CVGIP: IMAGE UNDERSTANDING
Vol. 53, No. 2, March, pp. 154–170, 1991

Determining Linear Shape Change: Toward Automatic Generation of Object Recognition Programs*

KATSUSHI IKEUCHI AND KI SANG HONG

School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213

Received March 27, 1989; accepted September 12, 1990

A 3D object localization task may be divided into two steps. First, one object appearance will be classified into one of the topologically equivalent classes of the 3D object appearances, referred to as aspects of the object *(aspect classification).* Then, the precise attitude and position of the object will be determined within one aspect *(linear shape change determination (LSCD))* We have been working on designing a compiler which automatically generates an object localization program from a given object and sensor model; the compiler adheres to the above two-step strategy of aspect classification and linear change determination. For the first step, the compiler generates a decision tree structure program. Each branch node of the tree represents a necessary decision, such as comparing the area size of a visible face, to classify one object appearance into a smaller number of aspect groups. Along this tree, an object appearance is to be classified into one particular aspect at a leaf node of the tree. This paper will investigate the design of the compiler component to generate the second step. The compiler extends each leaf node of the tree and connects several nodes so that it performs the LSCD. The compiler chooses the largest 3D face as the primal face among several visible faces at the aspect corresponding to a leaf node. The compiler has priority rules which will select one particular method out of several possible ones; this rule defines a face coordinate system on a primal face. By using these rules, it analyzes the primal face, defines the face coordinate system on it, and registers the defining method to a node connected to the leaf node. The compiler also embodies the transformation from the face to the body coordinate system at the node. In order to increase the accuracy of the attitude and position, the compiler further puts two more nodes at each branch of the program. The first node establishes correspondences between model edges and image edges. The second node iteratively solves the transformation equation to determine the object's attitude and position using these correspondences. We have prepared a program library, which is a collection of prototypical objects to perform tasks mentioned before. In compile mode, the compiler retrieves necessary strategies defined at each node, finds proper prototypical objects from the program library, instantiates and inserts these objects to the node. In run mode, these instance objects run and determine the object's attitude and position through message-passing between objects.

CONTENTS

1. INTRODUCTION

One of the critical issues in building a model-based vision system is how to quickly extract and organize the relevant knowledge of an object and to systematically turn it into a vision program. One method of achieving this goal is compiling an object and sensor model into a vision program automatically [5, 1, 13, 6]. That is, the relevant knowledge in the object and sensor model is extracted and converted into a vision program during compile time so that as little computation as possible is performed during run time.

We have been designing a class of vision algorithm compilers to generate object localization programs. Gen-

* This research was sponsored by the Defense Advanced Research Projects Agency. DOD, through ARPA Order No. 4976, and monitored by the Air Force Avionics Laboratory under contract F33615-87-C-1499. K. S. Hong was on leave of absence from POSTECH and was supported in part by KOSEF: Korea Science and Engineering Foundation. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or of the U.S. Government.

1049-9660/91 $3.00

erally speaking, a 3D object changes its appearance and generates an infinite number of appearances depending on viewer directions. However, these infinite appearances may be grouped into a finite number of topologically equivalent appearance classes. These equivalent classes are called *aspects* [17]. Using this fact, we can divide our localization strategy, which will be embodied into a program by the compiler, into two parts; classification of an object's appearance into one of the aspects *(aspect classification)*, and determination of the precise attitude and position of the object within the aspect (*linear shape change determination (LSCD)*). Within this framework, the following compiler modules should be developed:

- extracting aspects from a given object model;
- selecting a strategy for aspect classification;
- selecting a strategy for linear change determination;
- converting these strategies into a runnable program.

As for the first module, we proposed an exhaustive method for the compiler module [11]. Originally, an aspect was defined as a topologically equivalent class of object appearances. Here, we modified the definition of an aspect as a class of appearances which shares the same combination of visible faces. Based on this definition, the compiler module generates various appearances of an object from uniformly sampled viewer directions. It classifies the generated appearances into appearance classes by examining combination of visible faces. We define the classes as the aspects of the object.

As for the second module, we proposed a module performing recursive subdivisions of possible aspects by available features [15]. Although it generated a strategy for the aspect classification, it did not generate the optimal one. Recently, we have developed a module to generate the tree having the minimum cost of computation of features [8]. Given the costs of various feature extraction operations, the compiler searches over the space of possible classification strategies using the branch-and-bound method, and finds the minimum cost one.

The compiler represents the classification strategy as a tree, referred to as an *interpretation tree (IT)*. Each branch node of the tree represents a necessary decision, such as comparing the area size of a visible face, to classify one object appearance into a smaller number of aspect groups. At each leaf node, an object appearance is to be classified into one particular aspect. Here face correspondences between image and model faces are to be given if the tree is performed in run time.

This paper focuses on the third and the fourth compiler modules. First, the paper will overview the task of the compiler. The operation consists of two paths: generating strategy and program conversion. We will discuss these two issues in the order, in more details in the later sections.

2. COMPILER OPERATION

This section outlines the compiler operation for generating the LSCD part of the IT. The operation consists of two paths: generating strategy and program conversion.

The following are the underlining assumptions of designing the compiler modules:

- IT knows that the object appearance belongs to one particular aspect, and thus, IT knows the correspondence between model and image visible faces, because its aspect classification is based on face correspondences.
- IT does not know the correspondence between model and image visible edges.

2.1. Strategy Generation

2.1.1. *Estimating the* Body *Coordinate System*

The LSCD amounts to recovering the origin and the axis directions of the body coordinate system in space. One of the representative body coordinate systems includes those defined by the mass center and the inertial axis directions of an object. However, this class of body coordinate systems can be recovered only when a set of points on the whole surface of the object is given; it cannot be recovered directly from a partial observation. Thus, in order to recover the body coordinate system, the compiler needs to augment the information given by a partial observation with that given by a geometric model.

Our compiler defines a set of common visible faces over the appearances of an aspect. Thus, the compiler will choose the most appropriate face among these visible faces at each aspect and will define a coordinate system on the face. This will allow the IT to identify the face in the image in run time, recover the coordinate system on the face, and estimate the body coordinate system by using the relation from the face to the body coordinate system.

The compiler module chooses the largest 3D-area face among the visible faces and refers to this largest 3D-area face as the primal face of the aspect. Although it may appear that the largest 2D-area face visible is more appropriate, such apparent 2D area size may vary depending on viewer directions within one aspect: it may occur that several different faces are chosen as the primal face within an aspect.

The outline of the compilation is as follows:

- The compiler examines each aspect stored at a leaf node of the tree given by the aspect classification generation. It then determines the primal face at each aspect.

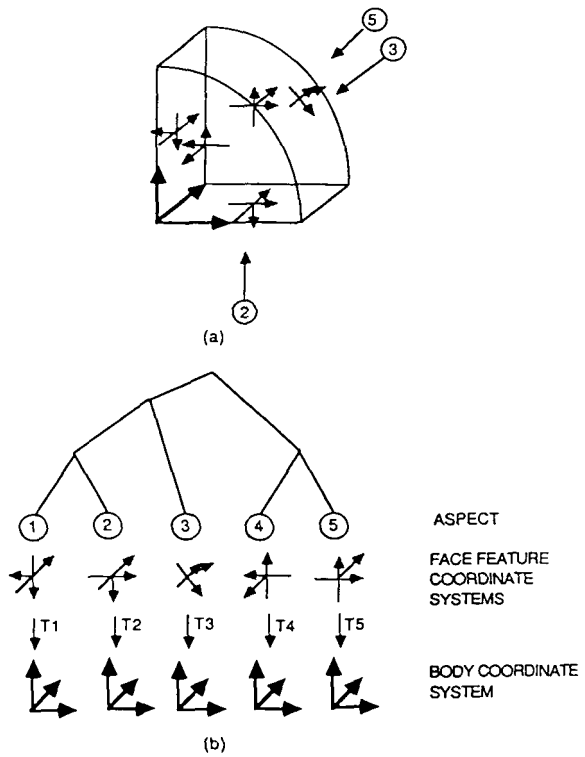


FIG. 1. Face coordinate system. (a) The relationship between the face coordinate systems and the body coordinate system. **(b)** Each leaf node of the IT, corresponding to one particular aspect, stores a method of defining the feature coordinate system and the transformation from it to the body coordinate system. The object model, thus. is represented as a collection of face coordinate systems and their relationship to the body Coordinate in the IT.

• The compiler analyzes each primal face, and determines a face coordinate system on it following the rule describe in section **3.**

• The compiler generates a new node and connects it to the previous leaf node.

• The compiler stores the resulting definition method at the newly generated node.

• The compiler stores the transformation from the face coordinate system to the body coordinate system at the node.

• The compiler connects a node to recover the body coordinate system from the face coordinate system.

Thus, the compiled IT represents the object model as a collection of face coordinate systems and their relationship to the body coordinate system, as shown in Fig. 1.

2.1.2. *Fine-Tuning by Edges*

The compiler will use edge information to increase the accuracy of the LSCD. We will refer to this step as *fine*-tuning.

In order to perform fine-tuning, the IT should establish

• correspondences between image and model edges

• recovery of the body coordinate system based on the edge correspondences.

At a leaf node of the IT, IT is supposed to have established the correspondences between image and model faces in run time. Although edges and faces are acquired through the same TV camera, they are extracted through two independent processes. A typical segmentation method shrinks each face for stable segmentation; due to smoothing operations and lighting conditions, a typical edge operator detects edges at slightly shifted places. Thus, the position of an edge and that of the corresponding face boundary are not aligned. Thus, IT has *to* establish the edge correspondences between model and image edges from scratch.

Each aspect defines a set of visible edges. The IT is required to search only the counterparts of the visible model edges in the image. Since the IT can predict **ex**pected positions of model edges from the estimated body coordinate system, it is possible to use a window method to make a rectangular area around each predicted edge and to establish a correspondence between the image edges within the area and the model edge [4]. However, this window method **1)** is sensitive to translation noise, 2) is sensitive to model error, and **3)** requires ad hoc parameters such as window size. Therefore, we have opted to use a configuration space method [18].

The configuration space method is similar to the Hough transformation. The **IT** prepares the space representing all possible body coordinate systems, which we refer to as the configuration ***space.*** For implementation purpose, the space is tessellated into many small cells. One image–model correspondence gives possible body coordinate systems. These possible body coordinate systems form a surface in the configuration space. The IT votes on the cells in this surface in the configuration space. The IT also registers which image–model edge pair votes for these cells. The IT repeats this process for all possible image-model edge pairs. The intersection point of these surfaces, indicated by the highest voted cell, gives the body coordinate system. The pairs who vote on this cell are the valid edge pairs.

The compiler will, thus, perform the following operations:

• The compiler creates a new node, connects it to the previous node, and registers the node as one to perform the configuration space based edge matching to the previous node for body coordinate estimation.

• The compiler creates and connects a node for solving the transformation equations, using the edge correspondences to the previous node.

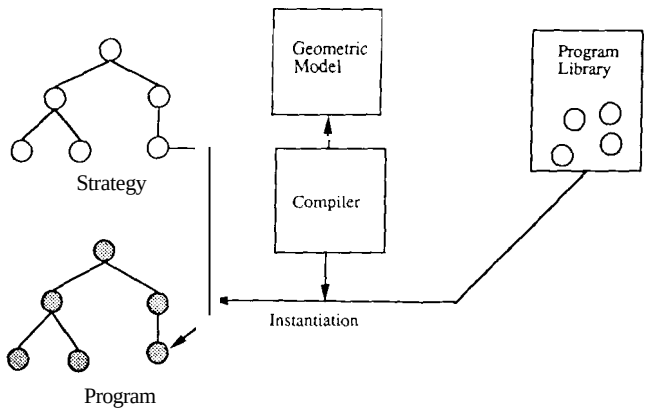


FIG. 2. Program library **and** compiler.

2.2. Program Conversion

Given a program library, we can develop a collection of prototypical objects of the object-oriented programming. The compiler retrieves necessary strategies defined at each node in the IT, finds proper prototypical objects in the program library, and instantiates and inserts these objects to each node. (See Fig. *2.)* In run mode, these instance objects are to run and determine the LSCD through message-passing between objects.

An object in object-oriented programming is a processing unit, which can store several internal values in slots. We can define demons for each slot, where a demon will be invoked implicitly when we retrieve/insert a value from/into the slot. We can define methods for each object, where a method will be invoked explicitly when we send a message associated with the method to the object. We can *instantiate* an instance object from a prototypical object, where the instance object will inherit some combinations of slot names, slot values, demon functions, and methods. Each prototypical object in the program library has slots for storing local values, such as model's z-axis direction, and methods, such as how to calculate the z-axis of an image face.

Basically, the program library consists of two classes of objects:

• event objects, which control the flow of the recognition process. This class of objects consists of the following three kinds of objects:

—objects which determine the body coordinate system from the face coordinate system
—objects which perform the fine-tuning to refine the body coordinate system from observed edges
—the supervisor object, which supervises the control of the flow in the LSCD.

• data objects, which represent data such as edges and faces in the image. This class of objects consists of the following two kinds of objects:

—face data objects
—edge data objects.

Due to the inheritance and demon mechanisms of the data objects, we can handle output from various sensors without changing any specification in the event objects. In this paper, we will not discuss these data objects further; they are discussed in greater detail elsewhere [4].

The following operations are applied to all nodes of the IT:

• The compiler examines one node of the IT, defines the necessary operation, and searches the program library for the prototypical object corresponding to the operation.
• The compiler instantiates the necessary instance object from the prototypical objects, and attaches it to the node.

The following sections will discuss the details of the method defined in the event objects in the program library.

3. GENERATION RULES

The aspect classification module of the compiler has completed one half of the IT; each leaf node corresponds to one particular aspect. The LSCD module of the compiler begins its task of completing the LSCD part of the tree from each leaf node. The compiler analyzes each primal face using the rules and determines the face coordinate system on it in this order: the origin, the z-axis, and the x-axis. This section explains the generation rules to be used for this defining operation by the compiler. These generation rules are implemented using CRL-OPS.

In order to discuss the generation rules, we need *to* explain several concepts to be used in them.

sampled appearances — The compiler generates possible aspects by classifying appearances sampled on the tessellated sphere by examining combination of visible faces. Those sampled appearances, which belongs to an aspect, are referred *to* as the sampled appearances of the aspect. Each aspect registers the sampled appearances which belong to it.

representative appearance — Among the sampled appearances of an aspect, the largest-sectional-area appearance is selected and referred to as the representative appearance of the aspect. Each aspect also registers its representative appearance.

TABLE 1
Rules for the Origin and the 2-Axis

Primal face class	Position/direction	Corresponding object
	Origin	
Planar	the mass center of the face	mass-center-of-a-planar-region
Curved	the mass center **of** the projected face	mass-center-of-a-curved region
	Z-axis direction	
Planar	the surface normal of the face	surface-orientation
Curved	the EGI mass center direction **of** the face	EGI-mass-center-direction

EGI of one face.—Originally, the EGI was defined as a spatial histogram of surface orientation distribution of the entire surface of an object over a unit sphere [9, 10]. Here we modify the definition of the EGI so that the EGI is accumulated from only one face.

3.1. Origin and Z-axis

Table 1 summarizes the rules for determining the origin and the Z-axis. The rules are classified into those for a planar primal face and those for a curved primal face.

If the primal face of an aspect is planar, the compiler uses its mass center position and its surface normal direction as the origin and Z-axis of the face feature coordinate system, respectively. Later, the compiler instantiates the mass-center-of-a-planar-region and surface-orientation object from the program library and connects them to the leaf node of the IT.

If the primal curved face is not self-occluded within an aspect, the compiler projects the face onto the tangential plane perpendicular to the its EGI mass center direction. The mass center of the projected region will be used as the origin. The EGI mass center direction is used as the Z-axis direction.

For the case of a self-occluded curved primal face, the compiler handles each sampled appearance independently; it independently defines the coordinate system on each primal face of each sampled appearance. The compiler instantiates the same number of the *mass-center-of-a-curved region* objects and *EGI-mass-center-direction* objects as the number of sampled appearances from the program library and connects them to the leaf node of the aspect in the IT. In run time, these objects are executed in parallel by sending execution messages to all these objects from the previous leaf object.

3.2. X-Axis Direction

In order to define the x-axis direction, we have to find a unique direction at each primal face. The following rules in Table **2** determine one unique direction on a primal face. The order follows the amount of the computation required to obtain such directions.

For the planar face, the search order is: more than two

TABLE 2
Rules for the X-Axis

Condition	x-axis direction	Corresponding object
More than two visible faces exist in the aspect	outer product of two vectors representing z axes of the two visible faces	region-direction-more-than-one
The primal face has a hole	hole direction with respect to the origin of the primal face	region-hole-direction
The primal face has a significantly large ratio of maximum inertia moment to minimum inertia moment	minimum inertia direction	region-momentum-direction
The edge histogram of the visible edges of the primal face has **a** significantly large peak direction	peak direction	region-edge-histogram
The primal face has a significantly large ratio **of** the maximum EGI inertia moment to the minimum EGI inertia moment	minimum EGI inertia direction	EGI-momentum-direction
None of the above is applicable	all directions (currently *36* directions)	parallel

TABLE 3
Object for Recovering Face Coordinate System

Object name	Method
Mass-center-of-a-planar-region	It calculates the mass of a region assuming the region is a projection of a planar face.
Mass-center-of-a-curved region	It determines the mass center of a region corresponding to a curved face. In the current implementation, the mass center of the projected area of a curved surface on the image plane is used instead of the tangential plane to the curved surface. Thus, the current implementation of this method is same as those of the planar face. Although this gives an only approximation of a mass center, it is enough because the fine-tune operation will be performed based on this approximation in the next stage.
Surface-orientation	It determines the average surface orientation over a region assuming that the region is **a** projection of a planar face.
EGI-mass-center-direction	It determines the **EGI** mass-center direction over a region corresponding to a curved primal face. This method is equivalent to making an average surface orientation over the curved surface.
Region-direction-more-than-one	It determines the second largest face around the target region, calculates the z axis direction of the region, and gets the outer product of the two vectors representing two z axes directions.
Region-hole-direction	It applies the swell-and-shrink operations several times to a region to erase noisy small holes, determines the largest hole within the region, and determines the direction of the hole's mass center with respect to the mass center of the region.
Region-momentum-direction	It calculates the region's inertia moment.
EGI-momentum-direction	It calculates the EGI momentum directions on the plane perpendicular to the EGI mass center direction.
Region-edge-histogram	It collects edges around a target region, accumulates edge directions, and determines the most often occurring direction. Note that we do not know which image edges belong to the target region Thus, the method expands the target region several pixels and use image edges included within the expanded region.
Parallel	It returns an arbitrary direction. In the current implementation, it returns the **x** axis direction of the image plane.

visible faces, a hole, momentum, edge, parallel. For the curved face, the search order is: more than two visible faces, a hole, momentum, EGI momentum, parallel.

4. OBJECTS FOR FACE COORDINATE SYSTEM

4.1. Object **for** Recovering Face Coordinate System

Corresponding to each action of the generation rules, we prepare the corresponding prototypical object in the program library. Note that the actions in the rules analyze a primal model face and store the defined coordinate system in the IT, while the methods in the prototypical objects in the program library analyze an image region and recover the coordinate system from it. Here the region to be processed by the IT is referred to as the ***target*** region. Currently, the program library contains the objects shown in Table 3 for this purpose.

4.2. Object **for** Estimating Body Coordinate System

Up to this stage, the compiler completes the part of the IT which is capable of recovering the face coordinate system from the image. It is necessary to have an object for recovering the body coordinate system based on the obtained face coordinate system. Thus, we prepare the *body-coordinate-estimation* object which estimates the body coordinate system with respect to the image world coordinate system in the program library. In compile mode, the compiler instantiates the object and connects it to each leaf node of the IT. The remainder of this subsection discusses the method of this object.

At the objects for recovering face coordinate system described in the previous section, the IT stores the rotation R_f and the translation X_f of the face coordinate system with respect to the model world coordinate system. (See Fig. 3(b).) The IT also stores those of the body coordinate system R_b, X_b with respect to the same coordinate system.

In run mode, the IT obtains the face coordinate system R_f', X_f' with respect to the image world coordinate system. (See Fig. 3(a).) At first, this method determines the rotation R_f as follows.

Let the unknown rotation matrix be δR,

$$\delta R R_f = R_f'. \tag{1}$$

(See Fig. 3 (a, b).)

Thus,

$$\delta R = R_f' R_f^{-1}. \tag{2}$$

Then, this method applies the rotation matrix to the body coordinate system in the model world coordinate system as

$$R'_b = \delta R R_b = R'_f R_f^{-1} R_b. \quad (3)$$

(See Fig. 3(c).)

After rotation of the model using this rotation matrix, the method determines the origin of the face coordinate system of the model, $\delta R X_f$, in order to determine the translation vector. Given the origin of the target region, X'_f, from the image, we can obtain the translation vector, δX, as

$$\delta X = X'_f - \delta R X_f. \quad (4)$$

The method translates that the body coordinate system in the model by δX so that it agrees with the body coordinate system in the image, (see Fig. 3(d)):

$$\begin{aligned} X'_b &= X_b + \delta X \\ &= X_b + X'_f - \delta R X_f \\ &= X_b + X'_f - (R'_f R_f^{-1}) X_f. \end{aligned} \quad (5)$$

Thus, the X'_b and R'_b give the body coordinate system of the model which aligns with the image. Using this body coordinate system, the object invokes VANTAGE [16] to generate an apparent image, from which the model edges used in the next edge correspondence task are extracted.

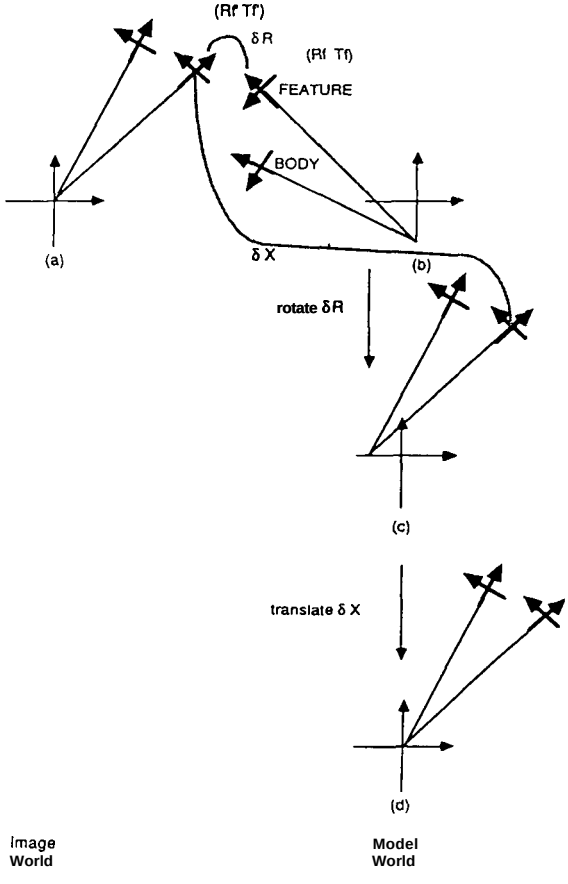


FIG. 3. The relationship between the image world coordinate system and the model world coordinate system.

5. OBJECT FOR FINE-TUNING

The body coordinate system obtained in the previous step is a rough estimate. In order to get a more accurate coordinate system, the compiler will use the information given by image edges. This section will explain two objects which handle this edge information: the *configuration* object and the *fine-tune* object.

In run-mode, the configuration object will identify edge pairs of model and image edges using the configuration space approach. Then, the fine-tune object will iteratively fine-tune the body configuration by iteratively solving a nonlinear equation given by edge correspondences.

In compile mode, the compiler instantiates the prototypical *configuration* object and connects it to each leaf node of the IT. Then, the compiler instantiates the *fine-tune* object and connects it to the instance configuration object in the IT.

5.1. Configuration Object (for Establishing Edge Pairs)

Theoretically, six dimensions are necessary for the configuration space: three dimensions in translation and three dimensions in rotation. However, the purpose of the use of the space is only to establish edge pairs and not to determine the object coordinate system. Thus, **a** three dimensional configuration space is prepared: two dimensions in translation and one dimension in rotation for simplicity.

Up to this point, the IT knows a rough body coordinate system. It is necessary to consider only the neighboring coordinate system around the estimated coordinate system. Thus, coordinate differences instead of coordinate systems are represented in the space. The method, currently implemented and prepared in the program library, uses a space which is tessellated into 10 by 10 by 10 cells.

At each image–model edge pair, a coordinate difference, the difference between translation and rotation, is obtained. First, the method will obtain the difference in rotation between the model and the image edge. This difference gives the plane on which the possible coordinate differences of the real body coordinate system from the estimated one exists. See Fig. 4(a).

The image edge will be rotated by the angle of the rotation difference to align it with the model edge. Here the center of the rotation is the origin of the body coordinate system Cstimated in the previous section. This operation corresponds to Fig. 4(c).

In order to obtain the translation difference, the method calculates as follows: the difference vectors are obtained from the distances between one end point of the image edge and two end points of the model edge,

The **3D** point of the model, (x_0, y_0, z_0), will be rotated to give

$$\begin{pmatrix} x_0' \\ y_0' \\ z_0' \end{pmatrix} = R' \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix} \tag{10}$$

Assuming the orthographic projection as the camera model, the projection of the **3D** model point on the image plane is expressed as

$$\begin{aligned} u_0 &= fx_0' + m \\ v_0 &= gy_0' + n \end{aligned} \tag{11}$$

where f and g are scale factors from the model to the image, and m and n are translation factors on the image plane.

Suppose the corresponding image edge has the equation

$$d = su + tv, \tag{12}$$

where $s^2 + t^2 = 1$, and d is the distance **of** the edge from the origin. The model point should satisfy Eq. (12) [19]. By expansion of this equation by the Taylor series, and substitution of the model point.

$$\begin{aligned} \boldsymbol{d} = {} & (sv_0 + tu_0) + s\left(\frac{\partial u}{\partial f}\Delta f + \frac{\partial u}{\partial g}\Delta g + \frac{\partial u}{\partial m}\boldsymbol{Am}\right. \\ & \left. + \frac{\partial u}{\partial n}\Delta n + \frac{\partial u}{\partial \alpha}\Delta\alpha + \frac{\partial u}{\partial \beta}\Delta\beta + \frac{\partial u}{\partial \gamma}\Delta\gamma\right) \\ & + t\left(\frac{\partial v}{\partial f}\Delta f + \frac{\partial v}{\partial g}\Delta g + \frac{\partial v}{\partial m}\boldsymbol{Am} + \frac{\partial v}{\partial n}\right. \\ & \left. + \frac{\partial v}{\partial \alpha}\Delta\alpha + \frac{\partial v}{\partial \beta}\Delta\beta + \frac{\partial v}{\partial \gamma}\Delta\gamma\right) \end{aligned} \tag{13}$$

Thus,

$$\begin{aligned} \boldsymbol{0} = {} & (-tzg)\Delta\alpha + (szf)\Delta\beta + (txg - syf)\Delta\gamma \\ & + sx'\Delta f + ty'\Delta g + s\Delta m + t\Delta n + d_0 - d. \end{aligned} \tag{14}$$

By using least mean square fitting, the method will obtain the unknowns $\Delta\alpha$, $\Delta\beta$, **Ay**, Δf, Δg, **Am, An.** Namely, the method will minimize

$$\begin{aligned} \boldsymbol{e} = \sum_i l_i (A_i\Delta\alpha + B_i\Delta\beta + C_i\Delta\gamma + D_i\Delta f \\ + E_i\Delta g + F_i\Delta m + G_i\Delta n + H_i)^2, \end{aligned} \tag{15}$$

where l_i is the length of the image edge i, and

$$\begin{aligned} A_i &= (-t_i z_i g) \\ B_i &= (s_i z_i f) \\ C_i &= (t_i x_i g - s_i y_i f) \\ D_i &= s_i x_i' \\ E_i &= t_i y_i' \\ F_i &= s_i \\ G_i &= t_i \\ H_i &= (d_{0i} - d_i), \end{aligned}$$

which can all be observed from the image edge ***i*** and the corresponding model edge i.

6. EXPERIMENTS

This section will illustrate how the compiler generates an IT from an object and a sensor model, what kind of representations are generated by an IT, and how the IT performs its tasks on real images.

In order to show the sensor independence of our system, we intentionally use two different sensors, dual photometric stereo [12] and an Erim range finder [7];[1] we also use sensor model (which are found elsewhere [14]) of both,

Since the scope of this paper is to examine the **LSCD** part, we will not discuss how to generate aspects, how to reduce the number of aspects, how to generate an aspect classification part, or the correct classification ratio of the aspect classification part. Some of these topics may be found in [15].

6.1. Dual Photometric Stereo: Car-Shaped Object

6.1.1. Generation Experiment

The car model, shown in Fig. 5(a), has 14 aspects as shown in Fig. 5(b) under the dual photometric stereo sensor model. Based on these aspects, the aspect-classification part of an IT is generated as shown in Fig. 5(c), where a leaf node **of** each branch contains one aspect. In order to allow redundancy, the IT has more than 14 leaf nodes.

The compiler generates **LSCD** branches from each leaf node using the generation rules described in Section 3. Each branch determines the body coordinate system through the face-coordinate system and the edge match. Figure 5(d) shows the completed whole IT for this sensor and object.

6.1.2. Execution Experiment

The car scene, shown in Fig. 6(a), is given to the IT. **A** needle map is obtained by photometric stereo; an edge

[1] **Yet, this compiler assumes that region and edge information is available from the sensor. In this respect. the compiler is still dependent on the sensor.**

$$\Delta \mathbf{x}_1 = \mathbf{X}_{\text{model},1} - \mathbf{x}_{\text{image}} = (\delta x_s, \delta y_s).$$
$$\Delta \mathbf{x}_2 = \mathbf{X}_{\text{model},2} - \mathbf{x}_{\text{image}} = (\delta x_e, \delta y_e). \quad (6)$$

An observed image edge may be a part of the corresponding model edge. Thus, any vector between the two difference vectors is a possible movement of the image edge aligning with the model edge, and is the possible translational coordinate difference. (See Fig. 4(d).) This method votes on all the cells which satisfy the equation in Eq. (7). Here, the weight of a vote is proportional to the length of an image edge (or how important the image edge is),

$$y = mx + b, \quad (7)$$

where

$$m = \frac{\delta y_e - \delta y_s}{\delta x_e - \delta x_s}$$

and

$$b = m\delta x_s + \delta y_s.$$

This voting operation is repeated between all possible pairs. After finishing the voting operations, the method will find the cell containing the maximum "weight." The edge pairs voting on this cell will be selected as the correct matching pairs. The method will give these correct

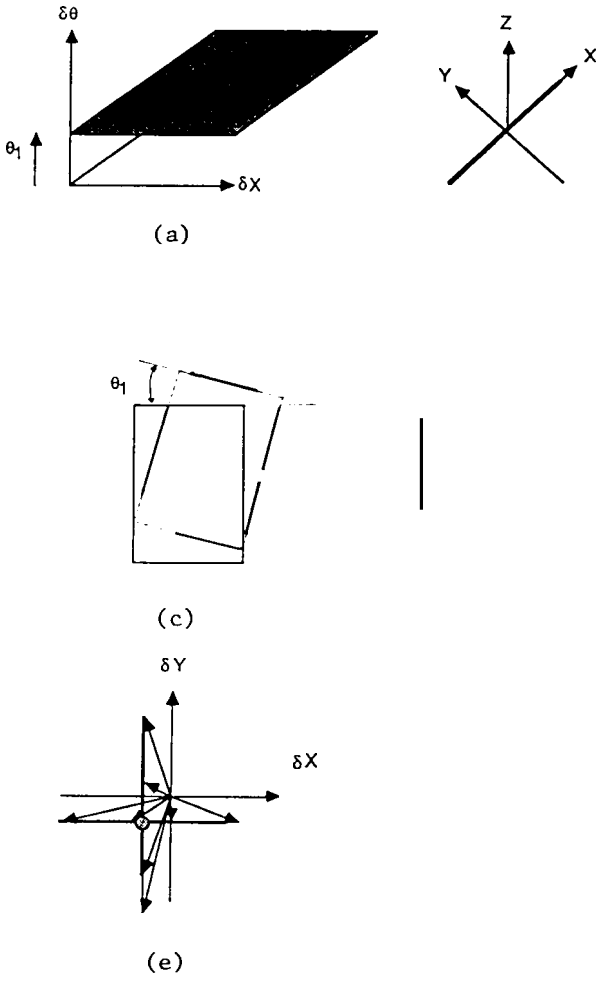


FIG. 4. Matching operations. (a) **3D** configuration space. (b) 2D edge feature coordinate system. The origin of this system is the center of the edge. The z-axis is perpendicular to the image plane. The x-axis aligns with the edge direction. (c) Difference in rotation. This gives the plane on which the configuration difference's trace exists. (d) An **observed** image edge may be a part of the corresponding model edge. Thus, any vector between the two different vectors is a possible movement of the image edge to align with the model edge. (e) The cell containing the maximum "weight" corresponds to the most likely coordinate system. The edge pairs that vote to this cell will be selected as the correct matching pairs. These correct edge pairs are given to the fine tuning object.

nate system from given edge pairs. The basic method is to adjust the object's appearance by matching 3D model edges with 2D image edges.

$$R = \begin{pmatrix} & & ca \text{ vers } \theta + b\sin\theta \\ & & cb \text{ vers } \theta - a\sin\theta \\ & & c^2 \text{ vers } \theta + \cos\theta \end{pmatrix} \quad (8)$$

using the rotation axis, described by the unit vector (a, b, c), and the rotation angle θ around the axis, where vers $\theta = (1 - \cos\theta)$.

Since the estimated object coordinate system is roughly aligned with the real one, the rotation θ is small; $\sin\theta = \theta, \cos\theta = 1$, and vers $= 0$. Substituting these values into equation (8), we have

$$R' = \begin{pmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{pmatrix}, \quad (9)$$

where $\alpha = a\theta$, $\beta = b\theta$, $\gamma = c\theta$.

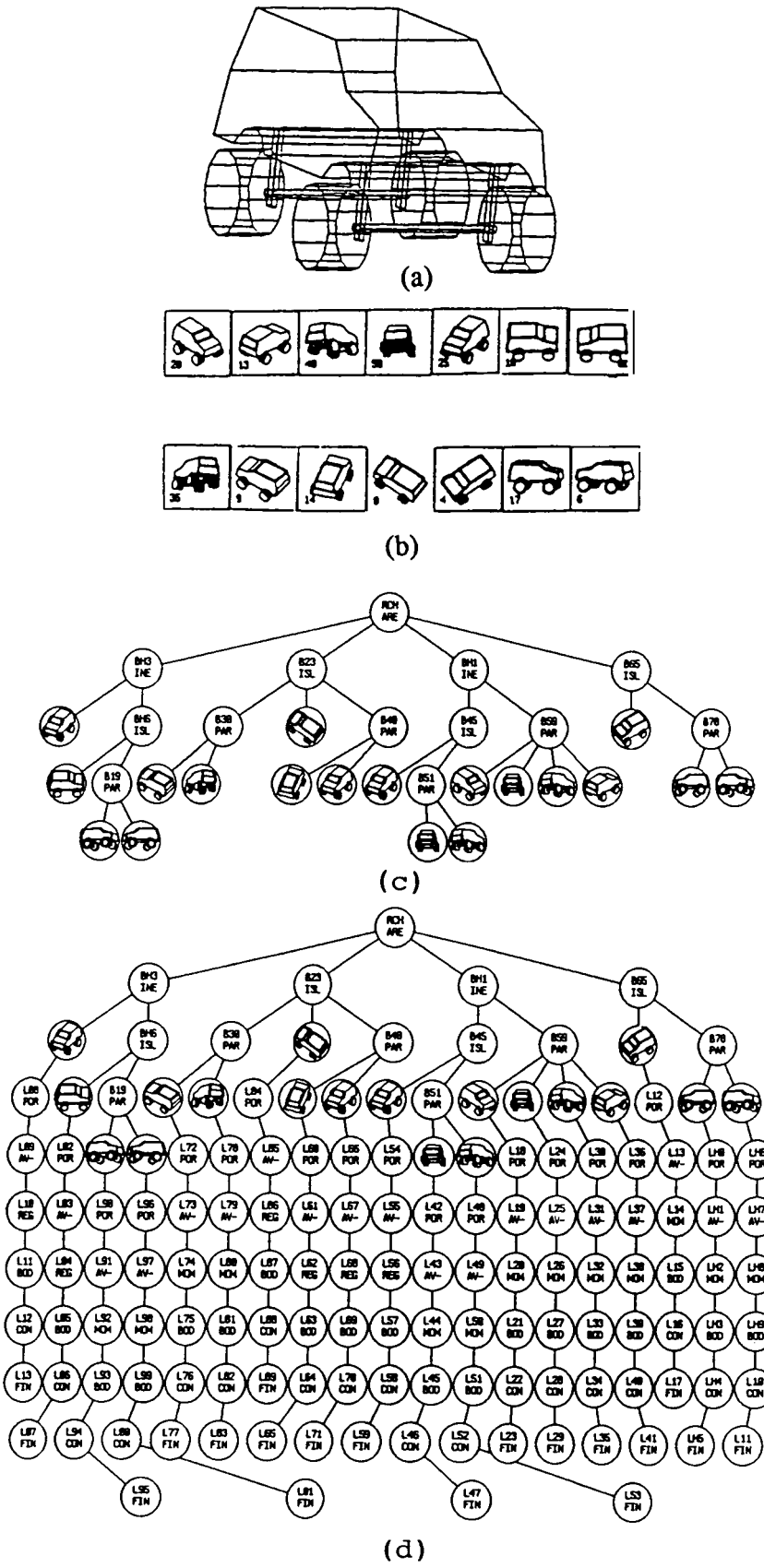
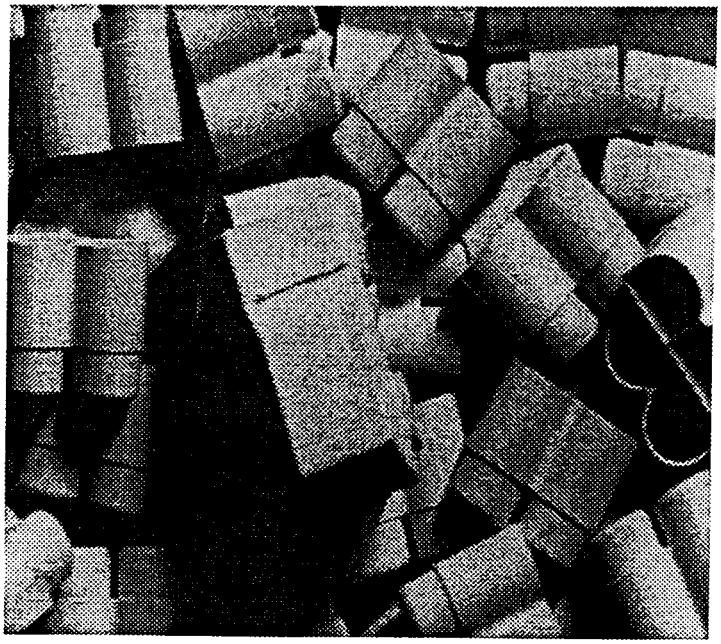
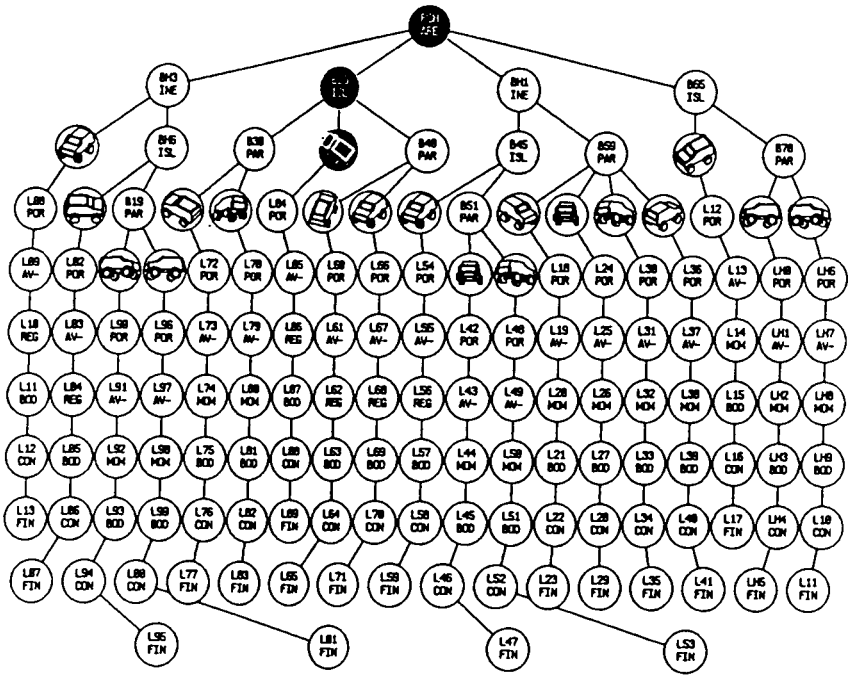


FIG. 5. Experiment 1. (a) Object model. (b) Aspects. (c) Aspect classification part of the interpretation tree. (d) Complete interpretation tree



(a)



(b)

```
(DEFSCHEMA LNODE86
        (IS-A                       LNODE)
        (LNODE-NAME                 LNODE86)
        (IS-A-CHILD-OF              LNODE85)
        (IS-A-PARENT-OF             LNODE87)
        (HYPOTHESIS                 HYPOTHESIS1)
        (ASPECT-COMP-NAME           ASPECT-COMP10)
        (EXECUTION                  REGION-MORE-THAN-ONE)
        (EXECUTION-DESTINATION      LBRANCH87)
        (EXECUTION-MATRIX           ((-0.07547908 0.99619484 0.043572128)
                                     (-0.5000054 -5.543232E-6 -0.8660222)
                                     (-0.8627267 -0.08715287 0.49810332))))
```

(c)

FIG. 6. Result of Experiment 1. (a) Input scene. (b) Aspect classification result. (c) Node contents. (d) Estimated attitude and position from the face feature coordinate system. (e) Edge pairs obtained by the configuration object. (f) Obtained precise attitude and position.

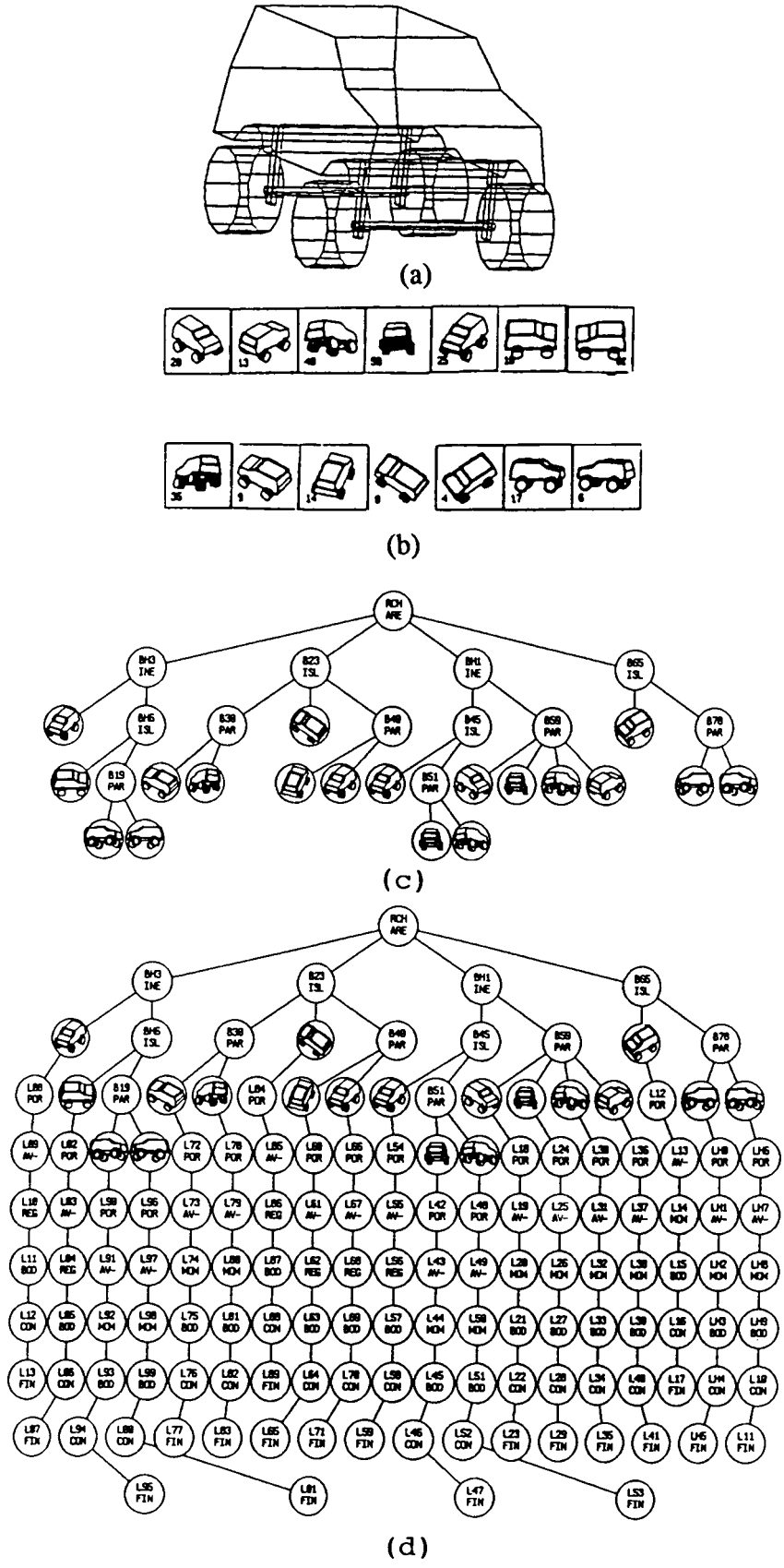
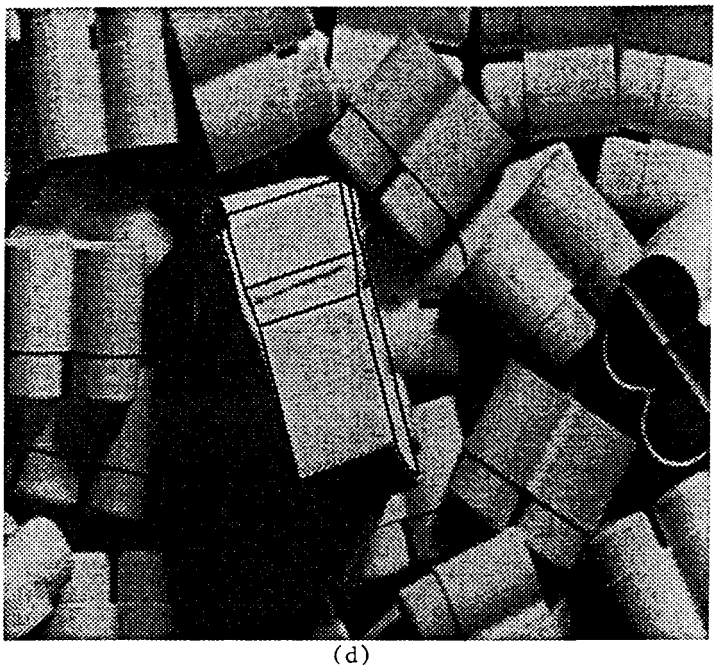
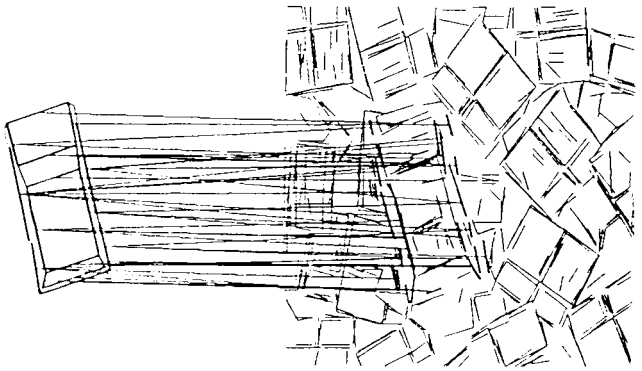


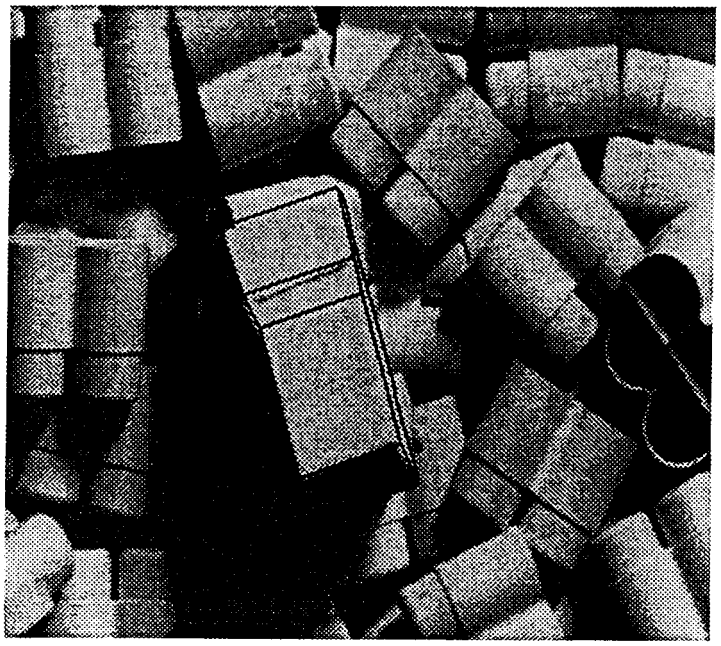
FIG. 5. Experiment 1. (a) Object model. (b) Aspects. (c) Aspect classification part of the interpretation tree. (d) Complete interpretation tree



(d)



(e)



(f)

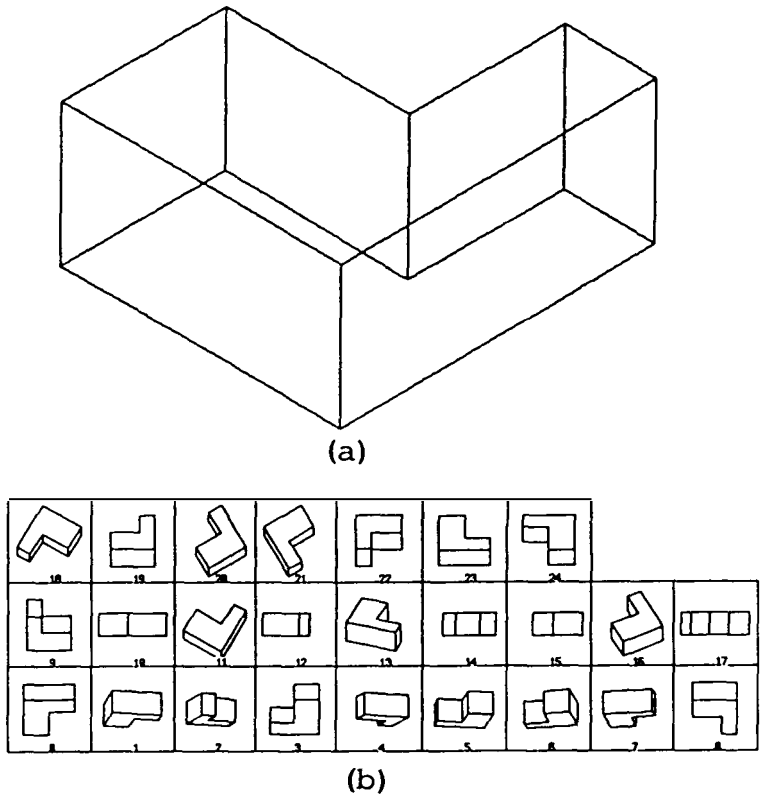


FIG. 7. Experiment **2.** (a) L-shaped object **model.** (b) Aspects. (c) Aspect classification part of the interpretation tree. (d) Complete interpretation tree.

map is obtained through an edge operator. **A** segmentation program divides the needle map into regions and generates a region distribution map. The IT chooses the nearest region to the sensor in the region map as the target region and begins the localization task from it. The aspect classification part of the IT classifies the region into the aspect indicated by the black node as shown in Fig. 6(b).

From the black node in the figure, the LSCD part begins. Node 184, marked as por, containing the *mass-center-of-a-planar-face* object, determines the mass center position of the region in the scene and declares the position as the origin of the face coordinate system. The ***surface-orientation*** object, marked as AV- in node 185 (AV- stands for the average surface orientation), determines the direction and declares it as the z-axis.

Figure 6(c) shows the contents **of** *l*86, represented in a knowledge representation language, Knowledge Craft. This node's relations in the IT are stored in slots *is-a-child-of* and *is-a-parent-of*, which were used in the tree generation. Slot *hypothesis* contains the current hypothesis *hypothesis*1, which contains the region name currently examined, other regions considered to belong to the same physical object, and the obtained face coordinate system.

Slot *execution* stores the method name to be executed by receiving the message *execution*. The *region-more-than-one* method calculates the neighboring region direction in the image, stores it in *region-x-axis* slot of *hypothesis*1, and declares it as the x-axis direction.

Then, this object sends the next *execution* message to node *l*87. Namely,

(call-method '*l*87 'execution 'hypothesis 1).

This object also contains the transformation from the face coordinate system to the body coordinate system in the *execution-matrix* **slot.**

The ***body-coordinate-estimation*** object in node 187 estimates the body coordinate system using the face coordinate system in *hypothesis1* and the transformation in *execution-matrix* of *l*86. Through this process, the body coordinate system is estimated as shown in Fig. 6(d).

The *configuration* object in node *l*88 obtains the edge pairs **as** shown in Fig. 6(e). Using these edge pairs, the *fine-tune* object in node 189 gives the position and attitude as shown in Fig. 6(f).

In order to evaluate the performance of the LSCD part of an IT generated by the compiler, ***we*** repeat the same execution process, mentioned above, 15 times. Namely,

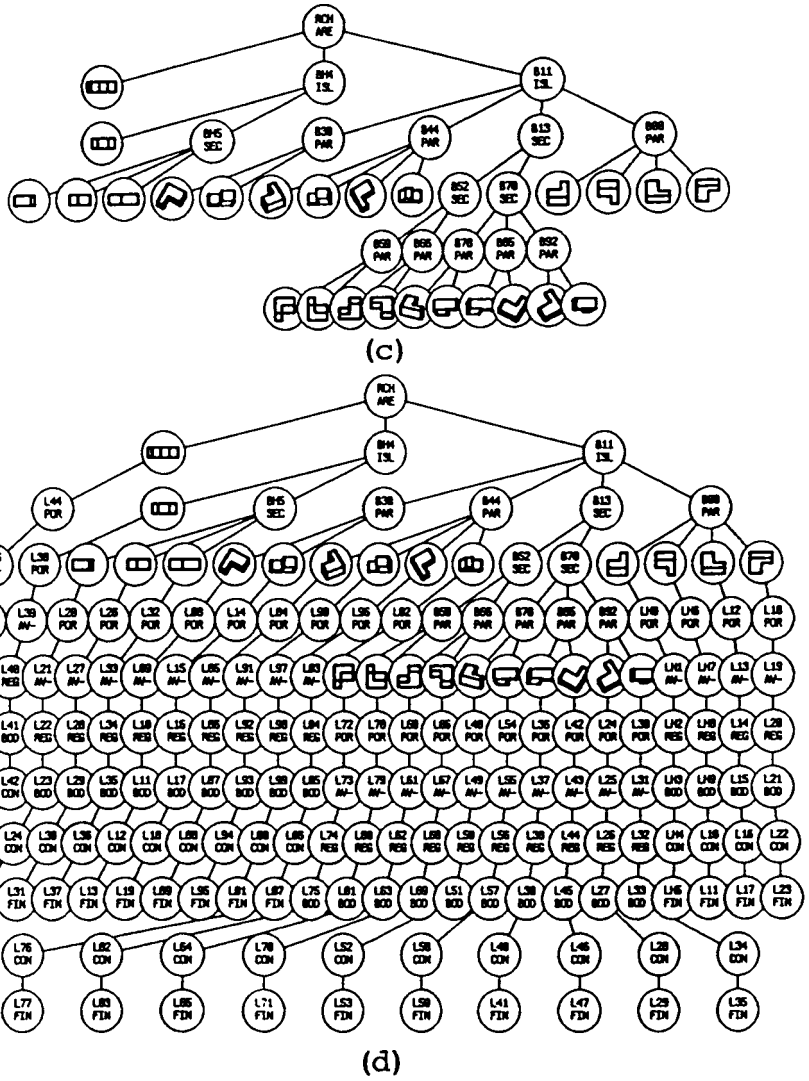


FIG. 7—*Continued*

we take the car scene is 15 times using the photometric stereo system. From these 15 sets of images, 15 real needle maps and, thus, 15 region maps are generated. We applied the IT to these 15 sets of region maps and obtained the object attitude 15 times. In our experiments, the average angular error, the average angular difference between the average attitude and each attitude, is 2.62 degrees.

6.2. Erim Range Finder: L-Shaped Object

In order to show the capability of the compiler with a different combination of object and sensors, we generate another IT for an Erim range finder and an L-shaped object.

The L-shaped object shown in Fig. 7(a) has 25 aspects as shown in Fig. 7(b) under the Erim range finder sensor model. Although this object is simpler than the previous car object, the detectable area of the Erim range finder, that is, those faces visible to the sensor, is larger than those of the dual photometric stereo [14]. Thus, the number of aspects in this example is larger than the number of aspects in the previous example. This is a typical example of the dependency of aspects upon the sensor model.

Based on these aspects, the aspect classification part of the IT is generated as shown in Fig. 7(c). At each leaf node of the IT, the compiler generates the LSCD part using the generation rules and completes the tree as shown in Fig. 7(d).

The L-shaped object, as shown in Fig. 8(a), is segmented into regions as shown in Fig. 8(b). Region R27, the nearest region to the sensor, is given to the IT. The aspect classification part of the IT classifies the region into the two possible aspects indicated by the black nodes as shown in Fig. 8(c).

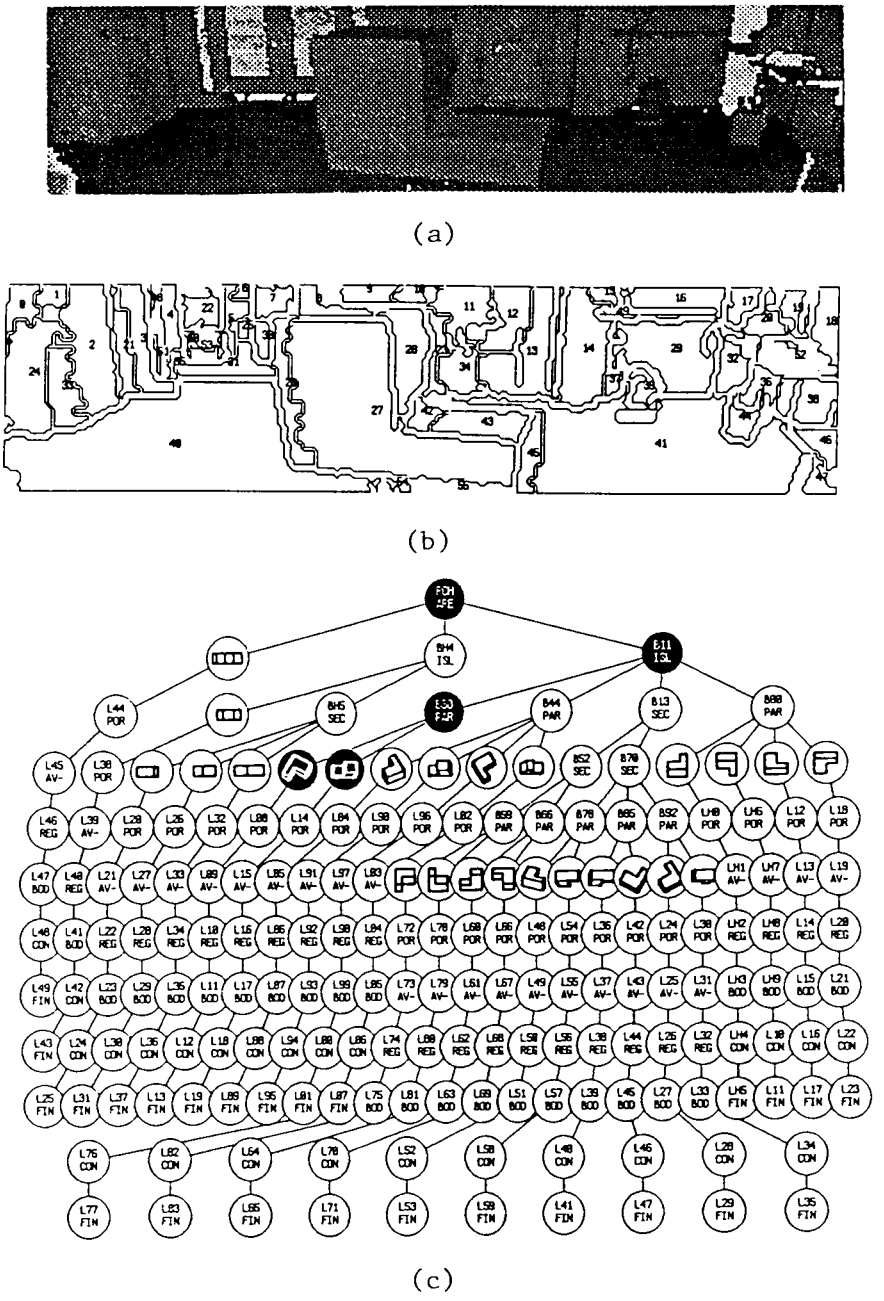
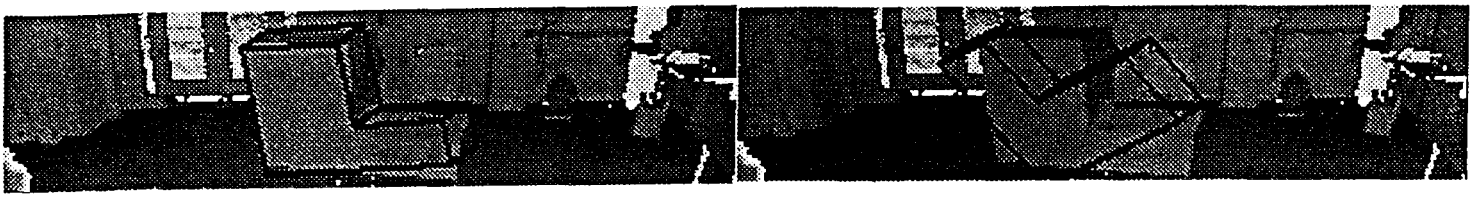


FIG. 8. Result of Experiment 2. (a) L-shaped scene (reflectance image given by ERIM range finder). (b) Segmentation result based on reflectance and depth. (c) Aspect classification result. (d) Two possible body coordinate systems. (e) Edge correspondences. (f) Obtained attitude and position.

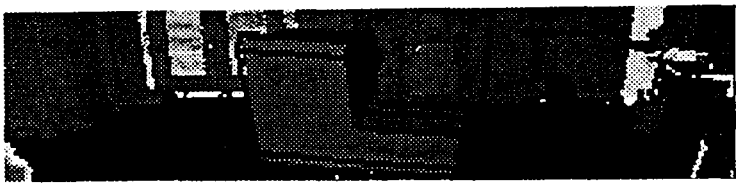
When the IT decides that it is impossible to discriminate some of the aspects, it executes the all **LSCD** branches corresponding to the possibilities aspects. In this example, two **LSCD** branches, beginning 108 and 114, are executed in parallel. Then, two possible body coordinate systems are estimated as shown in Fig. 8(d).

The *configuration* object obtains the edge pairs by searching the configuration cell containing the maximum voting score where a voting score represents the total length of edge pairs voting on the cell. Figure 8(e) shows the selected edge pairs in the node 112. The *supervisor* object compares the maximum voting scores among the possible branches and decides the most likely branch. In this example, it decides that the one containing the node 112 (the left figure in Fig. 8(d)) is more likely than the one containing the node 118 (the right figure in Fig. 8(d)).



(d)

(e)



(f)

FIG. 8—*Continued*

Then, it sends the execution message to the node 113. The ***fine-trine*** object in 13 node determines the result shown in Fig. 8(f).

7. SUMMARY

This paper has presented a vision algorithm compiler which generates an object localization program from an object and sensor model. In particular, this paper concentrates on a module which generates a part of a program to determine a LSCD part within an aspect.

In order to execute our compiler, we have to prepare

- object representation in VANTAGE geometric modeler,
- sensor representation in VANTAGE geometric modeler,
- feature operation cost for aspect classification,
- feature selection rule for LSCD, and
- program library.

By using this declarative knowledge, the compiler generates a program automatically.

Currently, all the modules of the compiler have been implemented. Without any human intervention, the compiler automatically generates

- aspect structures of an object under a sensor,
- aspect classification strategy,
- LSCD strategy, and
- a localization program.

We are currently working to improve the performance of the compiler as well as to increase ways in which the compiler can be used.

ACKNOWLEDGMENTS

Takeo Kanade provided many useful comments and encouragements. Martial Hebert provided us with ERIM range data and software which were used in our experiments. Keith Petersen and Kathryn Porsche proofread earlier drafts of this paper and provided many useful comments which have improved the readability of this paper. The authors also thank Shree Nayar, Purushothaman Balakumar, Jean-Christophe Robert, and the members of the task-oriented vision laboratory of the Robotics Institute, Carnegie Mellon University, for their valuable comments and suggestions.

REFERENCES

1. **R. C. Bolles** and P. Horaud. 3DPO: **A** three-dimensional part orientation system. *Three-Dimensional Machine Vision* (T. Kanade, **Ed.),** pp. 399–450, Kluwer, Boston, 1987.

2. A. J. Boyer, K. L. Vayda, and A. C. Kak, Robotic manipulation experiments using structural stereopsis for 3D vision, *IEEE Expert* 1(fall), 1986, 73–94.

3. J. B. Burns and L. J. Kitchen, Rapid object recognition from a large model base using prediction hierarchies, in *Proc. of DARPA Image Understanding Workshop,* pp. 711–719, Morgan Kaufmann, April 1988.

4. H. Chang. K. Ikeuchi, and T. Kanade, *Model-Based Vision System by Object-Oriented Programming, Technical Report CMU-RI-TR-88-3,* Carnegie Mellon University, The Robotics Institute, March 1988.

5. C. Goad, Special purpose automatic programming for 3D model-based vision, in *Proc. of DARPA Image Understanding Workshop,* pp. 94–104, DARPA, 1983.

6. C. Hansen and T. Henderson, CAGD-based computer vision, In *Proc. IEEE Computer Society Workshop on Computer Vision, Miami Beach, FL, December 1987,* pp. 100–105.

7. M. Hebert and T. Kanade, Outdoor scene analysis using range data, in *Proc. of Intern. Conf. on Robotics and Automation, San Francisco, April, 1986,* pp. 1426–1432.

8. K. S. Hong, K. Ikeuchi, and K. D. Gremban, Miminium cost aspect classification: A module of a vision algorithm compiler, in *10th Intern. Conf. on Pattern Recognition, Atlantic City, N.J., June 1990,* pp. 65–69.

9. B. K. P. Horn, Extended gaussian images, *Proc. IEEE* 72(12), 1984, 1671–1686.

10. K. Ikeuchi, Recognition of 3-d objects using the extended gaussian image, in *International Joint Conf. on Artificial Intelligence. Vancouver, B.C., Canada. 1981,* pp. 595–600.

11. K. Ikeuchi, Precompiling ageometrical model into an interpretation tree for object recognition in bin-picking tasks, in *Proc. DARPA Image Understanding Workshop, Los Angeles, CA, February 1986.*

12. K. Ikeuchi, Determining a depth map using a dual photometric stereo, *Internat. J. Robotics Res.* 6(1), 1987, 15–31.

13. K. Ikeuchi, Generating an interpretation tree from a CAD model for 3-D object recognition in bin-picking tasks, *Intern. J. Comput. Vision* 1(2), 1987, 145–165.

14. K. Ikeuchi, and T. Kanade, Modeling sensor performance for model-based vision, in *Proc. of 4th Inter. Sympos. of Robotics Research (R. Bolles and B. Roth, Ed),* pp. 255–263, MIT Press, Cambridge, MA, 1987.

15. K. Ikeuchi and T. Kanade, Towards automatic generation of object recognition program *Proc IEEE,* 76(8), 1988, 1016–1035.

16. T. Kanade, P. Balakumar, J. C. Robert, R. Hoffman, and K. Ikeuchi, Overview of geometric/sensor modeler VANTAGE, in *Proc. the International Symposium and Exposition on Robots, Sydney, Australia, November 1988.*

17. J. J. Koenderink and A. J. Van Doorn, Internal representation of solid shape with respect to vision, *Biol. Cybernrt.* 32(4), 1979, 211–216.

18. Y. Kuno, K. Ikeuchi, and T. Kanade, Model-based vision by cooperative processing of evidence and hypotheses using configuration spaces, in R. D. Juday, editor, *Proc. of SPIE Conf. Digital and Optical Shape Representation and Pattern Recognition.* (R. D. Juday, Ed.), Vol. 938, pp. 444–453. SPIE, 1988.

19. D. G. Lowe, Three-dimensional object recognition from single two-dimensional images, *Artificial Intelligence, 31,* 1987, 355–395

20. H. Plantinga and C. Dyer, *Visibility, Occlusion, and Aspect Graph,* Technical Report 1987, University of Wisconsin-Madison. Computer Science Department, 1987.

21. L. G. Shapiro, A CAD-model-based system for object localization, in *Proc. of SPIE Conf. on Digital and Optical Shape Representation and Pattern Recognition,* (R. D. Duday, Ed.), pp. 408–418.

22. J. H. Stewman and K. W. Bowyer, Aspect graphs for planar-face convex objects, in *Proc. of IEEE Workshop on Computer Vision, 1987,* pp. 123–130.

23. M. Swain, Recognition from a large database using a decision tree, in *Proc. of DARPA Image Understanding Worksliop.* pp. 490–686, 1988.