object, while the placement W_b is straddling the boundary between two surfaces. If window placement is such that it is located entirely within a smooth surface of the object, a plane fitting method should provide satisfactory surface normals, even when the surface is somewhat curved. However, if the window lies across a jump or crease boundary, the computed surface normal will be distorted because the fitted range data actually come from two different surfaces. The effect of this distortion is that normals will not be wholly discontinuous in traveling over an edge but will smoothly change.

This smoothing distortion can be virtually eliminated by adaptive placement of windows in the vicinity of edges. The key idea is to adaptively position the fitting window around the range point in question such that the window encloses the point without crossing any edges. To illustrate our point, consider the computation of surface normal at a surface point $\boldsymbol{p}_{i,j}$ lying on the vicinity of a crease edge by using a 5×5 fitting window as shown in Fig. 2.2. The crease edge divides the range pixels into two regions R_1 and R_2 representing two different surfaces, the range pixel in question, $\boldsymbol{p}_{i,j}$, being on surface R_1. With a window size of 5×5, there will be 25 windows,

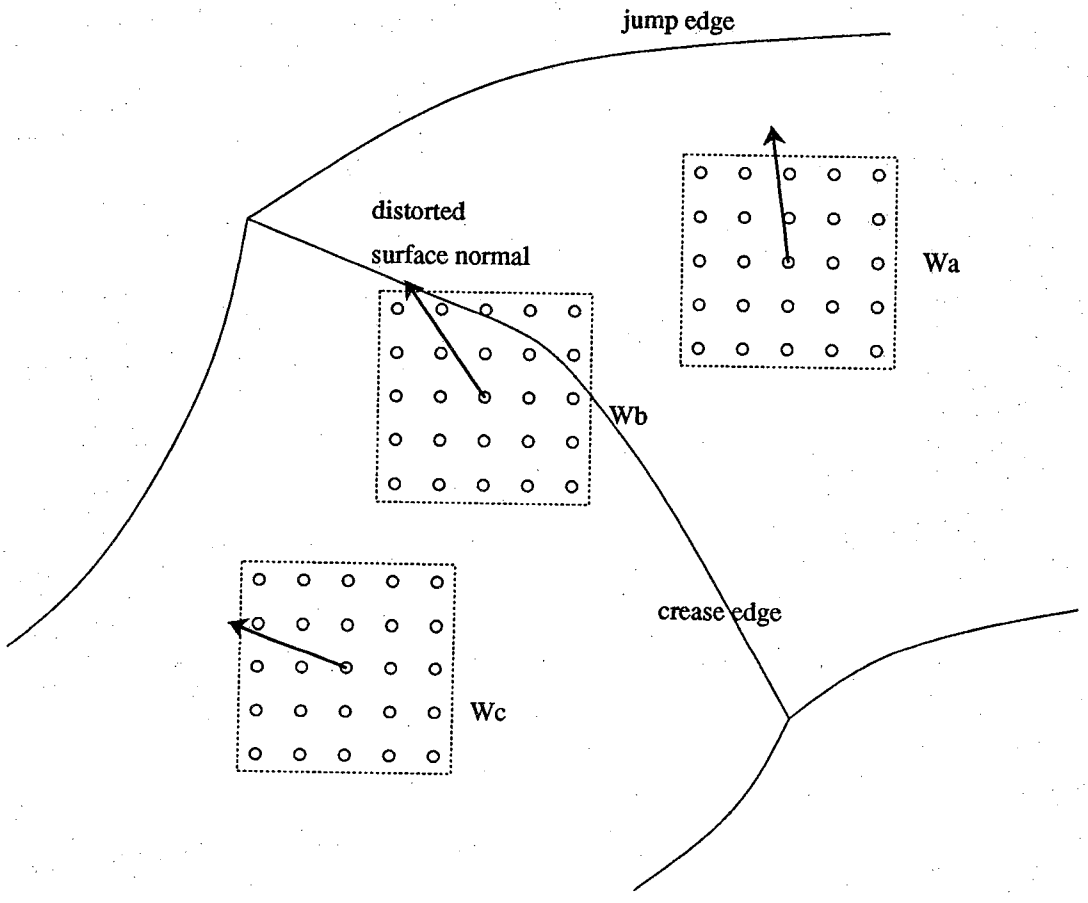


Figure 2.1. Fitting window over a crease edge causes the "smoothing" distortion.

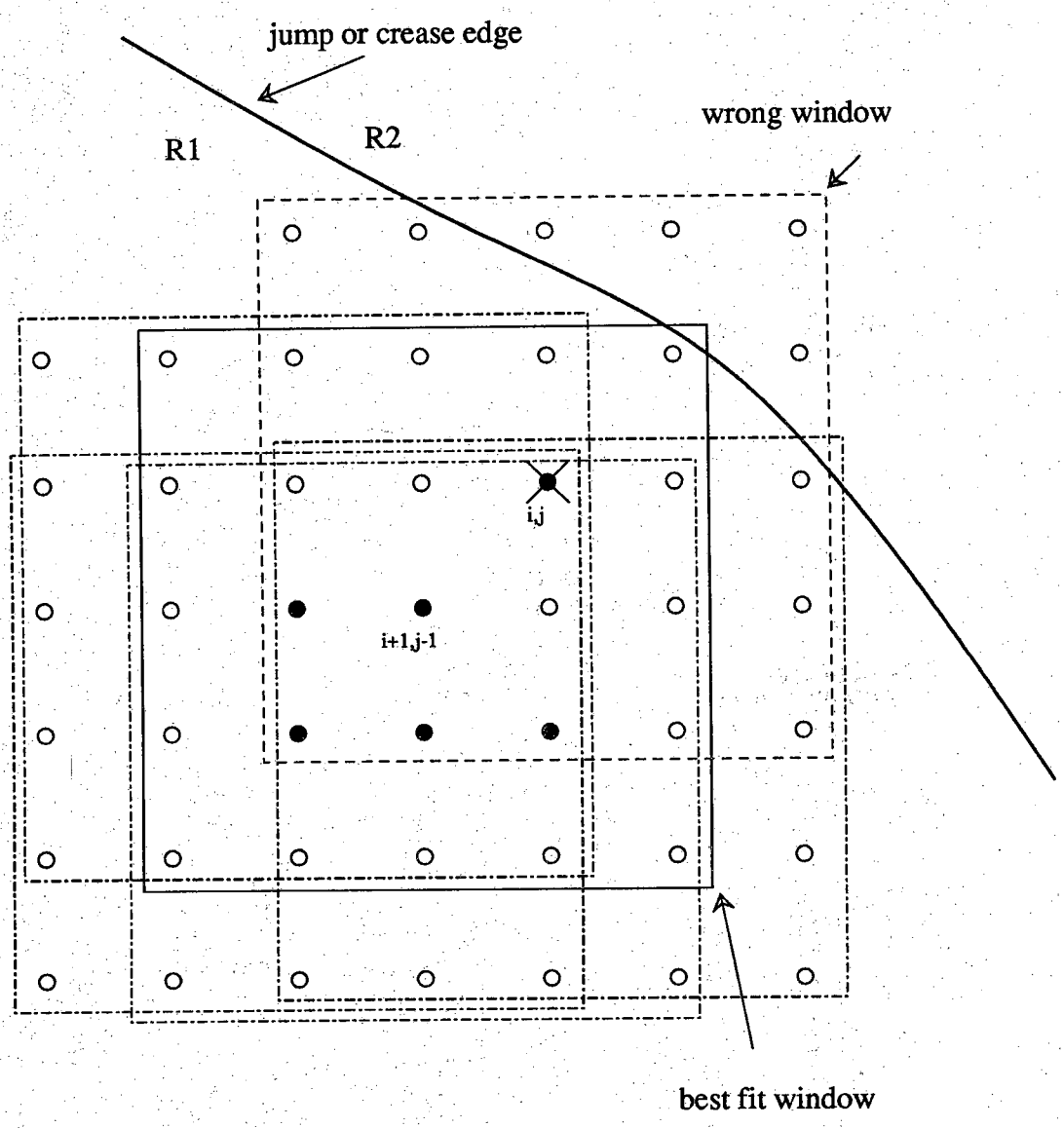


Figure 2.2. The range pixel $p_{i,j}$ has $N \times N$ candidate windows; the adaptive one should lie entirely on region R_1 and be as close as possible to $p_{i,j}$.

$$\{W_{k,l} \mid i-2 \leq k \leq i+2,\ j-2 \leq l \leq j+2\},$$

that can enclose the range pixel $\boldsymbol{p}_{i,j}$. In computing the surface normal of $\boldsymbol{p}_{i,j}$, we want to choose a window of its neighboring range pixels that best estimate the tangent plane at $\boldsymbol{p}_{i,j}$. Obviously, any window that includes the edge, such as $W_{i,j}$, will give a distorted result because it would contain range pixels from the other surface R_2. To obtain a meaningful result, we must insist that the fitting window lie entirely on the same side of the crease edge with $\boldsymbol{p}_{i,j}$. Thus, in this example there are only five windows can be considered for the computation of the surface normal at the range pixel (i,j), these windows being $W_{i+1,j-2}$, $W_{i+1,j-1}$, $W_{i+2,j-2}$, $W_{i+2,j-1}$, $W_{i+2,j}$. To select from amongst these windows, we bear in the mind the requirement that the center of the selected window should be as close to range pixel (i,j) as possible. On this consideration, of the five windows, only $W_{i+1,j-1}$ qualifies. This strategy for window selection raises the question of how to put the two criteria into a mathematical form such that we can evaluate each candidate window accordingly, the two criteria being that the window not cross any edges and that the center of the window be as close as possible to the range pixel in question.

To develop this mathematical form for evaluating potential windows for surface normal computation, we note that the planar patch fitting error is larger at those windows which cross an edge compared to those windows which do not; this is in keeping with the observation made by [*B&F*–81]. Hence, the fitting error associated with each window can provide a indication of whether or not the window has run over an edge. This observation translates into the following mathematical form for evaluating a window $W_{k,l}$ for consideration at range pixel (i,j)

$$w(i,j,k,l)\ \varepsilon(W_{k,l}) \tag{2.7}$$

where $w(i,j,k,l)$ is a weighting function inversely proportional to the distance between the parameter space location (i,j), which corresponds to the range pixel $\boldsymbol{p}_{i,j}$, and the location (k,l) corresponding to the center of window $W_{k,l}$, and $\varepsilon(W_{k,l})$ is the fitting error over window $W_{k,l}$. We choose that window that minimizes (2.7) among all such candidate windows, meaning all those windows that include the range pixel $\boldsymbol{p}(i,j)$. There are several ways to define the distance between two pixels in a 2-D array (see [*R&K*–82]). In our implementation, we have chosen to use the city block distance, so the weighting function w is defined as

$$w(i,j,k,l) = \frac{1}{c+|i-k|+|j-l|} \qquad c>1.$$

Here the constant c is chosen such that the distance weighting will be the dominant factor in the expression (2.7) if all the windows are within the same continuous object surface. On the other hand, if the planar patch fitting error is large, we want the second

term in expression (2.7) to dominate. The algorithm in a pseudo language is sketched below.

```
compute_surface_normal () {
    for each i,j in the image array
        compute ε(W_{i,j}) of the best fit plane
        tn_{i,j} = normal of the best fit plane

    for each i,j in the image array
        among all k,l ∈ W_{i,j}
            find k̂,l̂ that minimizes (w(i,j,k,l) ε(W_{k,l}))
        n_{i,j} = tn_{k̂,l̂}
}
```

We can see that the first part of the algorithm is basically the conventional approach except that the fitting errors, $\varepsilon(W)$, are recorded at all the range pixels, meaning for all the windows. The second part of the algorithm then looks at a certain neighborhood of each range pixel, and assigns that surface normal to it which corresponds to the minimum of the evaluation function. Note that the only overhead involved in this additional work, corresponding to the second part of the algorithm where the best window is found, lies in the determination of a minimum from amongst $N \times N$ values; this can be done by making $\log_2(N \times N)$ comparisons.

We will now show on experimental data the improvements made possible with the adaptive placement of windows. Fig. 2.3 shows a scene containing a wooden object illuminated with 85 stripes. The resolution of the camera used in this experiment was 480×512, resulting in a 480×85 array of offset data, with each offset value between 0 and 511. Since the resolution in row direction is much higher than that in the column direction, and since it really does not make much sense to work with unequal resolutions in the two orthogonal directions, we compress every three rows of the array into one and then compute a 160×85 range image from the resulting data; the details on how the offset information is converted into a range map are described in [*C&K*−87] and can be found in this report in Chapter 1. Note that although the array size is 'rectangular', being made of 160 rows and only 85 columns, the spatial resolution is approximately the same in both directions, the reason being that the camera looks at 'longer' extent space in the direction that corresponds to columns.

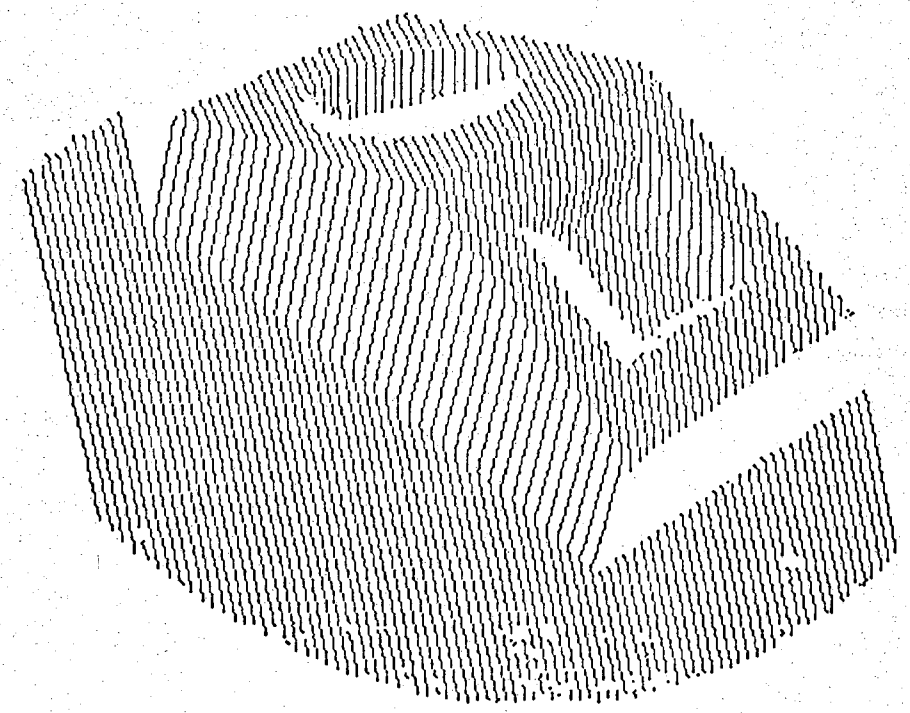


Figure 2.3. A stripe image of an object taken with 85 scans by a structured light range sensor.

To compare surface normal computation results with and without the adaptive technique, we show a needle diagram in Fig. 2.4 obtained with 5×5 windows using the traditional approach (the non-adaptive approach). On the other hand, when the adaptive approach is used, we get the needle diagram in Fig. 2.5. It is easy to see that the smoothing distortions have been virtually eliminated with the adaptive method.

2.3. Range Image Segmentation

In this section, we will present a region growing approach to the segmentation of range images. Our algorithm is very similar to the one presented earlier by Snyder and Bilbro [*S&B* –85].*

A segmentation algorithm must segment a range image into regions that correspond to smooth object surfaces. By definition, smooth surfaces are bounded by crease edges where surface normal discontinuities occur. In a range image, smooth object surfaces manifest themselves as regions of range pixels bounded by crease edges and jump edges. Suppose we are able to detect a range pixel corresponding to one of these surfaces in a given range image. To grow outwards from this pixel, we must provide termination conditions that indicate the occurrence of jump or crease edges. A jump edge occurs if there is a range discontinuity between two adjacent pixels. We can detect a jump edge by using the following predicate:

$$|\boldsymbol{p}_{i,j} - \boldsymbol{p}_{k,l}| > \mathit{range\_threshold},$$

where $\boldsymbol{p}_{k,l}$ is adjacent to $\boldsymbol{p}_{i,j}$. This range threshold should be a function of scanning resolution (spacing between two adjacent scans) and should be chosen properly so that two range points on a slanted surface would not be mistaken for a jump edge. We have chosen the range threshold to be 3 times the scanning resolution.

A crease edge is where surface normals suddenly change directions in the range image. In order to detect the occurrence of surface normal discontinuities, we must rely on the change rate of surface normal from one pixel to the next. This change rate may be regarded as a form of normal curvature measurement [*O* –66]. The following predicate captures the rate of change of surface normals and can be used as a stopping criterion for the detection of crease edges.

* We apologize to the reader for misusing the phrase 'region growing.' What we have done can be simply implemented by first detecting edges in a range image, followed by connective analysis of the non-edge pixels. The connectivity analysis could be implemented efficiently in the parameter space (i,j) directly. We felt compelled to use the phrase 'region growing' in connection with our algorithm since our algorithm is very similar to the so-called region growing method described in [*S&B* –85].

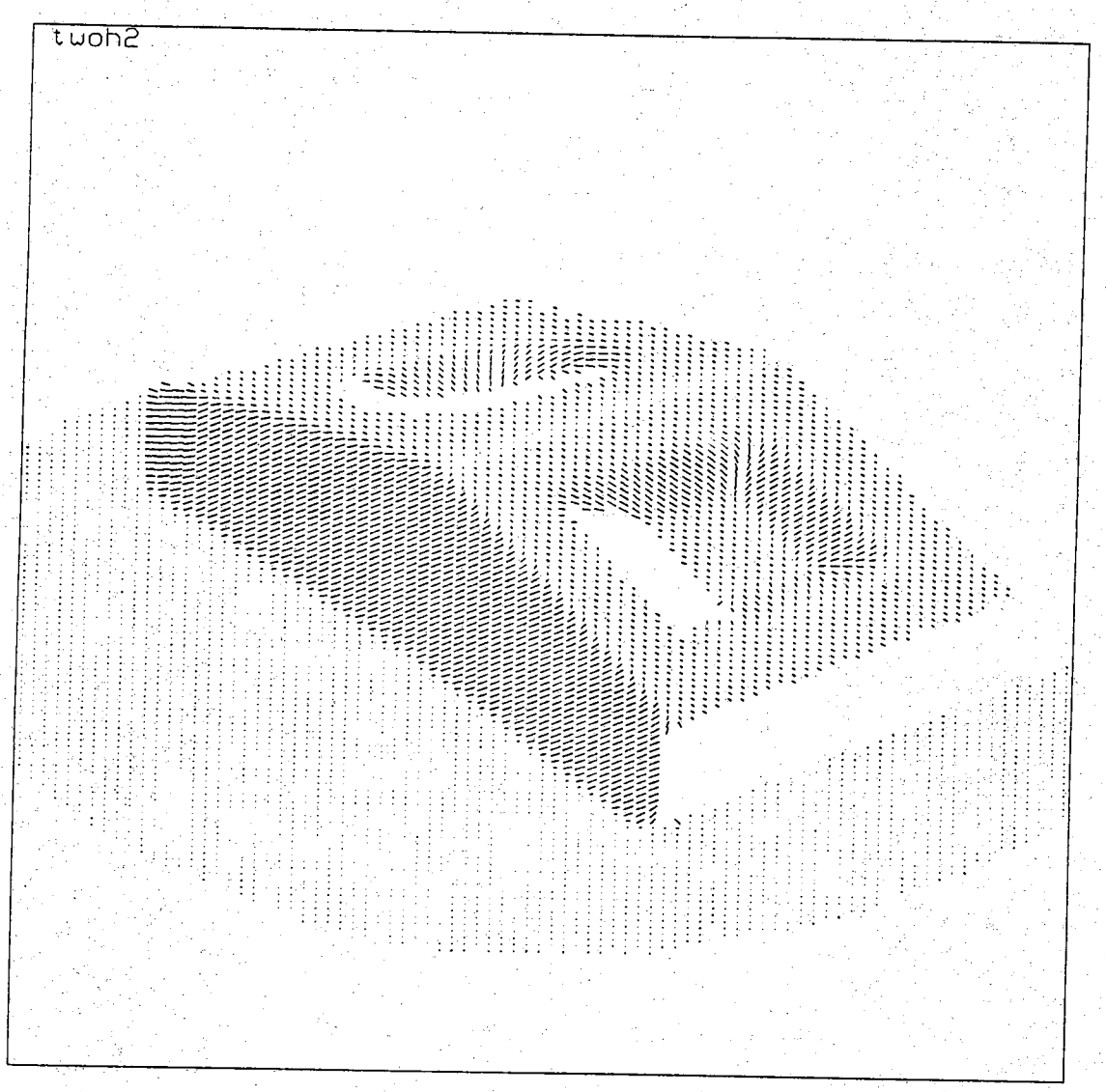


Figure 2.4. A needle image of the object, each needle represents the surface normal at a range pixel computed by fitting a plane equation to a 5×5 window of neighborhood. Notice the smoothing distortions in the vicinity of edges.

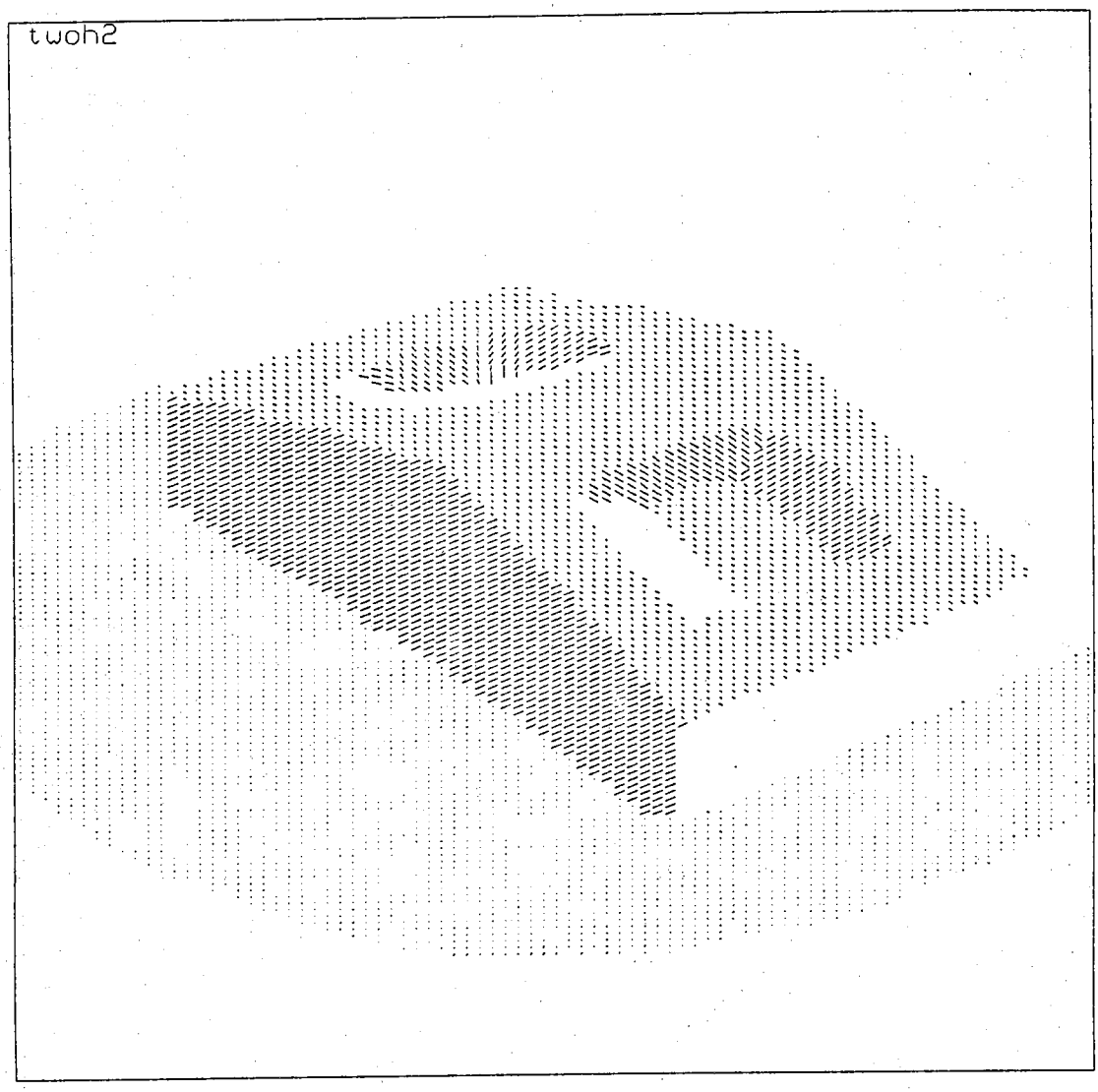


Figure 2.5. A needle image of the object obtained by using the adaptive window method. Notice the smoothing distortions have been virtually eliminated.

$$\frac{\cos^{-1}(\boldsymbol{n}_{i,j} \cdot \boldsymbol{n}_{k,l})}{|\boldsymbol{p}_{i,j} - \boldsymbol{p}_{k,l}|} > curvature\_threshold.$$

Theoretically speaking, as the points $\boldsymbol{p}_{i,j}$ and $\boldsymbol{p}_{k,l}$ get closer the predicate approaches the normal curvature at one of the two points. [Note that the normal curvature corresponds to the curvature associated with the curve obtained by "cutting" an object surface with a plane. In the curvature measure shown here, the cutting plane is that plane which passes through the points $\boldsymbol{p}_{i,j}$ and $\boldsymbol{p}_{k,l}$ on the object surface. The third constraint on the cutting plane is that the surface normal at either $\boldsymbol{p}_{i,j}$ or $\boldsymbol{p}_{k,l}$ also be contained within the plane.] This curvature threshold used should exceed the maximum of the maximal curvature of any of the smooth surfaces expected to be encountered in the scene. For example, assume that the most curved surface in the scene corresponds to a sphere of radius 3", then the curvature threshold should be no less than 1/3, the normal curvature of a sphere of radius r being $1/r$.

Range image segmentation proceeds by growing a region in all directions by recursively merging neighboring range pixels depending upon the two predicates defined above. Segmentation results on the range image shown in the previous section are illustrated in Fig. 2.6. In general, the performance of this growing procedure is highly dependent on the quality of range data and surface normals because only local information is used. However, since our adaptive window technique is capable of producing clear, nondistorted surface normals, this simple region growing method preforms quite well for the types of scenes we have worked with in connection with this research.

We can also extract vertices and edges from a segmented image. To do so, we trace the boundary of each segmented region clockwise and monitor the labels of its neighboring regions. A change of the labels signals the presence of a vertex, and any two adjacent vertices define an edge. The type (crease, occluding, or occluded) of each edge can be determined by comparing the range values on the two sides of the edge. Only a crease boundary is regarded as a real edge. The adjacency relationships between surfaces are also recorded during this boundary tracking.

2.4. Classification of Surfaces

Given a processed range image, we want to classify each segmented region into one of three types of surfaces, namely planar, cylindrical or conical; of course, if a region does not fit the criteria for any of these three categories, we would like the region to be classified as unknown. Theoretically, by fitting a quadric function to the range data of a segmented region and examining the coefficient of the best fit quadric function, we can determine which type of surface the region is. However, as noted by

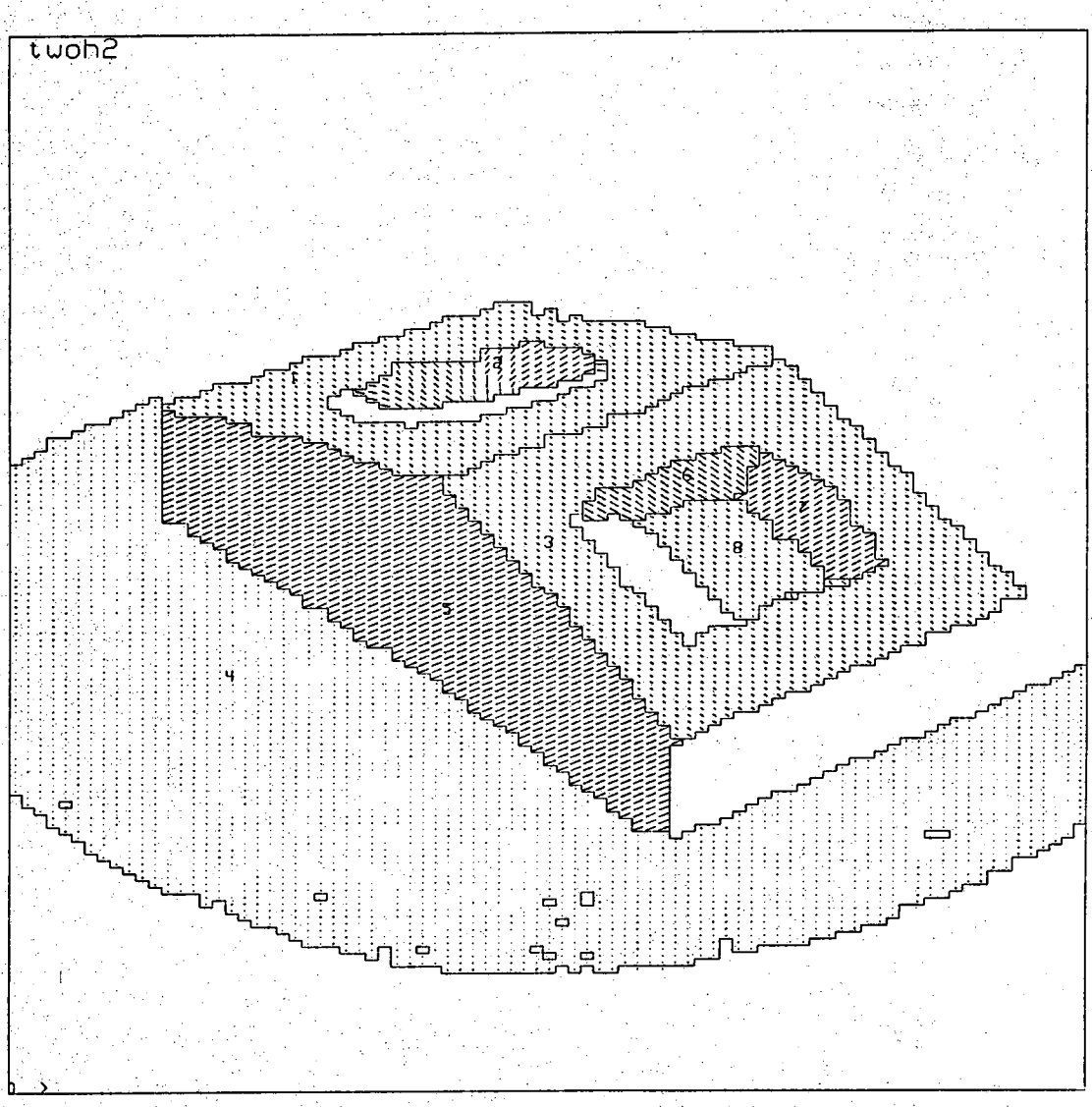


Figure 2.6. The segmentation result from the needle image in Fig. 2.5.

Hoffman and Jain [*H&J*–87], in practice, those coefficients are very sensitive to noise. In this section we present an efficient method to classify segmented regions into the three surface types based on the characteristics of their extended Gaussian images.

The extended Gaussian image of a surface is obtained by mapping the surface normal at every point of the surface onto a Gaussian sphere [*Ik*–83] [*Ho*–84]. For a planar surface, the EGI is a small patch whose orientation on the Gaussian sphere corresponds to the normal to the plane (see Fig. 2.7-a). For a cylindrical surface, its EGI is a great circle whose axis is parallel to the axis of the cylinder (see Fig. 2.8-a), by the axis of the circle is meant a unit vector perpendicular to the plane of the circle and passing through its center. And finally, for a conical surface, its EGI is a minor circle of radius less than one, the axis of the minor circle being again parallel to the axis of the conical surface, as shown in Fig. 2.9-a. In addition, as illustrated in Fig. 2.10, the distance from the center of the sphere to the plane containing the circle in each case is given by

$$d=\sin(\theta)$$

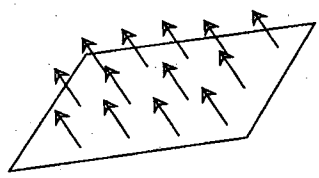
whereas the radius of the circle is given by

$$r=\cos(\theta),$$

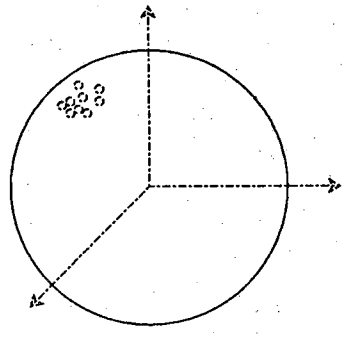
θ being the cone angle. It is useful to think of planar and cylindrical surfaces as two special cases of a conical surface. We can easily visualize a cone becoming a plane by letting the cone angle θ approach $\pi/2$. To see how a cone deforms into a cylinder of radius r, we first fix an orthogonal section of the cone where the radius equals r and then let θ approach 0 by pulling the tip of the cone away from the section. The effect of deforming a cone into a cylinder or a plane can can also be seen in its EGI. As the cone angle θ approaches 0, the circle on the Gaussian sphere becomes a great circle, and as θ approaches $\pi/2$ the circle shrinks to a point. In other words, we can regard the EGI's of planar and cylindrical surfaces as the two extreme cases of the EGI of a cone.

Based on the above observation, we can classify a region of range data by using the procedure sketched below. First, we test whether the EGI of the region fits a "generic" circle. If it doesn't, the region should be classified as unknown surface; otherwise, we compute d, the distance of the plane containing the circle from the center of the sphere, and classify the region into one of the three types according to the distance. Of course, in practice, for any of the three surface types, points on the sphere may not be available everywhere on the circle. Therefore, an assumption in our work is that a sufficiently large arc segment of the circle is available in order that we may find the plane that contains the circle.

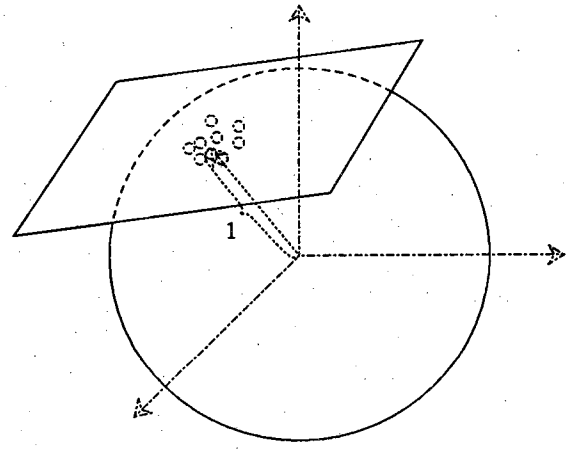
Clearly, our procedure for surface classification requires that we determine whether or not the points on the EGI fall on a circular arc and, if they do, then we must



A planar surface
with its surface normals

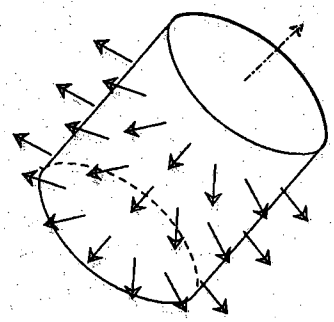


The surface normals are mapped onto
a small patch on the Gaussian sphere

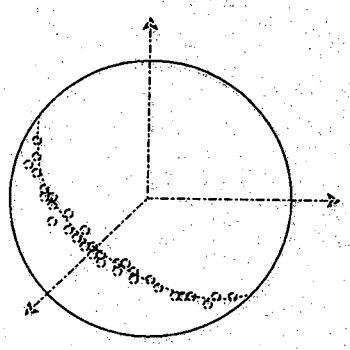


Fitting a plane to the points
on the Gaussian sphere

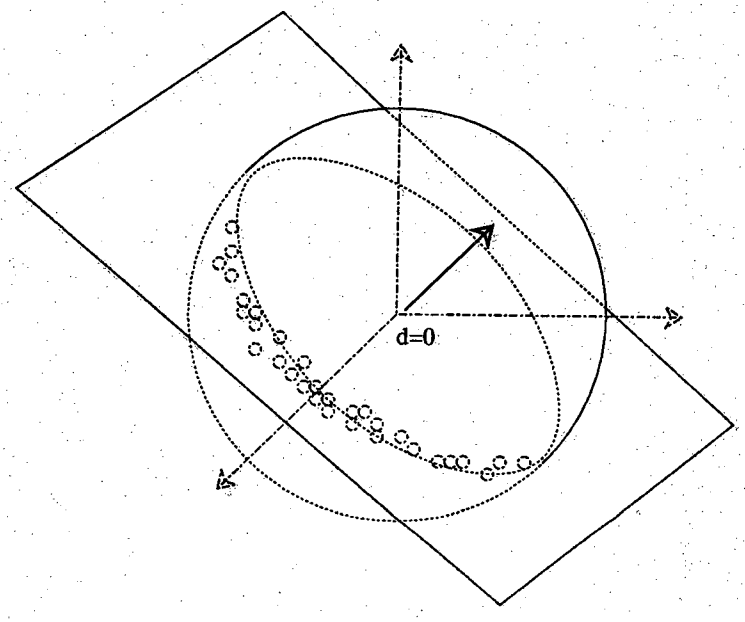
Figure 2.7. The EGIs of a planar surface and the fitting plane on the Gaussian sphere.



A cylindrical surface
with its surface normals

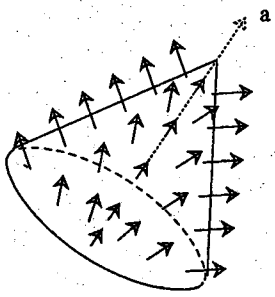


The surface normals are mapped onto
a great circle on the Gaussian sphere

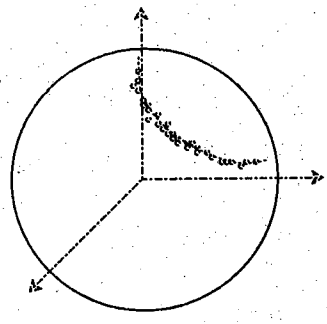


Fitting a plane to the points
on the Gaussian sphere

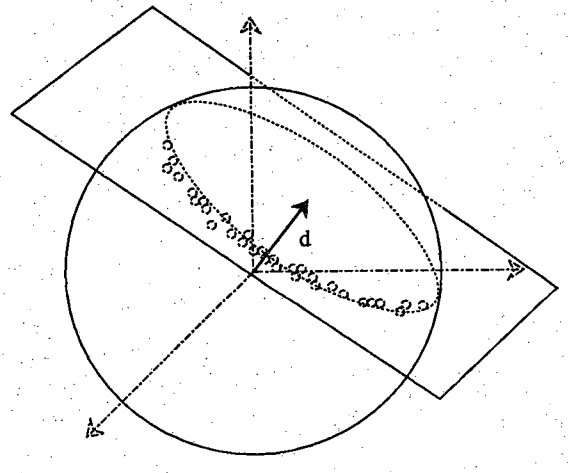
Figure 2.8. The EGIs of a cylindrical surface and the fitting plane on the Gaussian sphere.



A cone with its surface normals



the surface normals are mapped onto
a small circle on the Gaussian sphere



Fitting a plane to the points
on the Gaussian sphere; here, 0<d<1

Figure 2.9. The EGIs of a conical surface and the fitting plane on the Gaussian sphere.

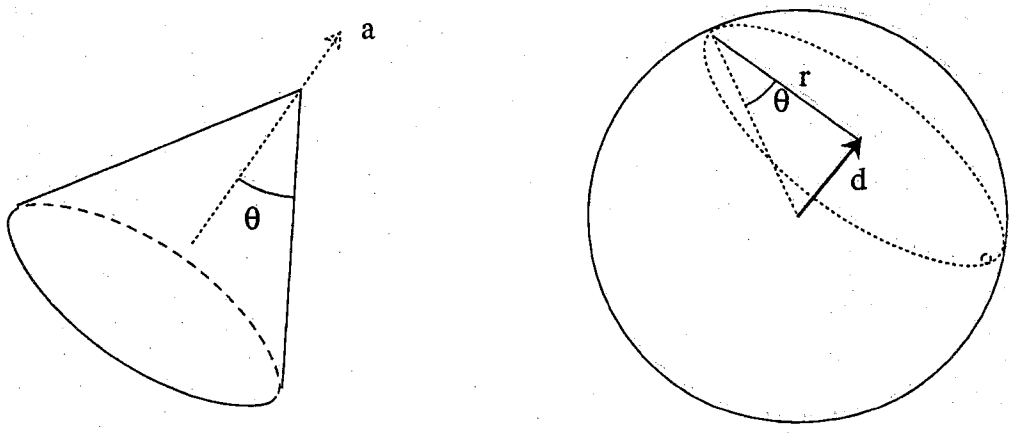


Figure 2.10. The radius of the circle, which is the EGI of a conical surface, is equal to cos(θ); the distance from the sphere center to the center of the circle is given by sin(θ), where θ is the cone angle.

compute the distance of the plane containing that circle from the center of the sphere. Observe that any point $\boldsymbol{n}$ of a perfect circle on a Gaussian sphere with axis $\boldsymbol{a}$ and perpendicular distance d must satisfy the following equation :

$$\boldsymbol{n} \cdot \boldsymbol{a} - d = 0. \tag{2.8}$$

This turns out to be a plane equation with normal $\boldsymbol{a}$ and distance from the center equal to d. Thus the circle can be interpreted as the intersection of the plane with the Gaussian sphere. Therefore, if we fit a plane to the EGI points for all three cases, for the case of a planar object surface the fitted plane would have to be tangential to the Gaussian sphere and hence its perpendicular distance from the center of the Gaussian sphere would equal 1, as shown in Fig. 2.7-b. On the other hand, the fitted plane for a cylindrical object surface must pass through the center of the Gaussian sphere (see Fig 2.8-b). And, finally, for a conical object surface, the fitted plane would be located at a distance between 0 and 1 (see Fig. 2.9-b). To accomplish this plane fitting, we minimize the following error function

$$\varepsilon = \sum_{i,j \in region} (\boldsymbol{n}_{i,j} \cdot \boldsymbol{a} - d)^2$$

At this point, the reader would note that this minimization problem is exactly the same as the plane fitting problem discussed in Section 2. Again, $\boldsymbol{a}$ and d can be found by solving the eigenvector of a matrix $\boldsymbol{R}$ which is a function of $\boldsymbol{n}_{i,j}$ as defined in Section 2. If the EGI of a surface is indeed an arc on a circle, then the estimated $\boldsymbol{a}$ and d should yield very small fitting error ε. On the other hand, if the EGI of the region is not part of a circle, meaning that the surface is none of the three primitive types, we can expect a significantly larger fitting error ε. Thus by properly thresholding the perpendicular distance d we can determine whether or not the region is a generic cone. The complete algorithm in pseudo language is given below:

---

classify_region (R_k) {
  find $\hat{\boldsymbol{a}}$, $\hat{d}$ which minimizes $\varepsilon = \sum_{i,j \in R_k} (\boldsymbol{n}_{i,j} \cdot \boldsymbol{a} - d)^2$
  compute $\hat{\varepsilon}$
  if $\frac{\hat{\varepsilon}}{//R_k//}$ > unknown_threshold
    return (unknow-type)
  else if $\hat{d}$> plane_threshold
    normal =$\hat{\boldsymbol{a}}$
    return (plane)
  else if $\hat{d}$ < cylinder_threshold

```
            cylinder_axis = â
            return (cylinder)
        else
            cone_axis = â
            cone_angle = sin^{-1}(d̂)
            return (cone)
    }
```

Note that the conditions we use for surface categorization are necessary but not sufficient. For example, a major circle on an EGI will be produced not only by a cylinder but also by a ruled surface.

CHAPTER 3
3D-POLY: A ROBOT VISION SYSTEM FOR RECOGNIZING 3-D OBJECTS IN LOW-ORDER POLYNOMIAL TIME

3.1. Introduction

The task at hand is to locate and identify instances of known model objects in a range image. The objects are assumed to be rigid. This task can, in general, be accomplished by matching features extracted from the image (scene features) with those describing models (model features). We will assume that the features are geometric in nature and can be characterized by shape, position and orientation; such features may include surfaces, edges, points, etc.

What are the desirable traits of a good system for object recognition and location? Clearly, it should be robust enough to work in multi-object scenes where the objects may be occluding one another. The complexity of the system should exhibit a low-order polynomial dependence, on, say, the number of features involved. The system should also be easy to train, meaning that it should be amenable to "learning by showing." In our context, that means that if we showed the system an object in all its external entirety then the system should automatically extract the relevant information that it would subsequently use for recognition and location determination.

A system with these traits is in operation in the Robot Vision Lab at Purdue. The system, called 3D-POLY, has been tested on scenes consisting of mutually occluding objects; Fig. 3.1 is an example of such a scene which is made of two different types of objects shown in Fig. 3.2. Evidently, these objects, whose surfaces are of planar and conical types in convex and concave juxtapositions, do not exemplify the most difficult that can be found in the industrial world; however, their recognition in occluded environments represents a significant advance in our opinion. The various frames in Fig. 3.3 illustrate a successful determination of location and identification of one of the objects whose surfaces are sufficiently visible in the scene of Fig. 3.1, and the manipulation of this object by the robot. For the objects in the heap, the models were automatically generated by the system — we call this learning by showing — by placing each object in a computer controlled scanner; each object was shown in many different

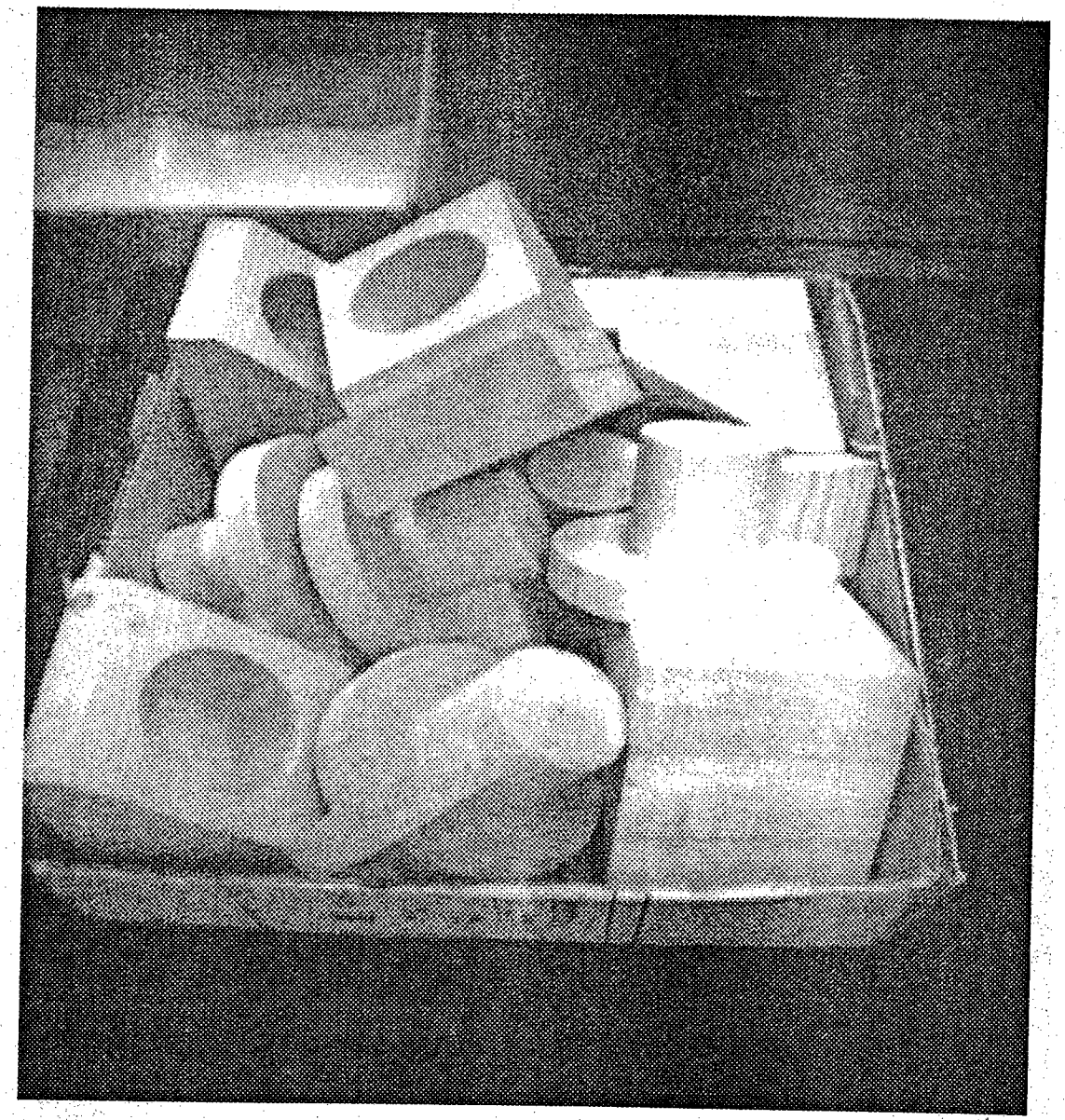


Figure 3.1. A scene of a pile of objects.

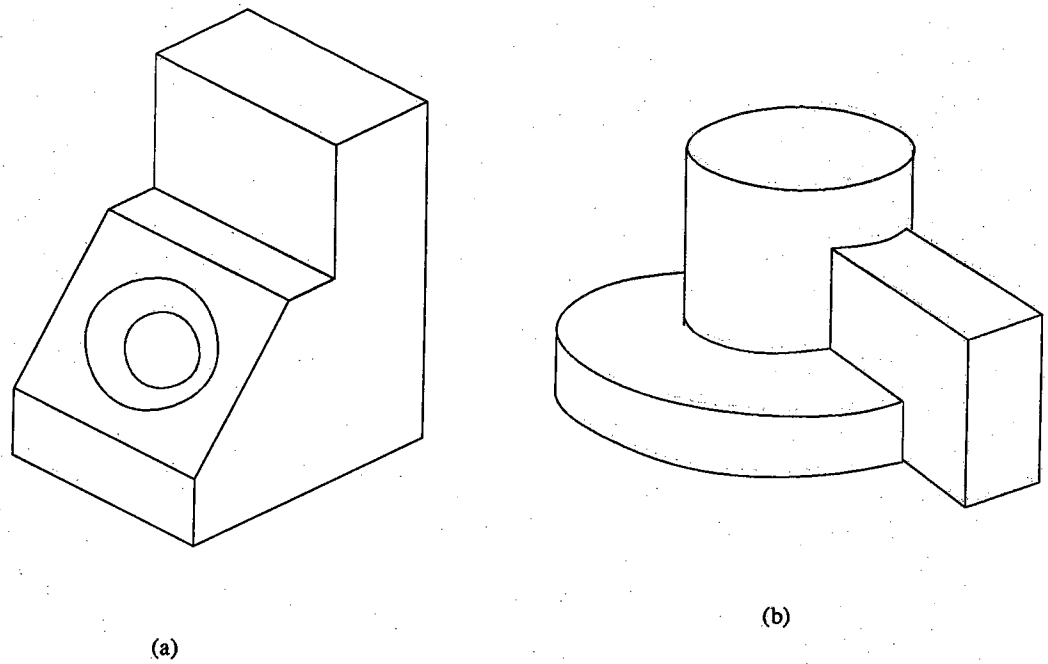


Figure 3.2. The two objects that make up the pile in the scene shown in Fig 3.1.

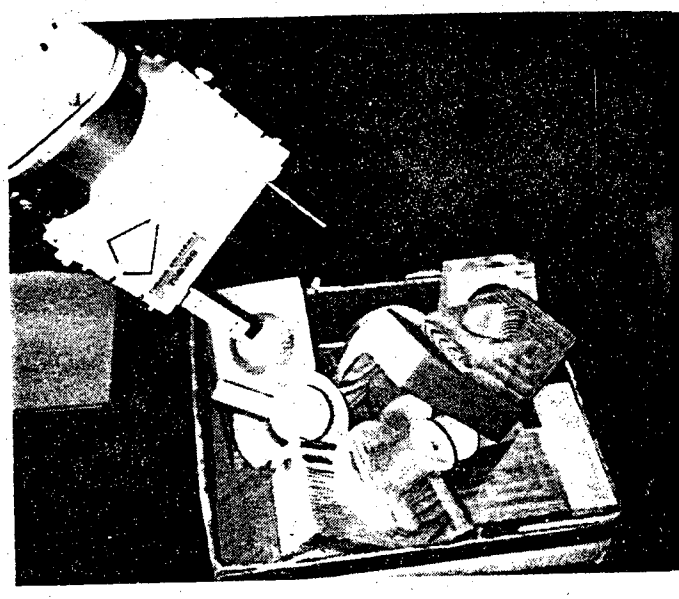
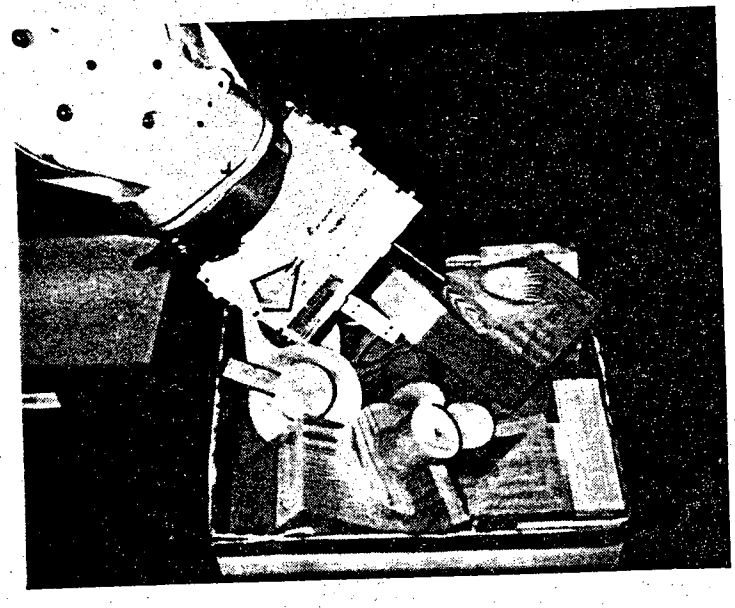





Figure 3.3. A sequence of frames shows the robot successfully picking up a recognized object.

configurations so that the system could build an integrated "whole view" model of the object. The details of the "learning" system will be presented in Chapter 4.

The aim of this chapter is not to describe the entire system that results in the type of sensor-guided manipulation shown in Fig. 3.3, but only those aspects that deal with strategies for hypothesis generation and the manner in which the model information is organized to facilitate verification. In the next section, we will state more formally the problem of determining the identity and location of an object. Against a background of this problem statement, in Section 3 we will discuss the literature related to our research. Finally, the rest of this chapter will then present the main theme of this article.

3.2. Problem Statement

Before we formally state the problem of object recognition, we would like to define the more important symbols used in our presentation.

- S or S_i: A scene feature will be denoted by S. When more than one feature is under discussion, the i^{th} feature will be denoted by S_i.
- M or M_j will denote model features.
- O_s will denote a scene object. For the purpose of explaining our hypothesis generation strategies and verification procedures, we will assume that the scene consist of a single object. However, as will be pointed out in Section 7, the entire method is easily generalizable to multi-object scenes. (Of course, its success in multi-object scenes would depend upon the extent to which an object is visible.)
- O_m will denote a candidate model object. Selection of a model object from the library of objects available to the system is part of hypothesis generation.
- n will denote the number of features extracted from the scene object.
- m will denote the number of features in the model object.
- Tr will represent the location (orientation and position) of the scene object; it is in fact a transformation consisting of a rotation $\boldsymbol{R}$ and a translation $\boldsymbol{t}$.† The transformation takes the object from the model coordinate system to its actual location in the world coordinate system (see Fig 3.4). Actually, the model coordinate system is the same as the world coordinate system, the only difference being that in the former all model objects are supposed to reside at the origin in some standard orientation. When we say a model object resides at the origin, we mean that some reference point on the object is coincident with the origin.

† We will also use Tr to denote the set of all possible solutions for the location of an object given the currently known constraints. The type of usage should be clear from the context.

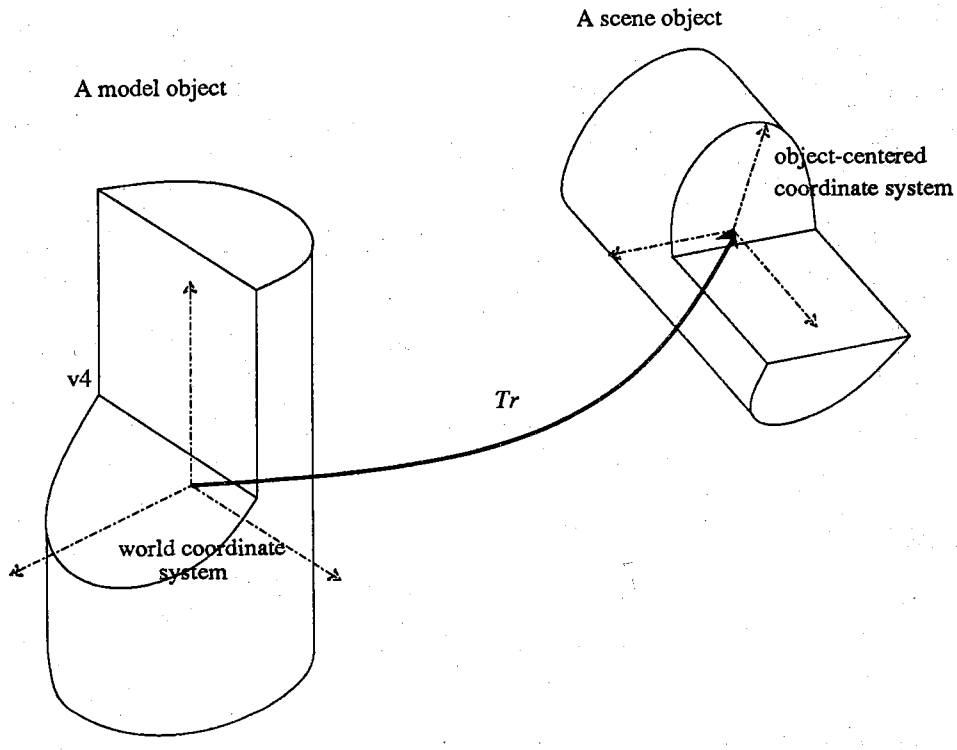


Figure 3.4. The relation between the object-centered coordinate system for the model data, the world coordinate system used for the scene data and the transformation that specifies the pose of object.

- c will denote a one-to-one mapping function between the scene features and the model features. In this paper, we will assume a one-to-one mapping between the scene features and the model features. Although there can certainly be situations where a one-to-one mapping may not be appropriate — for example, when scene edges are broken, one may have to map more than one scene edge to the same model edge — our segmentation algorithms do manage to produce features at almost the same level of connectivity as the model features, at least for the types of scenes depicted in Figs. 3.1. For example, Fig. 3.5 shows the edge and surface features of the objects in a heap.

With this notation, a scene object may be represented by

$$O_s = \{S_i \mid i = 1, \ldots, n\}$$

and a model object by

$$O_m = \{M_j \mid j = 1, \ldots, m\}$$

where the ordering of the features is unimportant at this time; we will have more to say about the subject of ordering later, since it plays an important role in the interpretation of multi-object scenes.

Since all objects will be assumed to be rigid, if object O_s is an instance of model O_m placed at location Tr in 3-D space, then every observed scene feature of O_s must be an instance of some model feature of O_m transformed by the rotation and translation specified by Tr. One may now state our problem more formally by saying that the aim of our system is to find a model object, O_m, determine its position and orientation transform Tr and establish a correspondence $c : O_s \rightarrow O_m$ such that

$$S_i <=> Tr \cdot M_{c(i)} \tag{3.1}$$

for all $S_i \in O_s$. Here <=> symbolizes a match between the two features S_i and $M_{c(i)}$. The criteria under which two features may be considered to match will, in general, depend on factors such as the types of features used, capabilities of the sensor, occlusion, noise, etc., and will be addressed later. Equation (3.1) provides a framework for discussing the problem of feature matching in general terms.

The problem of recognition and localization of a scene object may be decomposed into the following three subproblems:

(a) Select a candidate model O_m from the library; this generates the object identification hypothesis.

(b) Determine (estimate) the location Tr; this is the location hypothesis.

(c) Establish a correspondence c between O_s and O_m that satisfies equation (3.1). Such a correspondence constitutes verification of the hypothesis.

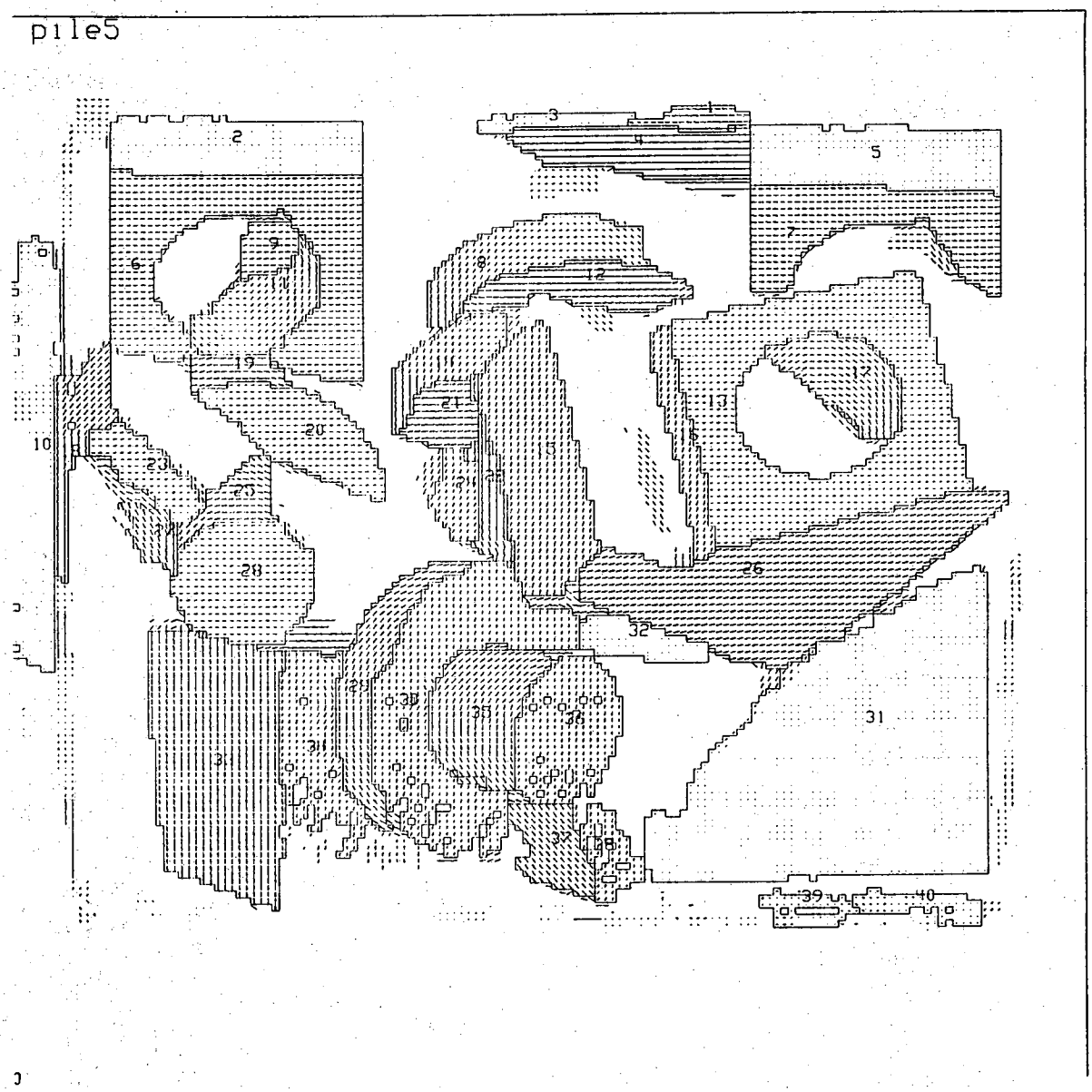


Figure 3.5. This processed range map of the scene of Fig. 3.1 shows the segmented regions. The needle orientations show the computed local orientations of the surfaces.

The size of the solution space for the first subproblem is proportional to the number of model objects in the model library. For the second subproblem, one has to contend with the six degrees of freedom associated with the transformation Tr, three of these being associated with the position vector $\boldsymbol{t}$ and the other three with the orientation matrix $\boldsymbol{R}$. If we use $\mathbb{R}^{Tr}$ to denote the solution space associated with the second subproblem, it is given by

$$\mathbb{R}^{Tr} = \mathbb{R}^3 \times [0, 2\pi] \times [0, 2\pi] \times [0, \pi]$$

where $\mathbb{R}$ stands for the real line, and, therfore, $\mathbb{R}^3$ stands for the solution space corresponding to all possible solutions for the translational vector $\boldsymbol{t}$. The solution space associated with the third subproblem is obviously of size m^n, since, in general, one must allow for all possible ways in which the n scene features can be matched with the m model features. Therefore, we can write the following expression for the solution space associated with the problem represented by equation (3.1):

$$\#\_of\_models \times \mathbb{R}^{Tr} \times m^n$$

We do not wish to give the reader an impression that strategies for range image interpretation must be founded on the problem decomposition shown here, only that it is a preferred approach for us (and a few other researchers in other laboratories). Approaches that do not conform to our problem decomposition do exist and some of these have been extensively investigated. In our literature survey in Section 3, we have alluded to these competing approaches.

In general, any solution to the problem of matching a scene object O_s with a candidate model O_m can, for the purposes of analyzing complexity and efficiency related issues, be conceived of as a tree search, as for example shown in Fig. 3.6. A traversal through the search tree may be referred to as a recognition process, each arc in the traversal representing an attempt at scoring a match between a scene feature and a model feature. Each node in a traversal may be considered to represent the current state of the recognition process, where the current state is specified by (c^*, Tr^*) with c^* being a partial correspondence list established so far and Tr^* representing the partial solution to the determination of object location. Clearly, the initial state of tree search, represented by the root, should be $(c^* = \varnothing,\ Tr^* = \mathbb{R}^{Tr})$. Through the tree search, we incremently construct the correspondence c^* on the one hand and contract the solution space of Tr^* on the other. A model object is considered to be an acceptable interpretation of a scene object if the traversal reaches one of the terminal nodes. Since reaching a terminal node merely means that all of the n scene features have been matched, it clearly does not constitute a sufficient condition for object recognition; all that can be said is that the model is an acceptable interpretation. If no traversal in the tree search is able to arrive at a terminal node then the candidate model object must be rejected. Note

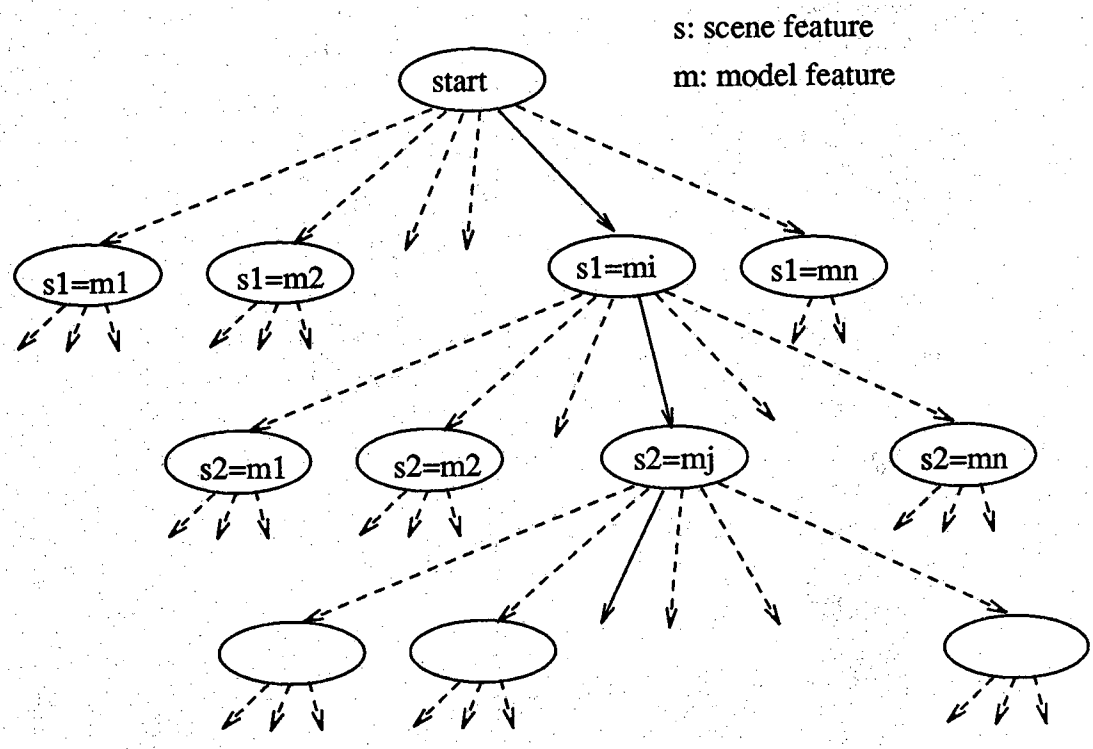


Figure 3.6. Data driven tree search.

that the tree search depicted in Fig. 3.6 represents data-driven approach to the recognition process, the search being data driven because the sequence of matches is controlled by the order of the scene features. Alternatively, the recognition process may be cast in a model-driven framework, as shown in Fig. 3.7, where the sequence of matches is controlled by the order in which the model features are invoked. The time complexity associated with model-driven search is $O(n^m)$. For recognizing isolated single objects, since n will be approximately equal to $m/2$, reflecting the possibility that only half the features would in most cases be visible from a viewpoint, and since $m^{(m/2)}$ is less than $(m/2)^m$, one can argue that for such cases a data-driven search might be more efficient than a model-driven search.

Going back to the data-driven tree search shown in Fig. 3.6, since each path in the tree corresponds to a possible solution to the correspondence c, and since in the worst case the search may have to sweep through the entire space (via, say, backtracking), and since the total number of nodes in the space is of the order of m^n, the time complexity of an exhaustive search on the tree is equal to $O(m^n)$. This exponential complexity, which is unacceptable for practically all applications, can be substantially reduced by using constraint propagation, of which hypothesis-and-verify is one example. In this paper, we will show that it is possible to establish a hypothesis-and-verify approach in such a manner that the time complexity reduces from the exponential to a low order polynomial.

3.3. Related Literature

Oshima and Shirai [$O\&S-83$] [$Sh-87$]represent both the scene and the models by graphs whose nodes represent planar or smoothly curved surfaces together with their attributes, and whose arcs represent relations between surfaces, the relations being of type adjacency, convexity or concavity of common edges, dihedral angles, distance between the centroids, etc. The attributes of the surfaces at the nodes include perimeter, area, mean region radii, etc. In fact, for the models a separate graph may be used for each "typical view of the object;" the authors do not make clear what they mean by a "typical view." Given a range image of unknown objects, their system first selects a "kernel", which consists of one or two most reliable surfaces for recognition, then performs an exhaustive search of all model graphs to find all compatible model kernels. Finally, the system performs a model-driven depth first search and attempts to find a correspondence between the remaining scene surfaces and the model surfaces. This approach is an example of hypothesis generation and verification, a hypothesis being a model object in a pose corresponding to what they call a typical view.

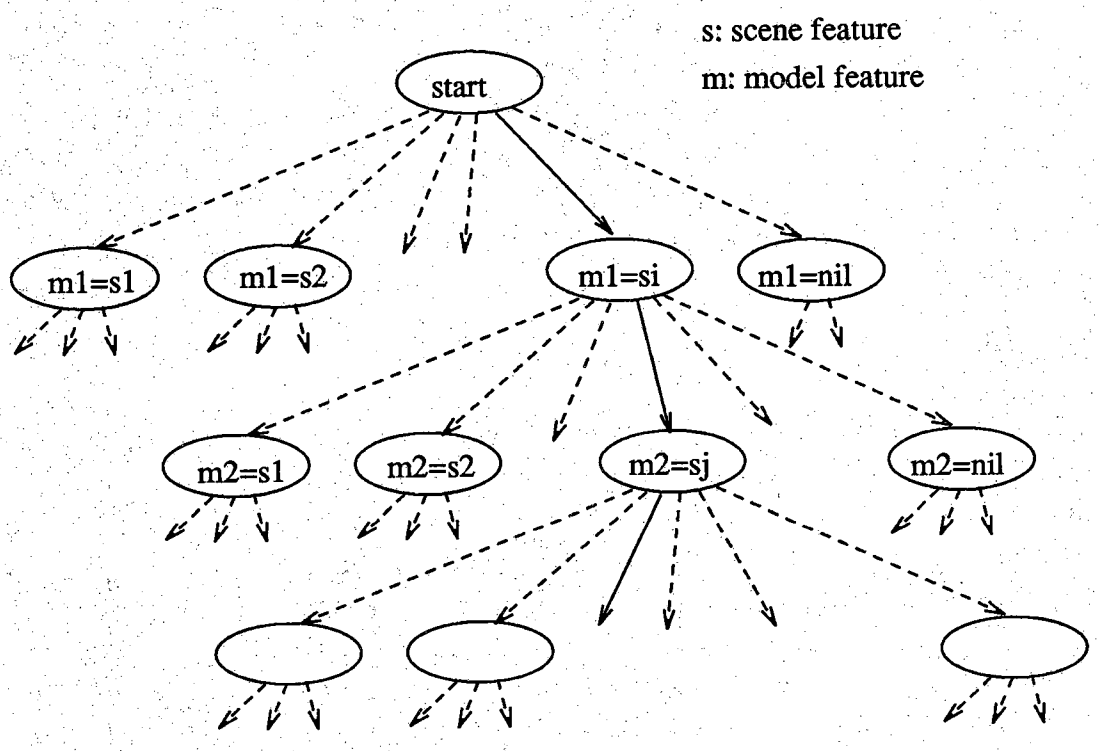


Figure 3.7. Model driven tree search.

Fan, Medioni and Nevatia [*F&et*–88] present a scheme for establishing the correspondence of objects and object surfaces between two range images representing two views of the objects. Note that this problem is similar to the one addressed by Oshima and Shirai, in which a model is actually a typical view of its corresponding object. They also describe scenes by graphs as Oshima and Shirai did. However, in their approach, before feature matching starts, each graph is segmented into a set of subgraphs, each subgraph representing a candidate object to be matched. This object segmentation is done by grouping surfaces mainly linked by convex and concave boundaries. Besides, they use a richer set of surface descriptors, such as average principal curvatures, 3-D areas, etc., and use transformation constraints, which are global constraints, in addition to the inter-surfaces constraints for graph matching.

Tomita and Kanade [*T&K*–84] present an edge-based vision system for recognition and localization of 3-D objects. An input range image is segmented into planar or conic surfaces and each surface is then descibed by its boundary segments. Each object model is described in the same manner from range images acquired during a learning phase. By matching boundary components through an exhaustive search, a transformation is then hypothesized. The candidate transformation is then tested by matching the other boundary segments for verification, the matching invoked during the verification phase being controlled by reliability, plausibility, etc.

Bolles and Horaud [*B&H*–86] in their 3DPO system emphasize the importance of feature ordering in reducing the complexity of matching. Their matching strategy starts with a distinctive edge feature, and then grows a match by adding compatible features, one at a time. Once a sufficient number of compatible features has been detected to allow a hypothesis to be formed, the verification procedure evaluates it by comparing the measured range data with the data predicted according to the hypothesis. The ordering of features to be matched is predetermined by an interactive feature-selection process, the ordering being done on the basis of uniqueness, cost of detecting it, likelihood of its detection, etc. Selection of a branch at a node of the search tree is hard-wired in this system. In other words, the sequence in which the scene and object features are matched is precompiled by the human operator.

Ikeuchi [*Ik*–87] presents a procedure for determining the unknown attitude of an object from 3-D measurements; the procedure uses an interpretation tree to guide the process of matching scene features with model features. Upper levels of the interpretation tree seek to categorize the attitude of the object with respect to one on its aspects (with respect to a given viewpoint, an aspect is a grouping all the attitudes that appear topologically similar from that viewpoint). Attitude categorization takes into account visible dominant surfaces and their interrelationships. Starting with the knowledge of

the aspect, lower levels of Ikeuchi's interpretation tree then try to calculate more precisely the object attitude. In the work reported in [*Ik*–87], the interpretation tree was specified by a human; however, we believe, it is now possible to compile it automatically from the CAD models of an object. The reader should note the important difference between an interpretion tree and the search tree shown in Fig. 3.6. Whereas the latter is a search space associated with the pairings of scene and model features, the former is a delineation of the steps of a procedure that must be invoked for interpretation, the alternative choices of the steps of the procedure being conveniently represented by the branching of a tree.

Hansen and Henderson [*H&H*–87] also propose a method for the automatic generation of recognition algorithms, also in the form of interpretation trees, based on the geometric properties of object shapes. They select and order features in the interpretation tree based on four qualities, namely, rarity, robustness, computational expense , completeness, and consistency. The synthesized interpretation tree has two parts, the first part is for matching the strongest set of view independent features; the second part then finds corroborating evidence in support of the hypothesis generated by the first part and thus completes the determination of the object's pose.

Faugeras and Hebert [*F&H*–83], [*F&H*–86] present a 3-D object recognition and localization system based on geometrical matching between primitive surfaces. Primitive surfaces in a scene are segmented by region growing based on planar or quadric surface fit. Each model object is also represented by a list of primitive surfaces. The recognition process consists of first selecting from the model object a feature, such as an edge or a surface; then they look for a scene feature to match the selected model feature. The next step consists of the selection of another model feature, such that this and the previous model features completely constrain the pose of the model object (clearly, if the two model features are edges, they cannot be parallel; or, if one model feature is a cylindrical surface and the other a planar surface, the principal axes of the two cannot be orthogonal). The system looks for a scene feature to match the second model feature. The two scene features thus found generate a hypothesis for the tranformation matrix for the scene object. The verification of the hypothesis consists of matching the remaining model features with the available scene features taking into account the position transformation. The verification stage carries out the minimization of an error measure, the measure being a sum of the errors between the predicted positions and orientations of the model features and their actual positions and orientations in the scene. The construction of this error measure is facilitated by organizing the scene features on a unit sphere which captures the relative orientations of the features and allows quick comparison with the transformed model features.

Grimson and Lozano-Perez [*G&L*–84] address the task of matching a set of observed range points, together with the measured local surface normal, with the surfaces of polyhedral objects. Matching consists of assigning observed points to object surfaces under the constraints corresponding to the upper and lower bounds on parameters such as the distances between two observed points, the angles between their assigned surfaces, etc. When a given set of surface assignments satisfies all the local constraints, a feasible interpretation of the measurements is generated — a feasible interpretation is like a hypothesis that must be verified subsequently. Local constraints constitute necessary but not sufficient conditions for the correct assignment of surfaces to the measured range points. Another way of expressing the same idea is that even if local constraints are satisfied, there may not exist a valid global transformation that agrees with the generated assignments of surfaces to the measured points. This is illustrated by Fig. 3.8, The measured points P_1, P_2, and P_3 can satisfy all the usual local constraints with regard to assignments (P_1,f_1), (P_2,f_2), (P_3,f_3), where f_1,f_2, and f_3 are three model surfaces. However, there does not exist a global transformation that would agree with the assignment. During verification phase, for every two measured points and the normals of their assigned surfaces, a rotation matrix is estimated; and, eventually, the centroid of all such rotation matrices is calculated to serve as a rotation transformation to bring the model and the scene objects into coincidence. In a similar manner, from every three measured points, a translation matrix is computed and, eventually, the centroid of all such translation matrices found. The average rotation and translation matrices thus computed are applied to transform the model object into the scene space. Finally, the validity of the average transformations is computed by projecting the measured points onto the corresponding model surfaces. During this process, the system calculates the normal distance between a measured point and its assigned surface; and, also, the system projects the point onto the assigned surface to carry out an "inside-outside" computation. If the projection is found to be outside the assigned surface, that is an indication that the feasible interpretation under test may not be so feasible after all.

In a more recent report, Grimson and Lozano-Perez have reported an extension of their system in which constraint propagation is used in conjunction with the local constraint satisfaction [*G&L*–87]. The basic idea here will be explained with the help of Fig. 3.9. We will assume that we have two range measurements and the associated surface normal measurements, as represented by the points P1 and P2 in Fig. 3.9. From these measurements, we can compute the distance d and the angle θ. Let us further assume that on the basis of the measured surface normals we have selected the two model surfaces shown in the figure. In the model database, we will associate a "base point" with each surface, as depicted by the points B1 and B2 in the figure. One can now show that the two measured points can occupy only particular positions with

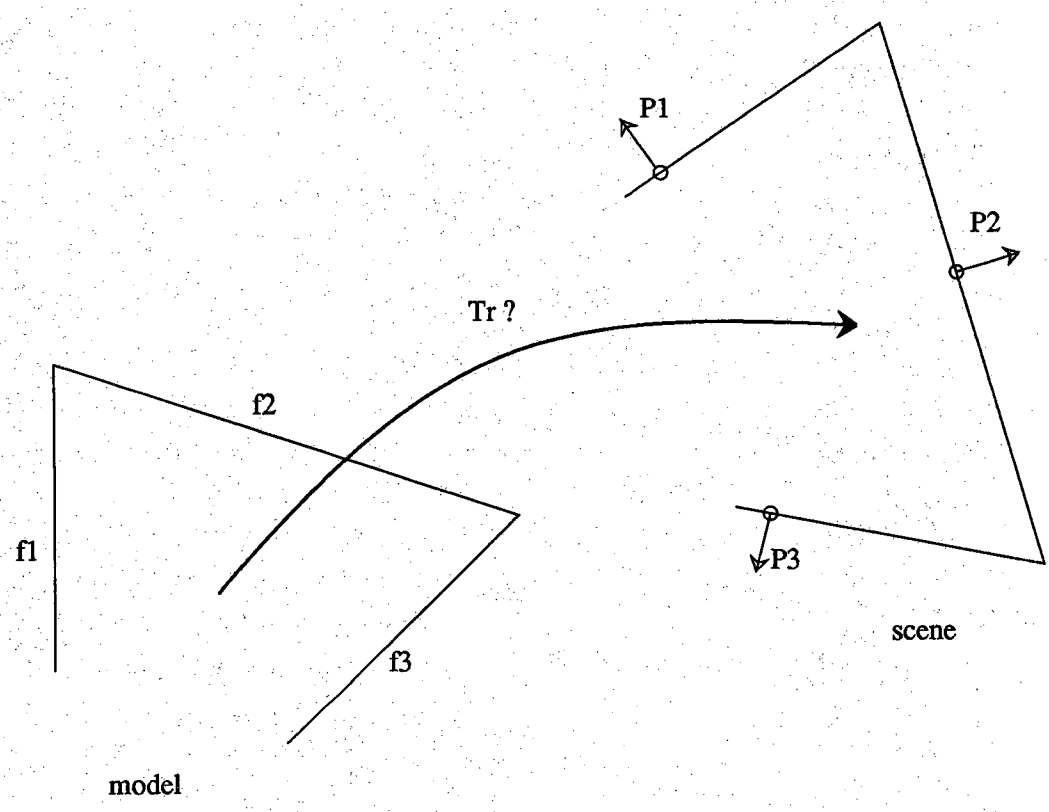


Figure 3.8. This figure shows that satisfying local constraints does not guarantee a global transformation between the model object and the scene object.

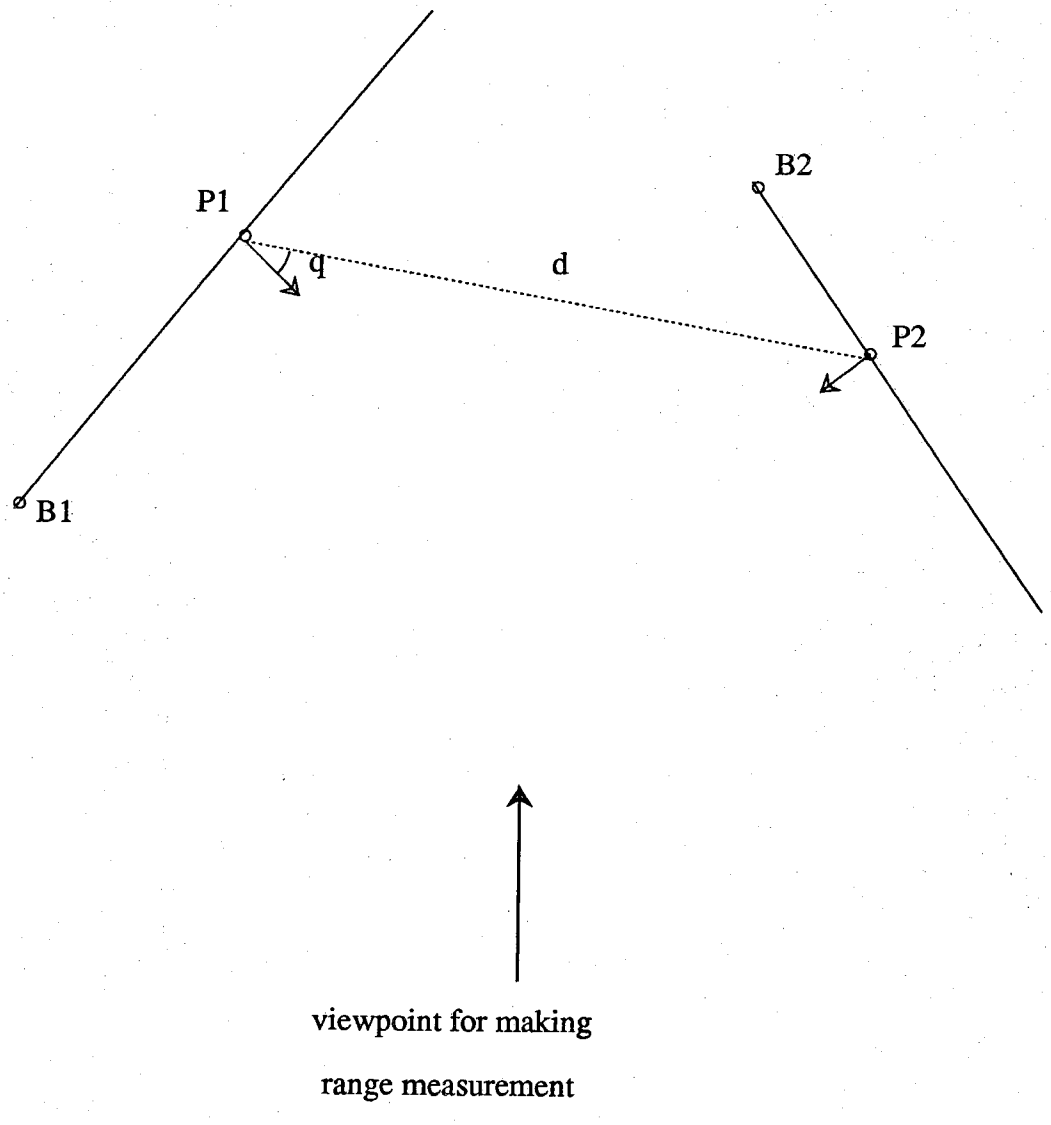


Figure 3.9. If we know the range values to the two points, P1 and P2, shown in the figure and the local surface normals, we can then compute the distance d and the angle θ. These two parameters are used in a constraint propagation approach to object interpretation.

respect to the base points on the respective surfaces. In other words, after the measured point P1 is assigned a location on surface 1 with respect to base point B1, then on surface 2 the measured point P2 must lie at a fixed location with respect to base point B2 — strictly speaking, this being true only in the absence of noise. In the presence of noise, one could predict an interval in which the second measurement must lie. The idea in constraint propagation is to compute this permissible interval as one goes down the interpretation tree and, of course, when the permissible interval for any assignment goes to zero, one need go no further along that path in the tree. We believe that this very elegant idea is quite complex to implement for the case of 3-D objects because of the algebraic difficulties with the representation of arbitrary shaped polygons in terms of parameters that can be bounded for the purposes of constraint propagation. However, basically, the idea is powerful and merits further investigation.

In their object recognition approach, Besl and Jain [*B&J*−86] use the idea that a smooth surface can be characterized by its Gaussian and mean curvatures, which are invariant to rotations and translations. Each surface point is classified into one of eight categories according to the signs of its Gaussian and mean curvatures. Their recognition procedure first extracts critical points from a range image, a critical point being defined by the intersection of zero-crossings of the two partial first derivatives of a range map with respect to the two parameters of surface parametrization. Only critical points with positive Gaussian curvature are used. Finally, a critical point together with its neighborhood, which carries the signed information about the Gaussian and mean curvatures, is used as a shape descriptor for the purpose of object characterization. Originally, the aim of these authors was to use such shape descriptors as view-independent features, since, after all, the Gaussian and mean curvature computations are view independent. However, as the authors now realize [*Ja*−88], the choice of the critical points is extremely sensitive to viewpoint selection. For that reason, this strategy may not yield acceptable results in practice.

Bhanu [*Bh*−84] uses stochastic relaxation for labeling the surfaces in a range map. Object surfaces are approximated by convex polygon faces, and the neighbors of each face are ranked according to their areas. Fundamental to any relaxation labeling is the definition of a compatibility measure which measures the appropriateness of assigning a model label to a scene surface given a label assignment at a neighboring surface. In Bhanu's work, this measure is defined as a function of the distances between the centroids of the scene surfaces on the one hand and between their assigned labels on the other; of ratios of the areas of the surfaces computed in a similar fashion; etc. Computation consists of two iterative stages. In the first stage, only a pairwise compatibility function is used, meaning, for example, that the compatibility function is a function of some attribute of two neighboring surfaces and their two corresponding labels. In the

second stage, the compatibility function depends upon three object surfaces and their three corresponding labels at the same time. In each stage, at the end of the iterative computation, those labels are retained that maximize the overall compatibility of the assigned labels.

Another interesting approach called "pose clustering" or "generalized Hough transform" is proposed by Stockman [*St*–87]. He uses a Hough space of 12 parameters which are the 12 unknowns of a 4×4 transformation matrix. The basic idea in this approach is as follows: Suppose we match a planar scene surface with a similar model surface, in general this would not generate a unique transformation because of the remaining three degrees of freedom, meaning that after the two surfaces are brought into coincidence, it will still be possible to translate and rotate one with respect to the other. Although a unique transformation is not generated, such a match does constrain the possible transformations; implying that in the parameter space such a match would lead to a subregion which must contain the correct transformation. By dividing the parameter space into bins and keeping track of contributions made to each bin by different matches, and retaining the bin with a maximal entry, it should theoretically be possible to determine the correct transformation. In practice, a direct implementation of this idea is difficult because of bin counting in high dimensional spaces. To get around this difficulty, Stockman matches a pair of non-coplanar edges from the scene with a pair of edges from the model, such a match generating a single point in the parameter space, since there can only exist one transformation matrix capable of taking two non-coplanar edges from the scene into two non-coplanar edges from the model. Stockman then applies a clustering algorithm to the resulting points in the parameter space to find the most plausible tranformation. The clustering algorithm consists of examining every point generated in the parameter space, constructing neighborhood around such points, and counting the number of neighboring such points in each neighborhood. The point with the largest number of neighbors is the desired unique transformation. The important issues in this computation are those that deal with the choice of the size of neighborhood, and the thresholds to be used for rejecting the most dense neighborhoods.

In the literature citations above, all the researchers have first extracted features, such as edges, planar, quadric and other surfaces, etc., from range images and then matched these features with those of models. There do exist other methods in which matching is not preceded by feature extraction. For example, in the extended Gaussian image approach of Horn [*Ho*–84] and Ikeuchi [*Ik*–83], surface normals measured from a scene are mapped directly onto a unit sphere and the resulting EGI is then compared with that of the model. The EGI approach has also been used in [*Y&K*–86] and [*Y&K*–86] for determining the identify, position and orientation of the topmost object in a pile of simple-shaped objects.

Then, there exist approaches where for every possible viewpoint an integrated value of some geometrical parameter is represented on a unit sphere for comparison with a similar representation of a model. For example, in the work of Fekete and Davis [*F&D*–84] and Korn and Dyer [*K&D*–87], an integrated value of some surface property seen from different viewpoints is recorded on a sphere and used for the estimation of object pose. Note that the feature sphere idea advanced in our work here bears no resemblance to this prior work. Another interesting approach, called geometric hashing, is proposed by Lamdan and Wolfson [*L&W*–88]. In that approach, every model feature is made accessible through a hashing table, the hashing function being derived from the positions/orientations of the features in relation to a coordinate system defined on a base which consists of a minimal set of features for determining an object pose.

Finally, there are many approaches designed for the recognition of 3-D or 2-D objects using 2-D imagery, e.g. the ruled-based ACRONYM system [*Br*–83], the characteristic views approach [*C&F*–82], and the automatic programming approach [*Go*–83], just to name a few. The reader is referred to the review articles by Chin and Dyer [*C&D*–86] and Besl and Jain [*B&J*–85] for general surveys of the subject.

3.4. Features for Object Recognition

The concept of a feature normally implies some saliency which makes it especially effective in describing objects and in matching. Since the recognition of objects will be solely based on shape, of primary interest are geometric features, such as edges, vertices, surfaces together with their mathematical forms, etc. Such features specify three dimensional shape, in contrast with features like surface texture, color, etc. These latter features, although important for recognition of objects by humans, will not be addressed in this paper, since they can not be detected in range images. In 3D-POLY we consider only those geometric features with which we can associate positions or orientations. For example, a vertex feature has associated with it a position vector, which is the vector from the origin to the vertex. Similarly, a cylindrical-surface feature has associated with it an orientation, which is the unit vector parallel to the axis of the cylinder. We will categorize the geometrical features into three different classes: primitive surfaces, primitive edges, and point features.

Primitive surfaces include planar surfaces, cylindrical surfaces and conic surfaces, which are three special cases of quadric surfaces. Primitive edges refer to straight-line features or ellipsoidal-curve features. Point features consist mainly of object vertices and those surface points that have distinctive differential-geometrical properties; surface points falling in the latter category exhibit maximal or minimal curvatures or can be saddle points. These three classes of features are effective in describing the shape of

an object, and, what is more, they can be reliably extracted from range images. Note that we are only using simple geometrical features, as compared to more complex ones, like generalized cylinders [*N&B*–77] and primitive solids [*Re*–80], that are often difficult to extract from range imagery.

3.4.1. Attributes of Features

We represent a feature by a set of attribute-value pairs, each pair being denoted by $<a:v>$, where a is the name of the attribute and v its value. The value of an attribute can be a number, a symbol, a set of labels of other features, or a list of some of these, depending on the nature of the attribute. For example, the surface feature $s2$ of the model object in Fig. 3.10 may be described by

$<surface\_type: cylindrical>$,
$<radius: 3>$,
$<axis: (0.0, 0.0, 1.0)>$,
$<area: 3>$,
$<point\_on\_axis: (0.0, 0.0, 0.0)>$,
$<adjacent\_region: \{s1, s3, s4\}>, \cdots$

The attributes of a feature, according to their geometric and topological characteristics, can be categorized on the bases of shape, relations and position/orientation. Each of these categories will be discussed in greater detail below.

- **Shape attributes**:

  A shape attribute, denoted by *sa*, describes the local geometric shape of the feature, For example, "surface_type", "radius", and "area" are some of the possible shape attributes of surface feature $s2$. Ideally, a shape attribute should be transformation invariant, i.e. independent of the object's position and orientation, in practice when a feature is seen through a sensor, some of its shape attributes may "look" different from different viewpoints. Therefore, we make a distinction between two different types of shape attributes, those that are viewpoint independent and those that are not. For example, the area of a surface and the length of an edge are viewpoint dependent, because they may vary with viewing direction due to occlusion. On the other hand, the attributes surface-type and radius are viewpoint independent. [Of course, we do realize that in high-noise and high-occlusion situations, a precise estimation of, say, the radius of a cylindrical surface may be difficult and may even become viewpoint dependent.] Clearly, when comparing shape attributes for matching a scene feature to a model feature, we should take this viewpoint-dependency into consideration.

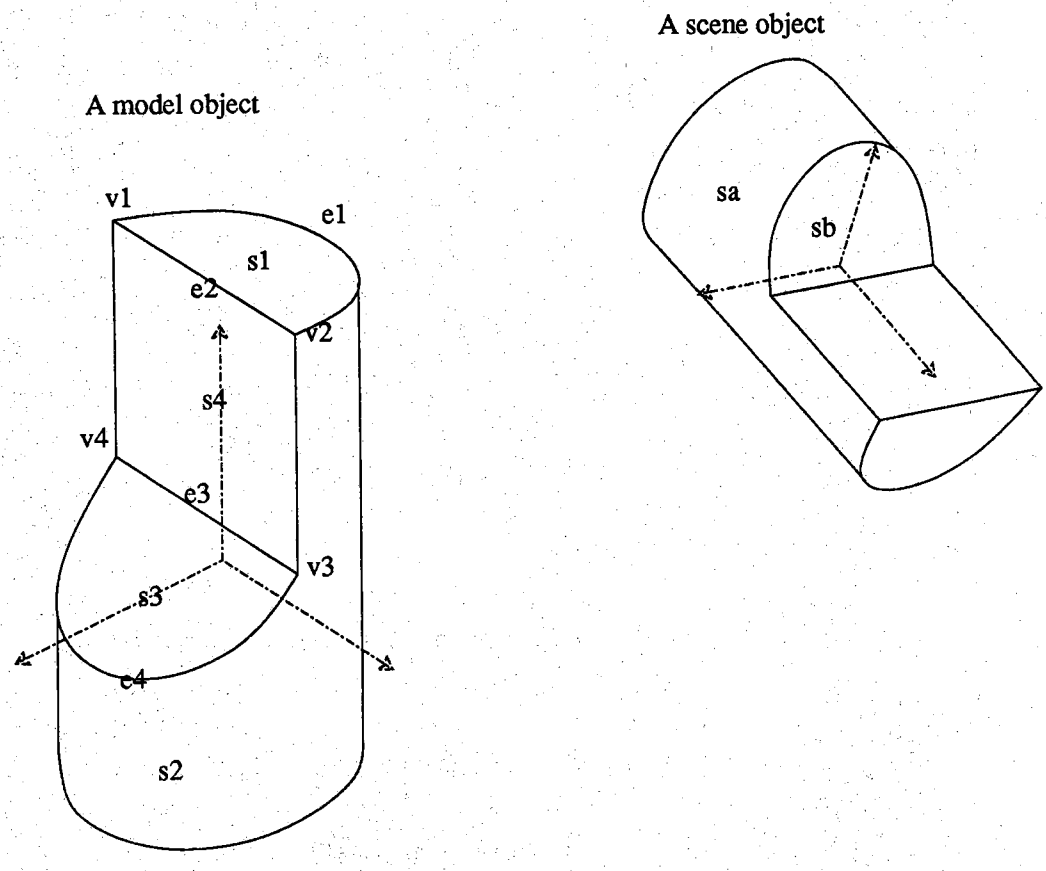


Figure 3.10. Labels of the primitive surfaces, primitive edges and the vertices of the object shown in Fig 3.4.

- **Relation attributes:**

  A relation attribute, denoted by *ra*, indicates how a given feature is topologically related to other features. For example, for the surface feature $s2$, the attribute "<*adjacent_to*: $\{s1, s3, s4\}$>" indicates its adjacency with three other surfaces. Relation attributes should also be independent of transformation. An attribute "<on_top_of: s1>" is not a proper relation attribute for feature $s2$ because it depends on the pose of the object.

- **Position/orientation attributes:**

  A position/orientation attribute, denoted by *la*, specifies position and/or orientation of a feature with respect to some coordinate system. In general, the position/orientation attributes of a scene feature are measured with respect to a world coordinate system, and those of a model feature are measured with respect to a model-centered coordinate system. As with shape attributes, some position/orientation attributes may be viewpoint-dependent, such as the centroid of a surface, or the midpoint of an edge; while others may be viewpoint-independent, as with the attributes surface-normal of a planar surface, or the axis of a cylindrical surface, etc. The viewpoint-dependent position/orientation attributes should be avoided in the estimation of *Tr* during hypothesis generation (see next section), however, they can be useful for rapidly eliminating the unmatched features during the verification process.

Since a feature may possess more than one shape attribute — for example, the feature $s2$ in Fig. 3.10 possesses the shape attributes surface-type, area, radius, etc. — we will use the symbol *SA* to denote the set of shape attributes associated with a feature. Similarly, we will use symbols *RA* and *LA* to denote the sets of relation and position/orientation attributes of a feature, respectively. Therefore, the feature edge $e2$ of the object in Fig. 3.10 may be described by

$$SA(e2) = \{<shape: straight>, <length: 3>, \cdots\},$$

$$LA(e2) = \{<direction: (0.0,1.0,0.0)>, <point\_on\_edge: (1.0,0.0,3.0)>, \cdots\},$$

and

$$RA(e2) = \{<end\_vertex: v1>, \cdots\}.$$

In practice, we may not need to use all the three categories of attributes but only those useful for a particular application. For example, Faugeras and Hebert in their geometric matching approach [*F&H*–86] have used only shape and position/orientation attributes. The set of attributes used in describing features should also depend on the sensor capability and the performance of the feature extractors. Using attributes which can not be reliably detected or measured by a sensor will not contribute much to solving the problem of object recognition. In any event, a minimal requirement in deciding which attributes to use for describing features is that no two features in an image or in a model be

allowed to have the same set of attribute-value pairs. This, of course, does not imply that a model or a scene not contain multiple instances of a particular feature type. To elaborate, the vertex features $v1$ and $v2$ for the model object in Fig. 3.10 are identical, but their attribute-value pairs will be different because of the differences in their position/orientation attributes.

3.4.2. Principal Directions of Model Features

Empirical observations show that an important characteristic associated with an object feature is what we call its principal direction. In an object-centered coordinate system, the principal direction of a feature gives us a fix on the directional position and/or orientation of the feature with respect to the other features on the object. Since, one must first establish an object-centered coordinate frame, a principal direction can only be assigned to a model object, or to a scene object which has been embedded in an object-centered coordinate frame via prior matches with some features.

In Section 5, we show how a useful data structure — we call them feature spheres — can be defined using the concept of principal directions. Here, we will define the principal directions more formally and make the reader aware of the fact that the manner in which such definitions are made are different for different classes of features, and within each class, for different types of features. As was mentioned before, different classes of features correspond to primitive surfaces, primitive edges and primitive points. Within the class primitive surfaces, we may have different types of features such as cylindrical, planar, spherical surfaces, and so on. While for some class and type of features, the principal direction represents their orientations, for others it represents their directional position on the object.

We will now formally define the prinicpal direction, denoted by Φ, for the three classes of features and for types within each class:

(1) Primitive surfaces:

- **Planar surface:**
  $\Phi \equiv$ *The direction of the outward surface normal*
- **Cylindrical surface or conic surface:**
  $\Phi \equiv$ *The direction of the axis*. The 180° ambiguity associated with this direction is resolved by choosing that direction which subtends an acute angle with the positive z-axis. For axes that are perpendicular to the z-coordinate, that direction is retained which is closest to the x-coordinate. And, the axis is perpendicular to both z and x, then we choose the +y direction.
- **Spherical surface:**
  Let o be the position vector to the center of the sphere in the object-centered

coordinate system. The principal direction is defined by the normalized form of this position vector:

$$\Phi \equiv \frac{o}{|o|}$$

Note that this principal direction is different in character from that defined for a cylindrical surface. We choose this form for Φ because it is not possible to associate an orientation vector with a spherical surface.

- **Quadric surface:**

A quadric surface has the form $x^t Ax + x \cdot B + C = 0$.

$$\Phi \equiv \frac{a_p}{|a_p|}$$

where a_p is the principal eigenvector of the matrix A.

In general, eigenvector associated with the largest eigenvalue is the principal eigenvector of A and the direction associated with this eigenvector usually defines the major axis of a surface. Since a quadric description includes the cases of planar, cylinderical and spherical surfaces defined above; the definition of the principal axis here must be used with care. The quadric definition is used only if a surface cannot be classified as being either planar, cylindrical or spherical.

(2) Primitive curves:

- **Straight line:**

$\Phi \equiv$ *line direction*

The 180° ambiguity associated with the direction of a line is resolved using the same criterion as for the axis of a cylinder.

- **Circular or ellipsoid curve:**

$\Phi \equiv$ *surface normal of the curve's plane*

The 180° here, too, is resolved as for the case of the axis of a cylinder.

(3) Point features:

Let p be the position vector of a point feature with respect to the object-centered coordinate system. The principal direction is defined by normalizing the position vector:

$$\Phi \equiv \frac{p}{|p|}$$

The important thing to note is that the parameters used in the definitions of principal directions are all extracted with relative ease from range maps. For example, if from a given viewpoint in a range map about 40 percent of the round part of a cylindrical

surface is visible, in most cases it is possible to make a good estimate of the direction of the axis of the cylinder.

3.4.3. Criteria for Feature Matching

We will now provide matching criteria for the matching problem expressed in equation (3.1) and express these in terms of the three attribute classes. In other words, we will express the conditions for each attribute class, conditions that must be satisfied for a scene object to match a model object. Our conditions are applicable strictly only under the noiseless case. For actual measurements, the comparisons implied by our conditions would have to treated in relation to some user specified thresholds, the magnitudes of the thresholds depending upon the noise and other uncertainties in the system.

- **Matching Criteria for Shape Attributes:**

  The reader will recall that we have two different types of shape attributes, those that are viewpoint independent and those that are not. A viewpoint independent shape attribute *sa* of a scene object feature is said to match the corresponding shape attribute of a model object feature if

  $$sa(S_i) = sa(M_{c(i)}). \tag{3.2}$$

  where the function sa(.) returns the value of the attribute sa for the feature that is its argument; S_i is a feature from the scene and $M_{c(i)}$ is the candidate model feature that is under test for matching with the scene feature. The above equality must be satisfied for each $sa \in SA(S_i)$. For viewpoint dependent shape attributes, clearly we cannot insist upon an equality in the above equation. In general, for such attributes, we require

  $$sa(S_i) \leq sa(M_{c(i)}).$$

  Note that since all the viewpoint dependent shape attributes are numerical in nature, we only have to use numerical inequalities and not, say, subsets, as would be case for symbolic features. For example, we would expect the length of a scene edge be equal to or less than the length of the corresponding model edge due to possible occlusion. Therefore, the matching criterion for this attribute can only be expressed as

  $$edge\_length\ (S_i) \leq edge\_length\ (M_{c(i)}).$$

- **Matching Criteria for Relation Attributes :**

  Relation attributes are also transformation invariant, thus if a scene feature S_i has relation *ra* with, say, scene feature S_l, then the model feature $M_{c(i)}$ must have the same relation *ra* with the model feature $M_{c(l)}$. More precisely, for every $ra \in RA(S_i)$

$$c(ra(S_i)) \subseteq ra(M_{c(i)}). \tag{3.3}$$

To justify the nature of this comparison, consider that the model object is as shown in Fig. 3.10. Further, suppose that in the scene the model surfaces s2 and s3 are visible and have been labeled as, say, S_a and S_b, respectively. Then, from the model description, we have

$$adjacent\_to(s3) = \{s2, s4\}$$

and, from the scene information,

$$adjacent\_to(S_b) = \{S_a\}$$

Let's say that during the hypothesis generation phase, an estimate was made for the transformation that takes the model object into the scene object. Let's further say that using this tranformation, we have already established the correspondence of the scene surface S_a with the model surface s_2, and, now, we are testing the correspondence of the scene surface S_b with the model surface s_3. We see that since $adjacent\_to(S_b) = S_a$, and since $c(S_a) = s_2$, a substitution in equation (3.3) yields for $ra = adjacent\_to$

$$\{s2\} \subseteq \{s2, s4\}$$

which being true implies that the scene surface S_b can indeed be matched with the model surface s_3, at least from the standpoint of satisfying relational constraints. The point to note is that in the matching criterion of equation (3.3) the features participating in a relation at a given scene feature S_i should be a subset of the features participating at corresponding model feature $M_{c(i)}$; it is not possible to replace the "subset" comparison with a strict equality because not all of the model surfaces may be visible in the scene.

- **Matching Criteria for Position/Orientation Attributes**:

For a viewpoint independent position/orientation attribute *la*, such as the location of a vertex or the direction of an edge, the matching criterion is described by

$$la(S_i) = \boldsymbol{R} \cdot la(M_{c(i)}) \qquad \text{if } la \text{ is an orientation vector,} \tag{3.4-a}$$

$$la(S_i) = \boldsymbol{R} \cdot la(M_{c(i)}) + \boldsymbol{t} \qquad \text{if } la \text{ is a position vector.} \tag{3.4-b}$$

for every $la \in LA(S_i)$. These criteria play a vital role in the localization of a scene object. Recall that the matching process starts with $Tr = \boldsymbol{R}^{Tr}$ and should end up with a unique solution which is the location of the scene object, else it should fail. Before Tr has converged to a unique solution equation (3.4-a) or (3.4-b) provide a system of equations to solve for Tr, but after that, they are nothing but two predicates which confirm or reject a match between the scene and model features.

Many important position attributes are not viewpoint independent, yet they are important. For example, the position attributes *point_on_axis* and *point_on_edge* shown in Section 4.1 are both viewpoint dependent since these points can be at arbitrary locations along their respective directions. Despite their arbitrary locations, these points play a vital role during the verification phase of matching. For illustrating this important point, let's say a scene edge is under consideration for matching with a model edge under a given pose transformation. Now, if the directions of the two are identical, that's not a sufficient criterion for the match to be valid, since the identity of directions merely implies that the scene edge is parallel to the model edge. To impose the additional constraint that the two edges be collinear, we need the *point_on_edge* attribute even if the point is arbitrarily located on the edge in the model space. The *point_on_edge* attribute is used in the following fashion: First the difference vector between the vector to *point_on_edge* and a vector to some arbitrary point on the scene edge is computed. Then the cross-product of the difference vector with the *direction* attribute of, say, the model edge is calculated. The magnitude of this cross-product should be close to zero for the match to be acceptable. Note that while the cross-product being zero guarantees the collinearity of the model and scene edges, it still allows one degree-of-freedom between the two. It is impractical to completely constrain this remaining degree-of-freedom since in the presence of occlusions the model edge may not be completely visible in the scene.

Before concluding this subsection, we should mention that, in a manner similar to edges, the position attribute associated with a planar surface, specified by giving the coordinates of an arbitray point on the surface, can be used to make sure that a model surface is coplanar with the corresponding surface from the scene; again, the identity of surface orientations is not sufficient, and to cope with occlusions, it is not possible to constrain the two any further.

3.5. Matching Strategy

As mentioned in the introduction, the recognition method employed in 3D-POLY is based on hypothesis generation and verification. In this section, we will explain how hypotheses are generated and then how each hypothesis is verified.

In what follows, we will first show that if hypotheses are generated by exhaustive search, meaning that a scene feature is tested against every possible model feature, then the time complexity of the recognition procedure is $O(n \times m^{(h+1)})$, where n is the number of scene features, m the number of model features, and h the number of features used for the hypothesis generation. Unfortunately, this complexity reduction is not sufficient for most practical purposes. We then proceed to show how by using the

notion of *local feature sets* for generating hypotheses and using the *feature sphere* data structure for verification, the complexity can be improved to $O(n \times m \times h!)$. Finally, we will show that when we use the vertex features to organize the local feature sets, the complexity is further improved to $O(n^2)$.

3.5.1. Hypothesis Generation and Verification

It is rather well known that only a small number of features is necessary to estimate the tranformation matrix *Tr* that gives the pose of a scene object in relation to the corresponding model object [*St*–86, *Bo*–84]. In our work, this small number of features will be referred to as a *hypothesis generation feature set* (HGF). Clearly, it is the position/orientation attributes for the features in an HGF that must be used for the estimation of *Tr*. We have shown some possible HGF sets and the position/orientation attributes used for determining *Tr* in Table 3.1. Note that the table is not an exhaustive listing of all possible HGF sets, but only those which are rather frequently used. How exactly a *Tr* may be constructed from the position/orientation attributes of the different possible HGF sets may, for example, be found in [*St*–86, *Bo*–84, *G&L*–84, *F&H*–86]; each of these references discusses the method used to calculate a *Tr* for the type of HGF used.

Let's assume that a recognition procedure needs a maximum of h features to construct a hypothesis for the pose transform *Tr* for a candidate model object. We will further assume that we have somehow selected a subset of cardinality h of the scene features, this subset will constitute the HGF set and will be represented by $\{S_1, S_2, ..., S_h\}$; we wish to generate hypotheses by using the features in this subset. We may then divide the search tree in Fig. 3.6 into two parts as shown in Fig. 3.11, the division occurring at level h on the tree. Note that at the first level of the tree, we try to match the scene feature S_1 against all possible model features from the candidate object. Then, at the second level, at each node generated by the first level, we try to match the scene feature S_2 with every possible model feature; and so on.

As depicted in the figure, after a hypothesis is formed with h features, we use the remaining $n-h$ features for verification. In principle, if a hypothesis is correct, i.e. the scene object O_s is indeed an instance of the candidate model O_m at location *Tr*, we should then be able to find all the remaining $n-h$ matched feature pairs using the transformation *Tr*. This implies that in the verification phase the scene feature at each level will match with exactly one model feature. This uniqueness is guaranteed by the requirement that no two features of a model have the same description. To reiterate what was said in Section 4, on account of the different position/orientation attributes this requirement is easily met even for those features that might otherwise be identical,

Table 3.1 Summary of HGF sets

Configuration of features	Position/orientation attributes
Three unique, noncollinear points.	The three positions vectors associated with the three points.
One straight edge & one non-collinear point.	The orientation attribute associated with the direction of the edge, the position attribute associated with a point on the edge, and the position attribute associated with the noncollinear point.
One ellipsoidal edge & one noncollinear point.	The orientation attribute associated with the edge, the position attribute associated with a point on the axis of the ellipsoidal edge, and a position attribute associated with the noncollinear point.
Two primitive surfaces & one point.	The two principal directions associated with the two surfaces, and a position attribute associated with the extra point.
Three non-coplanar primitive surfaces.	The orientation attributes associated with any two of the surfaces, and three position attributes associated with some three points, one on each surface.

NOTES ON TABLE 1:

The reader might like to note that for each HGF set, the position/orientation attributes shown constitute the least amount of information that is required for the calculation of the pose transform using that set. That this is so should be obvious for the first set. For the second set, we need to know the coordinates of at least one point that is arbitrarily located on the straight edge. Without this extra information, the rotation transform computed from just the edge directions would not also 'move' the edge from its model space to the scene space; the point on the straight edge helps us make the model edge become collinear with the scene edge. Then, we can use the extra non-collinear point to constrain the rotation of the object around the edge. The same argument applies to the third HGF entry. The attributes listed for the fourth HGF set should be obvious, especially in light of Appendix A. Finally, for the last HGF set, while two orientation vectors are sufficient to give us the rotation transform, coordinates to three points, located on each of the surfaces, are needed to constrain the translation vector. Note that these three points can be at arbitrary locations on the surfaces in the model space, the same being true in the scene space. (See the end of Section 4 for how a point located arbitrarily can be used to constrain the location of a surface.)

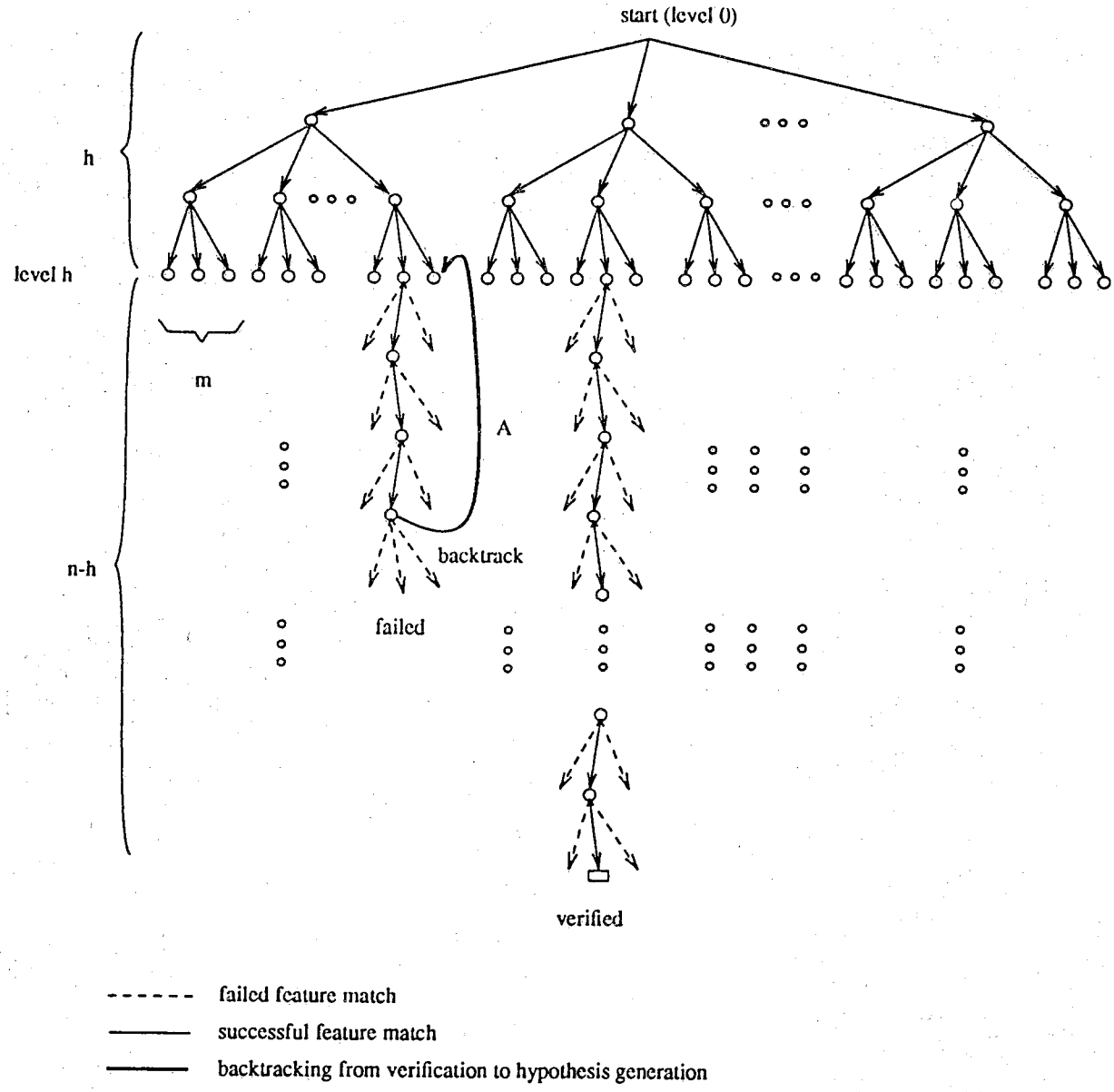


Figure 3.11. The data driven search tree is divided into two parts at level h on the tree. The first part represents the hypothesis generation stage while the second part represents the verification stage.

say, by virtue of their similar shapes.

On the other hand, if any one of the remaining $n-h$ features can not be matched to a model feature, that implies the current hypothesis is invalid, because either O_m is not the right object model or Tr is not the correct transformation. Therefore, when a scene feature, say, S_k, $k>h$, does not match any model features under the candidate Tr, it serves no purpose to backtrack to level $k-1$ or higher. Instead, the system should go back into the hypothesis generation phase, by backtracking over the first h levels, and try to generate another hypothesis for Tr, as illustrated by arc A in Fig 3.11. Clearly, if repeated backtracking over the first h levels fails to produce a valid Tr, the candidate model object should be discarded and new tree search begun with a new candidate object. In the rest of this subsection, we will explore the time complexity associated with this type of search.

This search process is exhaustive over the model features in the sense that at every node shown in Fig. 3.11, a scene feature must be compared with all the features of the candidate model object. Therefore, at each node, the complexity is proportional to m, the number of features in the model objects. The number m is also the fan-out at each node encountered during the hypothesis generation phase, i.e., in the first $h-1$ levels of the search space. However, the fan-out in the verification phase equals 1 because of our requirement that a match failure during verification implies going back into hypothesis generation.

Since backtracking is allowed to be exhaustive during the hypothesis generation phase, the time complexity associated with hypothesis generation is $O(m^h)$. The time complexity associated with the worst case verification scenario can be estimated by noting that each verification path has at most $n-h$ nodes, and, since at each node we must make m comparisons, the complexity of verification is $O(m \times n)$. Therefore, the overall complexity associated with this recognition process is

$$O(m^h) \times O(m \times n)$$

which is the same as

$$O(n \times m^{h+1})$$

For rigid objects, h will typically be equal to 3, although its precise value depends upon how carefully the HGF sets are constructed. Therefore, the expression for the complexity function becomes

$$O(n \times m^4)$$

Although one may consider this complexity function to be a substantial improvement over the $O(m^n)$ function associated with the search tree of Fig. 3.6, it is still not

acceptable for practical applications. In the next subsection, we will show how by constraining the selection of model features for matching we can make further reductions in the complexity.

3.5.2. How to Constrain the Selection of Model Features

In this subsection, we will explore the question of what constraints one should invoke to select model features for matching with a scene feature. Given that the comparison of attribute values plays a central role in the matching process, the constraints we are looking for should be derivable from the attributes. But, since we have three different kinds of attributes, namely, shape, relation, and position/orientation, the question that arises is which of these attributes are best suited for the required constraints.

To answer this question, we will take the reader through a two-dimensional example shown in Fig. 3.12. With the help of this example we will convince the reader that the attributes used for constraining the selection of model features should depend upon whether or not we know the tranform *Tr*. In other words, the constraints used in the hypothesis generation phase must, of necessity, be different from those used in the verification phase. We will show that for hypothesis generation phase, we must take recourse to an idea suggested and used by other researchers: the model feature that is invoked for comparison against a scene feature should depend upon its relations with the previous model features in the path traversed so far in the search space of Fig. 3.6. And, for the verification phase, we show that remarkable reductions in computational complexity can be achieved by using constraints derived from the principal directions of features — recall that the principal direction of a feature is derivable from its position/orientation features. We will then show that the invocation of constraints on the principal directions is greatly facilitated if the features are organized according to a special data structure we call the feature sphere.

To help explain our points, in Fig. 3.12 is shown a 2-D range image of a polygonal object. The viewpoint is from the top, as illustrated. Range mapping is orthogonal, meaning that the lines of sight for the determination of range values are parallel; for example, the range at scene point 1 is equal to the distance d1, and d1 is parallel to d2, the range at scene point 2. The model polygon is shown in Fig. 3.13. The problem is to recognize and locate the polygon in Fig. 3.12 given its model in Fig. 3.13. We will assume that the recognition system is using only vertex features (an example of primitive point feature type). Scene vertices will be denoted by integers 1, 2, 3, $\cdots$, and those of the model by letters, *a*, *b*, *c*, $\cdots$. From the viewpoint shown, only the model vertices *a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*, *k*, *l* are visible to the sensor. For the sake of argument, we will assume that of these vertices, the model vertex *d* is not detectable; therefore, its

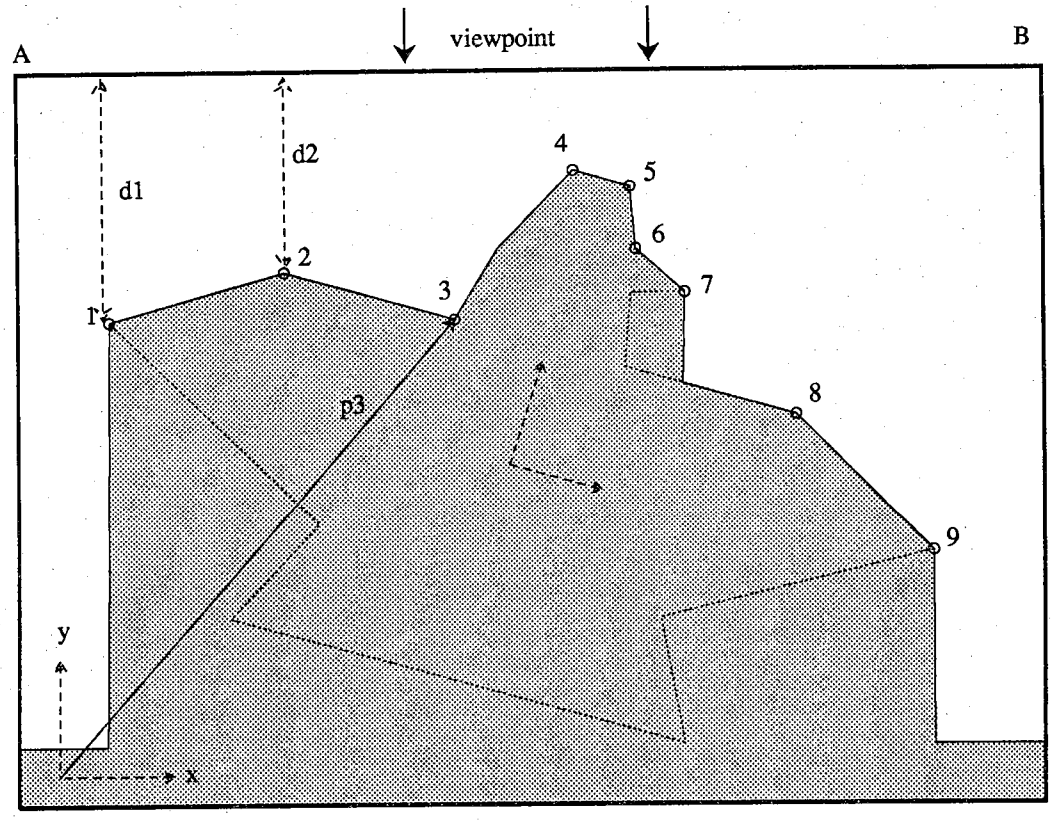


Figure 3.12. A 2-D range image of a polygon. The range values are proportional to the perpendicular distance from line AB. For example, the range corresponding to scene point 1 is equal to the distance d1.

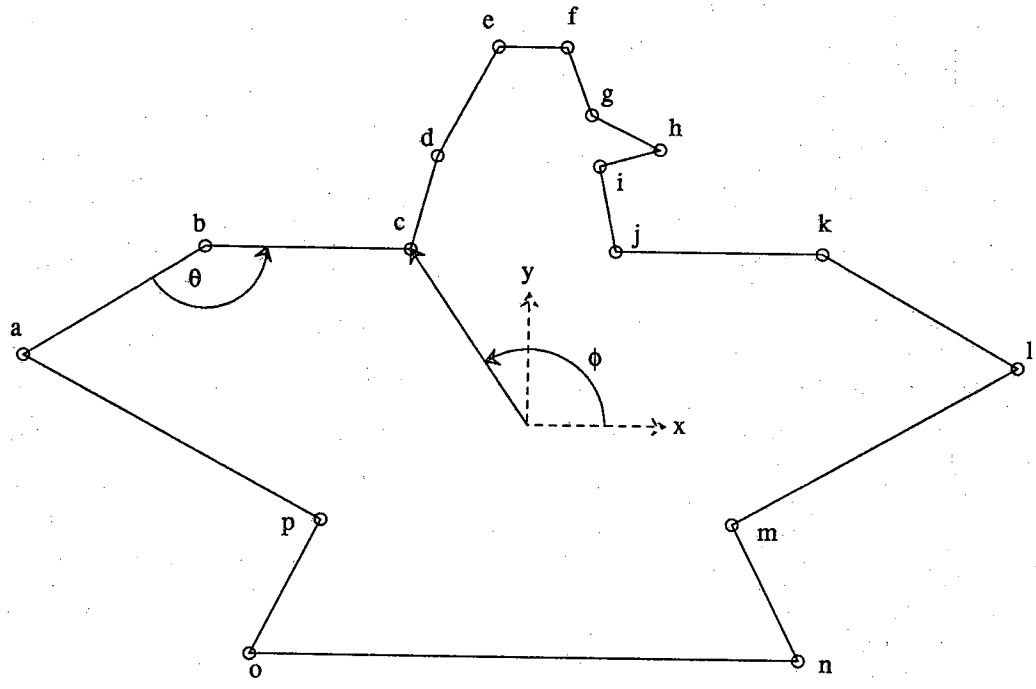


Figure 3.13. The model of the polygon shown in Fig. 3.12.

correspondent in Fig. 3.12 has not been given a label. The undetectability of d in the sensor data could be due to the fact that the angle defining that vertex is not very sharp. As a result, feature extraction from the sensor data will only yield the vertices 1, 2, 3, 4, 5, 6, 7, 8, 9. For the viewpoint shown, we must further assume the unavailability of angle measurements at vertices 1, 7 and 9.

We will subject the recognition of the polygonal object in Fig. 3.12 to the kind of hypothesis and verify approach depicted in Fig. 3.11, except that we will add constraints on the selection of model features at each node. We will first examine the possibility of using constraints derived from shape attributes.

3.5.2.1. Using Constraints Derived from Shape Attributes

Let's say each scene feature is characterized by the following set of shape attribute-value pairs:

$$SA = <sa_i, v_i>$$

In the absence of uncertainties, perhaps the most straightforward way of constraining the selection of model features in the matching process is to invoke only those model features whose sa_i values are the same as v_i. For the 2-D example, say that at a node in the search space the scene vertex 2 is under consideration. Now, a possible shape attribute for a 2-D vertex is the dihedral angle θ shown for one of the vertices in Fig. 3.13. Let's say the measured dihedral angle at the scene vertex 2 is θ_2. Given this shape information, it should be necessary to invoke only those model vertices whose dihedral angles, denoted here by θ_x, satisfy the constraint

$$\{x: |\theta_x - \theta_2| < \varepsilon\},$$

where ε represents the uncertainty in angle measurement. Given a judicious choice for ε, such a constraint might only invoke the model vertices b and k — a considerable improvement over having to compare, in the worst case, of course, — the scene vertex 2 with all the 16 model vertices.

A practical implementation of the above idea would require that we organize the model features according to the shape attributes. One could do so, for example, by sorting the model features by the values of the attributes. Then given a desired attribute value for a scene feature, the candidate model features could be retrieved by a binary search. Another way is to use an array with each array cell representing an interval of the attribute value; a model feature could then be assigned to an appropriate cell on the basis of the value of the attribute. The latter method would, in general, be more demanding on memory requirement, but the retrieval of candidate model features for a given scene feature attribute value would be more efficient.

Although, in some cases, it would certainly be possible to benefit from the ideas outlined in the previous two paragraphs, we have chosen to not use shape attributes for constraining the selection of model features. Our most important reasons are that the viewpoint independent shape attributes, for the most part, do not contain sufficient discriminatory power for adequately constraining the selection of model features; and the viewpoint dependent shape attributes are too prone to getting their values distorted by occlusion and, of course, the change in viewpoint.

For example, planarity of a surface is a viewpoint independent shape attribute. Now consider the example illustrated in Fig. 3.10 and assume that we are matching scene surfaces with model surfaces using planarity as a shape attribute. Clearly, all the model surfaces but $s2$ would become candidates for scene surface s_b, and there would be almost no gains in the computational complexity. On the other hand, a viewpoint dependent shape attribute, like the area of a surface would obviously be useless because the problems that could be caused by occlusion. For another illustration of the difficulties caused by using viewpoint dependent shape attributes, lets go back to the matching of vertices in Fig. 3.12. Although it may not seem so, the dihedral angle is viewpoint dependent, as, for example, evidenced by the vertices 1, 7 and 9. The dihedral angles at these vertices can not be measured from the viewpoint shown in Fig. 3.12 because of self-occlusion. Therefore, it would be impossible to use the most obvious shape attribute — the dihedral angle — for constraining search, as any of the nodes could suffer from self-occlusion, depending upon the pose of the object.

3.5.2.2. Using Constraints Derived from Relation Attributes

Let's say a scene feature S has the following relation attribute-value pair $<ra:\{S_1, S_2, \ldots, S_k\}>$, meaning that the scene features $S_1, S_2, \ldots, S_k$ are participating with S in the relation named *ra*. We will assume that of $S_1, S_2, \ldots, S_k$, the features S_1 through S_p, $p<k$, have already been matched with model features. Then we may consider only those model features for matching against S that enter into relation *ra* with the model features that match S_1 through S_p. More formally, a model feature M will be selected for matching with S provided one of the relation attribute for M is $<ra:\{M_{c(1)}, M_{c(2)}, \ldots, M_{c(p)}\}>$. Remember, the mapping function c gives us the correspondences between the scene features and the model features.

In the example of Fig. 3.12, assume that the scene vertices 4 and 5 have already been matched with the model vertices *e* and *f*, and that the scene vertex 6 is now under test for possible match with a model vertex. Since vertex 6 has relation attribute $<adjacent\_to: \{5, 7\}>$, a candidate model feature for matching with vertex 6 should posses the relation attribute $<adjacent\_to: \{f, *\}>$, where we have used an asterisk to

act as a place-holder for the yet unknown corresponding model vertex of vertex 7. For symmetric relations, such as *adjacent_to*, a search for those model features that satisfy the desired constraint can be easily conducted by examining the relations at the vertex *f*; we may thus conclude that the model vertex *g* is a candidate model feature for the scene vertex 6. For non-symmetrical relations, unless care is exercised in organizing the model feature with respect to their relations, in the worst case one may have to search through all the model features to determine those which satisfy the required constraints. However, even with such a search, one would gain from the subsequent savings in not having to match all the model features with a scene feature. In addition to having to search for the model features, there are other issues that play an important role in matching scene features with model features under relational constraints, especially when one also has to contend with the uncertainties associated with real data. Over the years, much has been done in this area and the reader is referred to [*S&H*–81, *K&et*–87, *W&L*–83] for futher details.

Although what we have said so far in this subsection may be construed as implying the appropriateness of relational constraints, the reader beware. We will now show that there can be situations when relational constraints may not help at all with the pruning of model features, and, further, in some cases they can lead to results that may be downright incorrect.

Going back to our example of Fig. 3.12, we just showed how the prior matches at scene vertices 4 and 5 help us constrain the search at vertex 6. One may similarly show that the matches at the vertices 5 and 6 help us with the selection of candidate model vertices at 7. Now, let's say, that the scene vertex 7 has been successfully matched with the model vertex *h*. Our next task is to find a list of candidate model vertices for the scene vertex 8. However, because of self-occlusion, there does not exist at vertex 8 the relation attribute <*adjacent_to* :..,7,..>. This means that prior matching history along the path being traversed in the search space would not help us at all at the scene vertex 8.

Now to show that relational constraints may lead to erroneous matches, consider the scene vertex 3, where we have the relation <*adjacent_to* :2,4>. We will assume that the vertex 4, being closest to the viewpoint line AB, has already been matched with model vertex *e*. The candidate model vertices for the scene vertex 3 must satisfy the relation <*adjacent_to* :*,*e*>, since *e* is the correspondent of 4 and since the vertex 2 has not been matched yet. This constraint would cause 3 to be matched with the model vertex d — an obviously incorrect result which would eventually cause an erroneous rejection of the model object.

3.5.2.3. Using Constraints Derived from Position/Orientation Attributes

If a scene feature possesses a position/orientation attribute-value pair $<la:v>$, then it follows from equations (3.4-a) and (3.4-b) that a potential candidate model feature must be characterized by the either or both of the following position/orientation attribute-value pairs

$$<la: \boldsymbol{R}^{-1}\cdot v> \qquad \text{if } v \textit{ is an orientation vector} \tag{3.5-a}$$

$$<la: \boldsymbol{R}^{-1}\cdot(v-t)> \quad \text{if } v \textit{ is a position vector} \tag{3.5-b}$$

where $\boldsymbol{R}$ and $\boldsymbol{t}$ are the rotation and translation components, respectively, of the tranformation Tr that takes the model object into the scene object. Clearly, an estimate of the tranformation Tr is required before a location constraint can be invoked.

We must again address the issue of how one might organize model features in order to efficiently invoke the location constraints. One approach would be to partition the space of all possible locations into cells and to assign model features to appropriate cells based on their locations. Since it takes three parameters to specify an orientation, two to specify the direction of the axis of rotation and another one to specify rotation around this axis, the location space for orientation vectors will consist of either the volume of a unit sphere, or, using the quaternion notation, the surface of a four-dimensional unit sphere. On the other hand, the location space for position vectors will be the 3-D Cartesian space.

While it would indeed be possible to use the position/orientation constraints in this manner to prune the list of candidate model features, difficulties arise in practice on account of the fact that it may not be possible to develop a unified organization of model features on the basis of position/orientation information, since some features may have only position attributes, other only orientation attributes, and still others both.

Fortunately, there is a way out of this impasse, by the use of principal directions defined in Section 4. For every feature, as shown in that section, we can derive its principal direction from either the position information or the orientation information. The principal direction can then be used, by the method discussed below, to organize the model features for efficient retrieval subsequently. In the rest of this subsection, we will use the 2-D example of Fig. 3.14 to introduce the idea of a *feature circle*, which is a means to organize, on the basis of their principal directions, the model features for the 2-D case.

For the 2-D example, we first compute the principal direction of each model vertex according to the definition in section 3-2. Since the space of direction vectors in 2-D space is a circle, we organize the model vertices along a unit circle as shown in Fig.

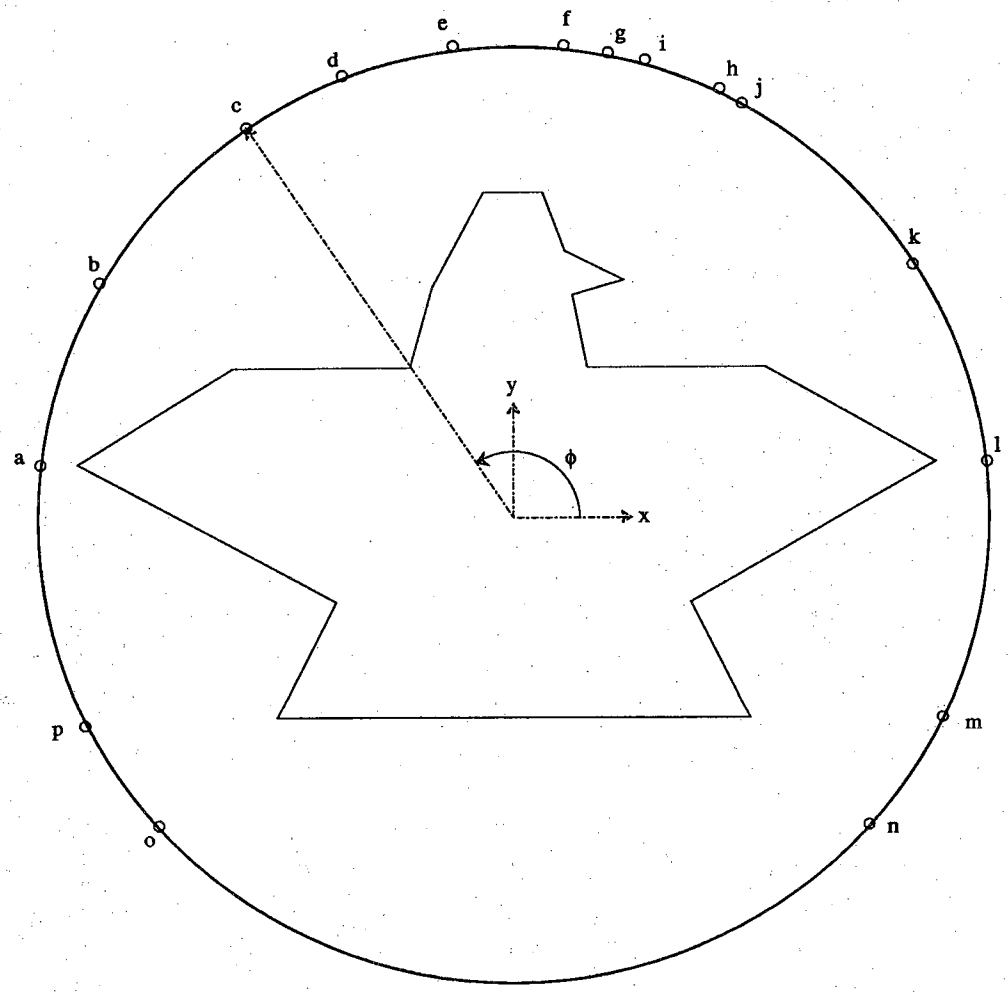


Figure 3.14. The vertices of the model polygon are pushed out to a unit circle which is the feature circle of the model.

3.14. This constitutes the feature circle for the model object. Suppose that the orientation $\boldsymbol{R}$ and position $\boldsymbol{t}$ of the scene polygon has been hypothesized by matching vertices 4 and 5 to model vertices e and f, respectively. Now, suppose we want to find the candidate model vertices for the scene vertex 3. Using equation (3.5-b), the position vector of a candidate model vertex for possible matching with the scene vertex 3 should be

$$\hat{\boldsymbol{p}} = \boldsymbol{R}^{-1} \cdot \boldsymbol{p}_3 - \boldsymbol{t}$$

The principal direction associated with this position vector is

$$\hat{\boldsymbol{\Phi}} = \frac{\hat{\boldsymbol{p}}}{|\hat{\boldsymbol{p}}|}$$

We can then access the feature circle of Fig. 3.14 and pull out those model features whose principal directions lie in the interval $[\Phi-\varepsilon\,,\ \Phi+\varepsilon]$, where ε depends upon the magnitude of uncertainty in the sensed data.

Of course, for the 3-D case, the organization of the model features would not be as simple as what is shown in Fig. 3.14, since the features would now have to be mapped onto the surface of a sphere on the basis of their principal directions. To handle the resulting complications, in Section 6, we will introduce the notion of a feature sphere which used a special indexing scheme for the tessellations on the surface of the sphere. The indexing scheme chosen reduces the complexity associated with finding the neighbors of a particular cell on the surface of a sphere.

3.5.2.4. Conclusion Regarding the Choice of Constraints

Before we present our complete feature matching strategy, we would like to summarize the conclusions that can be drawn from the preceding three subsections.

- Relatively speaking, shape attributes are not that useful for the purpose of selecting candidate model features because, when they are view independent, they often do not carry enough discriminatory power, and, when they are view dependent, they cannot be used for obvious reasons.
- When the pose transformation *Tr* is unknown, relation attributes can provide strong constraints for selecting model features; however, extraction of relation attributes may be too prone to artifacts.
- When the pose transformation *Tr* is given, the principal direction attribute, which can be derived from the position/orientation attributes, probably provides the best constraint for selecting model features. We use the adjective "best" to emphasize, in a qualitative sense admittedly, the fact that this attribute can be calculated in a fairly robust manner for most features, and, to emphasize its ability to provide strong discrimination amongst competing model features.

These conclusions form the foundation of our overall matching strategy, which we now present:

During hypothesis generation:

In this phase, we will use constraints on relation attributes to prune the list of model features. To get around the problems associated with exhaustive backtracking in the upper h levels of the search space shown in Fig. 3.11, *we will group immediately related model features into sets, to be called* Local Feature Sets *(LFS). Each LFS will be capable of generating a value for the transformation matrix Tr. The idea of using feature sets for constructing hypotheses about pose transformations is akin to the* local feature focus *idea used by Bolles and Cain [B&C–82] for the 2-D case and to the notion of* kernel features *used by Oshima and Shirai [O&S–83] for the 3-D case.*

During verification:

In this phase, we will use the principal direction constraint to select model features. For efficient retrieval on the basis of their principal directions, the model features will be organized on feature spheres.

In the next subsection, we will elaborate on the notion of Local Feature Sets for hypothesis generation. In the following subsection, we will then present a formal definition of the feature sphere data structure and present expressions for the complexity functions associated with our matching strategy.

3.5.3. Local Feature Sets for Hypothesis Generation

Ideally, an LFS is a minimal grouping of features that is capable of yielding a unique value for the pose transform which takes the model object into the scene object. The features in such a minimal grouping could, for example, correspond to one of the rows in Table 3.1.

More practically, it is desirable that the features in an LFS be in close proximity to one another, so that the probability of their being simultaneously visible from a given viewpoint would be high. In our implementation, we have found useful the following variation on the above idea, which seems to lead to particularly efficient hypothesis generation strategies for objects that are rich in vertices, such as the objects of Fig. 3.2. We allow our LFS's to be larger than minimal groupings and insist that each grouping contain a vertex and all the surfaces meeting at that vertex. [It would be equally easy to use edges in place of surfaces.] In Fig. 3.15, we first show labeled features for one of the objects of Fig. 3.2. For that object, the LFS's generated with this specification are

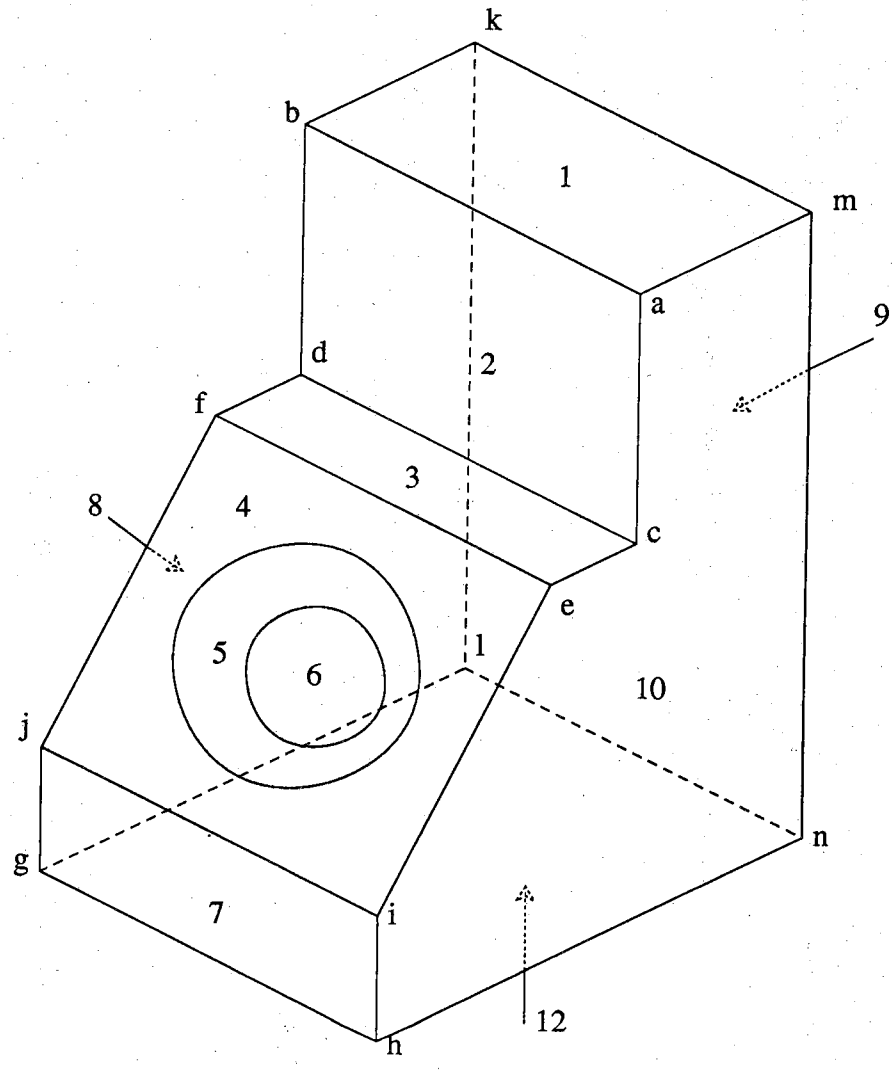


Figure 3.15. The labels of surfaces and vertices of the object in Fig 3.2 (a).

shown in Table 3.2. To explain the advantages of our approach, consider the LFS corresponding to the vertex *d* of the object in Fig. 3.15. The data record for this LFS will look like

---

Vertex c
flag: -1
xyz: ##
surfaces: 2 10 3
adjacent_vertices: a e d
edge_type: v v c

The flag value of -1 means that one of the three edges meeting at the vertex is concave. The variable xyz is instantiated to the coordinates of the vertex in the model coordinate system. In edge_type, v denotes convex and c concave, as there are two concave and one convex edges meeting at this vertex. This LFS subsumes at least three minimal feature groupings that are also capable of generating a unique value for the pose transform. For example, the grouping consisting of the surfaces 2 and 10, together with the coordinates of the vertex *a*, can yield a unique value for *Tr*. To answer the question why we use this particular construction for LFS's, we will first define a completely visible vertex.

In the scene, a vertex will be called *completely visible* if no occluding edges meet at the vertex. An example of a completely visible vertex is shown in Fig. 3.16-(a), while (b) shows the same vertex when it is not completely visible. Note that occluding edges in a range map are characterized by range discontinuities.

We believe that a completely visible vertex in a scene provides the strongest constraints for calculating the *Tr* associated with an object in a scene. Of course, theoretically, any two of the non-parallel surfaces coming together at the vertex, in conjunction with the vertex itself, are capable of specifying uniquely the *Tr* associated with a scene object. Therefore, theoretically at least, for the vertex shown in Fig. 3.16-(a), any two of the surfaces, together with the coordinates of the vertex, can yield *Tr*. However, in practice, it is difficult to calculate with great precision the position of the vertex itself, primarily because of the nature of discontinuities of some of the spatial derivatives at such a point. Therefore, our approach is that if a completely visible vertex can be found in a scene, it should immediately be used to calculate a *Tr*.

Of course, it is entirely likely that we may not find any completely visible vertices in a scene, meaning that for a vertex like the one shown in Fig. 3.16-(b), because of

Table 3.2 Local feature sets (LFS) of the object in Fig 3.15.

vertex	surfaces
a	1, 9, 2
b	1, 2, 8
c	2, 10, 3
d	2, 3, 8
e	3, 10, 4
f	3, 4, 8
g	7, 12, 4
h	7, 10, 12
i	4, 10, 7
j	4, 7, 8
k	1, 8, 9
l	9, 8, 12
m	1, 9, 10
n	10, 9, 12

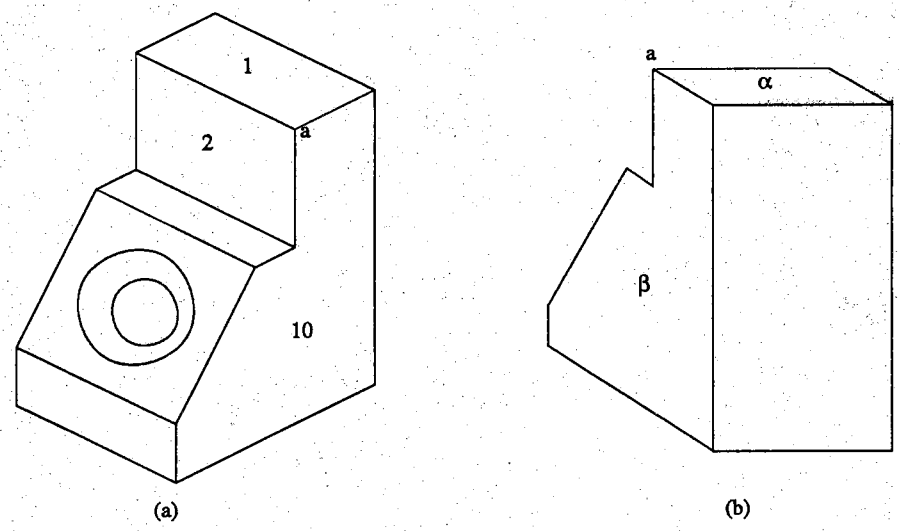


Figure 3.16. A completely visible vertex of a object in one view becomes partially visible in another view.

self-occlusion we may be able to see only two of the three surfaces. In such a case, the LFS for the vertex can still be used by assigning appropriate labels to the scene surfaces from the entries in the LFS. In general, if h surfaces meet at a vertex and only k of these are visible in a scene, then there are only h possibilities for matching the k scene surfaces, this happens because of the rotational adjacencies that have to be maintained. For example, again as illustrated in Fig. 3.16, the vertex a is formed by three surfaces 1, 10 and 2, if we see only two of the surfaces α and β as in (b), there are only three different labeling patterns for the two surfaces, namely,

$$\{(1\rightarrow\alpha,\ 10\rightarrow\beta,\ 2\rightarrow nil),\ (10\rightarrow\alpha,\ 2\rightarrow\beta,\ 1\rightarrow nil),\ (2\rightarrow\alpha,\ 1\rightarrow\beta,\ 10\rightarrow nil)\}.$$

In each of these patterns, the labels must maintain the same adjacencies that are in the model. Therefore, we can say that in matching k scene features with the h features of an LFS, the overhead is k, which is incurred in matching the k scene features with the potential correspondents from the LFS. Since this can only be done in h ways, the overall complexity associated with matching with an LFS is $O(h\times k)$.

Therefore, the complexity associated with generating hypotheses for an object which has N_{LFS} LFS's is $N_{LFS}\times O(h\times k)$. In practice, $N_{LFS} = O(m)$, where m is the total number of model features. Therefore, the overall complexity associated with generating all the hypotheses is

$$O(m\times h\times k) \quad = \quad O(m)$$

Before we conclude this subsection, we would like the reader to note that the gains achieved with the use LFS's as non-minimal feature groupings is at the cost of more complex flow of control during hypothesis generation. While with minimal groupings, it is possible to institute uniform control, with non-minimal groupings special cases must be handled separately depending upon how many of the features in an LFS can be matched with the scene features.

Also, we have said nothing about the mathematics of how to actually compute a Tr given that we have a match between some scene features and model features. In Appendix A, we provide formulations for estimating the transformation based on quaternion representation.

3.5.4. Feature Sphere for Verification

We want to organize model features of an object such that, given a candidate principal direction $\hat{\mathbf{\Phi}}$ computed from a scene feature, all the model features with the principal direction $\hat{\mathbf{\Phi}}$ can be accessed efficiently. Since a particular direction corresponds to a unique point on the surface of a unit sphere, similar to the way of organizing vertices on a circle in the 2-D example, a natural way is to record the model features on a unit

sphere as a function of their principal directions. We shall call such a sphere a feature sphere. There can, of course, be multiple number of features corresponding to a given point on the feature sphere, especially if more than one feature class is used for describing models. In our experience, programming becomes more efficient if a separate feature sphere is used for each class, meaning that we represent all the primitive surface features on one sphere, all the primitive edge features on another sphere, and all the primitive point features on yet another. Fig. 3.17 shows the vertex feature sphere and the surface feature sphere for the 3-D model object in Fig. 3.10.

After a hypothesis about the object's location Tr is generated, we want to verify or reject the hypothesis by matching the rest of the scene features to model features under Tr. Of the different scene features which will be used for verification, consider a scene feature S. According to equations (3.4-a) and (3.4-b), a model feature that is a candidate for matching with the scene feature S should be characterized by a principal direction $\hat{\mathbf{\Phi}}$ that is equal to the following for the different types of S.

- If S is a primitive surface (spherical surface excluded) or a primitive curve:

$$\hat{\mathbf{\Phi}} = \boldsymbol{R}^{-1} * \boldsymbol{v}(S), \tag{3.6-a}$$

  where $\boldsymbol{R}$ is the rotation component of Tr, and $\boldsymbol{v}(S)$ is the orientation direction of feature S, defined similarly as its principal direction but with respect to the world coordinate system.

- If S is point feature or a spherical surface:
  Let $\boldsymbol{p}(S)$ be the position vector of feature S with respect to a world coordinate system.

$$\hat{\boldsymbol{p}} = Tr^{-1} * \boldsymbol{p}(S) = \boldsymbol{R}^{-1} * (\boldsymbol{p}(S) - \boldsymbol{t}) \; ; \tag{3.6-b}$$

$$\hat{\mathbf{\Phi}} = \frac{\hat{\boldsymbol{p}}}{|\hat{\boldsymbol{p}}|} .$$

  where $\boldsymbol{t}$ is the translation component of Tr.

As previously mentioned, principal direction provide a very strong constraint for selection of candidate model features, i.e. each candidate principal direction computed from equation (3.6-a) or (3.6-b) will lead to a small number of candidate features. This is especially true for point features as we observed in the 2-D example in which a candidate principal direction addresses to only one candidate model vertex. For primitive surface or primitive edges, that number may depend on the configuration of object surfaces. In general, we may assume that the principal directions of a model's features are randomly distributed over the unit sphere. Although, the probability of any two features occupying the same spot on the unit sphere will be very low, for the sake of argument we may assume that on the average there will be k features for each principal direction,

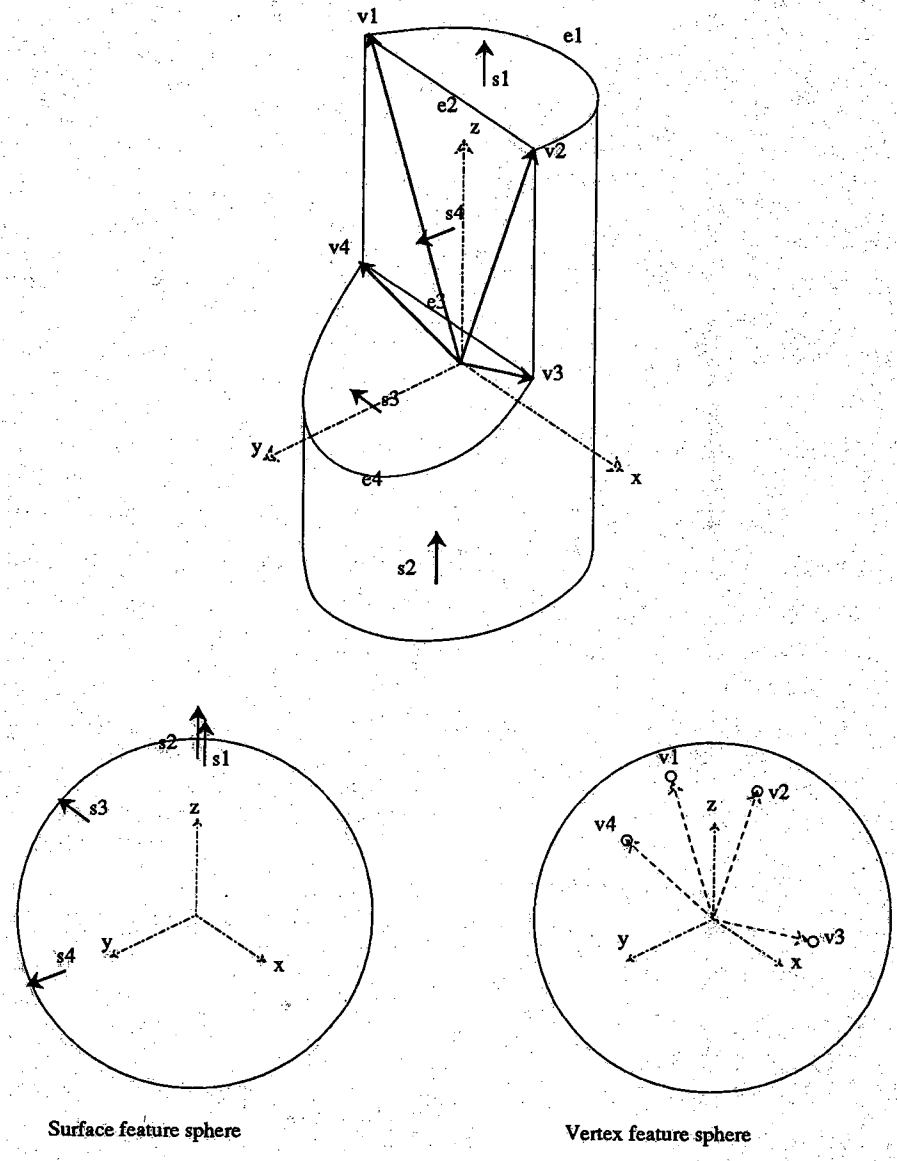


Figure 3.17. The surface and vertex feature spheres of the model object.

where $k \ll m$. Then the worst case time complexity for matching for verification will be $O(n \times k) = O(n)$. When combined with the complexity of hypothesis generation, as discussed in Section 5.3, this implies an overall complexity level of $O(mn)$. Since, $m = O(n)$, we can then conclude that the overall complexity with our approach for single object recognition to be $O(n^2)$.

In the next section we will present the implementation of feature sphere in computer in detail. It is interesting to note that if a model object is a convex polyhedron then its surface feature sphere representation is equivalent to its EGI (extended Gaussian image) [*Ho*−84], [*Ik*−83], and if a primitive curved surface is allowed to be added to a polyhedron then the surface feature sphere is similar to CSG-EESI representation proposed by Xie and Calvert [*X&C*−88]. In addition, if every surface point is regarded as a point feature, then the point feature sphere of a star-shape object is equivalent to the well-tessellated surface representation proposed by Brown [*Br*−79].

3.6. A Data Structure for Representing Feature Spheres

In order to implement feature spheres in a computer, we first need to tessellate the sphere and then create an appropriate data structure for representing the tessellations. In our case, each cell on the sphere will be represented by its center point, and the purpose of the data structure will be to allow us to efficiently access these points. In what follows, we will use the term *tessel* to refer to both a cell created by tessellating a sphere and to the central point of the cell. Before a data structure can be created for representing the tessels, we must bear in mind the following two kinds of operations that will be performed on the data structure for the purposes of feature matching.

First, during the model building process model features must be assigned to their respective tessels on the bases of their principal directions. Clearly, it is unlikely that the direction corresponding to one of the tessels would correspond exactly to that of a feature. For a given model feature, we must, therefore, locate the nearest tessel. In other words, we need a tessel assignment function, which will be denoted by $L(\Phi)$, that should return the label of a tessel to which a model feature of principal direction Φ is assigned.

Second, given a scene feature S in the verification process, we want to examine whether there is a corresponding model feature with direction $\Phi^* = Tr^{-1}(\Phi(S))$ in the model under consideration. Assuming the hypothesis is correct, idealy, we should be able to find such a model feature at $L(\Phi^*)$ on the feature sphere of the model. However, due to noise and other artifacts associated with the estimation of Tr, Φ^* will only be accurate to within some uncertainty interval. This directional uncertainty associated with Φ^* can be expressed as a cone whose axis is the computed direction itself, as

shown in Fig. 3.18. This implies that potential model features for matching with S should be all those that are within this cone. If we could assume the error processes associated with the uncertainties in Φ^* to be of zero-mean type, from within the cone one would first select that feature which was closest to $L(\Phi^*)$, and, if that match were to fail, select the next closest, etc. Clearly, this is a breadth first search rooted at $L(\Phi^*)$, and the depth of search (the farthest neighbors to examine) should correspond to the maximal allowable direction uncertainty.

It should be obvious that for implementing the above strategy for the selection of model features, we need a function that would be capable of directly accessing the immediate neighbors of a given tessel; we consider two tessels to be neighbors if they share a common edge in the tessellation. This function will be called *find-neighbors* function and will be denoted by N. So, we want

$$N(L_0) = \{L_1, L_2, \ldots, L_k\}$$

where $L_1, L_2, \ldots, L_k$ are the labels of the immediate neighbors of the tessel labeled L_0.

3.6.1. Previous Approaches To Data Structuring of Sphere Tessellations

In their work on EGI representation, Horn [*Ho*–84] and Ikeuchi [*Ik*–83] have discussed a hierarchical tree structure for representing a tessellated sphere based on icosahedron or dodecahedron. A drawback of this hierarchical data structure is that the adjacency relationship between neighboring tessels is not preserved. To get around this difficulty, Fekete and Davis [*F&D*–84] used a fairly complex labeling scheme, in this scheme each tessel is labeled by the pathname of its corresponding node in the tree. The neighbors of a tessel within one of twenty main icosahedral triangles are found by examining the pathname of the tessel, symbol by symbol, and synthesizing the pathnames of its neighboring tessels by the use of complicated state-transition rules and lookup tables. This procedure requires at least $O(n)$ operations, where n is the number of levels in the hierarchy. When the neighbors lie in an adjacent triangle, a different procedure is needed. Korn and Dyer [*K&D*–87] have also proposed a data structure for a tessellated sphere with a fixed number of subdivision levels. Twenty one-dimensional arrays, each of size 4^n, are used to represent the sampling points on the sphere, which implies that a sampling point is labeled by a number from 0 to 4^n–1. Their find-neighbors algorithm is essentially the same as that of Fekete and Davis.

In this section we will present a new data structure for representing a tessellated sphere based on icosahedron. Its main merit is that logical adjacency between elements of the data structure corresponds to physical adjacency between sampling points on the sphere. We will show that the neighbors of a given tessel can be found with a constant

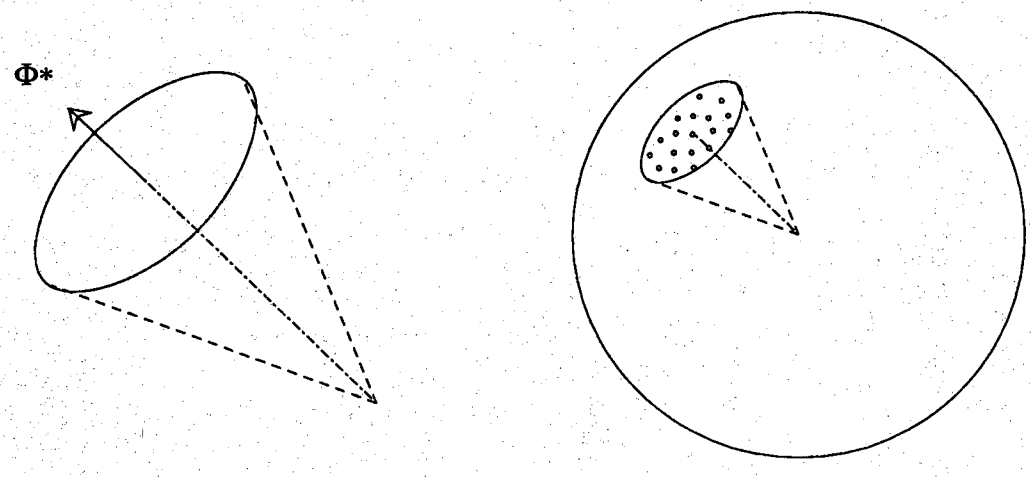


Figure 3.18. A cone represents directional uncertainty of the computed direction $\mathbf{\Phi}^*$, and the sampling points on the sphere lie within the uncertainty cone.

time complexity algorithm, regardless of sampling resolution. Furthermore, by using the find-neighbors function, the tessel-assignment function L can be implemented efficiently, too.

3.6.2. Tessellating a Unit Sphere

In this subsection, we will present the tesselations on which our data structuring is based. Subsequently, it should become evident to the reader that the regularity of the neighborhood patterns in the tesselations used allows us to devise a simple scheme for neighbor finding. However, first we will quickly review the considerations that go into the design of tesselations.

When a sphere is tesselated into cells, ideally we would like the cells to be symmetrical, be identical in shape, and possess equal areas; also, ideally, the tesselation scheme should maintain these attributes over a wide range of cell resolutions. However, it is well known that a tesselation scheme with these attributes does not exist. The best one can do is to use the techniques of geodesic dome constructions [*Ke*–76], [*Pu*–76]; these techniques lead to triangular cells that are approximately equal in area and shape. The geodesic tesselations are obtained via the following three steps:

(1) Chose a regular polyhedron, which usually is an icosahedron or a dodecahedron, and inscribe it in a sphere to be tessellated. If a dodecahedron is used, each of its pentagonal faces is divided into five triangular faces around its center to form a pentakis dodecahedron. Thus each face of the regular polyhedron will be a triangle.

(2) Subdivide each triangular face of either the icosohedran or the pentakis dodecahedron into subfaces by dividing each edge of a triangular face into Q sections, where Q is called the frequency of geodesic division. As a result, each triangular face is divided into Q^2 triangular subfaces. Finer resolution can be obtained simply by increasing the frequency of geodesic division. Usually, Q is a power of two.

(3) Project the subdivided faces onto the sphere. In order to make the projected triangle sizes more consistent, the edges of the triangles should be divided into sections such that each section subtends the same angle at the center of the sphere; as a consequence the lengths will be the same for the edge sections after they are projected onto the sphere [*Ke*–76].

To generate the tesselations used by us, we start out by implementing the above approach with an icosahedron. The geodesic polyhedron thus produced contains $20\,Q^2$ cells and $10\,Q^2$+2 vertices. Fig 3.19 shows an icosahedron and a tessellated sphere based on the icosahedron with frequency Q=4 of geodesic division.

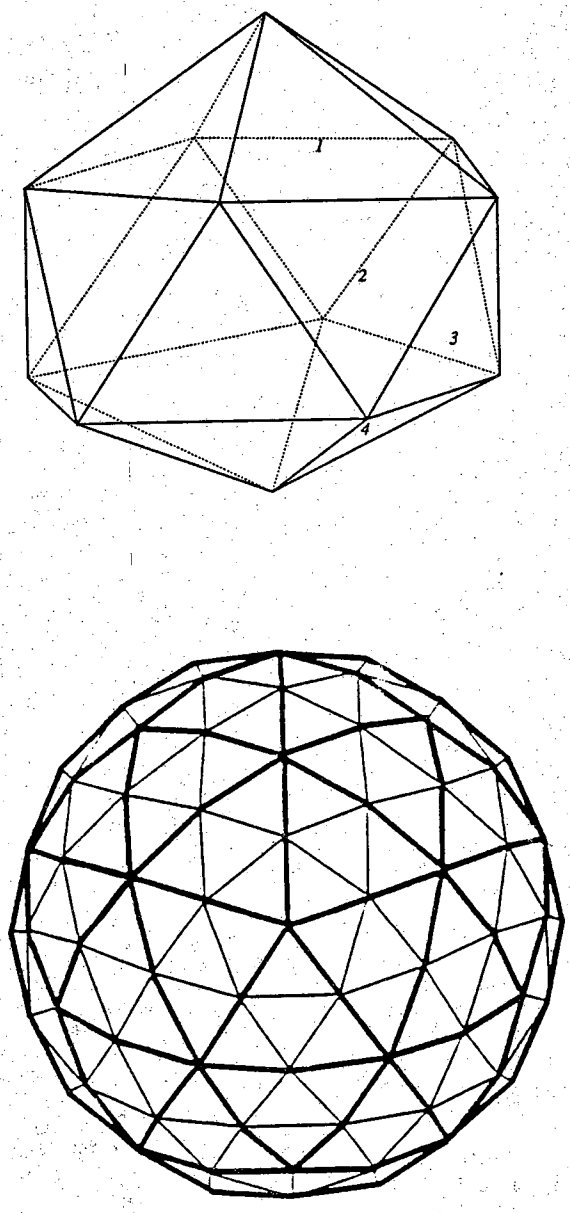


Figure 3.19. An icosahedron and a tessellated sphere based on the icosahedron.

Our next step is to construct a dual of the geodesic polyhedron produced by the above method. Note the dual of a polyhedron is also a polyhedron whose vertices correspond to the faces of the original polyhedron and whose faces correspond to the vertices of the original polyhedron. For example, a pentagon is the dual of an icosahedron. The dual geodesic polyhedron thus produced consists of $10\,Q^2+2$ cells, of which $10\,(Q^2-1)$ are hexagonal and the rest 12 are pentagonal. The 12 pentagonal cells of the dual polyhedron correspond to the 12 vertices of the original icosahedron. This dual polyhedron is then projected onto the unit sphere to produce the desired tesselations. As shown in Fig. 3.20, the center of each tessel serves as the sampling point for that tessel for the purpose of discretization. It is important to note that these sampling points correspond to the vertices of the original polyhedron, the one before the dual was constructed. This fact will prove to be most important to our derivations later.

As illustrated in Fig. 3.20, our tessels can be either pentagonal or hexagonal, the former has five neighbors, and the latter six. The average area of a tessel is given by $4\pi/(10\,Q^2+2)$. The radial angle between adjacent sampling points, which is an indication of sampling resolution, can be roughly estimated by

$$atan(2) \,/\, Q$$

where $atan(2)$ is the angular spread of an icosahedron's edge.

3.6.3. A Spherical Array for Representing the Tessellation

We will now present a spherical array data structure for the computer representation of the tesselation. This array will lead to easy and efficient implementations of the find-neighbors function N and tessel-assignment function L. The data structure will be constructed by first noting that the vertices of the geodesic polyhedron are the sampling points of the dual polyhedron; flattening out, as shown in Fig 3.21, the 20 triangular faces of the underlying icosahedron; and, finally, paralleling the development of the geodesic polyhedron on this flattened form. The flattened-out representation of the icosahedron consists of five connected parallelograms, each of them consisting of four triangular faces, each triangular face corresponding to one of the 20 triangles on the icosahedron. Each parallelogram is subdivided into $4\times Q^2$ triangular cells using Q for the frequency of geodesic division (Fig. 3.22). The vertices shown in Fig. 3.22 correspond to the vertices of the geodesic polyhedron, and also, therefore, to the sampling points of our tesselation for the case of $Q = 4$. The flattened-out representation, of which Fig. 3.22, is an example, will be referred to as the *spherical array*.

Each parallelogram in a spherical array consists of $(Q+1)\times(2Q+1)$ vertices. Obviously, the vertices in each parallelogram separately could be represented by a two dimensional array; however, note that the vertices on the border of the parallelograms

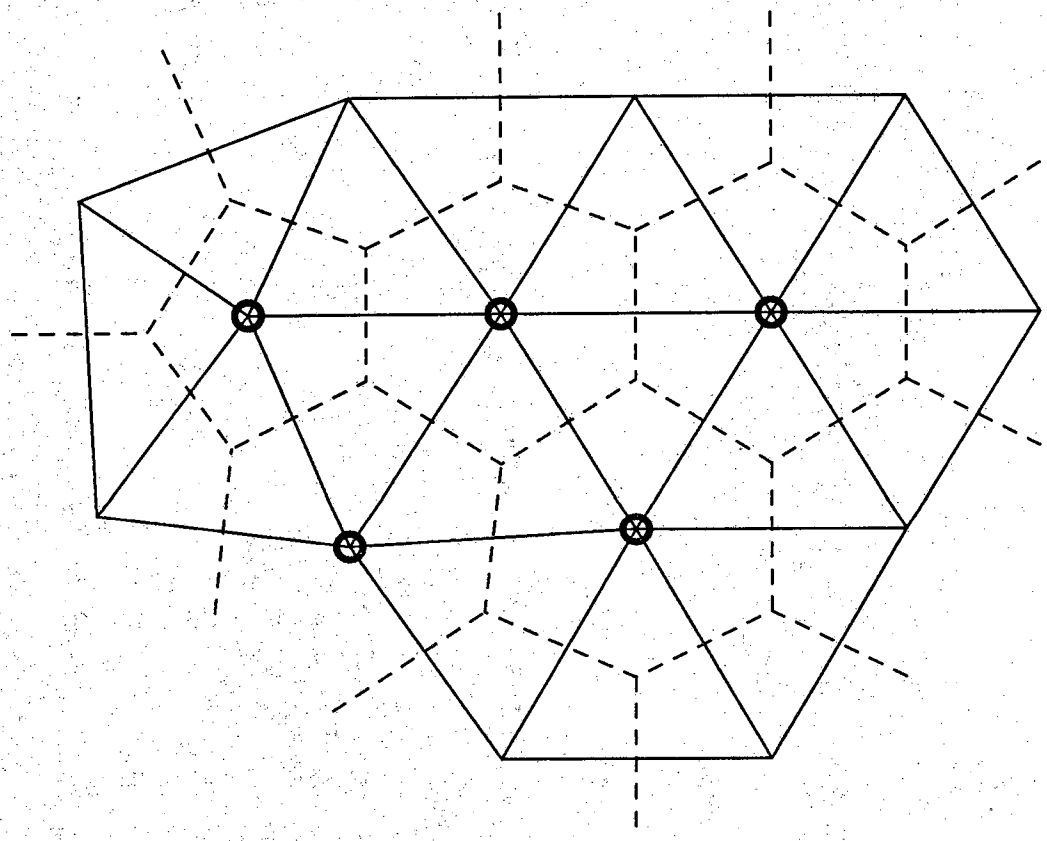


Figure 3.20. The dash lines indicate part of the dual polyhedron; the sampling points are defined at the vertices of the original polyhedron outlined by solid lines.

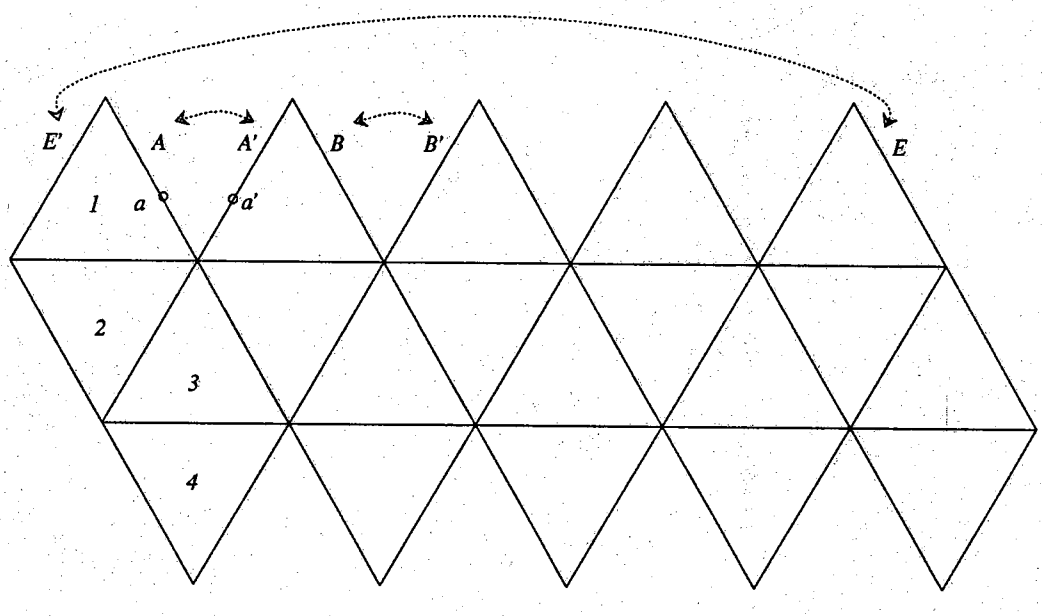


Figure 3.21. The original icosahedron is flattened out to form five connected parallelograms, each of them consisting of 4 triangular faces.

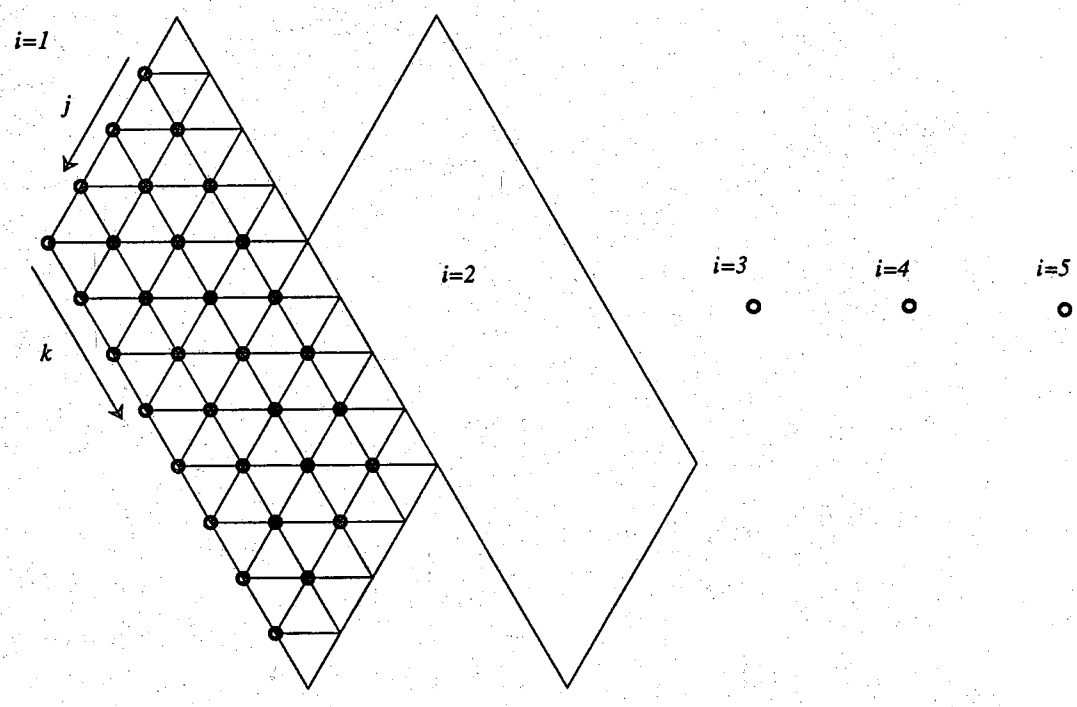


Figure 3.22. The assignment of the elements of a $Q \times 2Q$ array on a parallelogram.

are shared, meaning, for example, that the vertices a and a' on the edges A and A', respectively, are really the same vertex on the geodesic polyhedron. In other words, before the icosahedron is unfolded to form the spherical array, edge A is connected to edge A', edge B to edge B', edge E to edge E', and so on (Fig. 3.21).

The fact that each border vertex should appear only once in an overall indexing scheme for the vertices in a spherical array implies that the size of the index array for representing each parallelogram need only be $Q \times 2Q$. For example, for the case shown in Fig. 3.22, each parallelogram need only be represented by a 4×8 array. The assignment of array indices for the parallelograms is depicted in Fig 3.22 for the $Q = 4$ case. The index i specifies a parallelogram and the indices j and k specify a vertex within the parallelogram. Clearly, we have five $Q \times 2Q$ arrays, for a total of $10 \times Q^2$ indexed points on the spherical array, this number being two less than the total $10 \times Q^2 + 2$ vertices on the geodesic polyhedron. The two missing vertices correspond to the two common vertices of the five parallelograms, one at the top and the other at the bottom. We shall allocate two additional distinguished sets of indices to represent these two vertices and referred to them as the the zenith and the nadir (see section 6.3.2 for explanation) of the tessellated sphere.

The proposed indexing implies the following ranges for i, j and k:

$$[i,\ j,\ k] \qquad\qquad 1 \le i \le 5,\quad 1 \le j \le Q,\quad 1 \le k \le 2Q.$$

The zenith and the nadir are assigned the distinguished indices [0, 0, 0] and [−1, 0, 0], respectively.

3.6.3.1. The Find-Neighbors Function

As pointed out before, the simplicity of the proposed data structure lies in its preserving the physical adjacencies between the tessels. We will now show that simple relationships exist that yield a tessels's neighbors, regardless of the location of the tessel, and, more important, regardless of whether the tessel possesses six or five neighbors. Most tessels posses six neighbors, except for the 12 that correspond to the 12 vertices of the original icosahedron, each of latter type possessing five neighbors only. In general, the six neighbors of a tessel $[i,\ j,\ k]$ that is not on the border of any of the five parallelograms are given by:

$$\begin{cases} [i,\ j,\ k+1], \\ [i,\ j+1,\ k], \\ [i,\ j+1,\ k-1], \\ [i,\ j,\ k-1], \\ [i,\ j-1,\ k], \\ [i,\ j-1,\ k+1]. \end{cases} \tag{3.7}$$

Therefore, for the above set of indices to give us the neighbors, the indices j and k must obey the constraints $1<j<Q$ and $1<k<2Q$. If also used to find the neighbors of a border tessel, some of the above indices would take out of range values, implying that those neighboring tessels are vertices shared by another parallelogram and should really be assigned to the array for that parallelogram. To convert the out-of-range labels to the legitimate ones, we apply the following substitution rules:

$$\left.\begin{array}{ll} [i,\ j,\ 0] \Rightarrow [i-1^{m5},\ 1,\ j] & j = 1,\ ...,\ Q \\ [i,\ Q+1,\ k] \Rightarrow [i-1^{m5},\ 1,\ Q+k] & k = 0,\ ...,\ Q \\ [i,\ Q+1,\ k] \Rightarrow [i-1^{m5},\ k-Q,\ 2Q] & k = Q+1,\ ...,\ 2Q \\ [i,\ 0,\ k] \Rightarrow [i+1^{m5},\ k-1,\ 1] & k = 2,\ ...,\ Q \\ [i,\ 0,\ k] \Rightarrow [i+1^{m5},\ Q,\ k-Q] & k = Q+1,\ ...,\ 2Q+1 \\ [i,\ j,\ 2Q+1] \Rightarrow [i+1^{m5},\ Q,\ j+Q+1] & j = 1,\ ...,\ Q-1 \\ [i,\ 0,\ 1] \Rightarrow [0,\ 0,\ 0] & \\ [i,\ Q,\ 2Q+1] \Rightarrow [-1,\ 0,\ 0] & \end{array}\right\} \tag{3.8}$$

for $i = 1,\ ...,5$, where $i^{m5} = (i-1) mod(5) + 1$

Except for the zenith and the nadir tessels, it can be verified that equations (3.7) and (3.8) are also applicable to 10 of the 12 five-neighbor tessels. At a five-neighbor tessel, two of the six labels returned by equation (3.7) will turn out to be identical after applying the substitution rules in (3.8). The five neighbors for the zenith and the nadir are

$$[i,\ 1,\ 1] \quad i = 1, \ldots, 5 \quad \text{and}$$

$$[i,\ Q,\ 2Q] \quad i = 1, \ldots, 5$$

respectively.

The following two examples will illustrate the neighbor finding scheme described above. The example are for the case of $Q = 4$.

Example 1:

Find the neighbors of tessel [1,3,1]

$$[1,3,1] \rightarrow \begin{cases} [1, 3, 2] \\ [1, 4, 1] \\ [1, 4, 0] = [5, 1, 4] \\ [1, 3, 0] = [5, 1, 3] \\ [1, 2, 1] \\ [1, 2, 2]. \end{cases} \rightarrow \begin{cases} [1, 3, 2] \\ [1, 4, 1] \\ [5, 1, 4] \\ [5, 1, 3] \\ [1, 2, 1] \\ [1, 2, 2] \end{cases}$$

Example 2:

Find the neighbors of tessel [2,4,5]

$$[2,4,5] \rightarrow \begin{cases} [2, 4, 6] \\ [2, 5, 5] = [1, 1, 8] \\ [2, 5, 4] = [1, 1, 8] \\ [2, 4, 4] \\ [2, 3, 5] \\ [2, 3, 6] \end{cases} \rightarrow \begin{cases} [2, 4, 6] \\ [1, 1, 8] \\ [2, 4, 4] \\ [2, 3, 5] \\ [2, 3, 6] \end{cases}$$

It is worth noting that [2,4,5] happens to be a vertex of the original icosahedron and has only five neighboring tessels, exactly what the rules returned.

3.6.3.2. Directions of Sampling Points

In order to specify the tessel-assignment function, we will need formulas for the directions of the tessels, meaning the directions associated with each of the vertices on the spherical array. For that purpose, we will take advantage of the symmetry of the icosahedron and use a sphere-centered coordinate system whose positive z axis passes the zenith at ([0, 0, 0]) and whose z-x plane passes an icosahedral vertex [1, Q, 1] as shown in Fig 3.23. The direction of each tessel, denoted by $\Phi[i, j, k]$, in this coordinate system will be expressed in terms of the longitude and latitude angles (ϕ, θ). Because of the symmetry of the icosahedron, we have

$$\begin{cases} \theta[i, j, k] = \theta[i-1, j, k] \\ \phi[i, j, k] = (\frac{2\pi}{5} + \phi[i-1, j, k]) \ mod \ (2\pi) \end{cases} \tag{3.9}$$

for $i = 2, ...,5,\ j = 1, ...,Q,\ k = 1, ...,2Q$.

therefore we only need to compute the direction of the tessels in the first parallelogram

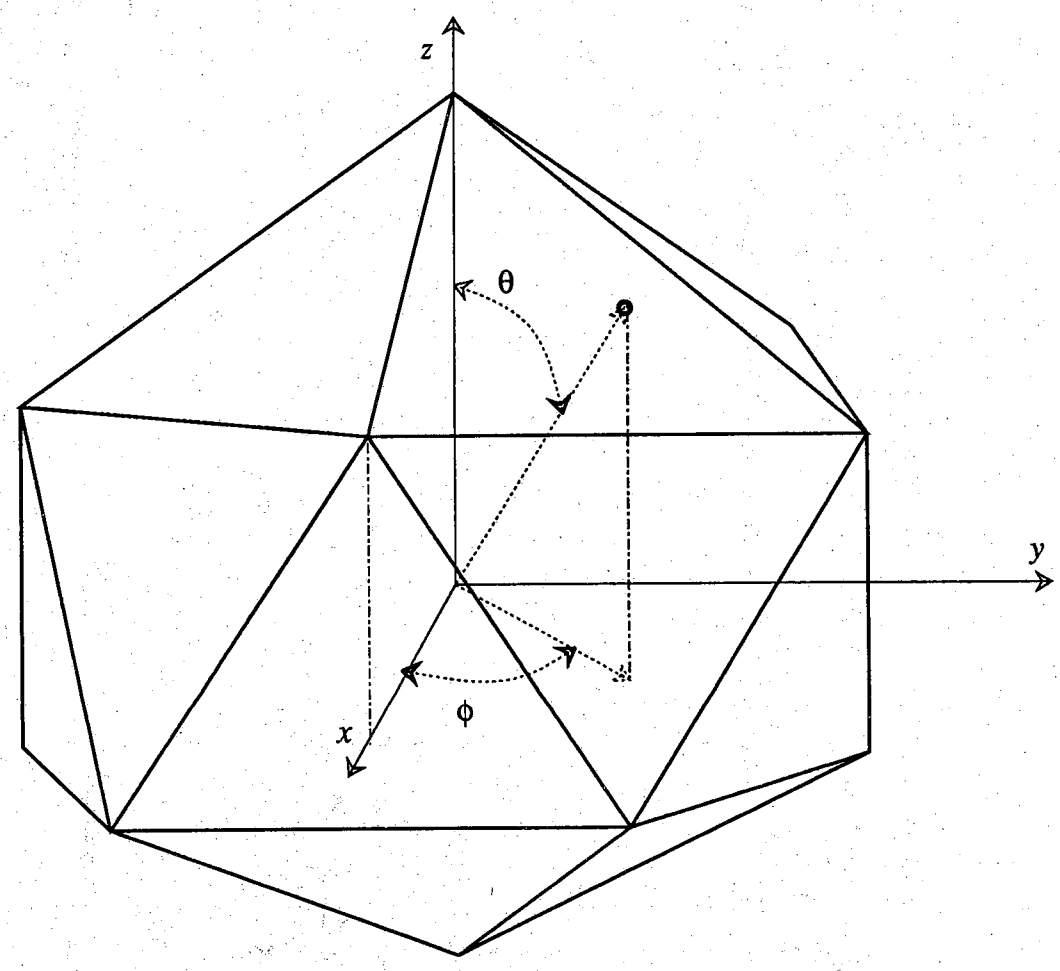


Figure 3.23. A spherical coordinate system defined on the original icosahedron.

(array).

It is easy to see that the direction of the five vertices of the first parallelogram are:

$$\Phi[1,\ 0,\ 1]^{*} = (0,\ -)$$

$$\Phi[1,\ Q,\ 1] = (atan(2),\ 0)$$

$$\Phi[1,\ 0,\ Q+1]^{*} = (atan(2),\ \frac{2\pi}{5})$$

$$\Phi[1,\ Q,\ Q+1] = (\pi - atan(2),\ \frac{\pi}{5})$$

$$\Phi[1,\ 0,\ 2Q+1]^{*} = (\pi - atan(2),\ \frac{3\pi}{5})$$

$$\Phi[1,\ Q,\ 2Q+1]^{*} = (\pi,\ -)$$

Recall that in the derivation of the geodesic polyhedron, we subdivided each edge of the triangles of the inscribed icosahedron into Q sections of equal radial angle. When $Q=2^r$, the result is equivalent to recursively subdividing r times a triangle into four triangles. Therefore, we can compute the direction of a new tessel by taking the averages of the known directions of the two tessels which are the end-points of the edge whose division led to the formation of the new tessel. This procedure can be applied recursively to compute the direction of every tessel. As an example, the three tessels which are the midpoints of the three edges of the upper triangle have directions:

$$\Phi[1,\ \frac{Q}{2},\ 1] = Mid(\Phi[1,\ 0,\ 1], \Phi[1,\ Q,\ 1])$$

$$\Phi[1,\ \frac{Q}{2},\ \frac{Q}{2}+1] = Mid(\Phi[1,\ 0,\ Q+1], \Phi[1,\ Q,\ 1])$$

$$\Phi[1,\ 0,\ \frac{Q}{2}+1] = Mid(\Phi[1,\ 0,\ Q+1], \Phi[1,\ 0,\ 1]).$$

Here $Mid(\Phi_1, \Phi_2)$ means to take the average direction of the two direction on the unit sphere. To save runtime computation, we may pre-compute the direction for all the tessels and store them in a lookup table.

3.6.3.3. The Tessel-Assignment Function

Given a particular direction Φ^*, its corresponding tessel in the spherical array should be the one whose direction is closest to Φ^*. The function L^* will return the indices of this tessel.

---

* Note that these labels are not legitimate in the spherical array data structure; we use them just to make the derivations clearer. The legitimate versions of these labels can be obtained by using the substitution formulas in equation (3.8).

$$\Phi[L^*] \cdot \Phi^* = \max_L (\Phi[L] \cdot \Phi^*)$$

The finding of the tessel L^* would thus involve a search process for the maximal dot product. Because the dot product is a monotonically increasing function toward the desired tessel, a local maximum must also be the global maximum. The local maximum can be found by an iterative climbing method from any tessel guessed initially. Since a good initial guess can reduce considerably the computations required to reach the maximum, we have provided in Appendix B

a linear approximation that translates a given Φ into a triple (i,j,k). Since the approximation has proved to be fairly good, the resulting indices are quite close to their actual values. Starting with these indices, one can then find the actual ones by conducting local search, as depicted by the following algorithm.

```
assign_tessel( Φ*) {
    L^0 = get_initial_guess( Φ*)
    L* = get_closer(L^0, Φ*)
    return L* }

get_closer(L, Φ*) {
    among all  L' in N(L)
        find L̂ which maximizes (Φ(L') · Φ*)
    if (Φ(L̂) · Φ* > Φ(L) · Φ*)
        get_closer( L̂, Φ*)
    else
        return L }
```

3.6.3.4. Building Feature Spheres on the Spherical Array

Note that since a feature is described by sets of attributes, a frame structure is used to store the attribute-value pairs. Each such frame structure is identified by a pointer which is stored at the corresponding tessel in the spherical array. The tessel address, as represented by the indices i,j, and k, is computed by applying the tessel-assignment-function to the principal direction of the future. It may happen that two or more neighboring features, neighboring in the sense of their possessing nearly identical principal directions, may have their points assigned to the same tessel. This conflict can be resolved by recording in the first registered feature a list of pointers for the features that share the same tessel address.

3.7. Recognition of Objects in the Presence of Occlusions

The discussion presented so far could be used directly for the recognition of single isolated objects. However, our main interest in 3D-POLY lies in recognizing objects under occluded conditions, as would be the case when the objects are presented to the sensory system in the form of heaps.

In general, when the range images to be interpreted are of scenes containing piles of overlapping objects, one has to contend with the following two problems: 1) The number of features extracted from a scene will usually be very large; and, 2) since different objects may be made of similar features, it would generally not be possible to set up simple associations between the scene features and the objects. To get around these problems in dealing with multiple object scenes, researchers previously have either performed object segmentation by exploiting range discontinuity information [*F&et*–88], or have used a *model driven** approach to group together scene features belonging to single objects [*F&H*–86], [*B&H*–86]. However, the former approach usually fails to work especially when the juxatopositions of multiple objects are such that there are no range discontinuities between them; and the latter is inefficient for reasons described in Section 2.

We will now present a data-driven approach for aggregating from a complex scene features belonging to single objects. The cornerstone of our approach is the idea that only physically adjacent scene features need be invoked for matching with a candidate object model. For this purpose, the notion of physical adjacency will be applied in the image space as opposed to the object space, implying, for example, that two surface regions sharing a common boundary, even if it is a jump boundary, will be considered adjacent to each other. Using this idea, we will now describe the complete method:

The algorithms uses two sets, UMSFS and MSFS, the former standing for the unmatched scene feature set and the latter for matched scene feature set. Initially, the algorithm assigns *all* the scene features to the set UMSFS. The process of object recognition starts with a local feature set (LFS) extracted from the UMSFS. The matching of this scene LFS with a model LFS generates a hypothesis about object identity and a pose transformation. The features in the scene LFS are then taken off from the UMSFS and assigned to MSFS; note that MSFS keeps a record of all the scene features matched so far with the current candidate model. Then during the verification stage, only those scene features in the UMSFS that are adjacent to the features in MSFS are selected for matching with the candidate model. During the verification state, if a UMSFS feature

*To be contrasted with the *data driven* procedure to be described in this section.

does match the candidate model feature, the scene feature is taken out of the UMSFS and added to the MSFS; otherwise the feature is marked as tested under the current hypothesis and left in the UMSFS.

The verification stage terminates when MSFS stop growing. Once the verification process terminates, the algorithm determines whether or not the features in the MFS constitute enough evidence to support the hypothesis on the basis of some predefined criterion. This criterion may be as simple as requiring a percentage, say, 30 %, of model features to be seen in the MSFS; or, as complicated as requiring a particular set of model features to appear in the MSFS; or, at a still more complex level, some combination of the two. If a hypothesis is considered verified, the features currently in MSFS are labeled by the name of the model and taken out of further consideration; otherwise, the hypothesis is rejected and every feature in the MSFS is put back into the UMSFS and the process continued with a new LFS. The entire process terminates after all the LFS's have been examined. The algorithm is presented below in pseudo C language:

```
Interprete_scene (I) {
    extract feature set {S} from I
    UMSFS = {S}
    while ( there exists a local feature set LFS_s in UMSFS )
        for each LFS_m in the model library
            if ( LFS_s matches LFS_m ) {
                estimate Tr by matching LFS_s with LFS_m
                candidate model O_m is the model corresponding to LFS_m
                MSFS = LFS
                Verify (O_m, MSFS, UMSFS, Tr) } }

Verify (O_m, MSFS, UMSFS, Tr){
 tagfor each untested S_i in UMSFS adjacent to MSFS {
        compute prinicipal direction Φ of Tr^-1(S_i)
        for each M_j registered in the neighborhood of L(Φ) on the feature sphere of O_m
            if ( Tr^-1(S_i) matches M_j) {
                add S_i to MSFS
                go to tag }
            else
                mark S_i tested
    if ( MSFS satisfies the recognition criterion ) {
        UMSFS = UMSFS - MSFS
        label every S in MSFS by the name O_m
        write_result (O_m, Tr)
```

```
        return (true) }
    else
        return (false) }
```

In our current implementation of this algorithm, the recognition criterion requires that at least 33% of a candidate model's features be present in the MSFS for a hypothesis to be considered valid. Note that the acceptance threshold can be no greater than 50% for most objects, especially those that have features distributed all around, since from a single viewpoint only half of an object will be scene. Therefore, 50% is a loose upper limit on the acceptance threshold. On the lower side, the threshold can not be set to be too low, since that would cause misrecognition of objects. We have found 33% to be a good compromise.

3.8. Experimental Results

This section presents experimental results obtained with our matching strategy; the results will also demonstrate in action the algorithm for recognizing objects in heaps. Although we have done experiments on a large number of scenes with 3D-POLY, only two such experiments will be presented to discuss the behavior of the algorithms.

3.8.1. The Models

The model library used consisted of two object models shown in Fig. 3.2. The object in Fig 3.2-(a) is given the name "square" and the one in (b) "round". The model knowledge was obtained by a "learning system" consisting of a special scanner in which the object is automatically rotated while illuminated by a number of translating laser beams. The data thus generated from many viewpoints is integrated and directly transformed into a feature sphere representation. Further details on the methods used for viewpoint integration and the transformations involved are presented in Chapter 4. For the two experiments discussed here, model data was generated by integrating six views for "square" and five for "round". For "square" object this resulted in a feature representation consisting of 14 vertex and 12 surface features. The model representation derived for "round" object consisted of 12 vertices and 10 surfaces.

Two feature spheres were derived for each model, one for surface features and the other for vertex features. The frequency of geodesic division, Q, of the spherical array discussed in Section 6.3 was chosen to be 16; this gave a resolution of about 4° per tessel in the spherical array representation. The vertex and surface features were used for the generation of hypotheses, while only the surface features were used for verification.

As described in Section 5.3, each object model must be associated with a list of LFS's for the purpose of hypothesis generation, an LFS being a set of surface features meeting at a vertex. In this prototype system, we have chosen to organize LFS's around convex vertices only, that is those whose edges are all convex. For "square" object, there are 12 LFS's which correspond to the 12 convex vertices, and for "round" object there are only four LFS's corresponding to the convex vertices *c*, *d*, *e*, *l*.

3.8.2. The Data

For the results that will be shown here, we had 10 overlapping objects, five of each type, in each of the two scenes. The objects were placed in a tray and, before data collection, the tray shaken vigorously to randomize the object placements. A typical scene was as was shown earlier in Fig. 3.1. Range images of the two scenes, shown in Figs. 3.24 and 3.25, were acquired by using a structured-light range sensing unit which is held by a PUMA robot for dynamic scanning; these images will be referred to as stripe images. Each stripe image consists of 150 stripes, with the inter-stripe spacing being 0.1"; this spacing is the distance the robot end-effector travels between successive projections.

Range maps for the scenes are obtained by converting each stripe point, which exists in image coordinates, into world coordinates using a calibration matrix by the method discussed in Chapter 1. Features are extracted from the range maps by a battery of low level procedures developed specifically for this research project. These procedures carry out surface normal computations, segmentations of surfaces of different types, surface classifications, etc., and are discussed in greater detail in Chapter 2. The output of preprocessing for the range map corresponding to the stripe image of Fig. 3.24 is shown in Fig. 3.5 in the form of a needle diagram and segmented surfaces. Fig. 3.26 shows the results for the stripe image of Fig. 3.25. Figs. 3.5 and 3.26 also display the labels given to the different surfaces.

3.8.3. Hypothesis Generation

For the purpose of hypothesis generation, each detected vertex in a scene is given a rank depending upon the number of surfaces meeting at the vertex and the convexity/concavity of the edges convergent at the vertex. The rank is greater, the larger the number of surfaces meeting at a vertex. Also, since we only use convex vertices for constructing the LFS's of a model, if a concave edge is found to be incident at a vertex, the rank of the vertex is made negative.

To generate hypotheses, the system first chooses the highest positively ranked vertex and then constructs an LFS by collecting all the surfaces meeting at the vertex. The

pile5

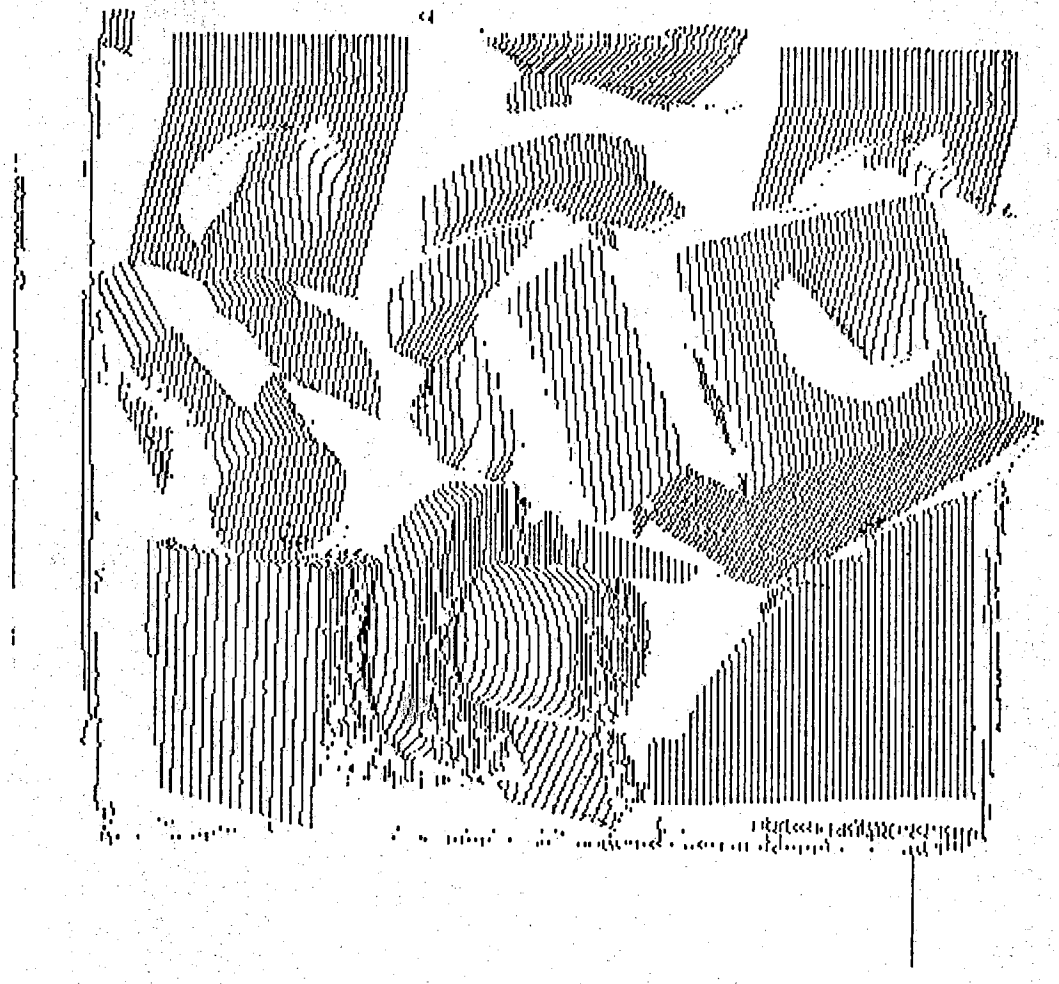


Figure 3.24. Stripe image of scene #1.

pile4

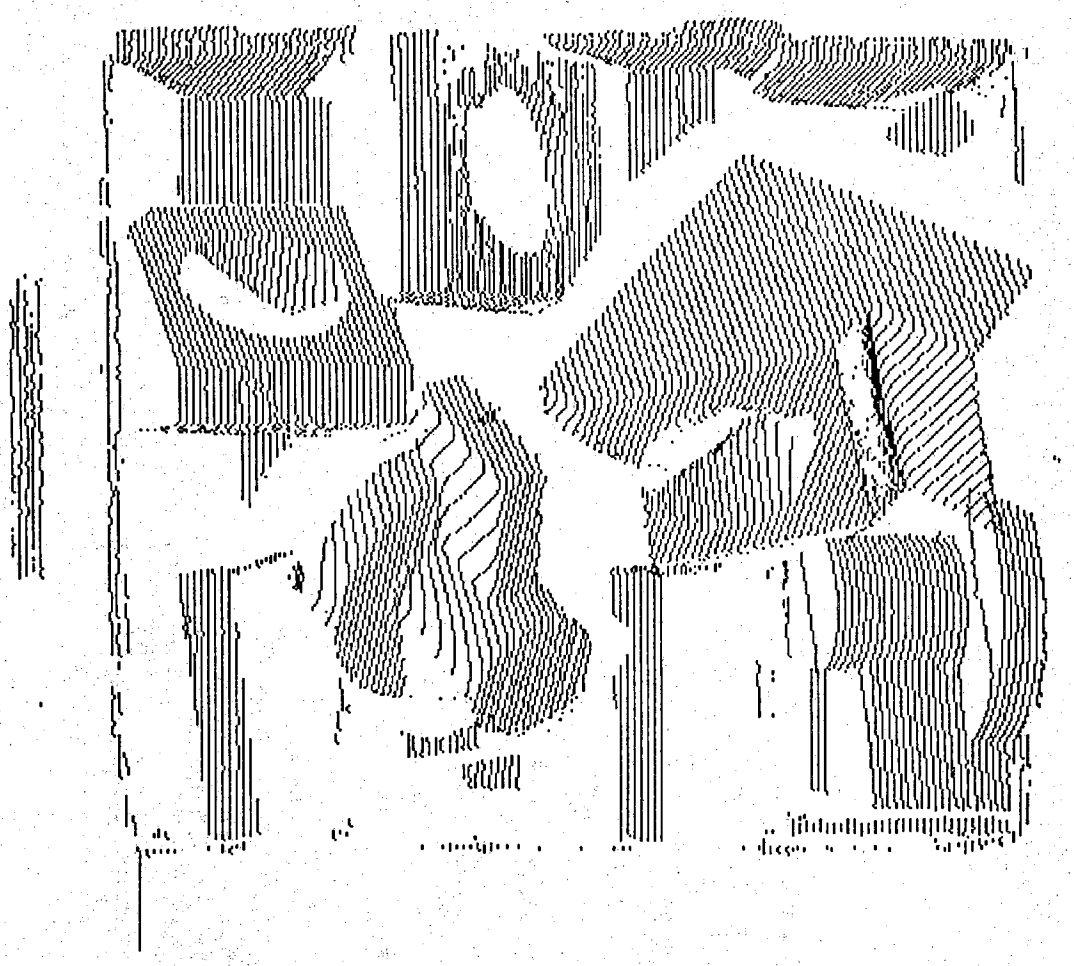


Figure 3.25. Stripe image of scene #2.

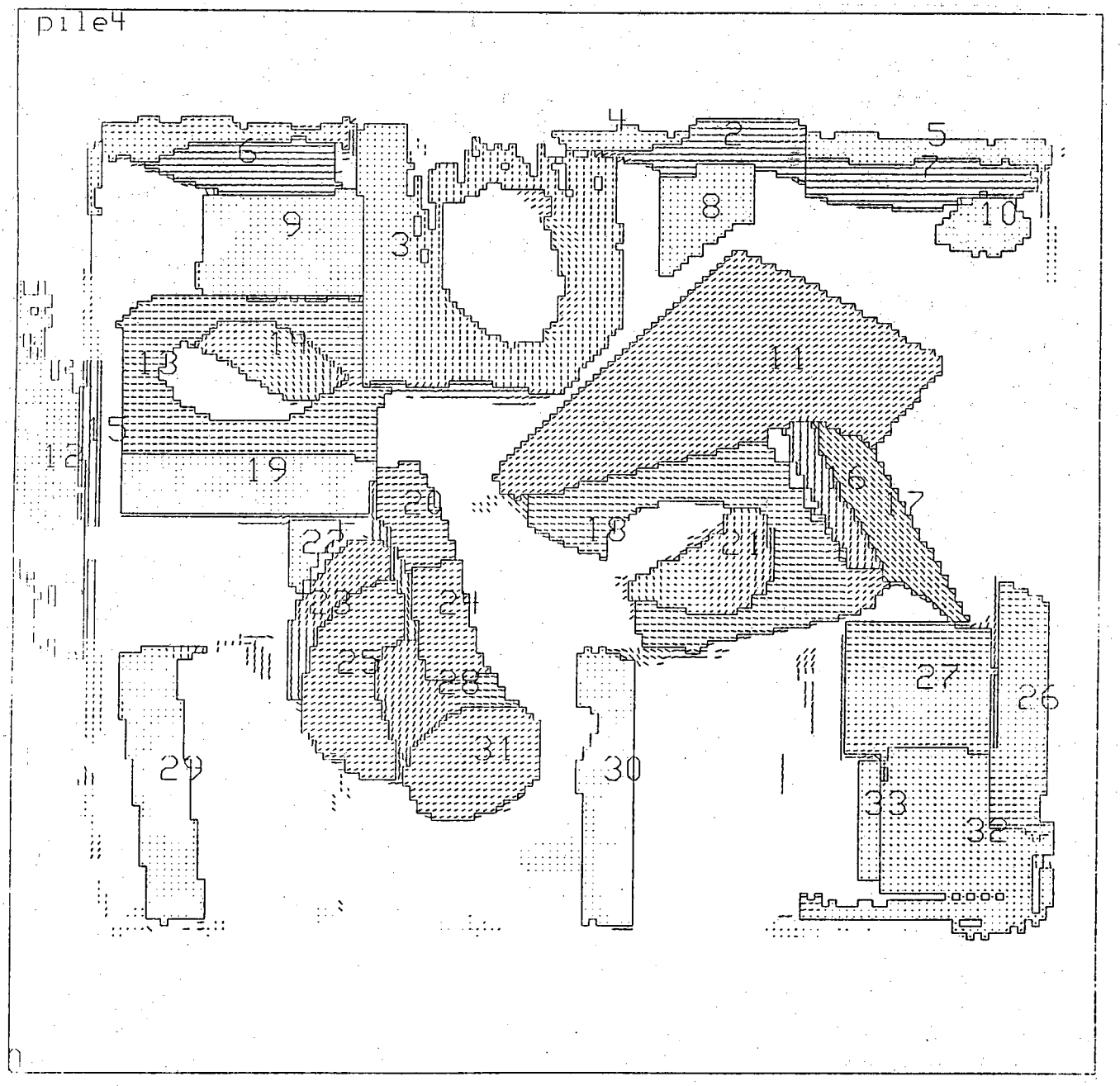


Figure 3.26. Result on feature extraction of scene #2.

scene LFS thus generated is matched with the LFS's of all the models, one by one. This matching between a scene LFS and a model LFS is carried out by a special procedure which tests the compatibility of the shape and relation attributes of the corresponding features in the two LFS's. Note that the maximal number of surfaces in an LFS for the objects in the experiments reported here is 3, thus there are three possible ways of establishing the correspondences between a scene LFS and a model LFS; all the three possibilities must be tested, each accepted possibility will lead to a different pose hypothesis.

For a given match between a scene LFS and a model LFS, the viewpoint independent position/orientation attributes of the features in the two LFS's are used for generating a candidate pose *Tr* for the scene object; further details on how exactly this is done can be found in Appendix A. For a candidate pose to translate into a pose hypothesis, the system checks the fitting error computed from the estimation of *Tr*; the error must be less than a predefined threshold.

In the preprocessed output shown in Fig. 3.5, there are 68 vertices, but only 36 of them are of convex type; in the output shown in Fig. 3.26 there are 22 convex vertices out of a total of 49 vertices. So, supposedly, in the worst case one would have to check 36 LFS's in the former case, and 22 in the latter. Since there are a total of 16 LFS's in the model library, 12 for "square" and 4 for "round", in the worst case one would have to carry out $16 \times 36 \times 3 = 1728$ LFS matches for the scene of Fig. 3.5, where the number 3 takes care of the aforementioned different ways of establishing correspondences between a model LFS and a scene LFS. Similarly, in the worst case situation, there may be $16 \times 22 \times 3 = 1056$ LFS matchings to be tested for the scene of Fig. 3.26. In practice, however, the number of LFS matches actually carried is far fewer on account of the following reason: An object hypothesis can be generated by any one of many LFS's, and when a hypothesis thus generated is verified, the system does not need to invoke any of the other LFS's for that object.

To give the reader an idea of the number of hypotheses generated, the system generated 156 hypotheses for the scene of Fig. 3.5, and 75 for the scene of Fig. 3.26.

3.8.4. Verification

Given the pose transformation *Tr* associated with a hypothesis, verification is carried out by computing the feature sphere tessel indices of those scene features that are "physically adjacent" to the LFS features, the notion of physical adjacency being as explained before, and matching each such scene feature with a model feature assigned to that tessel, assuming such a model feature can be found. [If more than one model feature may be assigned to a tessel, the scene feature must be matched with all of them.]

Of course, since measurement noise and other artifacts will always be present to distort the attribute values of a scene feature, the scene feature must be matched with all the model features belonging to tessels within a certain neighborhood of the tessel computed from the scene feature principal direction. The size of the neighborhood reflects the uncertainty in the feature measurements. For most of our experiments, we use all the model features within two tessels of the tessel assigned to a scene feature, corresponding approximately to a directional uncertainty of 8.0 °.

To illustrate the behavior of the algorithm, Table 3.3 shows the hypothesis generation and verification procedure in action. Each line entry, printed out upon the formation of a hypothesis, identifies the LFS used by the vertex chosen, and shows the surface correspondences established when the scene LFS was matched with a model LFS. For example, for the first hypothesis, marked hyp#1 in the table, the LFS matching established correspondences between scene surface 7 and model surface 2; and between scene surface 5 and model surface 1. The number 2 at the end of the line in the table indicates that the first hypothesis was failed during the verification stage after failures along two different paths in the search space, each failure caused by a mismatch of a scene feature, physically adjacent to one of the hypothesis generating LFS features, and the model feature located within the uncertainty range of the tessel corresponding to the scene feature. This is not to imply a fan-out of only 2 at the end of the hypothsis generating segment for hyp#1; only that for the other branches the scene features, again physically adjacent to one of the LFS features, had no corresponding model features on the feature sphere. This is also the reason for 0 at the end of many of the line entries in the table.

As mentioned in Section 7, the acceptance of a hypothesis is predicated upon our finding at least 33% of the model features from amongst those that are adjacent to the features in an LFS. As shown in the table, from among the 75 generated hypotheses only the hypothses #23 and #75 are verified and lead to the recognition of an instance of "square" in the first case, and to that of "round" in the other. During the verification of hypothesis #23, scene surface 16 fails to match model surface 3 of the square model, although scene regions 18 and 21 do match model regions 4 and 5, respectively. As is evident from the stripe image of Fig. 3.25, the difficulties with scene surface 16 are due to problems with the robust detection of stripes over that surface; these problems are probably caused by the rather very acute angle between the stripe projection direction and the surface. It is entirely possible that the surface labeled 16 in the scene is made of reflections of the stripes seen in adjoining surfaces. In other words, surface 16 is most likely a spurious surface and not matchable with its potential candidate model surface 3. During the verification of hypothesis 75, scene region 23 is not matched to any model region. This is because only a small portion (less than 25%) of the cylindrical

Table 3.3 Output listing of the interpretation of scene #2.

```
Verify hyp #1  Model: square  Vert: 12  Reg: (7->2) (5->1) ... failed - 2
Verify hyp #2  Model: square  Vert: 12  Reg: (7->1) (5->2) ... failed - 2
Verify hyp #3  Model: square  Vert: 12  Reg: (7->10) (5->4) ... failed - 2
Verify hyp #4  Model: square  Vert: 12  Reg: (7->4) (5->8) ... failed - 1
Verify hyp #5  Model: square  Vert: 12  Reg: (7->10) (5->12) ... failed - 1
Verify hyp #6  Model: square  Vert: 12  Reg: (7->4) (5->10) ... failed - 1
Verify hyp #7  Model: square  Vert: 12  Reg: (7->12) (5->9) ... failed - 1
Verify hyp #8  Model: square  Vert: 12  Reg: (7->9) (5->10) ... failed - 1
Verify hyp #9  Model: square  Vert: 12  Reg: (7->9) (5->12) ... failed - 1
Verify hyp #10  Model: square  Vert: 12  Reg: (7->12) (5->10) ... failed - 1
Verify hyp #11  Model: square  Vert: 12  Reg: (7->10) (5->9) ... failed - 1
Verify hyp #12  Model: round  Vert: 18  Reg: (17->9) (11->7) ... failed - 0
Verify hyp #13  Model: round  Vert: 18  Reg: (17->9) (11->1) ... failed - 0
Verify hyp #14  Model: round  Vert: 18  Reg: (17->7) (11->9) ... failed - 0
Verify hyp #15  Model: round  Vert: 18  Reg: (17->1) (11->6) ... failed - 0
Verify hyp #16  Model: round  Vert: 18  Reg: (17->6) (11->7) ... failed - 0
Verify hyp #17  Model: round  Vert: 18  Reg: (17->7) (11->6) ... failed - 0
Verify hyp #18  Model: square  Vert: 18  Reg: (17->2) (11->1) ... failed - 0
Verify hyp #19  Model: square  Vert: 18  Reg: (17->1) (11->10)
            scene region 18 matched to model region 4  ... failed - 2
Verify hyp #20  Model: square  Vert: 18  Reg: (17->10) (11->2) ... failed - 0
Verify hyp #21  Model: square  Vert: 18  Reg: (17->8) (11->1) ... failed - 0
Verify hyp #22  Model: square  Vert: 18  Reg: (17->1) (11->2) ... failed - 0
Verify hyp #23  Model: square  Vert: 18  Reg: (17->2) (11->8)
            scene region 18 matched to model region 4
            scene region 21 matched to model region 5  ... SUCCEED!!! - 2

Verify hyp #24  Model: square  Vert: 26  Reg: (19->4) (13->3) ... failed - 0
Verify hyp #25  Model: square  Vert: 26  Reg: (19->3) (13->4) ... failed - 0
Verify hyp #26  Model: square  Vert: 26  Reg: (19->7) (13->4) ... failed - 0
Verify hyp #27  Model: square  Vert: 27  Reg: (13->4) (19->3) ... failed - 0
Verify hyp #28  Model: square  Vert: 27  Reg: (13->3) (19->4) ... failed - 0
Verify hyp #29  Model: square  Vert: 27  Reg: (13->7) (19->4) ... failed - 0
Verify hyp #30  Model: round  Vert: 36  Reg: (24->7) (20->10) ... failed - 0
Verify hyp #31  Model: round  Vert: 36  Reg: (24->10) (20->9) ... failed - 2
Verify hyp #32  Model: round  Vert: 36  Reg: (24->9) (20->7) ... failed - 0
Verify hyp #33  Model: round  Vert: 36  Reg: (24->9) (20->1) ... failed - 0
Verify hyp #34  Model: round  Vert: 36  Reg: (24->1) (20->7) ... failed - 2
Verify hyp #35  Model: round  Vert: 36  Reg: (24->7) (20->9) ... failed - 0
Verify hyp #36  Model: round  Vert: 36  Reg: (24->7) (20->1) ... failed - 0
Verify hyp #37  Model: round  Vert: 36  Reg: (24->1) (20->6) ... failed - 2
Verify hyp #38  Model: round  Vert: 36  Reg: (24->6) (20->7) ... failed - 0
Verify hyp #39  Model: round  Vert: 36  Reg: (24->6) (20->10) ... failed - 0
Verify hyp #40  Model: round  Vert: 36  Reg: (24->10) (20->7)
            scene region 28 matched to model region 4  ... failed - 3
Verify hyp #41  Model: round  Vert: 36  Reg: (24->7) (20->6) ... failed - 0
Verify hyp #42  Model: square  Vert: 36  Reg: (24->2) (20->1) ... failed - 0
```

Table 3.3 Continued

```
Verify hyp #43  Model: square  Vert: 36  Reg: (24->1) (20->10) ... failed - 0
Verify hyp #44  Model: square  Vert: 36  Reg: (24->10) (20->2) ... failed - 0
Verify hyp #45  Model: square  Vert: 36  Reg: (24->8) (20->1) ... failed - 0
Verify hyp #46  Model: square  Vert: 36  Reg: (24->1) (20->2) ... failed - 0
Verify hyp #47  Model: square  Vert: 36  Reg: (24->2) (20->8) ... failed - 0
Verify hyp #48  Model: square  Vert: 36  Reg: (24->3) (20->10) ... failed - 0
Verify hyp #49  Model: square  Vert: 36  Reg: (24->10) (20->4) ... failed - 0
Verify hyp #50  Model: square  Vert: 36  Reg: (24->4) (20->8) ... failed - 0
Verify hyp #51  Model: square  Vert: 36  Reg: (24->8) (20->3) ... failed - 0
Verify hyp #52  Model: square  Vert: 36  Reg: (24->12) (20->7) ... failed - 0
Verify hyp #53  Model: square  Vert: 36  Reg: (24->7) (20->10) ... failed - 0
Verify hyp #54  Model: square  Vert: 36  Reg: (24->10) (20->12) ... failed - 0
Verify hyp #55  Model: square  Vert: 36  Reg: (24->4) (20->10) ... failed - 0
Verify hyp #56  Model: square  Vert: 36  Reg: (24->10) (20->7) ... failed - 0
Verify hyp #57  Model: square  Vert: 36  Reg: (24->9) (20->1) ... failed - 0
Verify hyp #58  Model: square  Vert: 36  Reg: (24->1) (20->8) ... failed - 0
Verify hyp #59  Model: square  Vert: 36  Reg: (24->8) (20->9) ... failed - 0
Verify hyp #60  Model: square  Vert: 36  Reg: (24->12) (20->9) ... failed - 0
Verify hyp #61  Model: square  Vert: 36  Reg: (24->9) (20->8) ... failed - 0
Verify hyp #62  Model: square  Vert: 36  Reg: (24->8) (20->12) ... failed - 0
Verify hyp #63  Model: square  Vert: 36  Reg: (24->10) (20->1) ... failed - 0
Verify hyp #64  Model: square  Vert: 36  Reg: (24->1) (20->9) ... failed - 0
Verify hyp #65  Model: square  Vert: 36  Reg: (24->9) (20->10) ... failed - 0
Verify hyp #66  Model: square  Vert: 36  Reg: (24->9) (20->12) ... failed - 0
Verify hyp #67  Model: square  Vert: 36  Reg: (24->12) (20->10) ... failed - 0
Verify hyp #68  Model: square  Vert: 36  Reg: (24->10) (20->9) ... failed - 0
Verify hyp #69  Model: round  Vert: 37  Reg: (20->7) (24->10) ... failed - 2
Verify hyp #70  Model: round  Vert: 37  Reg: (20->10) (24->9) ... failed - 0
Verify hyp #71  Model: round  Vert: 37  Reg: (20->9) (24->7) ... failed - 0
Verify hyp #72  Model: round  Vert: 37  Reg: (20->9) (24->1) ... failed - 2
Verify hyp #73  Model: round  Vert: 37  Reg: (20->1) (24->7) ... failed - 0
Verify hyp #74  Model: round  Vert: 37  Reg: (20->7) (24->9) ... failed - 0
Verify hyp #75  Model: round  Vert: 37  Reg: (20->7) (24->1)
                scene region 28 matched to model region 4
                scene region 31 matched to model region 2
                scene region 25 matched to model region 8  ... SUCCEED!!! - 6

Total number of feature matching tests for verification: 29
Process completed
Recognized_objects:
 square
 round
```

surface is visible in the scene, and the computed radius is off too much from its correct value to match to the candidate model region 5.

Note from Table 3.3 that most of the 75 hypotheses are rejected immediately during verification, without the computational burden of any feature matching. For each line entry in the table that ends in a 0, no features had to be matched during the verification stage; the hypothesis failed simply because no model features could be found in the vicinity of the tessels for the scene features used during verifications. In fact, as depicted at the end of the table, for the scene of Fig. 3.26, only 29 features had to be matched during the entire verification process. So, on the average, the system had to match only 0.387 features during each hypothesis verification. The largest number of features matched during any verification was 6, this was for hypothesis #75, confirming our $O(n)$ measure for the time complexity of verification.

This prototype system is programed in C language and runs on a SUN-3 workstation. The CPU time for interpreting a processed range image was 9 seconds for the scene of Fig. 3.5 and 4 seconds for the scene of Fig. 3.26. The CPU time is approximately propotional to the number of generated hypotheses, which in turn depends on the complexity of the scene.

3.9. Conclusions

In this chapter, we have presented feature matching and recognition strategies in 3D-POLY. For recognition, the system used an approach based on hypothesis generation and verification. The strategies used in the system lead to a polynomial time algorithm for the interpretation of range images.

The polynomial bound on the time complexity was made possible by two key ideas, one for hypothesis generation and the other for verification. The key idea in the former was the use of special feature sets, the spatial relationships between the features in these sets being such that the number of possible ways in which the scene feature could be matched to these in the sets was substantially curtailed. The key idea in the verification stage was the association of a principal direction with a feature and, after the establishment of a pose tranform, comparing a scene feature with a model feature only if the two agreed on the basis of their principal directions. This sharply reduced the number of scene and model features that had to be actually matched, leading to great savings in the computations involved.

To embed the notion of feature principal-direction in a computationally efficient framework, we represented the model features on a feature sphere. We advanced a data structure for feature spheres and presented efficient algorithms for finding

neighborhoods on the sphere and for assigning a tessel on the sphere to a measured principal direction.

We showed how our object recognition framework should be applied to scenes consisting of multiple objects in a heap. Finally, we discussed experimental results validating our complexity measures.

CHAPTER 4
LEARNING 3-D MODELS FROM MULTIPLE VIEWS OF OBJECTS

To be completely functional, a robot vision system must have access to a library of model representations of all possible objects that we want the system to recognize. Models may be specified either via CAD descriptions or the system may be provided with a capability to generate its own models ''by showing." In this chapter, we will take the latter approach and present a procedure which consists of placing an object in a computer controlled structured-light scanner capable of generating range maps of the object from many different viewpoints. We will show how the surface information from the different viewpoints is integrated into a full 3-D representation of the object. The learned representation thus generated consists of a feature sphere, which can then be directly used by the recognition procedure described earlier. In addition, it is also possible to draw wire-frame and full boundary representations from the feature spheres so obtained, although in this chapter we will only show the former.

4.1. Introduction

The recognition strategy of Chapter 3 must have available to it models of the objects that we want the robot vision system to recognize, and these models should preferably utilize the feature sphere representation. Of course, a superior goal would be drive the recognition system directly from the available CAD models of objects; however, CAD models cannot be used directly for vision applications because features that one may be able to pull out of an image are not explicitly defined in such models. Therefore, one must always install an intermediate representation to bridge CAD with vision, a point that was eloquently made in [*K&et*–87].

For the purpose of this chapter, we will assume that we want our models to be in the feature sphere form. Clearly, for someone to drive our recognition method with CAD, they will have to write their own ''translator" to convert CAD data structures into feature sphere data structures. Much work has been done by other researchers in developing these types of translators, translators that convert CAD representations into those which are more suitable for use by vision algorithms. The most notable work along these lines was done by Bhanu and Ho [*B&H*–87]; they have discussed

procedures for converting a CAD boundary representation into vision-oriented representations such as EGI, octrees, etc. Similarly, Xie and Calvert [*X&C*–88] have presented a rule-based system that can convert a boundary representation of an object to their so called "CGS-EESI" (constructive geometric solid-extended enhanced spherical image) representation. In the works of Hansen and Henderson [*H&H*–87] and Ikeuchi [*Ik*–87], an interpretation tree is built from a CAD model of the object.

As was mentioned before, the focus of this chapter is on presenting a technique in which the model of an object is learned by showing, the learned model being directly in the form that can be used in the recognition procedure. To construct a full 3-D model, an object must be shown to the system in many different orientations and the information obtained from all the orientations somehow integrated, leading to the notion of viewpoint integration. At the simplest level, as was done by Oshima and Shirai [*O&S*–83], viewpoint integration may consist of simply collecting every possible view of an object, each view being represented by a graph of the surface-features extracted from that view. Similar approaches for model construction are also typical of those used in the recognition of 3-D objects from 2-D imagery. An obvious drawback of such approaches is the large size of the resulting model library, which can degrade the performance of the recognition system.

At a more sophisticated level, learning a model consists of actually merging the feature information gleaned from different viewpoints. As an example of this approach, Baker [*Ba*–77] has presented a scheme for model building in which points of curvature irregularity extracted from different views of an object are correlated. Martin and Aggarwal [*M&A*–83] have presented a method in which a volumetric model of an object is constructed by intersecting the bounding volumes, each such volume being specified by the silhouette in a view. In the scheme advanced by Potmesil [*Po*–83], bi-cubic surface segments from various views of an object are integrated into a complete boundary representation of the object. Another scheme has been presented by Herman and Kanade [*H&K*–86] in which 3-D models of buildings are built incrementally from sequences of stereo images. A similar approach has been presented by Xie and Calvert [*X&C*–86] for generating 3-D models of office scenes. To the best of our knowledge, the models constructed by all these methods were used only for displaying the objects thus synthesized and not for driving a recognition system. In fact, it is not obvious that it would at all be easy to use the model representations constructed in these works for recognition. How the generated models may be used for recognition is more obvious in the contributions made by Underwood and Coates [*U&C*–75] and Dane [*Da*–82]. Underwood and Coates [*U&C*–75] have presented an agorithm for constructing graph representations of convex polyhedra from multiple views of the objects, a node in the graph being a planar face and an arc expressing adjacency between two

faces. In the mothod of Dane [*Da*–82], viewpoint information is merged to generate planar and quadric surfaces.

An important issue in multiview integration is the determination of whether a feature seen in a new view has been seen in any previous views. Clearly, this problem is a variation on the feature matching problem in object recognition. In [*U&C*–75] and [*Da*–82], this type of feature matching is carried out by exhaustive search, meaning that a feature seen in a new view has to be tested against every feature seen in all previous views. In this chapter we show that the highly efficient feature matching strategy presented in Chapter 3 can be applied to this problem with little modification. Another important issue in multiview integration is how to merge two pieces of information carried by two features in two different views when they are in effect the same feature. We will propose solutions to both problems in this chapter.

The physical setup of our model learning system, shown in Fig. 4.1, consists of a computer-controlled turntable and a structured light range scanner. The feature extraction component of this system is the same as that described for the object recognition system described earlier. The feature matching and merging component is a modification of the feature matching routine employed earlier. In the next section, we will give an overview of the setup and the system. Then in Section 3, we show how one must establish an object-centered coordinate system in which the 3-D model is ultimately systhesized; the manner in which the coordinates are selected must take into account considerations such as any wobble in the turntable. In Sections 4 and 5, we will then discuss how to initiate a model from the first view of the object and, subsequently, how to update a partial model with data from successive views. Finally, in Section 6, we will use an experimental example to illustrate the working of the entire model learning system.

4.2. General Strategy and System Overview

In multiview integration for model building, the number of views from which an object is viewed must satisfy two requirements, which we will now state. (As the reader will notice, the second requirement will subsume the first.) For the first requirement, it is necessary that every object feature be visible in at least one view. And, for the second requirement, features must overlap between successive views for the purpose of establishing the adjacency relationships between the features. Obviously, if a given feature was visible in only a single view, it would be impossible to discover what other features the given feature might be adjacent to. The second requirement also points to the fundamental problem in model building by multiview integration, viz., how to relate features in one view to features in another view, keeping in mind the likelihood that

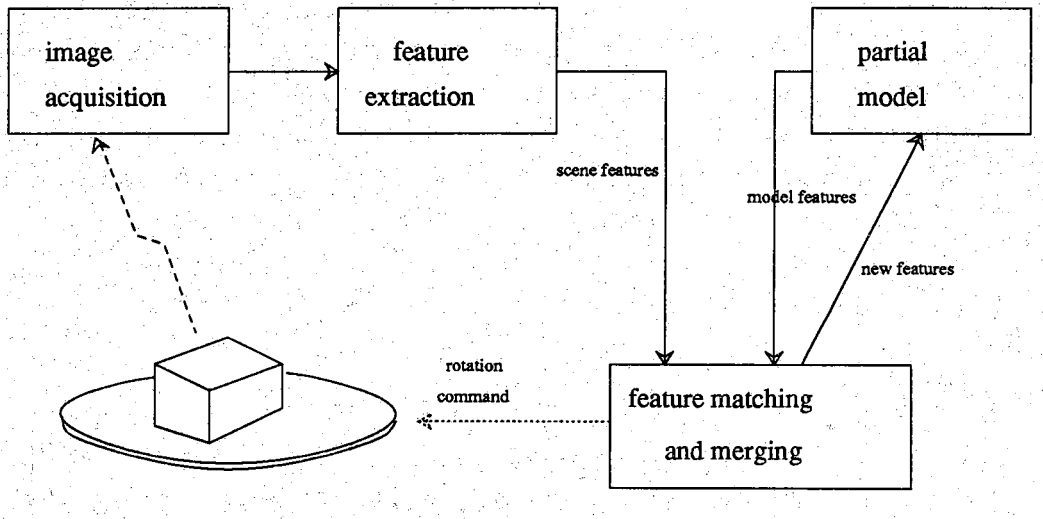


Figure 4.1. A model learning system. The image acquisition and the feature extraction components of this system are the same as in Fig. 0.1.

some of the features in the two views are the same. Our strategy to solving this problem is a modified version of the strategy for object recognition described in Chapter 3 and consists of the following steps: From the first view of the object, we first define an object-centered coordinate system. We then establish a feature sphere (or, feature spheres, if more than one feature type is being considered) corresponding to this coordinate system and assign the extracted features to the appropriate tessels on the sphere. Each feature sphere is then incrementally updated by using features and their attribute-value pairs gleaned from each successive view.

For incrementally updating a feature sphere, we first determine the transformation between the fixed world coordinate frame in which each view is taken and the object-centered coordinate frame which rotates with the object and in which the feature spheres are built. For the most part, this transformation is easily derived from the rotation of the turntable. However, if the object is manually turned upside down to integrate in the information from the underside of the object, the system must automatically compute this transformation. Once the transformation is known, the feature matching criteria discussed in Chapter 3 can be used to determine whether or not a feature was seen in a previous view. If a feature in the new view is matched to a feature in the currently known partial model, the system must merge the two features together; otherwise, the partial model must be updated by the addition of the new feature.

The following pseudo-code represents this model learning strategy:

```
build_model (O_m, {I})) {
    for each new view I in {I}
        extract a feature set {S} from range image I
        if (first view)
            initiate_model ({S}, O_m)
        else
            compute transformation Tr
            update_model ({S}, O_m, Tr)
    for each new view I in {I}
    collect LFS's }

initiate_model ({S}, O_m) {
    define an object-centered coordinate system Tr_m
    allocate a feature sphere for each class of feature
    for each S_i in {S}
        add (Tr_m^-1 (S_i)) to O_m }
```

update_model ($\{S\}$, O_m, Tr) {
  for each S_i in $\{S\}$
    if there is an $M_j \in O_m$ such that $Tr^{-1}(S_i)$ <=> M_j
      then modify M_j with S_i
    else add ($Tr^{-1}(S_i)$) to O_m }

Note that in the last step in "build_model" the system collects all the *local feature sets* (LFS) from the synthesized 3-D model. In accordance with the discussion in Chapter 3, these LFS's play an important role in the object recognition system described there. In particular, the LFS's are used for generating pose transformation hypotheses.

Fig. 4.2 depicts the two essential elements of the model learning system, the turntable and the structured-light unit. An object whose model is to be created is placed at the center of a turntable and its range images are then taken for different rotational positions of the object; for each range image the structured light scanner is translated linearly, as illustrated by the straight arrow in the figure, the direction of motion corresponding to the y-axis in the world coordinate system. The rotations of the turntable and the translations of the scanner are all under computer control and can be varied depending upon the complexities of the object shape. (For simple objects, the different rotational views can be as far apart as 90° and a 3-D model synthesized with just four views.) The axis of the rotation of the turntable is approximately parallel to the +z direction of the world coordinate system. To provide a good coverage of both the sides and the top of the object, both the center of the laser beam and the optic axis of the camera make angles that are roughly 45° with the z-axis.

Unless the object has a flat bottom, it often becomes necessary to also model those surfaces that would be invisible when the object is first placed on the turntable due to their being in contact with the table. To make a complete 3-D model for such objects, after all the views are collected in the first position, the object is manually turned upside down and scanning resumed. In general, just as many views are collected in the new position as in the first. The important point to note is that the system automatically computes the transform Tr that relates the object-centered coordinate frame for the object in an upside-down position to the world coordinate frame by matching common features.

4.3. Determination of Transformations

Clearly, the first thing that must be done in model learning is to establish a coordinate system in which the model will be synthesized. For this purpose, we set up an

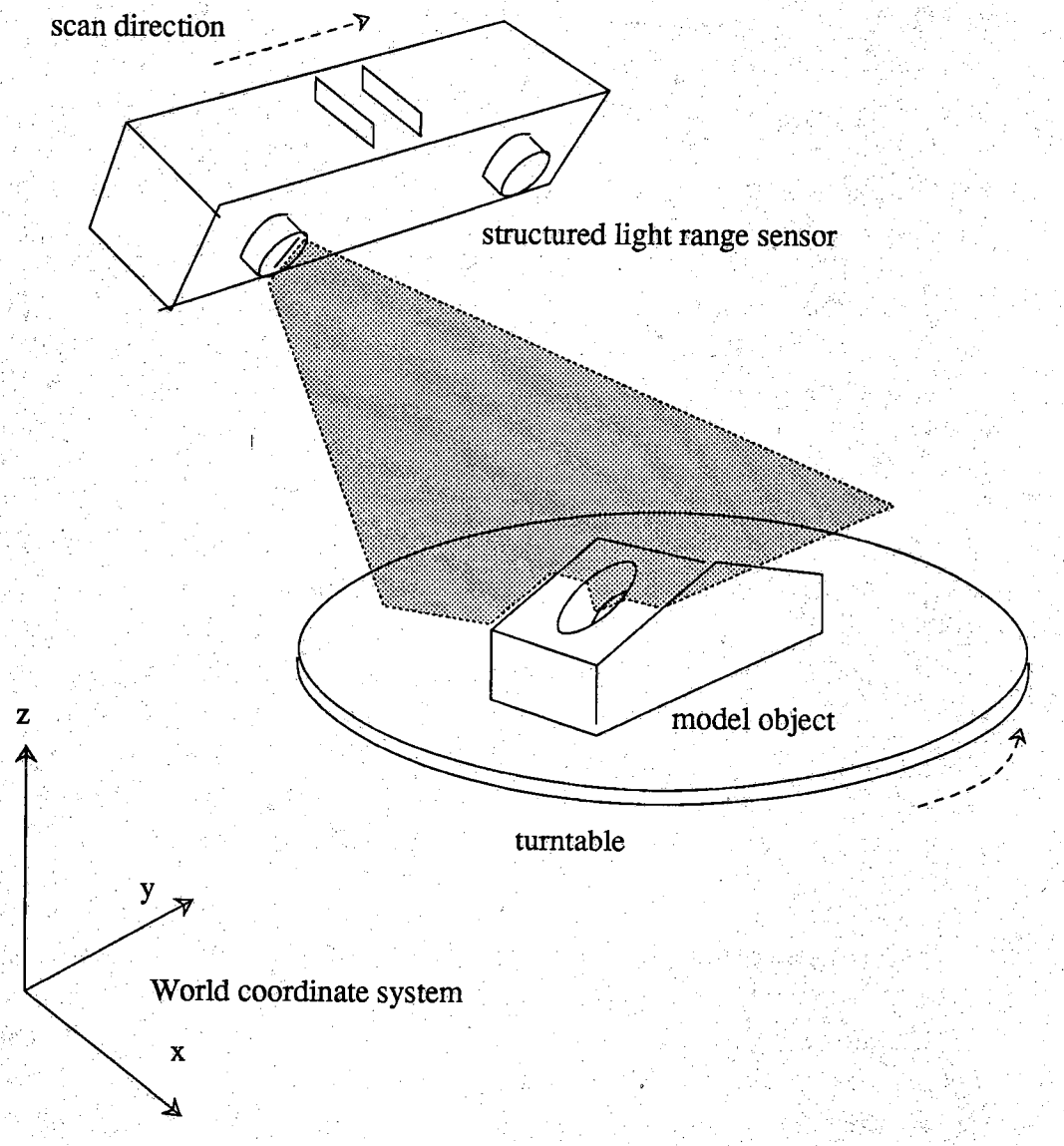


Figure 4.2. The physical setup of the model learning system. The structured light range sensor is calibrated with respect to the world coordinate system.

object-centered coordinate system. The process of model synthesis is greatly facilitated if this coordinate system is defined in such a manner that its z-axis is coincident with the axis of rotation of the turntable. It also helps to define the origin of this coordinate system at a point that is half way between the highest point on the object and the turntable. What follows is a procedure for defining such an object-centered coordinate system given the first view of the object. Note that this procedure relies solely on the reading of the range sensor of the system, and does not require any manual measurements.

The reader beware that we are making a distinction between the object-centered coordinate frame, in which all the views are pooled for model synthesis, and a world coordinate frame, in which sits the scanner. Therefore, it is the world coordinate frame in which we specify the scan directions. We will assume that the range sensor has been calibrated [*C&K*−87] with respect to the world coordinate system. The origin of the world coordinate system can be anywhere.

Of course, since the scanner resides in the world frame, we must first establish the world coordinate system before we can set up the object-centered coordinates. The world frame is established essentially by the human operator. In our experiments, the world frame is as shown in Fig. 4.2, with the y-axis corresponding to the translational movement of the structured-light unit and the z-axis nominally perpendicular to the turntable.

Next, the object-centered coordinate frame must be established. We will now describe a procedure for doing so. The first step consists of determining the axis of rotation of the turntable, since this axis will serve as the z-axis of the object-centered system. The rotating axis can be specified by a unit direction vector, $\boldsymbol{a}$. We will also assume the existence of a center of rotation, denoted by $\boldsymbol{p}_0$; this will be a point on the rotation axis located at the intersection of the axis with the plane of the turntable. Note that the rotation center, as specified by $\boldsymbol{p}_0$, will not by itself be used for the origin of the object-centered system, but only as an intermediate step toward obtaining that origin.

To determine $\boldsymbol{a}$, we take two range images of the face (which is planar) of the turntable; for the second image the turntable is rotated by 180°. In each of these images, a plane is fit to the turntable range data, and the surface normals computed. Let the two surface normals be denoted by $\boldsymbol{n}^0$ and $\boldsymbol{n}^{180}$. If the surface of the turntable was perfectly perpendicular to the rotation axis for all angular positions of the turntable, the two normals would be identical and parallel to $\boldsymbol{a}$. In general, this condition is not satisfied because of the slight wobble that might be present when the turntable is rotated. In the presence of the wobble, the directions of the two normals are symmetric with respect to $\boldsymbol{a}$. In either case, the direction of the rotation axis can be computed from

$$\boldsymbol{a} = \frac{(\boldsymbol{n}^0 + \boldsymbol{n}^{180})/2}{|(\boldsymbol{n}^0 + \boldsymbol{n}^{180})/2|}.$$

The position vector $\boldsymbol{p}_0$ to the center of rotation can be found from a range image in which the center of the turntable is marked somehow. To be able to localize the center precisely, we usually place a rectangular box on the turntable with one of its corner touching the center. We then take a range image of the box and the turntable and detect the corner in the resulting data.

Since we want the origin of the object-centered coordinate frame to be about half way between the highest point on the object and the turntable, the system utilizes the first view of the object for locating this origin. If we denote by h the maximum height of any object point in the first view, the location of the origin of the object-centered coordinate system is then given by

$$\boldsymbol{o} = \boldsymbol{p}_0 + (\frac{h}{2})\,\boldsymbol{a}$$

Since the object is placed roughly at the center of the turntable, for many objects this origin will be roughly at the volume center of the object. We then specify as follows the three axes, x', y' and z', of the object-centered coordinate system

$$\boldsymbol{z}' = \boldsymbol{a}$$

$$\boldsymbol{y}' = \frac{\boldsymbol{a} \times [1\ 0\ 0]^t}{|\boldsymbol{a} \times [1\ 0\ 0]^t|}$$

$$\boldsymbol{x}' = \boldsymbol{y}' \times \boldsymbol{z}'$$

where $[1\ 0\ 0]^t$ represents the x axis of the world coordinate system. Thus the object-centered coordinate frame in the first view is related to the world coordinate frame by the following transformation

$$Tr_0 = \begin{bmatrix} \boldsymbol{x}'_x & \boldsymbol{y}'_x & \boldsymbol{z}'_x & \boldsymbol{o}_x \\ \boldsymbol{x}'_y & \boldsymbol{y}'_y & \boldsymbol{z}'_y & \boldsymbol{o}_y \\ \boldsymbol{x}'_z & \boldsymbol{y}'_z & \boldsymbol{z}'_z & \boldsymbol{o}_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For a given position vector in the object-centered frame for the first view, this transformation helps us find its corresponding coordinates in the world frame. In other words, Tr_0 takes from the object-centered frame to the world frame. Therefore, Tr_0^{-1} takes us from the world frame to the object-centered frame for the first view. Suppose, in the range map for the first view -- the range map for every view will be defined in the fixed world frame -- we locate a feature at, say, the vector $\boldsymbol{v}$, then the coordinates of this vector in the object-centered frame will be given by $Tr_0^{-1}\boldsymbol{v}$.

As the object is rotated on the turntable for a different view, the object-centered frame also rotates with the object; however, the range map is still in the same world frame. Let the angle of rotation of the turntable, measured counterclockwise in the xy-plane of the world frame, for the i^{th} view be θ_i. The transformation that takes us from the fixed world frame to the object-centered frame for the i^{th} view is given by Tr_i^{-1}, where

$$Tr_i = Tr_0 \, Rot_{z'}(\theta_i) \tag{4.1}$$

with $Rot_{z'}$ defined by

$$Rot_{z'}(\theta_i) = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note that the matrix $Rot(\theta)$ rotates a vector through an angle θ counterclockwise in the xy-plane. Therefore, if in the i^{th} range map, an object feature is found to be located at, say, the vector $\boldsymbol{v}$, then the corresponding vector in the object-centered frame is $Tr_i^{-1}\boldsymbol{v}$.

In the derivation of Equation (4.1), we have made use of the knowledge that the object undertakes a rotation around a known axis through a known angle from one view to the next. Clearly, when the object is flipped for generating information on its underside, this transformation will cease to be valid. We therefore must have recourse to some algorithm that can re-establish the transformation of the object in the new setting. The algorithm that is used for this purpose is very similar to the one used for generating pose-transformation hypotheses in the object recognition system described in Chapter 3. We will now describe how exactly we find the transformation that takes us from the world frame to the object-centered frame after the object is flipped.

Assume that in the first set of scans, before the object is flipped, the system has collected all the possible LFS's from the partial model constructed so far. Further assume that at least one of these LFS's is visible in the the first view after the object is flipped. The algorithm then proceeds as follows: Extract every LFS from the range image and try to find a matched LFS in the partial model built so far from the topside views. Each such match will, in general, lead to a different pose transformation for the object in its flipped position. For each such pose transformation, we count the number of features extracted from the range image that can be matched with the partial model. The pose transformation yielding the largest count is accepted as the transformation that takes us from the world frame to the object-centered frame. The algorithm for computing this transformation is sketched below in pseudo language:

```
reestablish_transform ( I, O_m ) {
    extract features {S} from range image I
*   extract a new LFS_s from {S}
        if (LFS_s matches an LFS in O_m)
            estimate T̂r
            count # of matching features between {S} and O_m under T̂r
        go to *
    return the T̂r which yields the maximal count }
```

Once this transformation, denoted by Tr_t, is found, for subsequent rotations of the object the transformation that takes us directly from the fixed world frame, in which all range maps are constructed, to the object-centered frame are determined as before. In other words, if

$$Tr_i = Tr_t \; Rot_{z'}(\theta_i) \tag{4.2}$$

then, Tr_i^{-1} takes us from the fixed world coordinates to the object-centered coordinates for the i^{th} view taken after the object is flipped. At the risk of being repetitious, we would like to elaborate by saying that if in the i^{th} range map after the object is flipped, if a feature is located at the vector $\boldsymbol{v}$ in the world frame, then the corresponding vector in the object-centered frame is given by $Tr_i^{-1}\boldsymbol{v}$.

4.4. Model Initiation in the First View

Once the object-centered coordinate system is established, the procedure for initiating a partial object model from the first view is rather straightforward. We first transform the position/orientation attributes of every detected object features in the image from the world coordinate system to the object-centered coordinate system by multiplying the attributes from the left by Tr_1^{-1}. In short, we perform the following feature translation:

$$Tr_1^{-1}(S_i) \rightarrow M_{c(i)} \tag{4.3}$$

where S_i is a scene feature and $M_{c(i)}$ is the translated model feature relabeled as $c(i)$.

As mentioned in Chapter 3, a feature can generally be described by three sets of attributes: shape, relation and position/orientation. Of these, only position/orientation attributes are transformation dependent. We thus can rewrite expression (4.3) in terms of the three sets of attributes as follows:

$$sa(S_i) \rightarrow sa(M_{c(i)}) \qquad \text{for } all \;\; sa \in SA(S_i)$$

$$c(ra(S_i)) \rightarrow ra(M_{c(i)}) \qquad \text{for } all \;\; ra \in RA(S_i)$$

$$Tr^{-1}(la(S_i)) \rightarrow la(M_{c(i)}) \qquad \text{for } all \;\; la \in LA(S_i)$$

Note that, as pointed out in Chapter 3, some of the shape attributes and position/orientation attributes are viewpoint dependent, i.e. the values of those attributes are subject to occlusion. Therefore when adding a feature to the model we must take note whether the feature is occluded in the scene. For example, a surface region in an image may have been occluded by some other surfaces if any one of its boundaries is an occluded boundary; consequently, some of the attributes, say, area and centroid, of the region may not be accurate. If a feature in the image is found occluded, we must regard its viewpoint dependent attributes as "weak" attributes, meaning that their values will be overwritten or modified if a more complete version of the feature is detected again in any of the subsequent views of the object. In our current implementation, we do not explicitly mark attributes as "weak" when occlusions are detected; or, one might say, we treat every attribute as "weak." To explain, suppose from a given view the area of a surface has been extracted, and then if for the same surface a larger area becomes available in a subsequent view, the larger value will overwrite the earlier smaller value. Of course, there are attributes that are not amenable to this "overwrite" formula; more on this subject in the next section.

After the features extracted from the first range map are translated, pointers to them must be recorded on a feature sphere, or a set of features spheres if different classes of features are used. To accomplish this, the principal direction of each feature is calculated from its position/orientation attributes in accordance with the formulas presented in Chapter 3. From the principal direction of a feature, its corresponding cell on the sphere is found by using the *tessel–assignment* function, also described in Chapter 3.

4.5. Updating the Model

We will now discuss how the features extracted from a new view are used to update the partial model built from the previous views. This implies that the system must first decide whether a feature detected in the new view is indeed "new" to the partial model; if it is, the system should add the feature to the partial model, otherwise the new information on what is an aleady existent feature must somehow be merged with the old information.

The newness/oldness of a feature, extracted from the new view, with respect to the partial model can only be determined by comparing the feature with those already stored in the partial model. Clearly, this comparison of features is the same as the feature matching problem discussed in Chapter 3. Recall that for each view of the

object, the transformation that takes us from the world coordinates, in which the scene features are extracted, to the object-centered coordinates used for the feature spheres is known to the system, as it can be calculated from equation (4.1) or (4.2). Given the new feature, we first translate it to the object-centered coordinate frame via Tr^{-1} by using the formulas shown earler; we then compute its principal direction. This principal direction will correspond to a particular tessel on the feature spheres of the model. The neighborhood of this tessel is searched for any registered model features to answer the question whether the new feature is the same as one of the old features. Clearly, the size of this neighborhood should depend on the uncertainty in the computation of the principal direction, and every model feature in the neighborhood is a candidate for testing against the new scene feature. As in object recognition, feature comparisons are made on the basis of three criteria — shape, relation, position/orientation. If no candidate features can be found in the neighborhood on the feature spheres, or if all the candidate features in the neighborhood fail to match, the new feature is considered to be new information about the model, and is then added to the model as were the features during the model initiation stage in the first view.

On the other hand, if a feature extracted from the current view of the object can be matched to one of the features on the partial model, we must then decide what to do with the feature. Although, the simplest solution would be to totally ignore the new feature, one has to bear in mind the possibility that the new attribute values might be "superior" to the old values, in the sense that they might be more free of occlusion, or may be less distorted due to noise and other artifacts. We therefore need some mechanism for "combining" the old and the new attribute values in such situations.

For attributes that are viewpoint independent, the new and the old attribute values are best combined by taking an average of the two, assuming that we have as much confidence in the new attribute value as in the old. For example, if the attribute *radius* of a feature *cylindrical-surface* already exists in the partial model, and if from the current view a new value becomes available for this attribute, then, since this attribute is viewpoint independent, we should update the value of *radius* by averaging the two values.* Similar updating would have to be done for other viewpoint independent attributes like the normal of a planar surface or the position of a vertex.

An entirely different strategy is required for viewpoint dependent attributes. In this case, if both the partial model and the new values are unoccluded, the system takes an

* Of course, this is a very simple strategy that suffices when the same feature would not be seen in more than two or three views. For those features that might be visible in a large number of views, in any combination the new attribute values would have to given a weight that would be inversely proportional to the number of updates already made for that attribute value.

average of the two. On the other hand, if one of the values is occluded, we retain only the unoccluded one. And, if both are occluded, the system takes a weighted average of the two, each weight being proportional to the number of pixels visible to the sensor.

Note that the notion of averaging, weighted or unweighted, for updating the value of an attribute can only be applied to numerical attributes. For non-numerical attributes, how the new information is combined with the old is decided on a case-by-case basis. For example, when merging two sets of features that are the values of two adjacency attributes, we take the union of the two sets.

4.6. Experimental Results

This section will present an example for illustrating how the entire model learning process is carried out. The object used, displayed in Fig. 4.3, consists of thirteen planar faces and one conical surface. To give the reader a rough idea of the size of the object, the length and the width of the object are approximately 7" and 3.5", respectively; the height to the heighest point, in the middle of the object, is about 3". The model learning example discussed here generates all the surfaces from only the top views; in other words, the object was not flipped. For modeling the underside, the system used the default assumption that the underside was a planar surface, which in this case happens to be a fact. The system was commanded to take six views of the object, 60 degree apart. The range image of each view consisted of 85 scans with 0.1" scan resolution, meaning that between successive positions of the structured-light unit during a translation, the distance traveled by the unit along the y-axis of the world coordinate system was 0.1". Fig. 4.4(a-f) are the stripe images for the six views, and Fig. 4.5(a-f) are their segmented needle maps. The label of each segmented region is displayed near the center of the region. The features extracted from each view consist of primitive surfaces and vertices; the model representation will be based on these two classes of features. For clarity, we will focus our discussion mostly on surface features, though we will mention vertex features when relevant. Notice that region 5 of view 1, region 4 of view-2, region 2 of view-3, etc. are the face of the turntable so they are excluded from the model learning process.

4.6.1. Initiating the Model

Before we start to process the first view, an object-centered coordinate system is defined according to the procedure described in Section 3 with respect to the world coordinate system in which the structured-light scanner is calibrated. Two spherical arrays with frequency-of-geodesic-division equal to 16, which leads to 2562 cells on each sphere, are then created to represent two feature spheres, one for surface features,

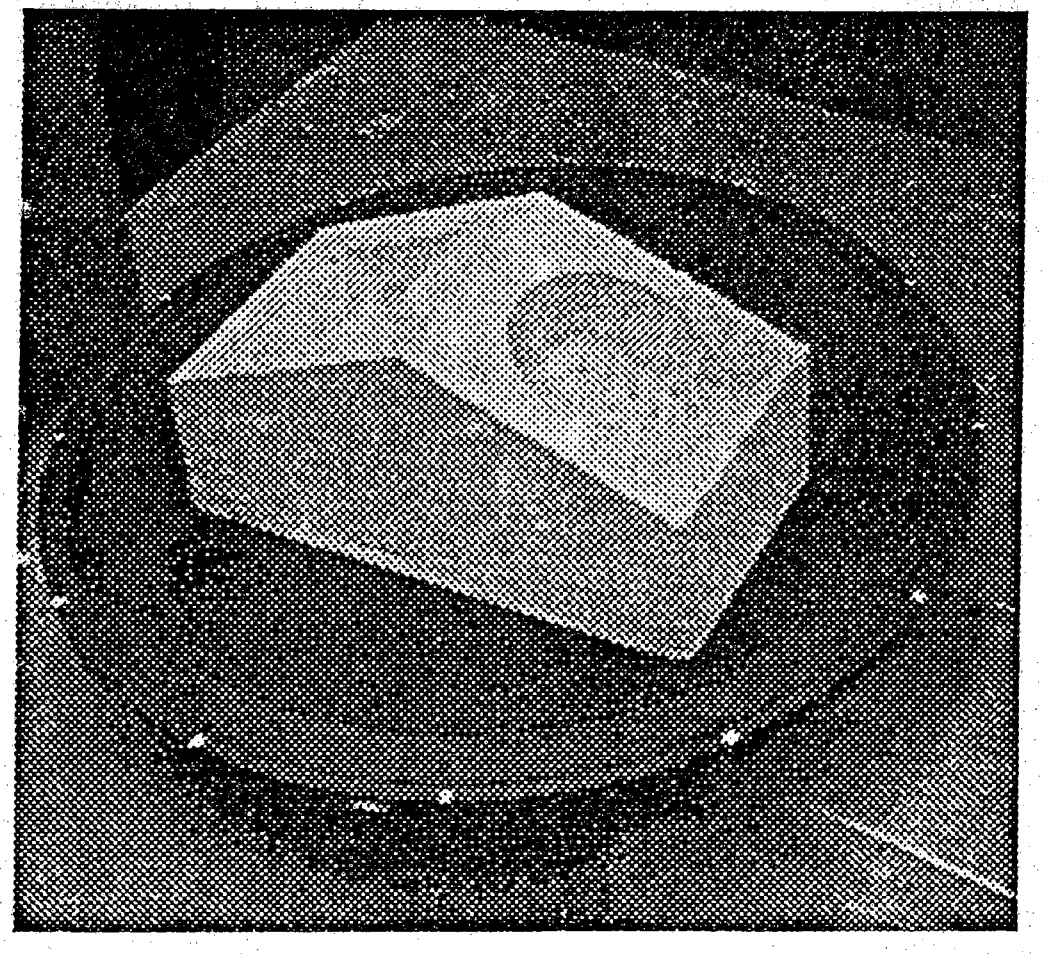
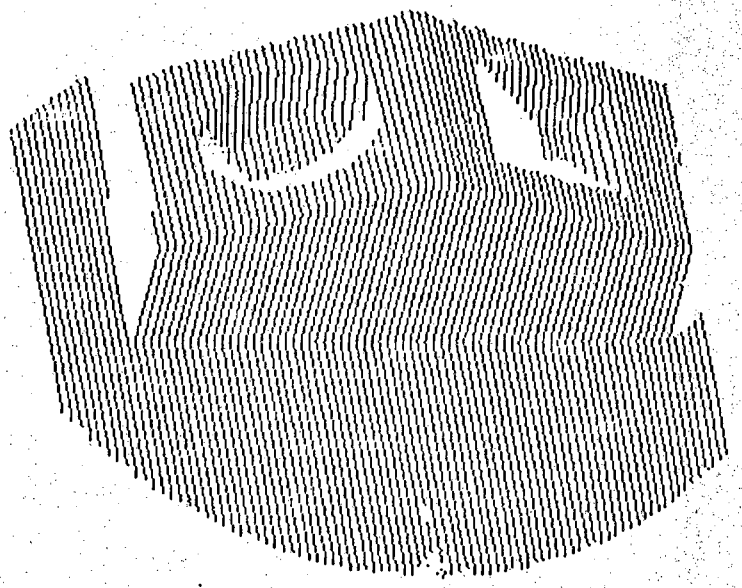


Figure 4.3. Object used for the model learning experiment discussed in the text.

(a)



(b)

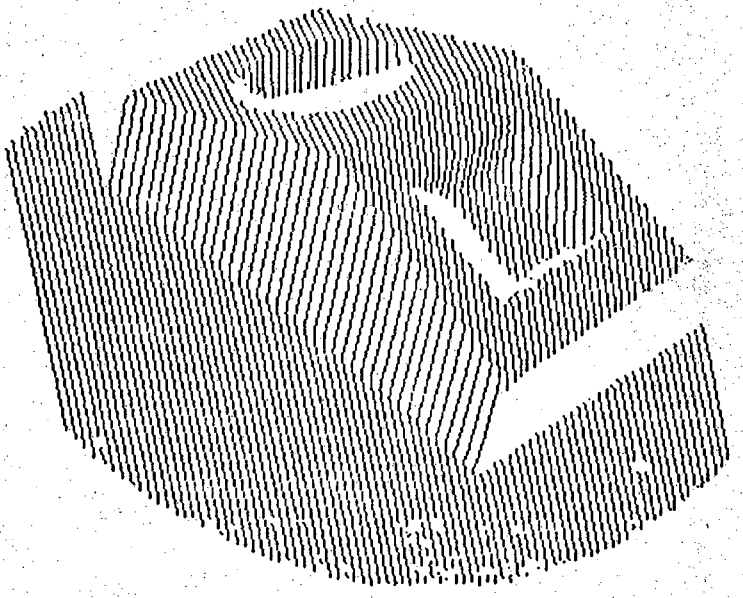


Figure 4.4. Stripe images of the six views of the object.

(c)

(d)

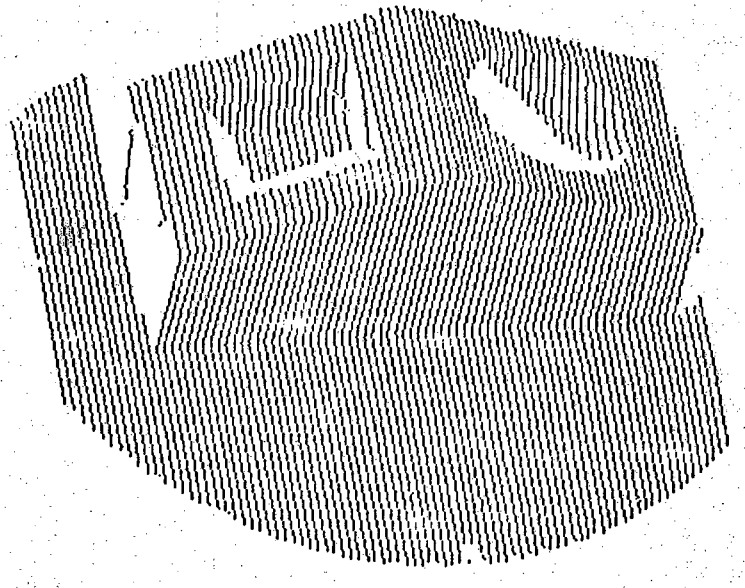
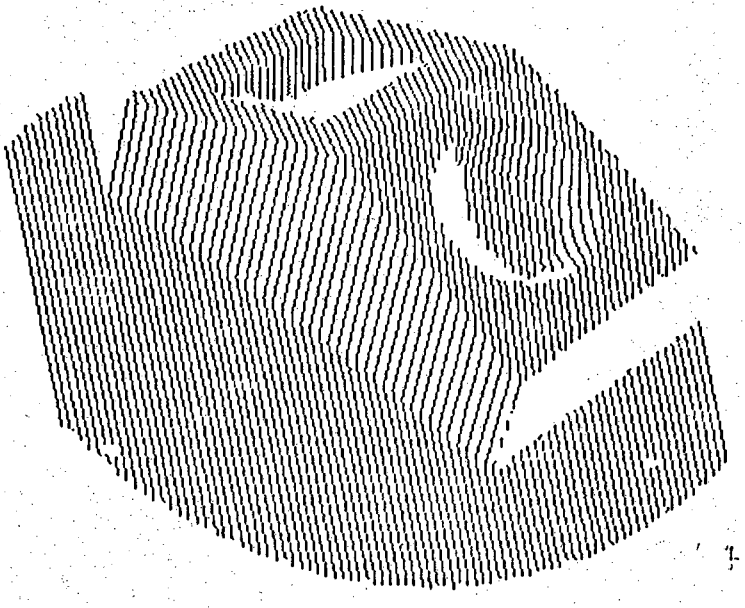


Figure 4.4. Continued

(e)



(f)

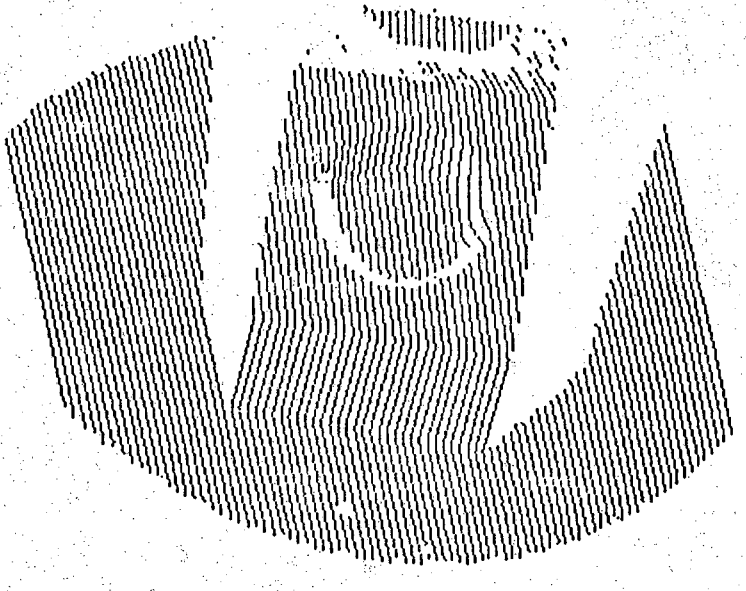


Figure 4.4. Continued

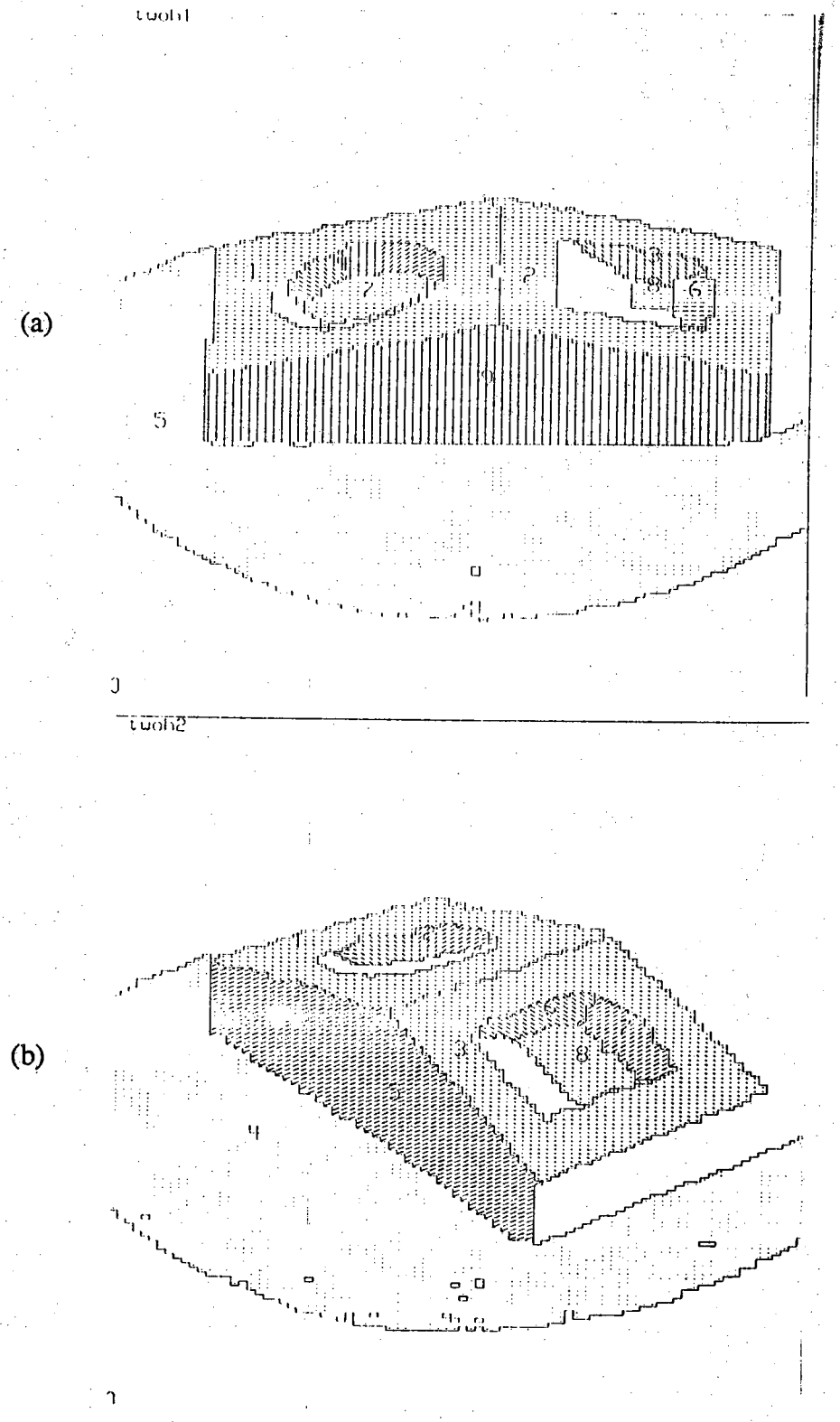
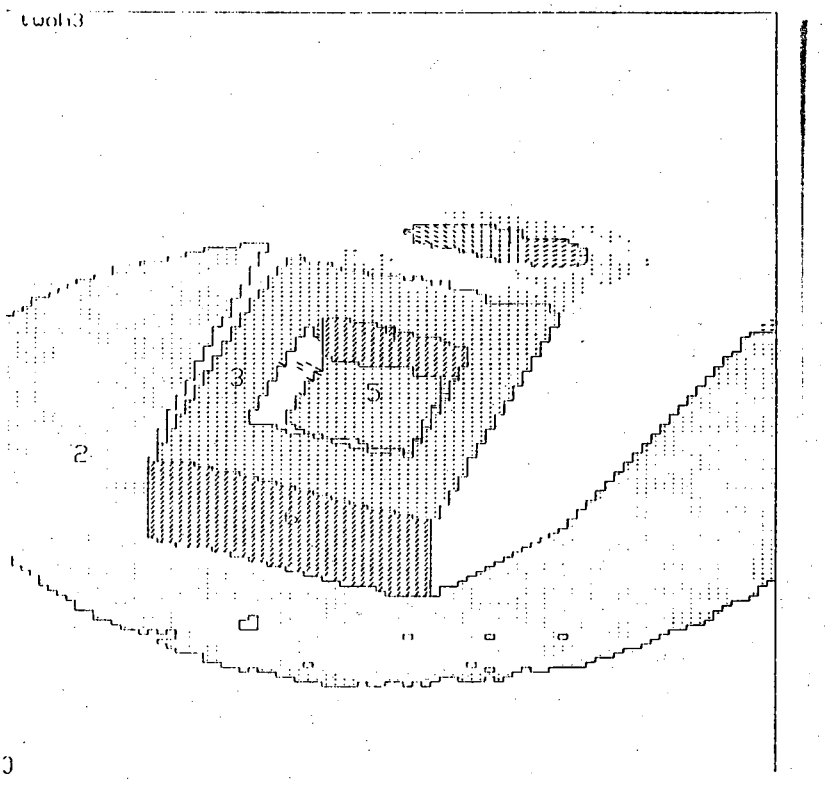


Figure 4.5. Segmented needle maps of the six views of the object.

(c)



(d)

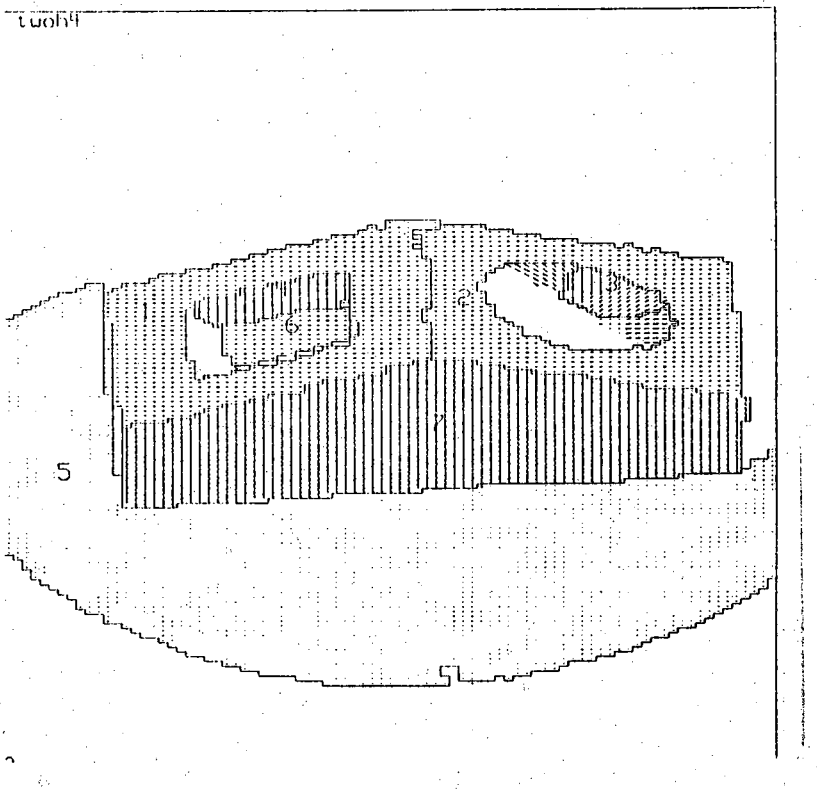
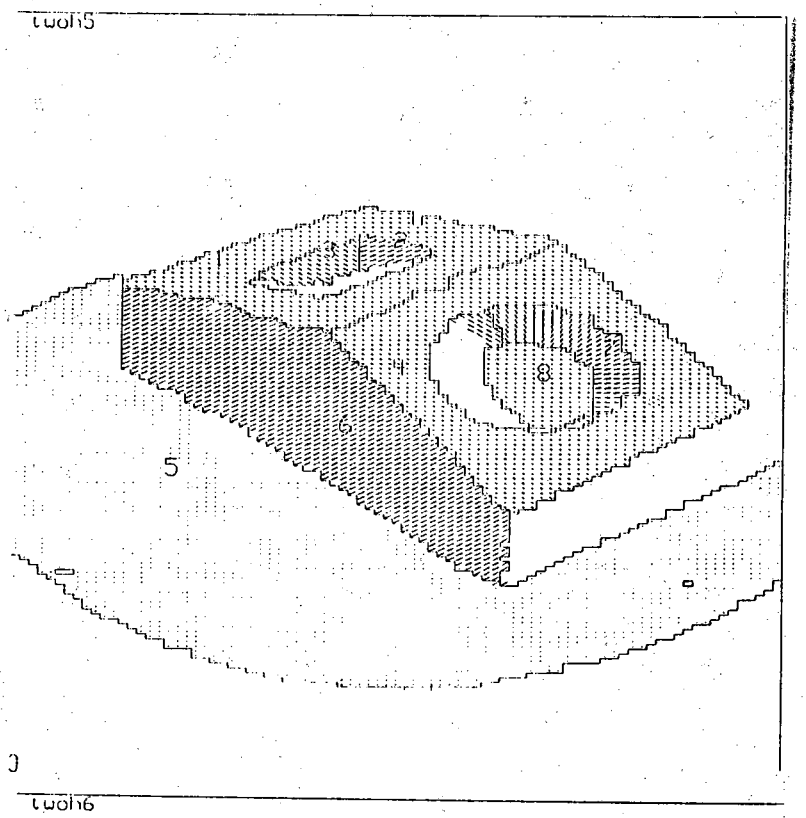


Figure 4.5. Continued

(e)



(f)

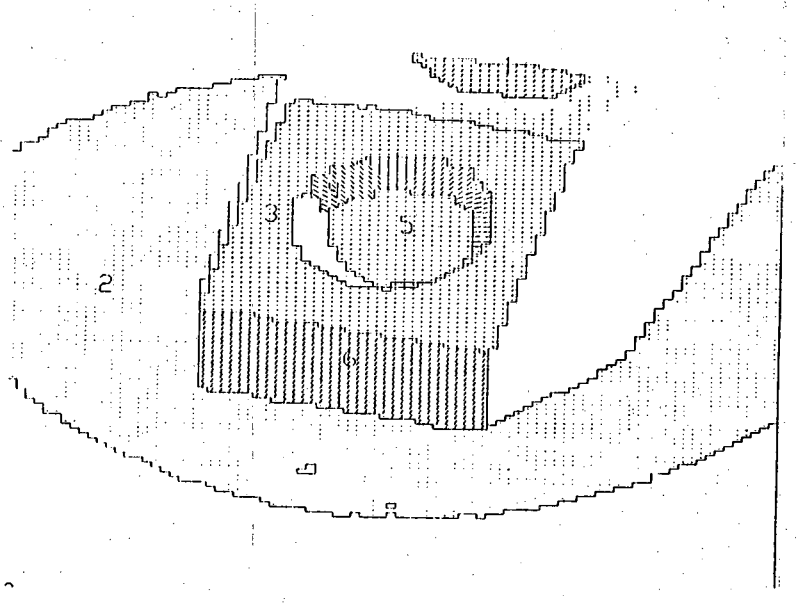


Figure 4.5. Continued

and the other for vertex features. The two feature spheres form the bases for feature matching in this model learning process. As shown in Fig 4.5-a, 9 regions and 16 vertices, the vertices are not labeled, are detected in the first view. Region 5, which is the face of the turntable, will not be considered as a surface feature of the object, so only 8 surface features are passed on to the learning process. Furthermore, we have chosen to disregard scene vertices formed between a curved surface and a planar/curved surface because they usually are spurious junctions caused by occlusion.*

As was mentioned in Chapter 3, each feature is represented in the computer memory by a frame data structure. For example, the attribute frame for the region 1 extracted from the first view is as follows:

Region 1
Type: planar
Number_of_pixels: 743
Number_of_adjacent_regions: 3
Adjacent_regions: (2 9 4)
Type_of_edge_with_adjacent_region: (convex convex convex)
Vertices_between_adjacent_regions: ((1, 2) (2, 3) (12, 11))
Normal: (-0.00503 -0.34456 0.93875)
Moment_direction: (0.33467 0.88405 0.32627)
Region_center: (-2.11130 17.15708 4.67800)

A couple of entries in the attribute set need clarification. To establish adjacency relationships between regions, the bounding contour of each region in a range map is traced in a clockwise direction and a record made of the common edges and vertices of a given region with other regions. During boundary tracing, note is also made of the start-vertex and the end-vertex when a common edge is found with another region. For example, the list (1, 2) in the value of the attribute *Vertices_between_adjacent_region*, 1 is the label of the start vertex of the common edge between regions 1 and 2; 2 is the label of the end vertex. Also, note that the nature of this common edge is *convex*. Another attribute

* Note that, in general, a vertex feature in a scene is defined either as a junction of three surfaces, or a junction of two surfaces and an occlusion. However, when an occlusion is involved and one or both of the surfaces meeting at a junction is curved, that vertex is ignored, because, usually, that is not a real vertex in the scene. Such false vertices become evident when, for example, a range map is made of a hole from above with the interior of the hole only partially visible.

that might bear some explanation is *moment_direction*; the direction refers to the direction of the line about which the moment of inertia is a minimum (in most cases, this is the direction along which the surface is most elongated). The attribute *Normal* applies, of course, to only planar surfaces, which is the case here. For, say, a cylindrical surface, instead of *Normal*, the relevant attribute would be *Axis*, whose value would be the direction of the axis of the cylinder.

In a similar vein, the attribute frame for vertex 1 extracted from the first view is:

Vertex 1
Position: (-3.9276 18.6921 5.2745)
Belongs_to_regions: (2 1 *)
Adjacent_vertices: (2 * *)
Edge_type: (convex * *)

In the example here, we made the assumption that exactly 3 surfaces had to meet at a vertex. Since it is likely that for a vertex not all the converging surfaces may be visible in a given view, we must leave place-holders for those that are not. This has been accomplished by the use of the symbol '*' in the above frame. Therefore, '*' denotes an uninstantiated attribute value.

The reader has probably noticed that adjacency information about surfaces, vertices, and edge-type of common edges is redundantly recorded on the attribute frames of the surface features as well as the vertex features. The reason for this is time efficiency, especially when it comes to the use of adjacency and edge-type information on vertex features for the generation of LFS's. The reader may recall from Chapter 3 that LFS's are used for hypothesis generation.

Now we will describe a step that is particular to model learning. While the attribute frames we showed above correspond to scene features in the object recognition discussed in Chapter 3, for the purpose of model learning each scene feature must either become a model feature or must be merged with one of the existing model features. Since, we are at this time discussing the first view of the object, all scene features become model features and are used to initiate the model. To convert a scene feature into a model feature, we must translate its position/orientation attributes from the world coordinate frame in which the data are taken into the object-centered frame in which the model is built. During this process of translating features into the object-centered coordinates, the features are also assigned new labels, this being done for purely cosmetic reasons. For example, initially, as shown in Fig. 4.5(a), the extracted surfaces

are labeled 1, 2, 3, 4, 6, 7, 8 and 9, with label 5 corresponding to the turntable. After dropping region 5, and translating the remaining surface features into the object-centered coordinates by multiplying the position/orientation vectors from the left by Tr_1^{-1}, the new surface labels as stored in the model become 1, 2,... , 8. As an example, model surface 1 has the following attribute frame:

Surface 1
Type: planar
Nomal: (-0.0050 -0.3446 0.9387)
Moment_direction: 0.3347 0.8841 0.3263
Region_center: 0.0137 -1.7699 0.8990
Number_of_adjacent_regions: 3
Adjacent_regions: (2 8 4)
....
....

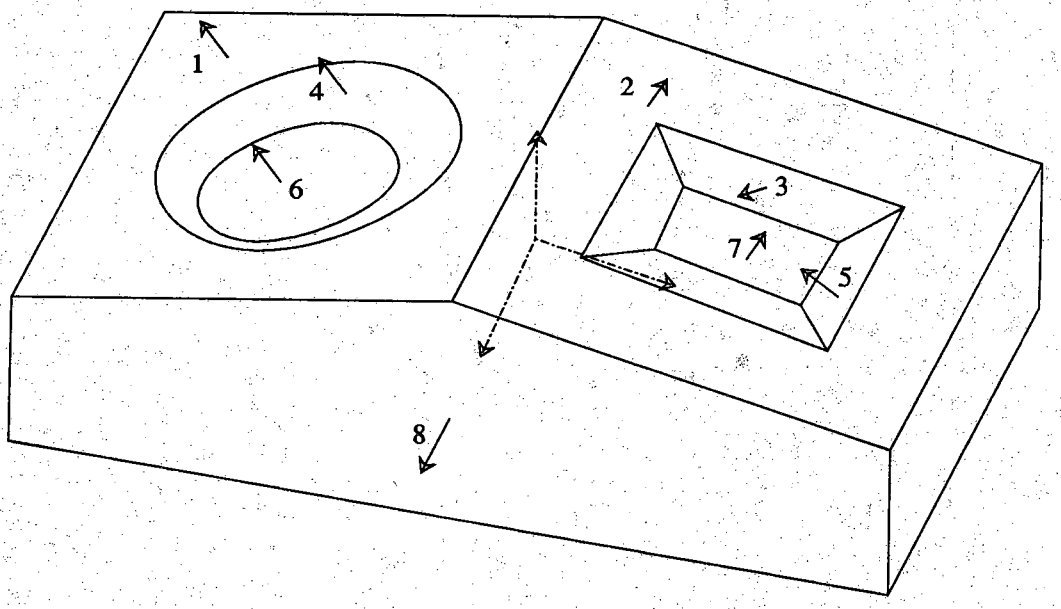
---

and corresponds to the scene surface 1 shown in Fig. 4.5(a). Similarly, all the vertex features extracted from the scene in the first view are transformed and relabeled.

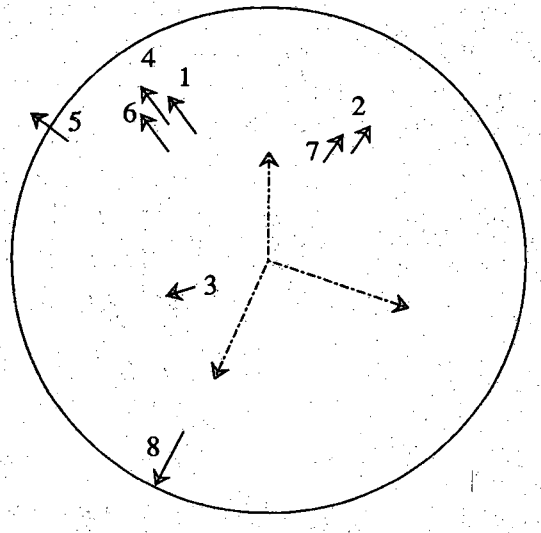
After the conversion of scene features into model features, the principal direction of each surface and vertex model feature is computed in the object-centered coordinate system. On the basis of the principal direction, each feature is assigned a pointer on the corresponding tessel on the appropriate feature sphere. Fig. 4.6 shows the surface feature sphere constructed from the features gleaned from the first view. To help the reader associate the different surfaces with the entries on the feature sphere, we have shown in (a) the different surfaces of the object and their labels as generated by processing the first view. Note that (a) is not a synthesized model, but only a means to transmit to the reader the surface-label association at the end of view 1.

4.6.2. Updating the Model

Now as each new view of the object is taken, we can use it to update the model initiated with the data from the first view and updated by all the previous views. If there are any common features between the new view and the partial model built so far, they must be discovered by matching. Of course, if there are no common features, then all we need do is to merely add the new features to the feature spheres built from the prior views. Consider, for example, view 2, which in the example under discussion is at an angle of 60° clockwise from view 1. Fig 4.5(b) shows that the range image for this view is segmented into 8 regions; except for region 4 each of these regions represents a surface features of the object visible from the view point corresponding to view 2. By



(a) The partial model built from view 1



(b) Surface feature sphere of the partial model

Figure 4.6. A partial model built from the features extracted from the first view. /" pn 159 for table

comparing with 4.5(a), one can immediately observe that region 6 is the only new surface feature seen in view 2, while the remaining 6 are seen in the first view and should already have been recorded in the partial model. The model learning process "learns" this facts by matching features in view 2 with the partial feature sphere constructed from view 1. This learning process consists of transforming the position/orientation information of each view 2 feature into the object-centered coordinate system via Tr_2^{-1}, and then computing the principal direction associated with the feature.* A small neighborhood on the feature sphere centered at the tessel corresponding to the computed principal direction is then searched for a compatible partial model feature. Currently, this neighborhood is of radius 2 tessels which corresponds to an allowable uncertainty of 8° in the principal direction.

Consider, for example, the region marked surface 5 in Fig. 4.5(b). On the basis of its principal direction, this surface is found to match the partial-model surface 8 shown in Fig. 4.6(a). For region 1 in Fig. 4.5(b), there are three candidate model features, these are marked 1, 4 and 6 in Fig. 4.6(a); all three of these partial-model features fall into the same tessel on the feature sphere, as depicted in Fig. 4.6(b). In this case, partial-model surface 4 excluded from further consideration on the basis of the surface types. Partial-model surface 6 is eliminated as a possible match for scene surface 1 on the basis of the values of the normal distances of the surfaces involved in the matching process. What is being said here is that if we take the dot product of normal to partial-model surface 6 with the position vector to any point on the surface 6, we will obtain the normal distance to the surface 6 (remember, that the point necessary for this calculation is stored as one of the attributes for planar surfaces). Now, if we carry out the same calculation for view-2 surface 1, the normal distance computed will be different from that calculated for partial-model surface 6, making the two unmatchable. So, ultimately, we can find that view-2 surface 1 must correspond to partial-model surface 1.

Continuing the above matching process with each of the features in view 2, we eventually conclude that surface 6 in Fig. 4.5(b) is new and was not seen in view 1. This surface is added to the partial model and given the label 10 (not shown in figures).

Now consider surface 8 in Fig. 4.5(b). It is found to match partial-model surface 7 shown in Fig. 4.6(a). However, the area of the surface in the second view is much larger than that in the first view. Therefore, in this case the process updating the partial model consists of overwriting the value of the attribute *area*. After view-2 surface 8 is merged

* Note that we could not have first computed the principal direction in the world coordinate frame and then transformed the resulting vector into the object-centered coordinate frame. The reason for this is that principal directions are defined with respect to the origin of the object-centered coordinate frame and therefore can only be computed in this frame.

with partial-model surface 7 in this manner, we recalculate the surface normal and the region center associated with the updated model surface 7. We must also update the adjacency information associated with the model surface 7, since the corresponding view-2 surface 8 was found adjacent to view-2 surface 6, which is now partial-model surface 10. This updating of adjacency information also takes place for partial-model surfaces 2 and 3. The vertex features are updated the same way as the surfaces.

Each of the remaining four views is used to update the partial model in the same manner. The final surface labels are depicted in Fig. 4.7. Note, as was the case with Fig. 4.6, this figure is only intended to help the reader associate labels with the surfaces of the object, the underlying object itself was not synthesized from the final model. Table 4.1 shows for each view the mappings from the view features to the partial-model features.

4.7. Discussions

In order to display the final result obtained by integrating all the six views, we derived a wire-frame representation of the object from the final feature sphere. This wire-frame exists in three dimensions and can be rotated for display. Two views of the wire-frame are shown in Figs. 4.8(a) and (b) together with vertex labels. Notwithstanding the fact that some of the object vertices came out disjointed in the images of the "learned model" shown in Figs. 4.8(a) and (b), the wire-frame is constructed readily from the vertex feature sphere. While the vertices 13 and 18 shown in Fig. 4.8 correspond to the same object vertex, they came out separated in the learned model because of occlusions. The same is true of the vertices 14 and 21 in Fig. 4.8. This difficulty could probably have been eliminated if we had used more views. It is important to realize that the wire-frame shown is used only for display and plays no role in any of the object recognition strategies, only feature spheres being used for that purpose.

The reader is probably curious about how we managed to show the curved edges in the wire-frame in Fig. 4.8. An ad hoc algorithm had to be written for this purpose and consisted of finding the intersection boundary of the planar and the conical surfaces, these two surfaces existing on the surface feature sphere. Note that the intersection of a cone and a plane forms an elliptic curve when the plan cuts through the cone, which is the case when the normal to the plane close to being parallel with the axis of the cone. The intersection would be hyperbolic when the plane makes a glancing cut of the cone. Since in our case the former condition is satisfied, we need to determine the parameters of the ellipse, from these parameters one could then generate a wire-frame representation of the ellipse. In general, a 3-D ellipse is described by its center, the plane it lies

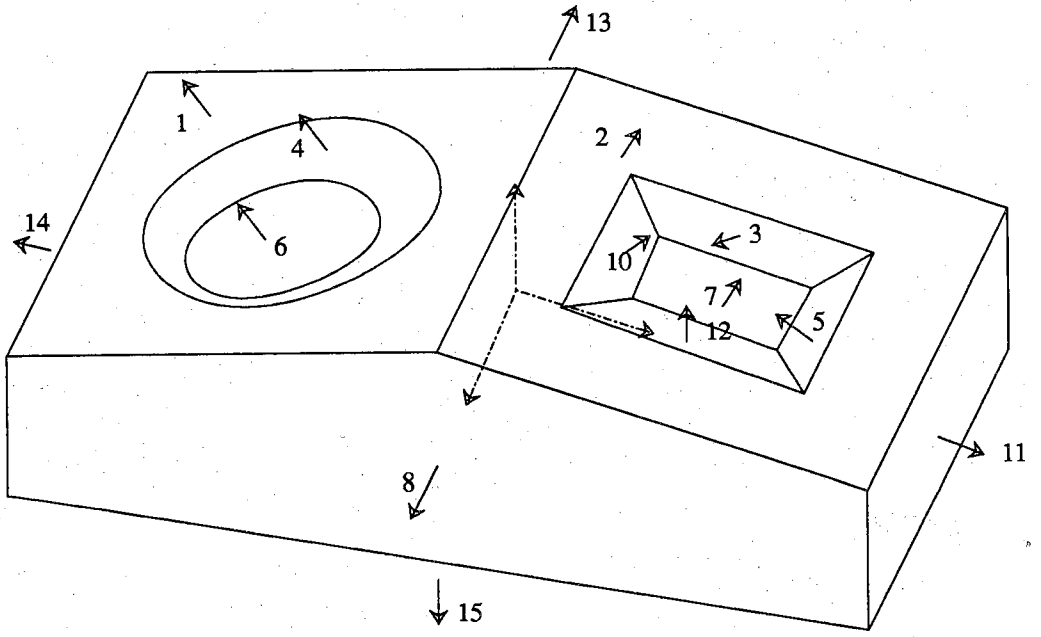


Figure 4.7. The completely built model using all six views of the object.

Table 4.1 This table shows mappings from surface labels in each view to the labels used for partial-model surfaces.

view	mapping from scene features to partial-model features
view 1	1 -> 1, 2 -> 2, 3 -> 3, 4 -> 4, 6 -> 5, 7 -> 6, 8 -> 7, 9 -> 8
view 2	1 => 1, 2 -> 9, 3 => 2, 5 => 8, 6 -> 10, 7 => 3, 8 => 7
view 3	1 => 9, 3 => 2, 4 => 10, 5 => 7, 6 -> 11
view 4	1 => 2, 2 => 1, 4 -> 12, 6 => 7, 7 -> 13
view 5	1 => 2, 2 => 12, 3 => 5, 4 => 1, 6 => 13, 7 => 4, 8=> 6
view 6	1 => 5, 3 => 1, 4 => 4, 5 => 6, 6 -> 14

NOTES:

For the first view, the view-1 label 5 corresponds to the turntable and therefore has no mapping to the partial model. The turntable labels for other views are also eliminated from mapping. While in the first view, the partial- model labels correspond mostly to the surface labels in the scene, for subsequent views, any matchings found between the surfaces in the view and surfaces in the partial-model determine the mappings shown in the table. Of course, when there are no matches, new labels must be used for scene features. The distinction between the two is brought out by the use of single-stemmed (->) and double-stemmed (=>) arrows, the former corresponding to the case when a new partial-model labels must be used for a scene surface, and the latter to the case when partial model label used is decided by the existence of a match.

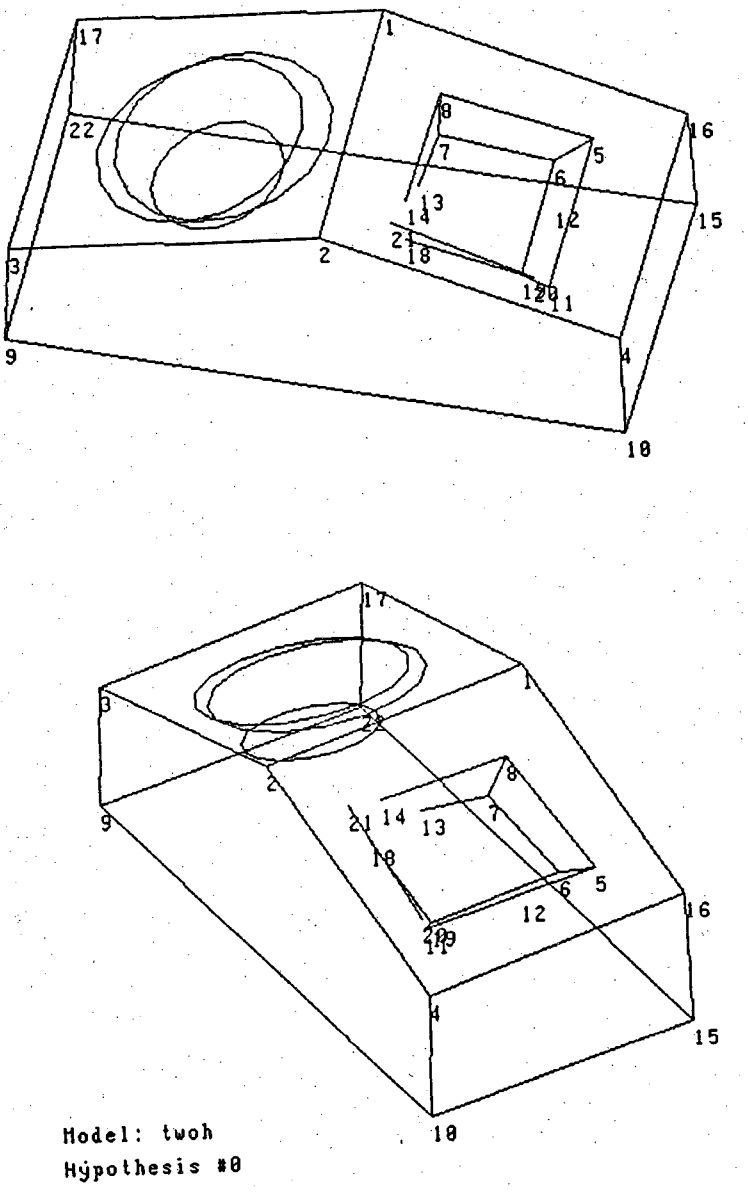


Figure 4.8. Two views of a wire-frame representation derived from the built model.

on, its long axis, and its two radii. Clearly, the plane the ellipse lies on is the plane making the intersection, and the center of the ellipse is the intersection point of the axis of the cone and the plane. The long axis of the ellipse lies on a plane surface that is formed by the normal to the intersecting plane and the axis of the cone; at the same time, the long axis of the ellipse is perpendicular to the normal to the intersecting plane. To determine the two radii, we first compute the length from the apex of the cone to the center of the ellipse, and the angle of intersection; the angle of intersection is defined as the angle between the normal to the intersecting plane and the axis of the cone. Then the two radii can be computed approximately from the length, the angle of the cone and the angle of the intersection. Given the parameters of the intersecting ellipse, the ellipse can be represented in a parametric form

$$\boldsymbol{p} = \boldsymbol{v}_l r_l \cos(\alpha) + \boldsymbol{v}_s r_s \sin(\alpha) + \boldsymbol{p}_o$$

where $\boldsymbol{p}$ is the position vector to a point on the ellipse, $\boldsymbol{v}_l$ and $\boldsymbol{v}_s$ the unit vectors along the major and the minor axes of the ellipse, r_l and r_s the two radii corresponding to the major and the minor axes, p_0 the position vector to the center of the ellipse, and, finally, α the angle for parametrizing the ellipse equation. The parametric form is easily converted into a wire-frame representation by discretizing the angle α. For small enough intervals in α, the segments of the ellipse would be linear.

The final model, as shown in Fig. 4.8, consists of 15 surfaces and 22 vertices. Note that the 22 vertices include only those that exist on the vertex feature sphere; the artificial vertices introduced to give wire frame representations to curved edges are not included. The number of surfaces in the generated model is one more than the number of surfaces on the object. The extra surface in the model corresponds to the region labeled 2 in view-2 image shown in Fig. 4.5(b). When the partial model is updated with view-2, this region is not recognized to be the same as region 4 in view-1 shown in Fig. 4.5(a). The reason for this mismatch is that in view-2 it is not possible to obtain an accurate estimation of the direction of the axis of the conical surface. In other words, the cone axis direction computed for region 2 in view-2 is too different from the direction of the cone axis for region 4 in view-1. As a result, when the system sees region 2 in view-2, it treats the region as a new feature, the pointer to this feature residing in the tessel corresponding to the computed axis direction.

The original object had only 18 vertices, however our model found 22. These extra vertices are quite visible in Fig. 4.8 and correspond to locations where object vertices appear disjointed. As was mentioned before, these extra vertices are caused by occlusion. In Fig. 4.5(b), for example, there is a vertex formed at the junction of regions 6, 8 and the occluded region. (Note that in analyzing a scene, a vertex is defined as a junction formed by either three surfaces, or two surfaces and an occluded region.)

Evidently, this vertex is false since in a different view the occlusion present could shift the location of this junction. We do do some topological reasoning to replace such spurious vertices with real vertices during the updating process. What is being said here is that if for the vertex formed by regions 6, 8 and occlusion we could in a later view discover all three surfaces meeting at the real vertex, then the false vertex would be replaced by the real vertex. Although, this reasoning was able to eliminate some of the false vertices, it proved not be effective for some, including the one formed by the junction of regions 6, 8 and occlusion in Fig. 4.5(b), because the three surfaces meeting in the vicinity of that point are never visible simultaneously in any of the views used.

Clearly, the number of views used must be such that the resulting model is toplogically consistent. Strictly speaking, because of disjointed vertices the model in Fig. 4.8 is not topologically consistent. Future research is planned to examine the generated models for their topological correctness. If a learned model is found to be incorrect, that should initiate a finer sampling of the viewpoint space.

For future research, one must also bear in mind that topological consistency while necessary may not be sufficient for a learned model to represent a real object -- the condition of geometric consistency must also be satisfied. For example, as illustrated by the truncated pyramid example shown in [*Mc*−82], a three dimensional entity may be topologically consistent, yet not geometrically so. Our future research will also aim at discovering what reasoning strategies should be implemented for making checks on geometric consistency.

It is important to realize that the learned model in Fig. 4.8, despite all its deficiencies, is adequate for object recognition in difficult and cluttered scenes, such as the one shown in Fig. 3.1 in Chapter 3. The data driven nature of our recognition strategies makes them forgiving of small errors in the model information. To remind the reader again, in a data-driven approach we select a feature from the scene at a time and then try to confirm its presence on a model feature sphere. Suppose, an extra vertex or an extra bit of a surface appeared on the model feature sphere, it may not necessarily pose any difficulties, depending, of course, upon how much discrimination is required between different object models. The accuracy requirements on model generation as a function of the discriminatory power of a 3-D object recognition strategy is yet another avenue for future research.

CONCLUSIONS

This report presented the 3D-POLY system for object recognition and model learning. The report addressed the four main issues listed in the Introduction in connection with our discussion there on the design of a robot vision system.

We mathematically analyzed the process of structured light imaging. The result of this analysis was a novel procedure for the calibration of structured light equipped robots; the procedure yields in a straightforward manner a calibration matrix that directly converts the image coordinates of an illuminated point in the scene into its world coordinates.

The report presented in Chapter 3 a hypothesis generation and verification strategy whose complexity possesses a low polynomial bound for single object recognition. It is important to realize that the manner in which identity and pose hypotheses are formed and verified is independent of what types of features are used for describing objects. The features described in Chapter 3 and currently used in 3D-POLY merely serve to illustrate how our hypothesis generation and verification scheme should be used. It is very likely that the types of features we have discussed may not be appropriate to industrial objects with shiny metallic surfaces, since the surfaces on such objects can not be easily imaged with structured light scanners. It is possible that for such objects a recognition strategy should be solely based upon lower level features such as vertices and edges and should not employ surface type features.

An important key to hypothesis verification in 3D-POLY is the use of spherical data structures, these data structures were used to store pointers to feature frames on the basis of the principal directions associated with the features. Of course, we could not have used this data structure had we not been able to present constant time algorithms for finding neighborhood over spherical tesselations. We believe the algorithms we presented in connection with this data structure will also prove useful in other situations where spherical representations are needed, for example, for aspect graphs for objects, used primarily for grouping topologically similar viewpoints, and for the Hough space for representing orientations.

LIST OF REFERENCES

[*Ag*–85] G. J. Agin, "Calibration and use of a light stripe range sensor mounted on the hand of a robot," The Robotics Institute, Carnegie-Mellon University, Tech. Rep. CMU-RI-TR-85-20, 1985.

[A&B-73] G. J. Agin and T. O. Binford, "Computer description of curved objects," in *Proc. 3rd Intl. Joint Conf. on Artificial Intell.*, pp. 629-640, 1973.

[*A&H*–82] G. J. Agin and P. T. Highnam, "Movable light-stripe sensor for obtaining three-dimensional coordinate measurements," in *Proc. SPIE Int. Tech. Symp.*, San Diego, CA, Aug 21-27, pp. 326-333, 1982.

[A&et-87] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-D point sets," *IEEE Trans. Pattern Analysis and Machine Intell.*, Vol. 9, No. 5, pp. 698-700, 1987.

[*Ay*–67] R. Ayres, Jr., *Projective Geometry,* Schaum Publishing Co., 1967.

[Ba-77] H. H. Baker, "Three-dimensional modeling," in *Proc. 5th Intl. Joint Conf. on Artificial Intell.*, pp. 649-655, 1977.

[B&H-87] B. Bhanu and C. C. Ho, "CAGD-based 3-D object representation for computer vision," *IEEE Computer,* Vol. 20, No. 8, pp. 19-36, August 1987.

[B&B-82] D. H. Ballard and C. M. Brown, *Computer Vision,* Englewood Cliffs, NJ: Prentice-Hall, 1982.

[Be-88] P. J. Besl, "Range imaging sensors," Research publication, CS Dept. General Motors Research Lab, GMR-6090, 1988.

[B&J-88] P. J. Besl and R. C. Jain, "Segmentation through variable-order surface fitting," *IEEE Trans. Pattern Analysis and Machine Intell.*, Vol. 10, No. 2, pp. 167-192, 1988.

[B&J-86] —, "Invariant surface characteristics for 3D object," *Computer Vision, Graphics, and Image Processing,* 33, pp. 33-80, 1986.

[B&J-85] —, "Three-dimensional object recognition," *Computing Survey,* Vol. 17, No. 1, pp. 75-145, March 1985.

[Bh-84] B. Bhanu, "Representation and shape matching of 3-D objects," *IEEE Trans. Pattern Analysis and Machine Intell.*, Vol. 6, No. 3, pp. 340-350, 1984.

[B&H-84] S. D. Blostein and T. S. Huang, "Estimating 3-D motion from range data," in *Proc. 1st Conf. Artificial Intelligence Application,* pp. 246-250, Dec 1984.

[B&C-82] R. C. Bolles and R. A. Cain, "Recognizing and locating partially visible objects: the local-feature-focus method," *Intl. Journal of Robotics Research,* Vol. 1, No. 3, pp. 57-82, 1982.

[B&F-81] R. C. Bolles and P. Horaud, "A RANSAC-based approach to model fitting and its application to finding cylinders in range data," in *Proc. 3rd Intl. Joint Conf. on Artificial Intell.,* pp. 637-643, 1981.

[B&H-86] R. C. Bolles and P. Horaud, "3DPO: a three dimensional part orientation system," *Intl. Journal of Robotics Research,* Vol. 5, No. 3, pp. 3-26, Fall 1986.

[Bo-84] B. A. Boyter, "Three-dimensional matching using range data," in *Proc. 1st Conf. Artificial Intelligence Application,* pp. 221-216, Dec 1984.

[Br-83] R. A. Brooks, "Model-based three-dimensional interpretations of two-dimensional images," *IEEE Trans. Pattern Analysis and Machine Intell.,* Vol. 5, No. 2, pp. 140-149, 1983.

[Br-79] C. M. Brown, "Fast display of well-tessellated surfaces," *Computer & Graphics,* Vol. 4, pp. 77-85, 1979.

[C&F-82] I. Chakravarty and H. Freeman, "Characteristic views as a basis for three-dimensional object recognition," in *Proc. SPIE Conf. on Robot Vision,* Vol. 336, pp. 37-45, 1982.

[C&K-87] C. H. Chen and A. C. Kak, "Modeling and calibration of a structure light scanner for 3-D robot vision," in *Proc. IEEE Intl. Conf. on Robotics and Automation,* pp. 807-815, Apirl 1987.

[C&D-86] R. T. Chin and C. R. Dyer, "Model-based recognition in robot vision," *Computing Survey,* Vol. 18, No. 1, pp. 68-108, March 1986.

[C&H-83] R. T. Chin and C. A. Harlow, "Automated inspection of printed circuit board: a survey," *IEEE Trans. Pattern Analysis and Machine Intell.,* Vol. 4, No. 6, pp. 557-573, 1983.

[*Da*–82] C. A. Dane III, "An object-centered three-dimensional model builder," Ph.D. dissertation, Computer and Information Science, Univ. of Pennsylvania, 1982.

[*D&H*–73] R. O. Duda and P. E. Hart, *Pattern Recognition and Scene Analysis,* New York: Wiley. pp. 379-423, 1973.

[F&et-88] T.J. Fan, F. Medoni and R. Nevatia, "Matching 3-D objects using surface descriptions" in *Proc. IEEE Intl. Conf. on Robotics and Automation,* pp. 1400-1406, Apirl 1988.

[F&H-86] O. D. Faugeras and M. Hebert, "The representation, recognition, and locating of 3-D objects," *Intl. Journal of Robotics Research,* Vol. 5, No. 3, pp. 27-52, 1986.

[F&H-83] —, "A 3-D recognition and positioning algorithm using geometrical matching between primitive surfaces," in *Proc. 8th Intl. Joint Conf. on Artificial Intell.,* pp. 996-1002, 1983.

[F&et-83] O. D. Faugeras, M. Hebert and E. Pauchon, "Segmentation of range data into planar and quadric patches," in *Proc. 3rd Computer Vision and Pattern Recognition Conf.*, pp. 8-13, 1983.

[F&D-84] G. Fekete and L. S. Davis, "Property spheres: a new representation for 3-D object recognition," *IEEE workshop on Computer Vision,* pp. 192-201, 1984.

[Go-83] C. Goad, "Special purpose automatic programming for 3D model-based vision," in *Proc. of the DARPA Image Understanding Workshop,* pp. 94-104, June 1983.

[G&L-87] W. E. L. Grimson and T. Lozana-Perez, "Localizing overlapping parts by searching the interpretation tree," *IEEE Trans. Pattern Analysis and Machine Intell.*, Vol. 9, No. 4, pp. 469-482, 1987.

[G&L-84] —, "Model-based Recognition and localization from sparse range or tactile data," *Intl. Journal of Robotics Research,* Vol. 3, No. 3, pp. 3-35, Fall 1984.

[Ha-69] W. R. Hamilton, *Elements of Quaternions,* New York: Chelsea Publishing Co., 1969.

[H&H-87] C. Hansen and T. Henderson, "CAGD-based Computer Vision," *IEEE workshop on Computer Vision.* pp. 100-105, 1987.

[H&P-82] M. Hebert and J. Ponce, "A new method for segmentating 3-D scenes into primitives," in *Proc. 6th Intl. Conf. on Pattern Recognition,* pp. 836-838, 1982.

[H&K-86] M. Herman and T. Kanade, "Incremental Reconstruction of 3-D scenes from multiple, complex images," *Artificial Intelligence,* 30, pp. 289-341, 1986.

[H&J-87] R. Hoffman and A. K. Jain, "Segmentation and classification of range images," *IEEE Trans. Pattern Analysis and Machine Intell.*, Vol. 9, No. 5, pp. 608-620, 1987.

[Ho-84] B. K. P. Horn, "Extended Gaussian Image," *Proceeding of IEEE,* Vol. 72, No. 12, pp. 1671-1686, 1984.

[H&et-88] S. A. Hutchinson, R. L. Cromwell, and A. C. Kak, "Planning sensing strategies in a robot work cell," in *Proc. IEEE Intl. Conf. on Robotics and Automation,* pp. 1068-1075, 1988.

[Ik-87] K. Ikeuchi, "Generating an interpretation tree from a cad model for 3D-object recognition in bin-picking tasks," *Intl. Journal of Computer Vision.* Vol. 1, No. 2, pp. 145-165, 1987.

[Ik-83] —, "Determining attitude of object form needle map using extended gaussian Image," MIT AI Lab Memo No. 714, April, 1983.

[Ja-88] R. C. Jain, personal communication, September, 1988.

[Ka-85] A. C. Kak, "Depth perception for robots," in *Handbook of Indusrical Robotics,* Edited by S. Y. Nof, New York: Wiley, pp. 272-319, 1985.

[K&et-87] A. C. Kak, A. J. Vayda, R. L. Cromwell, W. Y. Kim, and C. H. Chen, "Knowlege-based robotics," in *Proc. IEEE Intl. Conf. on Robotics and Automation,* pp. 637-646, 1987.

[Ke-76] H. Kenner, *Geodesic math and how to use it,* University of California Press, 1976.

[K&J-86] T. F. Knoll and R. C. Jain, "Recognizing partially visible object using feature indexed hypothesis," *IEEE Journal of Robotics and Automation,* RA-2, No. 1, pp. 3-13, 1986.

[K&D-87] M. R. Korn and C. R. Dyer, "3-D multiview object representations for model-based object recognition," *Pattern Recognition,* Vol. 20, No. 1, pp. 91-103, 1987.

[L&W-88] Y. Lamdan and H. J. Wolfson, "Geometric hashing: a general and efficient model-based recognition scheme," in *Proc. 2nd Intl. Conf. on Computer Vision,* pp. —, Dec. 1988.

[Ma-82] D. Marr, *Vision,* New York: Freeman, 1982.

[Mc-82] A. K. Mackworth, "Reasoning about surface orientations," in *The Handbook of Artificial Intelligence,* edited by P. Cohen & E. Feigenbaum, Vol. 3. pp. 173-182, Heuristech, Stanford, CA, 1982.

[*M&et*-81] H. A. Martin, J. R. Birk and R. B. Kelley, "Camera models based on data from two calibration planes," Computer Graphics and Image Processing, Vol 17, pp. 173-180, 1981.

[M&A-83] W. N. Martin and J. J. Aggarwal, "Volumetric description of objects form multiple views," *IEEE Trans. Pattern Analysis and Machine Intell.,* Vol. 5, No. 2, pp. 150-158, 1983.

[M&B-80] D. L. Milgram and C. M. Bjorklund, "Range image processing: planar surface extraction," in *Proc. 5th Intl. Conf. on Pattern Recognition,* pp. 912-919, 1980.

[M&et-84] S. Mori, K. Yamamoto, and M. Yasuda, "Research on machine recognition of handprinted characters," *IEEE Trans. Pattern Analysis and Machine Intell.,* Vol. 6, No. 6, pp. 386-405, 1984.

[N&M-79] M. Nagao and T. Matsuyama, "Edge preserving smoothing," *Computer Graphics, and Image Processing,* 9, pp. 394-407, 1979.

[N&B-77] R. Nevatia and T. O. Binford, "Description and recognition of curved objects," *Artificial Intelligence, Vol. 8, No. 1,* pp. 77-98, 1977.

[O-66] B. O'Neill, *Elementary Differential Geometry,* New York: Academic, 1966.

[O&S-83] M. Oshima and Y. Shirai, "Object recognition using three-dimensional information," *IEEE Trans. Pattern Analysis and Machine Intell.,* Vol. 5, No. 4, pp. 353-361, 1983.

[P&M-86] B. Parvin, and G. Medioni, "Segmentation of range images into planar surfaces by split and merge" in *Proc. 6th Computer Vision and Pattern*

Recognition Conf., pp. 415-417, 1986.

[Po-83] M. Potmesil, "Generating models of solid objects by matching 3D surface segments," in *Proc. 8th Intl. Joint Conf. on Artificial Intell.*, pp. 1089-1903, 1983.

[Pr-87] H. Printz, "Finding the orientation of a cone or cylinder," *IEEE workshop on Computer Vision*. pp. 94-99, 1987.

[Pu-76] A. Pugh, *Polyhedra: a visual approach,* Univ. of California Press, Berkeley, CA, 1976.

[Re-80] A. A. Requicha, "Representations for rigid solids: theory, methods, and systems" *Computing Surveys,* Vol. 12, No. 4, pp. 437-465, 1980.

[R&K-82] A. Rosenfeld and A. Kak, *Digital Picture Processing,* Vols. 1 and 2, New York: Academic, 1982.

[S&J-84] I. K. Sethi and S. N. Jayaramamurthy, "Surface classification using characteristic contour," in *Proc. 8th Intl. Conf. on Pattern Recognition,* pp. 438-440, 1984.

[S&H-81] L. G. Shapiro and R. M. Haralick, "Structural descriptions and inexact matching," *IEEE Trans. Pattern Analysis and Machine Intell.*, Vol. 3, No. 5, pp. 504-519, 1981.

[S&et-84] L. G. Shapiro, J. D. Moriaty, and R. M. Haralick, "Matching three-dimensional objects using relational paradigm" *Pattern Recognition,* Vol. 17, No. 4. pp. 385-405, 1984.

[Sh-87] Y. Shirai, *Three-Dimensional Computer Vision,* Springer-Verlag, Berlin, 1987.

[S&B-85] W. E. Snyder and G. Bilbro, "Segmentation of three dimensional images," in *Proc. IEEE Intl. Conf. on Robotics and Automation,* pp. 396-403, 1985.

[St-87] G. Stockman "Object recognition and localization via pose clustering" *Computer Vision, Graphics, and Image Processing,* 40, pp. 361-387, 1987.

[Su-79] K. Sugihara "Range-data analysis guided by a junction dictionary," *Artificial Intelligence,* Vol. 12, No. 1, pp. 41-69, 1979.

[*T&K*–84] F. Tomita and T. Kanade, "A 3D vision system, generating and matching shape descriptions in range images," in *Proc. 2nd Intl. Sym. of Robotics Research,* pp. 35-42, 1984

[*Ts*–86] R. Y. Tsai "An efficient and accurate camera calibration technique for 3D machine vision," in *Proc. Computer Vision and Pattern Recognition Conf.*, pp. 364-374, 1986

[U&C-75] S. A. Underwood and C. L. Coates, "Visual learning from multiple views," *IEEE Trans. Comput.*, Vol 24, No. 6, pp. 651-661, 1975.

[W&L-83] A. K. C. Wong and S. W. Lu, "Representation of 3-D objects by attributed hypergraphs for computer vision," in *Proc. Intl. Conf. on Sys. Man Cybern.*, pp. 49-53, 1983.

[X&C-88] S.-E. Xie and T. W. Calvert "CSG-EESI: a new solid representation scheme and a conversion expert system" *IEEE Trans. Pattern Analysis and Machine Intell.*, Vol. 10, No. 3, pp. 221-234, 1988.

[X&C-86] —, "Constructing 3-D models of a scene from planned multiple views," in *Proc. SPIE Intelligent Robots and Computer Vision,* Vol. 726, pp. 233-239, 1986.

[Y&K-89] H. S. Yang and A. C.Kak, "Edge extraction and labeling from structured light 3-D vision data," in *Selected Topics in Signal Processing,* ed.: S. Haykin, Prentice-Hall, Inc. pp. 148-192, 1989.

[Y&K-86] —, "Determination of the identity, position and orientation of the topmost object in a pile," *Computer Vision, Graphics, and Image Processing,* 36, pp. 229-255, 1986.

[Y&K-86] —, "Determination of the identity, position, and orientation of the topmost object in a pile: some further experiments" in *Proc. 1986 IEEE Intl. Conf. on Robotics and Automation,* pp. 38-48, 1986.

Appendix A
Determination of Transformation

In this appendix we will present the formulation for estimating the transformation that brings a set of model features into a corresponding set of scene features. The location/orientation attributes of the features will be used for this purpose. Clearly, only those location/attributes can be used that are viewpoint independent; implying that we should not use attributes like the surface centroid, mid-point of an edge, etc. Let us denote a position attribute of a feature by $\boldsymbol{p}$, which is a position vector, and a orientation attribute by $\boldsymbol{a}$, which is a direction vector (unit vector). If a scene feature S is matched to a model feature M, then under noise-free condition we should have

$$\boldsymbol{R} \cdot \boldsymbol{p}_m + \boldsymbol{t} = \boldsymbol{p}_s \tag{A.1}$$

$$\boldsymbol{R} \cdot \boldsymbol{a}_m = \boldsymbol{a}_s \tag{A.2}$$

where $\boldsymbol{p}_m$ and $\boldsymbol{p}_s$, and $\boldsymbol{a}_m$ and $\boldsymbol{a}_s$ are the corresponding location attributes and orientation attributes of the model feature and the scene feature, and $\boldsymbol{R}$ and $\boldsymbol{t}$ are the rotational and translational components of the transformation Tr, respectively. Note that both equation (A.1) and (A.2) are in vector form. We will assume that $\boldsymbol{R}$ is a 3×3 matrix and $\boldsymbol{t}$ a 3-vector. Although the following form will not be used explicitly in our work, the reader might find it informative to know that when a rigid body is rotated clockwise through an angle θ about an axis whose direction is given by the unit vector $\boldsymbol{n}$, the matrix $\boldsymbol{R}$ takes the form

$$\begin{bmatrix} n_x^2+\cos\theta(1-n_x^2) & n_x n_y(1-\cos\theta)-n_z\sin\theta & n_z n_x(1-\cos\theta)+n_y\sin\theta \\ n_x n_y(1-\cos\theta)+n_z\sin\theta & n_y^2+\cos\theta(1-n_y^2) & n_y n_z(1-\cos\theta)-n_x\sin\theta \\ n_z n_x(1-\cos\theta)-n_y\sin\theta & n_y n_z(1-\cos\theta)+n_x\sin\theta & n_z^2+\cos\theta(1-n_z^2) \end{bmatrix}$$

This matrix has three unknowns, the angle θ and two of n_x, n_y, and n_z, since the magnitude of $\boldsymbol{n}$ is unity.

1. Solution for the Transformation

Now given the correspondence between a set of scene features $\{S\}$ and a set of model features $\{M\}$, we want to determine (or estimate if noise is present) the $\boldsymbol{R}$ and $\boldsymbol{t}$. Without loss of generality we can assume the matching of $\{S\}$ to $\{M\}$ results in the correspondences between $\boldsymbol{p}_s^i$ and $\boldsymbol{p}_m^i$, $i = 1, \ldots, k$, and between $\boldsymbol{a}_s^j$ to $\boldsymbol{a}_m^j$, $j = 1,\ldots,l$. Note that k does not have to agree with l since the number of position attributes in each feature might be different from the number of orientation attributes. From equation (A.1) and (A.2), we now have

$$\boldsymbol{R} \cdot \boldsymbol{p}_m^i + \boldsymbol{t} = \boldsymbol{p}_s^i \tag{A.3}$$

for $i = 1,...,k$, and

$$\boldsymbol{R} \cdot \boldsymbol{a}_m^j = \boldsymbol{a}_s^j \tag{A.4}$$

for $j = 1,...,l$.

Since $\boldsymbol{R}$ is present in only equation (A.4) while both $\boldsymbol{R}$ and $\boldsymbol{t}$ are present in equation (A.3), it is natural to decompose the problem of solving Tr into two stages: first solve for $\boldsymbol{R}$ by using equation (A.4) and then solve for $\boldsymbol{t}$ by using equation (A.3).

A question that arises here is that under what conditions can we gaurantee a unique solution for $\boldsymbol{R}$ and $\boldsymbol{t}$. Let us first investigate the case of $\boldsymbol{R}$. Since each orientation vector $\boldsymbol{a}$ is a unit vector in 3-D space, it can be completely specified by two parameters. Consequently, each instance of equation (A.4) can provide two independent scalar equations in terms of $\boldsymbol{R}$. Furthermore, as mentioned in Section 3, a rotation $\boldsymbol{R}$ has three degrees of freedom. Therefore, in order to to completely solve $\boldsymbol{R}$ we need at least two instances of equation (A.4), i.e. two corresponding pairs of orientation vectors, providing that the two vectors are not linearly dependent (parallel orientation vectors will lead to linearly dependent equations). Given two equations of the type shown in (A.4), we will actually have four equations for the three unknown of $\boldsymbol{R}$. If the correspondences between the scene surface orientations and the model surface orientations are correct, then these four equations are not really independent because the orientation vectors must obey the following additional constraint:

$$\boldsymbol{a}_m^1 \cdot \boldsymbol{a}_m^2 = \boldsymbol{a}_s^1 \cdot \boldsymbol{a}_s^2$$

In other words, this constraint must be derivable from the four equations. In practice, this constraint is used to verify the accuracy of the surface correspondences prior to solving the equations.

If l, the number of orientation vectors in the correspondence, is greater than 2, then the corresponding orientation vectors must obey the following pairwise constraints: i.e.

$$\boldsymbol{a}_m^i \cdot \boldsymbol{a}_m^j = \boldsymbol{a}_s^i \cdot \boldsymbol{a}_s^j \qquad \text{for } \textit{all } i,j \leq l \tag{A.5}$$

So, suppose by matching scene surfaces with model surfaces we have set up l correspondences that satisfy the above costraints. Now, the question is what is the best way to solve the l vector equations of the type shown in (A.4) for the unknown $\boldsymbol{R}$. One could lump together all the l equations into the following composite form

$$\boldsymbol{R} \cdot [\boldsymbol{a}_m^1 \; \boldsymbol{a}_m^2 \; \cdots \;] = [\boldsymbol{a}_s^1 \; \boldsymbol{a}_s^2 \; \cdots \;]$$

which could be written in a more compact form as

$$\boldsymbol{R} \cdot \boldsymbol{A}_m = \boldsymbol{A}_s$$

leading to the following least squares solution for $\boldsymbol{R}$

$$\boldsymbol{R} = \boldsymbol{A}_s \boldsymbol{A}_m^T \left[\boldsymbol{A}_m \boldsymbol{A}_m^T \right]^{-1} \tag{A.6}$$

Supposedly, a correct least squares solution obtained in this manner should minimize the metric

$$\left[\boldsymbol{R}\boldsymbol{A}_m - \boldsymbol{A}_s \right] \left[\boldsymbol{R}\boldsymbol{A}_m - \boldsymbol{A}_s \right]^T \tag{A.7}$$

or, in other words, lead to a solution of the equation

$$\frac{\delta}{\delta \boldsymbol{R}} \left[\boldsymbol{R}\boldsymbol{A}_m - \boldsymbol{A}_s \right] \left[\boldsymbol{R}\boldsymbol{A}_m - \boldsymbol{A}_s \right]^T = 0 \tag{A.8}$$

Unfortunately, the solution represented by the equation (A.6) and the rationale leading up to it are faulty for the main reason that the metric in equation (A.7) is really not an error metric since it is a 3×3 matrix and not a scalar. What we really want to minimize is not what is shown in (A.6) but the following form

$$\boldsymbol{E}^2 = \sum_{j=1}^{l} | \boldsymbol{a}_s^j - \boldsymbol{R} \cdot \boldsymbol{a}_m^j |^2 \tag{A.9}$$

In the next subsection, following a derivation originally given by Faugeras and Hebert [*F&H*–83] we will show how an elegant solution to the minimization of $\boldsymbol{E}^2$ can be obtained by the use of quaternions. The reader should note that other methods also exist for solving equation (A.9); see, for example, [*A&et*–87, *G&L*–84].

We would like to make one more comment about the inappropriateness of (A.6) for the solution we desire. Even if the equation in (A.8) made sense, the least squares optimization would be with respect to all the nine elements of the matrix $\boldsymbol{R}$. Since these nine elements do not constitute independent variables -- in fact, there are only three independent variables involved amongst these nine elements -- the solution obtained may be entirely meaningless.

2. Estimation of the Rotation Matrix

We clearly want an $\boldsymbol{R}$ that would satisfy

$$\frac{\partial \boldsymbol{E}^2}{\partial \boldsymbol{R}} = 0 \quad |_{\boldsymbol{R} = \hat{\boldsymbol{R}}}$$

Following Faugeras and Hebert, we will use quaternions to represent rotations, and obtain a solution for $\boldsymbol{R}$ in the form of the principal eigenvectors of a matrix in terms of $\boldsymbol{a}_s$ and $\boldsymbol{a}_m$. A quaternion [*Ha*–69] is defined as

$$Q \equiv (s, v)$$

where s is a scalar, and v is a vector of 3 elements. The conjugate of a quaternion Q is denoted by $\overline{Q}$ and defined as

$$\overline{Q} \equiv (s, -v)$$

The multiplication of two quaternions Q and Q' is also a quaternion and given by

$$Q * Q' \equiv (ss' - v \cdot v',\ v \times v' + sv' + s'v)$$

From the above two definitions, we have

$$\begin{aligned} \overline{Q} * \overline{Q'} &= (ss' - (-v)\cdot(-v'),\ (-v)\times(-v') + s(-v') + s'(-v)) \\ &= (s's - v' \cdot v,\ -(v' \times v + sv' + sv')) \\ &= \overline{Q' * Q} \end{aligned}$$

Let us assume that the rotation expressed by R is carried out along an axis n and with angle θ. Then the rotation of a by R, given by the vector $R \cdot a$, can be represented by

$$(0, R \cdot a) = Q_R * (0, a) * \overline{Q_R} \tag{A.10}$$

where we now have quaternions on both sides and where

$$Q_R = (cos\frac{\theta}{2}, \sin\frac{\theta}{2} n).$$

and vector a has been written in the quaternion form $(0, a)$. It is easy to verify that

$$|Q_R|^2 \equiv Q_R * \overline{Q_R} = 1$$

By substituting (A.10) for $R \cdot a_m^j$ in equation (A.9), we obtain

$$\Sigma^2 = \sum_{j=1}^{l} |(0, a_s) - Q_R * (0, a_m) * \overline{Q_R}|^2$$

Note, for the simplicity of notation we have dropped the superscript j on vectors a_m and a_s. Since $|Q_R|^2 = 1$, for any quaternion Q_x, we have

$$\begin{aligned} |Q_x|^2 &= Q_x * \overline{Q_x} \\ &= Q_x * (1, 0) * \overline{Q_x} \\ &= Q_x * Q_R * \overline{Q_R} * \overline{Q_x} \\ &= Q_x * Q_R * \overline{Q_x * Q_R} \\ &= |Q_x * Q_R|^2 \end{aligned}$$

Therefore, we can post-multiply Q_R with both terms in the above equation and obtain

$$\Sigma^2 = \sum_j^l |(0, a_s)*Q_R - Q_R * (0, a_m)|^2 \tag{A.11}$$

Now we can minimize Σ^2, which has become a quadratic function of Q_R, with respect to

$$Q_R = (\alpha,\eta)$$

This minimization must satisfy constraint

$$\alpha^2 + |\eta|^2 = 1$$

which is a consequence of $Q_R * \overline{Q_R} = 1$.

From the definition of quaternion multiplication, we can expand the term $(0, a_s)*Q_R - Q_R * (0, a_m)$ in equation (A.11) as

$$\begin{aligned}
&(0, a_s) * (\alpha,\eta) - (\alpha,\eta) * (0, a_m) \\
&= \left[-a_s \cdot \eta\ ,\ \alpha a_s + a_s \times \eta\right] - \left[-a_m \cdot \eta\ ,\ \alpha a_m + a_m \times \eta\right] \\
&= \left[\eta \cdot (a_m - a_s),\ -\alpha (a_m - a_s) + (a_m + a_s) \times \eta\right]
\end{aligned} \tag{A.12}$$

The above expression is a linear function of α and η; thus we should be able to express the above expression in the form of matrix multiplication as

$$(\alpha,\eta) \cdot B$$

where B is a 4×4 matrix in terms of a_s and a_m.

This is done by converting vector cross product to matrix multiplication as follows. Define a cross matrix X of a vector $v = (x,y,z)$ as

$$X(v) = \begin{bmatrix} 0 & z & -y \\ -z & 0 & x \\ y & -x & 0 \end{bmatrix}$$

It is easy to verify that the cross product of two vectors v and η can be expressed in the form of matrix multiplication as

$$v \times \eta = \eta \cdot X(v)$$

Note again that η is a row vector and not a column vector. In fact, all the vectors used in the formulation presented here are row vectors, unless, of course, stated otherwise. The expression in (A.12) can now be rewritten as

$$(\eta \cdot (a_m - a_s),\ -\alpha (a_m - a_s) + \eta \cdot X(a_m + a_s))$$

which is the same as

$$(\alpha,\eta) \cdot \begin{bmatrix} 0 & -(a_m - a_s) \\ (a_m - a_s) & X(a_m + a_s) \end{bmatrix}$$

Thus we have

$$\boldsymbol{B} = \begin{bmatrix} 0 & -c_x & -c_y & -c_z \\ c_x & 0 & b_z & -b_y \\ c_y & b_z & 0 & -b_x \\ c_z & b_y & b_x & 0 \end{bmatrix}$$

where

$$\boldsymbol{b} = \boldsymbol{a}_m + \boldsymbol{a}_s \qquad \text{and}$$

$$\boldsymbol{c} = \boldsymbol{a}_m - \boldsymbol{a}_s$$

Now equation (A.12) can be rewritten as

$$\begin{aligned} E^2 &= \sum_j^l |\boldsymbol{Q}_R \cdot \boldsymbol{B}|^2 \\ &= \sum_j^l \boldsymbol{Q}_R \cdot \boldsymbol{B} \cdot \boldsymbol{B}^T \cdot \boldsymbol{Q}_R^T \\ &= \boldsymbol{Q}_R \cdot \boldsymbol{A} \cdot \boldsymbol{Q}_R^T \end{aligned}$$

where

$$\boldsymbol{A} = \sum_{j=1}^{l} \boldsymbol{B}\,\boldsymbol{B}^T \tag{A.13}$$

The quaternion rotation $\boldsymbol{Q}_R$ that and minimizes equation (A.13) will be the eigenvector associated with the minimal eigenvalue of the matrix $\boldsymbol{A}$. Since the magnitude of $\boldsymbol{Q}_R$ is unity, we must, of course, normalize the solution eigenvector. Assuming the computed eigenvector after normalization is $[\alpha,\beta,\gamma,\delta]$, then the rotation angle

$$\theta = 2\cos^{-1}(\alpha)$$

and the rotation axis

$$\boldsymbol{n} = (\beta,\gamma,\delta) \,/\, \sin(\frac{\theta}{2})$$

In addition, the minimal enginvalue of the matrix $\boldsymbol{A}$ will be equal to the minimized E^2, which is the fitting error. Hence, based on that enginvalue we can determine how well the scene surfaces correspond to the model surfaces. If the enginvalue is greater than some predefined threshold, we should reject the correspondences established, and, therefore, reject the matching of $\{S\}$ with $\{M\}$.

3. Estimation of the Translation Vector

After the rotation is determined, we can similarly estimate the translation by minimizing the following error function

$$E_t^2 = \sum_{j=1}^{k} |p_s^j - R \cdot p_m^j - t|^2 \qquad (A.14)$$

$$= \sum_{j=1}^{k} \left[p_s^j - R \cdot p_m^j - t \right] \left[p_s^j - R \cdot p_m^j - t \right]^T$$

Setting the derivative $\delta \frac{E_t^2}{\delta t}$ to zero lead to the following solution

$$\hat{t} = \sum_{j=1}^{k} p_s^j - \hat{R} \cdot (\sum_{j}^{k} p_m^j)$$

The minimized fitting error can be readily calculated by plugging $\hat{t}$ into equation (A.14). Again, if this fitting error is greater than certain threshold, we should reject the matching.

The reader will recall that pose transformation hypotheses are generated by first extracting vertices from a scene and then matching a scene vertex, together with its associated surfaces, with an LFS for the model. As was mentioned in Chapter 3, an LFS is composed of a model vertex and all the surfaces that come together at that vertex. The surface features in an LFS will always possess orientation attributes but may or may not possess any position attributes, especially viewpoint independent position attributes. (For example, the centroid of a surface won't do since it is viewpoint dependent due to the fact that its calculation is greatly influenced by the extent to which the surface might be occluded.) This implies that in an LFS we may not have available to us many location vectors such as p_m^j. For this reason, the position attribute of the vertex itself becomes of prime importance. For a scene vertex where three surfaces meet, the coordinates of the vertex can be computed very robustly by finding the intersection the three surfaces. For those vertices where only two surfaces meet, there does not exist a reliable method of computing the coordinates; therefore, we simply use the nearest range measurement from the structured-light data. In either of these cases, k equals 1 in equation (A.14) and therefore the translation vector is given by

$$\hat{t} = p_s - \hat{R} \cdot p_m$$

Appendix B
Initial Guess for Tessel Assignment

Given a principal direction $\mathbf{\Phi}$, in this appendix we show how its corresponding tessel indices (i,j,k) can be computed from a linear approximation. Note that the indices thus computed are only supposed to place us in the vicinity of the true tessel. As explained in Section 6.3.3, the approximately located tessel is used as a starting point for getting the exact tessel corresponding to $\mathbf{\Phi}$.

We will present our approximation for the first of the parallelograms shown in Fig. 3.22, the approximations for the other parallelograms are identical in their j and k dependences by virtue of symmetry; the dependence on i is different and will be shown below.

The approximation for the first parallelogram in Fig. 3.22 actually consists of three separate approximations, one for each of the three zones that we will now identify. The first zone consists of the triangle marked 1 in Fig. 3.21, the second zone of the triangles 2 and 3, and the last zone of the triangle marked 4.

According to equation (9) in Section 7, the index i is independent of indices j and k in the computation of θ and ϕ for a given triplet (i,j,k). For a given (θ,ϕ), we can therefore separate the determination of index i from that of j and k. The procedure that follows consists of three steps:

(1) First determine the identity of the zone to which the direction belongs.

(2) Next, determine the index i corresponding to the parallelogram in which the direction (θ,ϕ) lies.

(3) Estimate the indices j and k.

For the first two steps, the following formulas are used: (Let $\kappa = \frac{2\pi}{5}$, $\tau = atan(2)$ and assume $0 \le \theta \le \pi$ and $0 \le \phi \le 2\pi$.)

---

if $(0 \le \theta < \tau)$ /* $\mathbf{\Phi} \in$ *zone*1 */

$$i = \left\lceil \frac{\phi}{\kappa} \right\rceil \ mod(5)$$

else if $(\tau \le \theta < \pi - \tau)$ /* $\mathbf{\Phi} \in$ *zone*2 */

$$i = \left\lceil \frac{\phi - (\theta - \tau) \times \kappa \,/\, (2\pi - 4\tau)}{\kappa} \right\rceil \ mod(5)$$

else /* $\mathbf{\Phi} \in$ *zone*3 */

$$i = \left\lceil \frac{\phi - \kappa/2}{\kappa} \right\rceil \ mod(5)$$

For step 3, we will allow j and k to take non-integer value in the following formulas. During computations, the non-integer values are truncated to yield the integer values. First, let

$$\phi' = \phi - (i-1)\times 2\kappa$$

If $\Phi \in$ zone 1,

$$k = \phi' \times \frac{(\theta \times Q / \tau)}{\kappa} + 1$$

$$j = \theta \times Q/\tau - k + 1$$

If $\Phi \in$ zone 2, assume

$$\theta = aj + bk + c$$
$$\phi' = dj + ek + f$$

Solving for a,b,c,d,e,f at the four corners of zone 2, we have

$$\theta = (j+k-1)\frac{\pi-2\tau}{Q} + 3\tau - \pi$$

$$\phi' = (k-j-1)\frac{\kappa}{2Q} + \frac{\kappa}{2}$$

Then j and k can be obtained by

$$j = Q \times [\frac{(\theta+\pi-3\tau)}{(\pi-2\tau)} - (\frac{2\phi'}{\kappa} - 1)]$$

$$k = j + 1 + Q \times (\frac{2\phi'}{\kappa} - 1)]$$

And for zone 3, we use the formulas similar to those of zone 1, except that j and k are swapped, and angles θ and ϕ are appropriately offset.