COMPUTER VISION AND IMAGE UNDERSTANDING
Vol. 61, No. 3, May, pp. 387–416, 1995

Interactive Learning of a Multiple-Attribute Hash Table Classifier for Fast Object Recognition

LYNNE GREWE AND AVINASH C. KAK

Robot Vision Laboratory, Purdue University, 1285 EE Building, West Lafayette, Indiana 47907-1285

Received May 15, 1994; accepted January 11, 1995

Multiple-attribute hashing is now considered to be a powerful approach for the recognition and localization of 3D objects on the basis of their invariant properties. In the systems developed to date, the structure of the hash table is fixed and must be created by the system developer—an onerous task especially when the number of attributes is large, as it must in systems that use both geometric and nongeometric attributes. Another deficiency of previous systems is that uncertainty is treated as a fixed value and not modeled. In this paper, we will present a system, named MULTI-HASH, which uses the tools of decision trees and uncertainty modeling for the automatic construction of hash tables. The decision-tree framework in MULTI-HASH is based on a hybrid method that uses both qualitative attributes, such as the shape of a surface, and quantitative attributes such as color, dihedral angles, etc. The human trainer shows objects to the vision system and, in an interactive mode, tells the system the model identities of the various segmented regions, etc. Subsequently, the decision-tree-based framework learns the structure of the hash table. © 1995 by Academic Press, Inc.

1. INTRODUCTION

At the simplest level, the problem we are addressing may be explained as follows. Let us say we have only two classes of 2D objects, such as wooden chips, all the objects in the first class being red and all the objects in the second blue. Let us also say that we have a vision system that knows about color, texture, shape, etc., and that also knows how to carry out segmentation on the basis of constancy of these attributes. We show instances of these two classes to a vision system and, for each showing, tell the system the class identity of the object. Now, at the end of this exercise, we want the system to infer that the best way to distinguish between the two classes is on the basis of color, and that texture, shape, etc., are not relevant to the recognition task. Using the language of hash tables, referring specifically to the manner in which researchers, including ourselves, have talked about hash tables in [19, 26–28, 73], the problem may be reexpressed as asking the vision system to construct a hash table, which in this case would be one-dimensional, and would only involve the color attribute. The one-dimensional hash table would have only two bins, the bins containing the information "Class 1" and "Class 2," respectively. We believe that it would be quite feasible to construct such a system today.

Now consider the more difficult problem of 3D objects. Automatic learning of a hash table here is complicated by the fact that a 3D object can present itself to a vision system in one of a large number of poses. The hash table now must also be able to distinguish between the different poses of an object. Of course, one can conceive of simple situations, such as when we have two classes of 3D object, the objects in one class being of color red and those in the other class of color green. If all that is expected of the system is to learn how to distinguish between the two classes, the system would be no different from the case discussed above. Such simple situations, unfortunately, do not arise in practice, and vision systems must be able to reason about both poses and identities.

In this paper, we will discuss how the recognition of an object that can appear in any of an infinity of poses can be solved by entering feature groupings into the bins of a hash table, each grouping being such that it permits calculation of the pose of the hypothesized object. This system, called MULTI-HASH, assigns to each feature grouping a bin in a multidimensional hash table on the basis of the most discriminatory attribute values associated with the features. MULTI-HASH uses interactive learning to figure out which attributes to use for setting up the hash table and where to establish boundaries between the different bins. During the learning phase, all that a human user has to do is to tell MULTI-HASH which feature groupings in the scene correspond to which groupings on a model object and the identity of the model object. By using the well-known principles of decision trees, MULTI-HASH proceeds to efficiently create a good hash table.

While interactive learning is one of the most distinguishing characteristic of MULTI-HASH, the notions that MULTI-HASH shares most closely with the other

1077-3142/95 $6.00

recently reported vision systems are those of feature groupings for constructing object hypotheses and geometric hashing for speeding up the process of hypothesis formation.

One of the better known vision systems that first used the notion of feature groupings for hypothesis formation is the 3DPO system by Bolles and Horaud [3], an extension to 3D of their 2D recognition system reported in [2]. Subsequently, our laboratory also used feature groupings in the form of local feature sets in a system called 3D-POLY for fast model-based 3D object recognition and localization [7]. A local feature set (LFS) in 3D-POLY consists of two or more surfaces, which are adjacent and rotationally ordered, and the vertex at which they meet. The same kinds of LFSs are used in MULTI-HASH. Another contribution that has also used the notion of feature groupings in a sense similar to ours is by Flynn and Jain [19].

As we said before, the other similarity between MULTI-HASH and some of the modern systems is the use of hashing, a notion whose genesis in the context of model-based vision dates back to the work of Lambdan, Schwartz, and Wolfson (see [48, 49]). That seminal work addressed the problem of affine-invariant recognition of flat objects from 2D data, where affine-invariance allowed flat objects to be at slant angles with respect to the camera. Lambdan, Schwartz, and Wolfson use for features the high curvature points on object boundaries in 2D images. These interest points are described by their positions. Off-line, a coordinate frame is generated for each possible triplet of a model's interest points and then the coordinates of the other interest points are calculated in this frame. These new coordinates which are invariant to affine transformations of the object are used as indices into a hash table in which the triplet used in forming the coordinate frame and the corresponding model identity are stored. On-line hypotheses are created via a voting procedure to find which (model, triplet) pair best matches some triplet of scene interest points. One drawback of this approach is that the only features used are high-curvature boundary points and thus many of the interesting attributes of an object such as surface shape, size, color, etc., are ignored. The reader is referred to the work of Grimson [29], Clemens and Jacobs [9], Lambdan and Wolfson [50], Rigoutsos and Hummel [62, 63], and Costa *et al.* [11] for further discussion on the high computational complexity and the robustness issues associated with the hashing method of [48]. An interesting extension of the basic geometric hashing approach of [48, 49] has been presented by Califano and Mohan [5], where the authors have also expounded on the benefits of using multiple attributes for constructing a hash table.

The geometric hashing approach used in MULTI-HASH is totally different from the one suggested originally in [48]. The main reason that has motivated this difference is that MULTI-HASH is designed for the recognition of 3D objects from 3D data. Previously published parallels to the manner in which we use hashing lie more in the work of Stein and Medioni [73] and Flynn and Jain [19].

In the hashing approach used by Stein and Medioni [73], features such as range edges are decomposed into smaller features consisting of contiguous straight edge segments; attributes such as angles between the adjacent straight segments are then used for forming a hash table. In contrast to the approach in [19] and the way in which we use hashing in MULTI-HASH, Stein and Medioni's method is tantamount to forming hypotheses from observations that are too local and possibly too microscopic in relation to the size and complexity of an object. The use of features that are too local for hypothesis formation shifts the burden of computation to verification. More specifically, the complexity of verification for this method grows quadratically with respect to the number of hypotheses generated. For comparison, in MULTI-HASH the computational effort for verification grows only linearly in the number of hypotheses formed. However, it must be said in defense of the method of Stein and Medioni that when objects are of free-form shape, theirs is probably the only approach to geometric hashing that would work today.

That brings us to the system of Flynn and Jain [19] in which hashing is used in a manner parallel to ours. Hypotheses are formed [19] by using feature groupings consisting of any three surfaces belonging to a model object. Model feature groupings are stored in the bins of a two-dimensional hash table. For polyhedral and conical surfaces, the two dimensions of the table correspond to the two dihedral angles, one between the first and the second surfaces and the other between the second and the third. If one of the surfaces in a feature grouping is spherical, the distance between the centroids of the surfaces is used instead of the dihedral angle. The problem with using just the dihedral angles for constructing a hash table is that such attributes may not carry adequate discriminatory power for many useful model libraries, such as when objects have a large number of surfaces that meet at 90°. In [26], where we reported on a predecessor to MULTI-HASH, we demonstrated how the use of nongeometric attributes, such as color, in addition to geometric attributes, such as shape, is useful in the construction of hash tables. The reader is also referred to the work of Flynn and Jain [20] for an extension of their work in [19] where they explored on-line ranking of attributes and the fact that some objects possess symmetry to reduce the number of hypothesis generated.

We believe that MULTI-HASH improves upon the hashing systems of [19, 26, 73] by eliminating one of their main shortcomings, namely, the use of fixed hash table structure. A fixed hash table, as used in [19, 26, 73], is

one in which each axis of the multiple-dimensional hash table is partitioned uniformly with fixed intervals. Unfortunately, for objects and sensors of practical interest, fixed-structure hash tables result in the formation of hot-spots, meaning some bins with too many entries, resulting in the retrieval of too many hypotheses. Fixed structure hash tables make sense only when it can be assumed a priori that attribute values are uniformly distributed in the table. This assumption is almost never realized in practice.

Two other areas in which MULTI-HASH improves upon the hashing schemes of [19, 26, 73] is in how it addresses the question of which attributes to use for the hash table and how to incorporate attribute uncertainties. MULTI-HASH automatically determines which attributes to use as a function of their discriminatory power with respect to the model-base in question. Another shortcoming of the previous hashing schemes is that they treat attribute value uncertainty as a fixed value. For example, in [19] no attention is paid to attribute uncertainties during the construction of the hash table, but, after the table is in place, hypotheses are retrieved by extracting all bins that fall within a fixed range of the attributed values; the size of the range reflects some a priori assumptions regarding how much uncertainty may be associated with the attributes. Using a fixed range during retrieval is tantamount to modeling the uncertainty by a fixed interval independent of the attribute value. There are two problems associated with modeling uncertainty with fixed intervals. One, it is difficult for a human user to conjure up the interval to use during the process of retrieval, and, two, if the interval used is too large, too many hypotheses will be retrieved; on the other hand, if the interval is too small, valid hypotheses will be missed. MULTI-HASH allows for attribute uncertainties to be dependent on attribute values. This is accomplished by modeling the uncertainty distributions and extracting the parameters of these models during the interactive learning process. A most important benefit of modeling uncertainties, as is done in MULTI-HASH, is that it now becomes relatively easy to reason about the discriminatory power of the various attributes. The ideas presented here could also be incorporated into the geometric hashing systems of the flavor described in [48, 49].

An overview of the MULTI-HASH system is given in the next section. In Section 3, which attributes should be used and how uncertainty is modeled in MULTI-HASH is discussed. Section 4 then begins with a discussion of optimality as it applies to the construction of a hash table and then launches into how decision trees are used in MULTI-HASH for constructing good hash tables. The MULTI-HASH system has been used to successfully and efficiently guide a robot for recognizing and removing objects from piles in the presence of occlusion and against cluttered backgrounds. The results of such experiments using MULTI-HASH are described in Section 5, where we have also compared the performance with our previous results in [26].

2. OVERVIEW OF MULTI-HASH

In this section, we will provide the reader with an overview of MULTI-HASH, emphasizing the overall flow-of-control and the interactive learning aspects.

Objects in MULTI-HASH are represented by surface and vertex features, the edges being stored implicitly in the descriptions of vertices. Each feature is represented by a set of attribute-value pairs. The attribute-value frame for a surface feature is

{
⟨Shape: *Planar or Cylindrical or Other*⟩
⟨Area: *in pixels or square inches*⟩
⟨Color: *3 tuple*⟩
⟨Principal Direction: *3D normal vector*⟩
⟨Planar Parameter: *measures elongation*⟩
⟨Cylindrical Parameter: *radius in inches*⟩
⟨Adjacent Surfaces: *list of surface pointers*⟩
⟨Angles: *between adjacent surfaces in radians*⟩
}

and the attribute-value frame for a vertex feature is

{
⟨Location: *3D position vector*⟩
⟨Adjacent Vertices: *list of vertex pointers*⟩
⟨Outgoing Edge Types: *list of Concave/Convex labels*⟩
⟨Surroundings Surfaces: *list of surface pointers*⟩
}

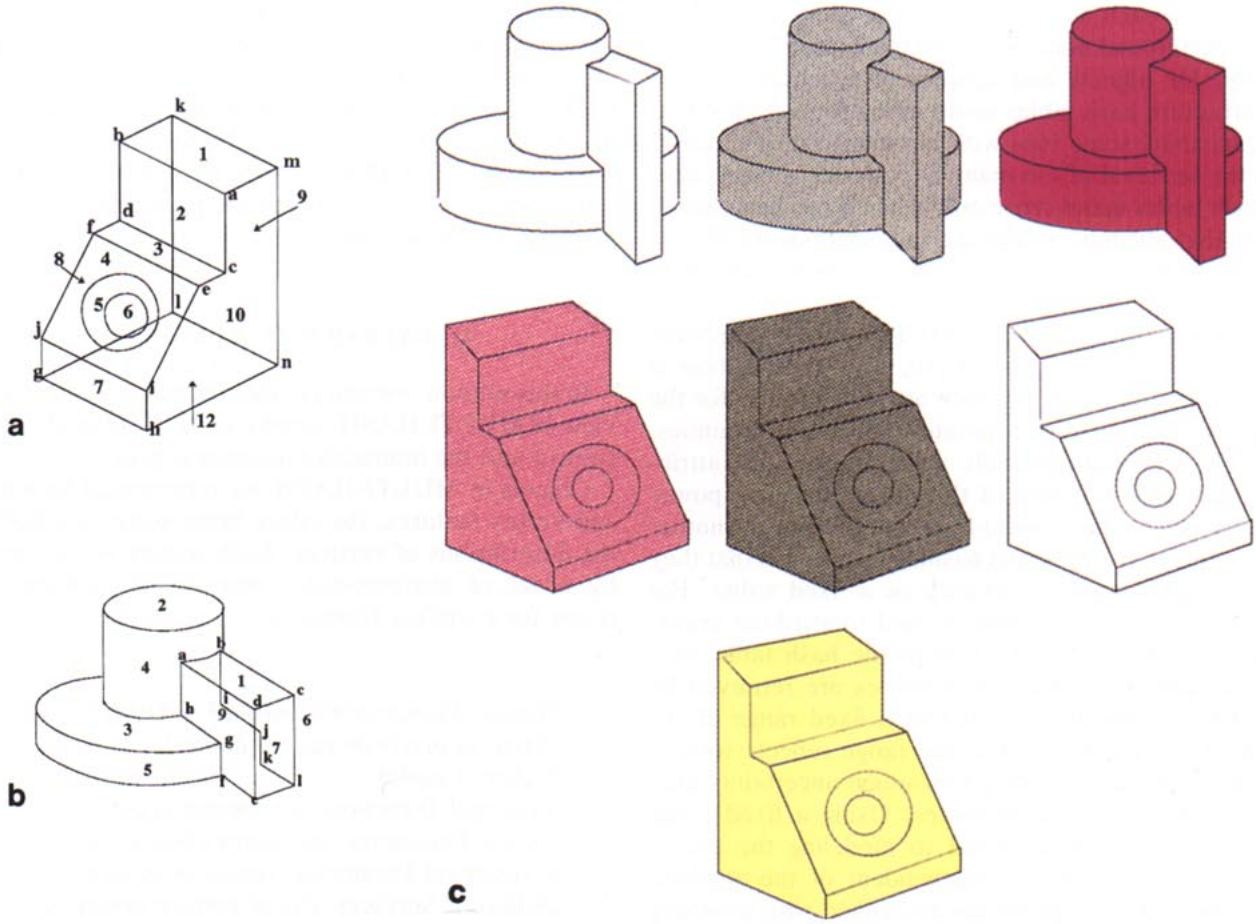
The "values" shown against each attribute represent either all the choices available for that attribute or the nature of the numerical data to be used. An attribute may be

qualitative OR quantitative
AND
geometric OR nongeometric OR relational
AND
viewpoint dependent OR independent.

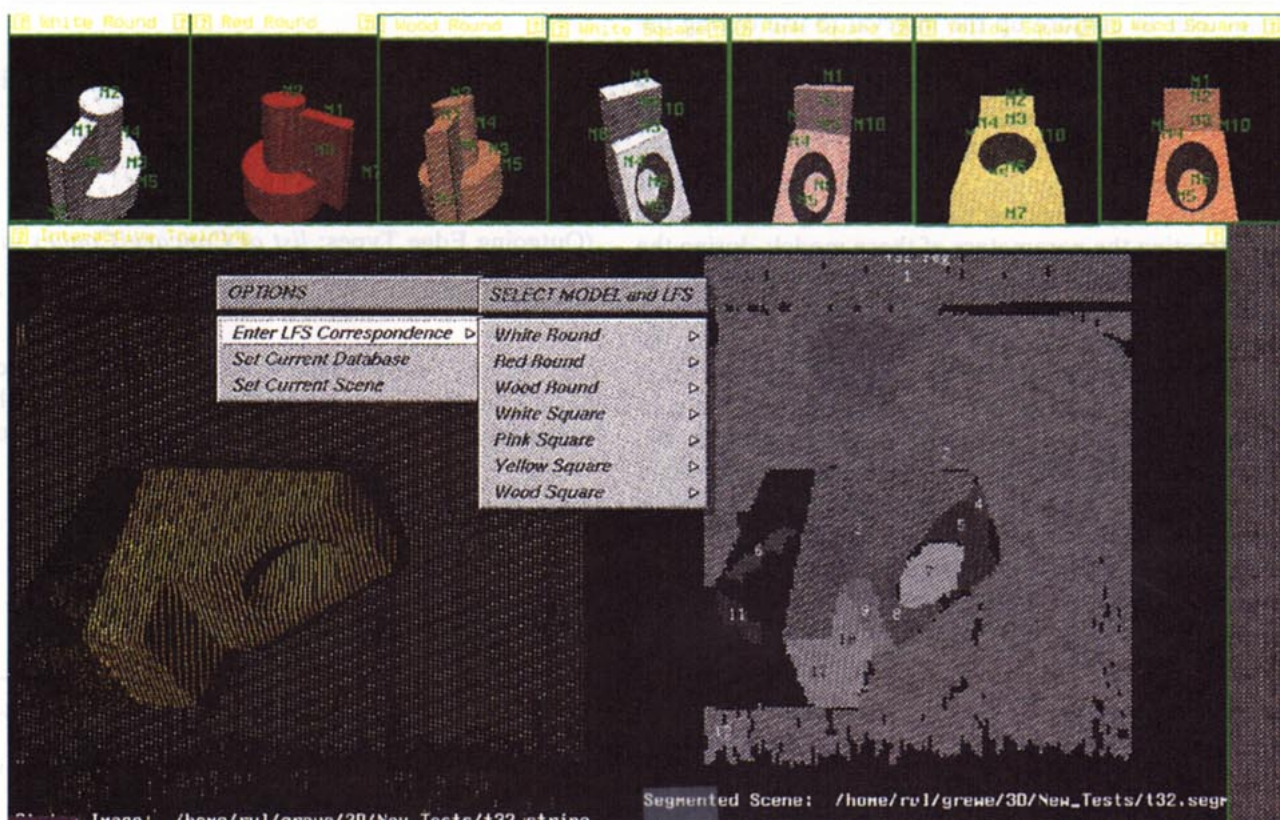
FIG. 1. The model-base used contains objects that differ with respect to both geometry and color. (a) Square-shaped object. (b) Round-shaped object. (c) Colors that the differently shaped objects appear in, including dark and light woods, white, pink, yellow, and red colors.

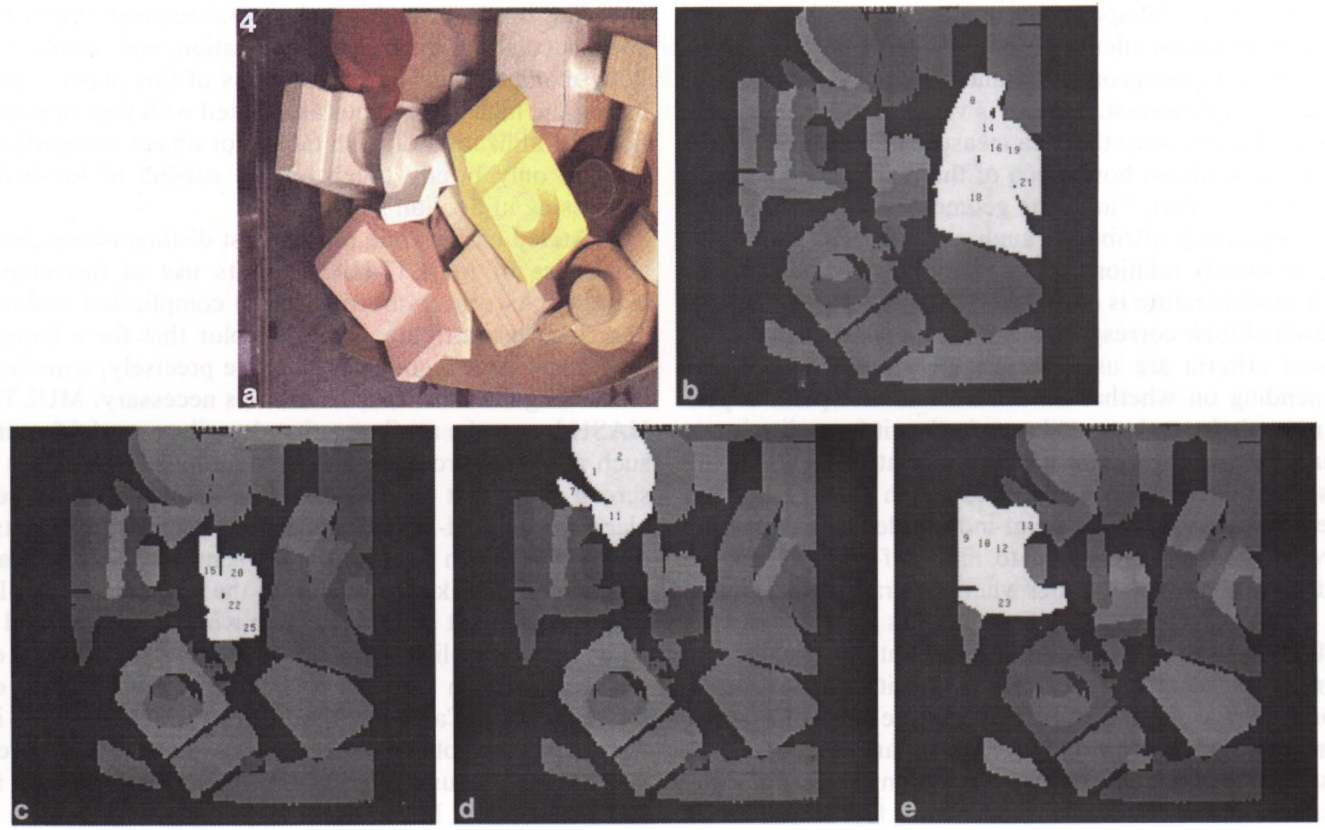
FIG. 3. A screen dump from an SGI machine displaying how interactive training takes place in MULTI-HASH. On the left in the main window is a structured light image of the training scene and on the right is the segmented image.

1


a
b
c

3


OPTIONS
Enter LFS Correspondence
Set Current Database
Set Current Scene
SELECT MODEL and LFS
White Round
Red Round
Wood Round
White Square
Pink Square
Yellow Square
Wood Square
Segmented Scene: /home/rvl/grewe/3D/New_Tests/t32.segr



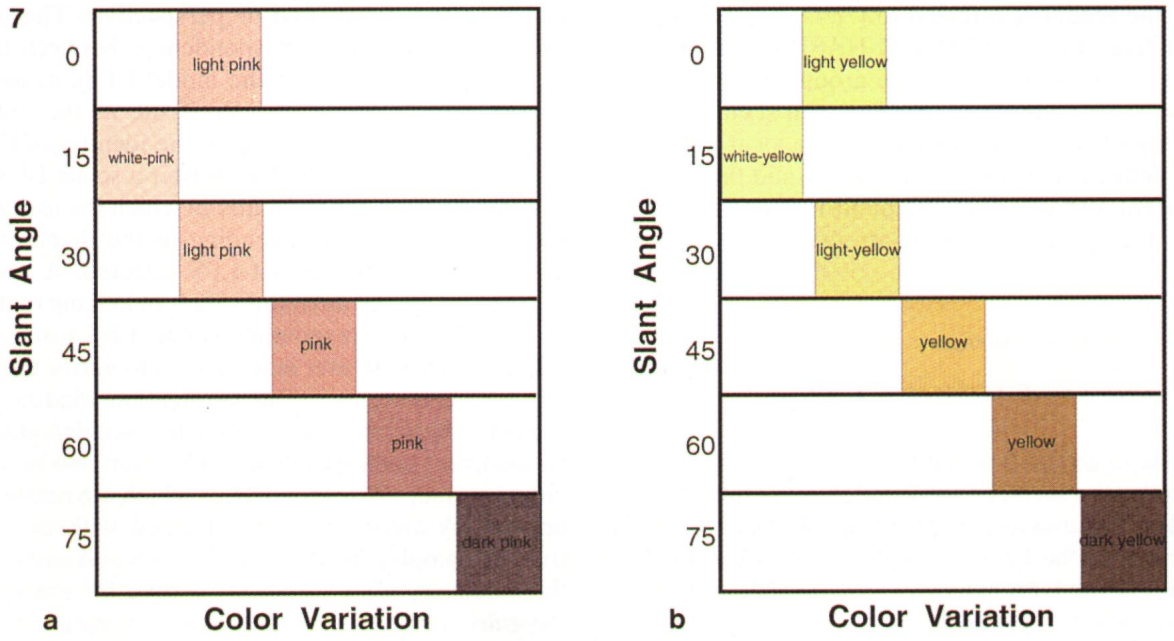


FIG. 4. (a) A pile containing objects from the model-base shown in Fig. 1. (b) First object recognized: square, yellow object. (c) Second object recognized: round, white object. (d) Third object recognized: round, red object. (e) Last object recognized: square, white object.

FIG. 7. (a) Change in measured color as a function of the slant angle for a pink-colored surface. (b) Change in measured color as a function of the slant angle for a yellow-colored surface.

For example, "shape" is a qualitative, geometric, and viewpoint-independent attribute, whereas "color" is a quantitative, nongeometric, and viewpoint-independent attribute. In contrast, "area" is viewpoint-dependent for scene objects, since the area measured for a surface obviously depends on how much of that surface is visible to the sensor. Also, "area" is geometric and quantitative. The adjacency attributes, such as "adjacent surfaces," are obviously relational.

A model feature is considered to match a scene feature if each of their corresponding attribute values match. Different criteria are used for comparing attribute values depending on whether an attribute is viewpoint-dependent or -independent, and on whether it is qualitative or quantitative. Viewpoint-independent attributes that are also qualitative are considered to match if their labels are identical, whereas viewpoint-independent and quantitative attributes are deemed to match if their values fall within a range of each other where this range is a function of the uncertainty modeling process that is part of MULTI-HASH. Viewpoint-dependent attributes are considered to match if certain inequality or subset relationships are met. For example, if the area of a scene surface which is viewpoint-dependent and quantitative is less than the a model surface's area then in terms of these attributes the two features match. For further details on the matching criteria used for viewpoint-dependent attributes, the reader is referred to [7].

Object hypotheses in MULTI-HASH are formed by matching certain kinds of feature groupings, called local feature sets, extracted from a scene with such groupings from the model-base. As defined in [7], a local feature set (LFS) is defined as a vertex of an object and the rotationally adjacent surfaces that surround it. Considering the object in Fig. 1a, the following are two LFSs of the object:

$$\{vertex = i,\ surfaces = 4,\ 10,\ 7\} \quad \text{and}$$

$$\{vertex = a,\ surfaces = 1,\ 10,\ 2\}.$$

A property of an LFS $= \{vertex = v_1,\ surfaces = S_1, S_2, S_3, \ldots\}$ is that its surfaces are listed in a rotationally adjacent order. Rotationally adjacent means that by starting at surface S_1 of the LFS, if one were to walk around the vertex, v_1, of the LFS in a clockwise manner, surface S_2 would be visited followed by S_3.

Figure 2 shows a block diagram of MULTI-HASH. This system can be divided roughly into three parts: (i) The low-level processing ending in feature extraction and grouping into LFSs; this part of MULTI-HASH is in the upper left of Fig. 2. (ii) An off-line part for interactive learning that results in the construction of a multiple-attribute hash table; this part is shown in the dotted box in upper right. (iii) The part for forming hypotheses by querying the hash table, and the subsequent stages for pose calculation, hypothesis verification, and, finally, robotic manipulation. Since the focus of this paper is primarily the high-level issues associated with the construction and utilization of hash tables for object recognition, we will only briefly touch on the subject of low-level processing in Section 5.

As stated earlier, one of the most distinguishing characteristics of MULTI-HASH is its use of interactive learning. As objects become more complicated and are described by attributes such as color that for a human model builder are not easy to define precisely, a method of learning these attributes becomes necessary. MULTI-HASH learns the attributes that describe a model feature such as color through interactive training. Figure 3 is a screen dump that displays how interactive training takes place in MULTI-HASH. The user places the objects, either singly or in piles, under the sensor system. A scan of the scene is taken and fed into the training system. In Fig. 3, on the left side of the main window is displayed a color structured-light image produced by such a scan of the scene and on the right is shown the segmentation of the range data. Candidate model objects are displayed in a row at the top of the screen; each object thus displayed is chosen by the user from a menu listing all the objects in the model-base. Each object is initially displayed in a standard pose and the user is provided with menu options to rotate the object and/or perspective. The user tells MULTI-HASH the correspondences between the LFSs in the viewed scene and the model LFSs using a set of menus. This is done by identifying in the model-base menu, also displayed in Fig. 3, the identity of the model and LFS the user wishes to register a scene LFS with. A window pops up subsequently in which the user types the scene surface id corresponding to the displayed model surface id from the model LFS selected. A verification window listing the attribute values belonging to the model LFS and the corresponding scene LFS just entered is displayed and the user is requested to verify the registration between the two. This verification window acts as a security check, in case the user has accidentally entered an incorrect correspondence. The user also sets up individual surface correspondences which are needed in verification. A database is thus formed with one or more training samples for each model LFS and surface. Using the database, MULTI-HASH learns, for example, what the color yellow means and how to separate it from the color red. How MULTI-HASH does this and how it uses this knowledge to construct a multiple attribute hash table is the topic of the following sections.

After MULTI-HASH has constructed a hash table whose bins contain pointers to model LFSs, hypotheses are created for a given scene LFS by using the values of the attributes used in the formation of the table to access a bin and by retrieving the model LFSs contained in it.

Before accepting a hypothesis retrieved from a bin, MULTI-HASH makes sure that the model LFS matches the scene LFS in question in terms of all of the attributes and not just those used for the formation of the table. For an accepted hypothesis, the pose of the scene objects is estimated and then the system proceeds with the verification of the hypothesis. Pose estimation and verification take place in exactly the same manner as discussed in [7, 8]. The reader is also referred to the appendix of [44] for a tutorial discussion of how pose is estimated.

In addition to interactive learning, another unique aspect of MULTI-HASH is its use of color as an attribute in hashing. Color is a good attribute to use because of its discriminatory power in many model-bases. As pointed out in [75], other reasons for using color include the fact that it is a local surface property that is somewhat independent of resolution and, with appropriate processing, viewpoint invariant. Further discussion regarding this processing will be presented in the next section.

As discussed in Section 5, registered range and color images are collected by a specially built structured-light scanner. The scene is illuminated alternately by a laser light stripe and a broader white light stripe. The laser light stripe yields the range values. For the extraction of color information, the white light stripe is sampled at exactly those points that were previously illuminated by the laser stripe.

In the rest of this paper, we will discuss how MULTI-HASH uses the tools of interactive training, uncertainty modeling and decision trees to construct a good hash table. To whet the reader's appetite for what is to come in the rest of the paper, we show in Fig. 4a a typical test scene; MULTI-HASH retrieves for this scene a total of 161 hypotheses, which is a significant reduction from the 318 hypotheses retrieved for the same scene by an earlier version of this system [26] where a fixed structure was used for the hash table. The other images in Fig. 4 highlight in the segmentation map the scene features used for recognizing the four objects found in the scene.

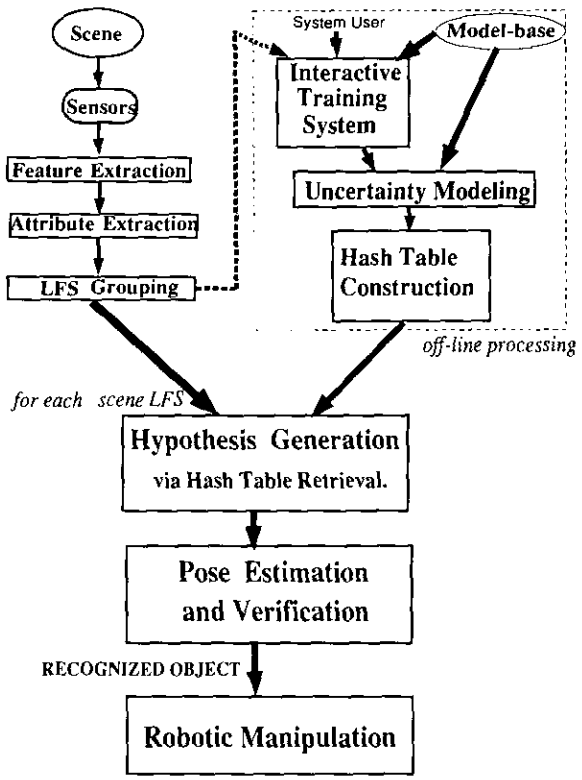


FIG. 2. Block diagram of MULTI-HASH, a 3D model-based object recognition system.

3. ATTRIBUTES AND MODELING THEIR UNCERTAINTIES

The MULTI-HASH system automatically selects the attributes used in the hash table. The methodology used for this automatic selection requires that the ever-present uncertainties in the measurement of attribute values be modeled appropriately. In this section, we will focus on the issue of attribute value uncertainties.

First, let us consider the question of which attributes from an LFS to use in the table. Evidently, the attributes used must be viewpoint invariant since the LFS's in the bins of the table are used for forming object and pose hypotheses and since, initially, the relationship of the object pose to the viewpoint used is not known. From all the viewpoint-invariant attributes, MULTI-HASH selects a subset based on their discriminatory power for making the required distinctions between objects and object poses for the model-base in question. Figure 5 shows histograms for two different attributes for the LFSs for the objects shown in Fig. 1; the numbers shown in the bins of, say, the histogram of Fig. 5b indicate the number of model LFSs for which value of the blue color for one of the surfaces falls in that bin. Clearly, if a vision system had to make a choice between the two attributes shown in Fig. 5, it would choose the blue component of color because of its greater discriminatory power, as the histogram for this attribute is more uniformly distributed. The uniformity of the histogram becomes even better if both these attributes are used together, as illustrated by Fig. 6. In the next section, we will discuss how the tools of decision trees and the results of uncertainty modeling can be used for discovering which attributes are more important and therefore which attributes should play more significant roles in the construction of the table.

In what follows, we will first discuss the sources of the uncertainties in attribute values. Next, we will discuss how these uncertainties are modeled in MULTI-HASH.

Attribute value uncertainties are created by the usual problems that arise from dealing with real-world environ-

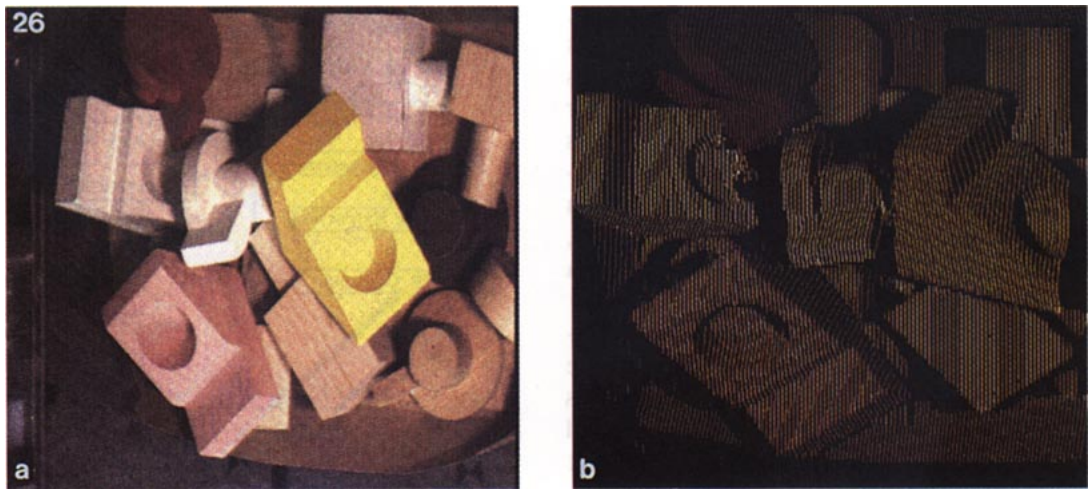


FIG. 26. (a) A scene containing objects from the model-base shown in Fig. 1. (b) Color-composite light-stripe image of the scene. (c) 3D plot of points detected. (d) Segmented image.

FIG. 32. (a) Composite color structured-light image of a typical test scene. (b) The segmentation map.

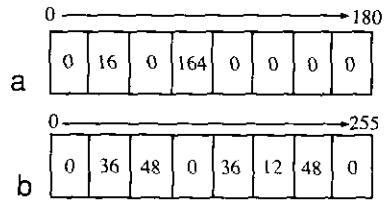


FIG. 5. Histograms illustrating discriminatory power of two attributes. The number shown in each bin is the number of model LFSs for which the attribute value in question falls in that bin. (a) Angle attribute (range 0° to 180°). (b) Blue component of color attribute (range 0 to 255).

ments. For instance, when an object surface is illuminated with a stripe of light, the intensity and the spectral composition of the light reflected in the direction of the camera depend on the slant angle of the surface in relation to both the light source and the camera. To demonstrate this effect, we took the object in Fig. 1a, positioned one of its surfaces at different slant angles and, for each slant angle, we recorded with a color camera the light reflected by the surface. Shown in Fig. 7a is the color of the returned light for the different slant angles, where the slant angle is the angle the surface normal makes with respect to the plane of light emitted by the sensor (see Section 5.1 for a brief discussion of our sensor).

To understand the phenomenon being depicted in Fig. 7a, consider the surface shown in Fig. 8 where, for a given incident light beam, we have shown reflected light consisting of specular components and diffuse components, the components being additive. In the more modern literature dealing with colored light, the specular and the diffuse components are referred to as the surface and the body components, respectively [24, 25, 30–33, 45, 51, 53, 55]. Light reflected by homogeneous materials, such as metals, is dominated by the surface component, whereas light reflected by inhomogeneous materials is dominated by the body component.

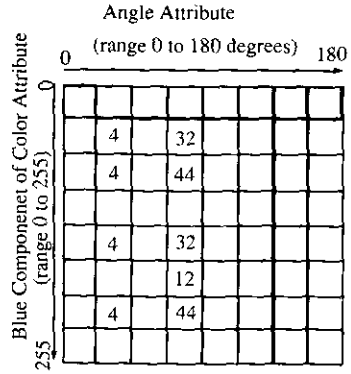


FIG. 6. Discrete histogram for the case when the two attributes shown in Fig. 5 are used simultaneously.

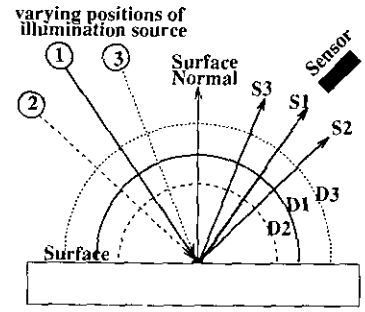


FIG. 8. *Si* represents the specular reflection resulting from illumination source *i*. Diffuse reflection for any illumination source radiates in all directions. The radius of the arc labeled *Di* represents the magnitude of the diffuse reflection component from illumination source *i*. Note that this magnitude is a function of the cosine of the angle the illumination source makes with the surface normal.

When the camera is situated in a direction where the angle of reflection is nearly the same as the angle of incidence, the surface component will usually dominate and the spectral composition of this component will be approximately the same as that of the illumination source. Since our illumination source for the measurement of color is white light, the color shown in Fig. 7 for the slant angle of 15° has the most white in it. The angle between the illumination source and the optical axis of the camera is approximately 30° and, when the surface is at a slant angle of 15°, the camera registers the largest specular component. This is explained approximately by the law of reflection, which says that the angle of reflection subtended by specular component of light with respect to the surface normal must equal the angle of incidence of the illumination light with respect to the normal. More accurately, this phenomenon is also explained by the Torrance–Sparrow model of specular reflection [76]. In [30], a modified Torrance–Sparrow model of specular reflection is discussed where the geometric dependence of the specular components is

$$R_S = \frac{F\,P\,G}{\hat{N}\cdot\hat{V}}, \tag{1}$$

where F is the Fresnel reflectance (see [30]). In this model, it is assumed that the surface is composed of

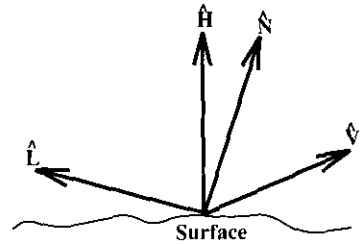


FIG. 9. Geometry of reflection.

small, randomly oriented, mirrorlike microfacets and P is a probability distribution function of the orientation of these facets. G is the geometrical attenuation factor that accounts for the shadowing and masking of surface facets by adjacent facets and is expressed as

$$G = \min\left\{1, \frac{2(\hat{N}\cdot\hat{H})(\hat{N}\cdot\hat{V})}{(\hat{V}\cdot\hat{H})}, \frac{2(\hat{N}\cdot\hat{H})(\hat{N}\cdot\hat{L})}{(\hat{V}\cdot\hat{H})}\right\}, \quad (2)$$

where $\hat{N}$ as shown in Fig. 9 is the normal vector to the surface, $\hat{V}$ is the line of sight to the camera or viewer, $\hat{L}$ is the vector to the illumination source, and $\hat{H}$ is the bisector of the $\hat{V}$ and $\hat{L}$ vectors. Note that G is maximum when the normal vector, $\hat{N}$, is coincident with the vector $\hat{H}$, this being the case when the angle of incidence is equal to the angle of reflectance. The Torrance–Sparrow model of Eq. (1) describes the fact that for larger values of the incident angle, the peak in the specular reflection component will not occur when the angle of reflectance is equal to the angle of incidence but, when the angle of reflectance is somewhat greater than the angle incidence. The data used in Fig. 7 was taken by tilting the surface in question and keeping the $\hat{L}$ and $\hat{V}$ directions constant. Consequently, at the relatively small slant angle of 15°, which is also the angle of incidence, we would expect that the peak of R_S to occur when the angle of reflectance is close to the angle of incidence, in this case 15°, and this is the situation in Figs. 7a and 7b.

When the slant angle of the surface becomes large, the reflected light consists mostly of the body component. But the intensity of this light diminishes rapidly as the surface slant angle approaches 90°. This explains the "darkness" of the result shown for 75°. This diminishing effect is a result of the fact that, as discussed in [38], if we assume a perfectly Lambertian surface, the magnitude of the body component varies as a cosine of the angle of incidence, which is our slant angle. The geometric dependence of the body components in more modern literature is described by the Kubelka–Munk (K–M) theory of the scattering and absorption of light in the colorant layers of the material body. The body reflectance, R_B, is described by an extended version of the K-M theory in [61] as

$$R_B(\theta_i, \lambda) = (1 - R'_S)\frac{C(\theta_i, \lambda)(1 - r_i)[R_\infty(\lambda) - D(\theta_i)]}{2[1 - r_i R_\infty(\lambda)]\cos(\theta_i)} \quad (3)$$

and

$$C(\theta_i) = \frac{w(\lambda)\cos(\theta_i)(2\cos(\theta_i) + 1)}{1 - 4[1 - w(\lambda)]\cos^2(\theta_i)} \quad (4)$$

$$D(\theta_i) = \frac{2\cos(\theta_i) - 1}{2\cos(\theta_i) + 1}, \quad (5)$$

where λ is the wavelength and θ_i the angle of incidence between the light source and the normal to the surface. R'_S is the amount of light that enters the body of the surface. r_i is the internal diffuse surface reflectance and is a function of the index of refraction of the material. R_∞ is the reflectance predicted by the original K–M model of diffuse light and is not a function of θ_i. Note that as the angle of incidence, θ_i, increases R_B will decrease which explains why at large angles, where the body component dominates, the color becomes darker. Shown in Fig. 7b is a similar effect for a yellow-colored surface.

Besides the causes of the variations illustrated in Fig. 7, color measurements can also be affected by the presence of shadows, interreflections between scene surfaces and differences in illumination sources. The capability of humans to perceive the same color for an object independent of at least some of these factors is commonly called color constancy and is a topic of current research [24, 25, 30–33, 45, 51, 53, 55]. Unfortunately, the algorithms that have been proposed for color constancy require cumbersome measurements, in the sense of requiring more or different measurements than what are output by a typical color camera, and tend to be computationally expensive. Fortunately, the manner in which data is collected in MULTI-HASH makes our system somewhat immune to some of the problems with achieving color constancy. As discussed previously, a special structured-light scanner was built for MULTI-HASH that has the ability to construct registered range and color images of a scene. For the acquisition of both the range and the color information, the scene is illuminated one stripe at a time, and this restricted illumination reduces both the presence of shadows in the scene and the effects of mutual illumination caused by interreflections from scene surfaces. Also, being a learning system, MULTI-HASH has a built-in capability to adapt to different white-light illumination sources.

As pointed out by Healey [30], the kind of variations we have shown in Fig. 7 resulting from the changes in the viewing angle with respect to the surface normal can be minimized by two operations: (i) the removal of specular highlights from the measured data; followed by (ii) the normalization of the remaining color measurements. Due to its computational burden, the former we have not implemented. The latter, we have taken care of by normalizing the measured r, g, b values as follows:

$$\hat{r} = \frac{r}{\sqrt{(r^2 + g^2 + b^2}} \times 255 \quad (6a)$$

$$\hat{g} = \frac{g}{\sqrt{(r^2 + g^2 + b^2)}} \times 255 \quad (6b)$$

$$\hat{b} = \frac{b}{\sqrt{(r^2 + g^2 + b^2)}} \times 255. \quad (6c)$$

Figure 10 illustrates the reduction in the variation as achieved by normalization of the measured color for a set

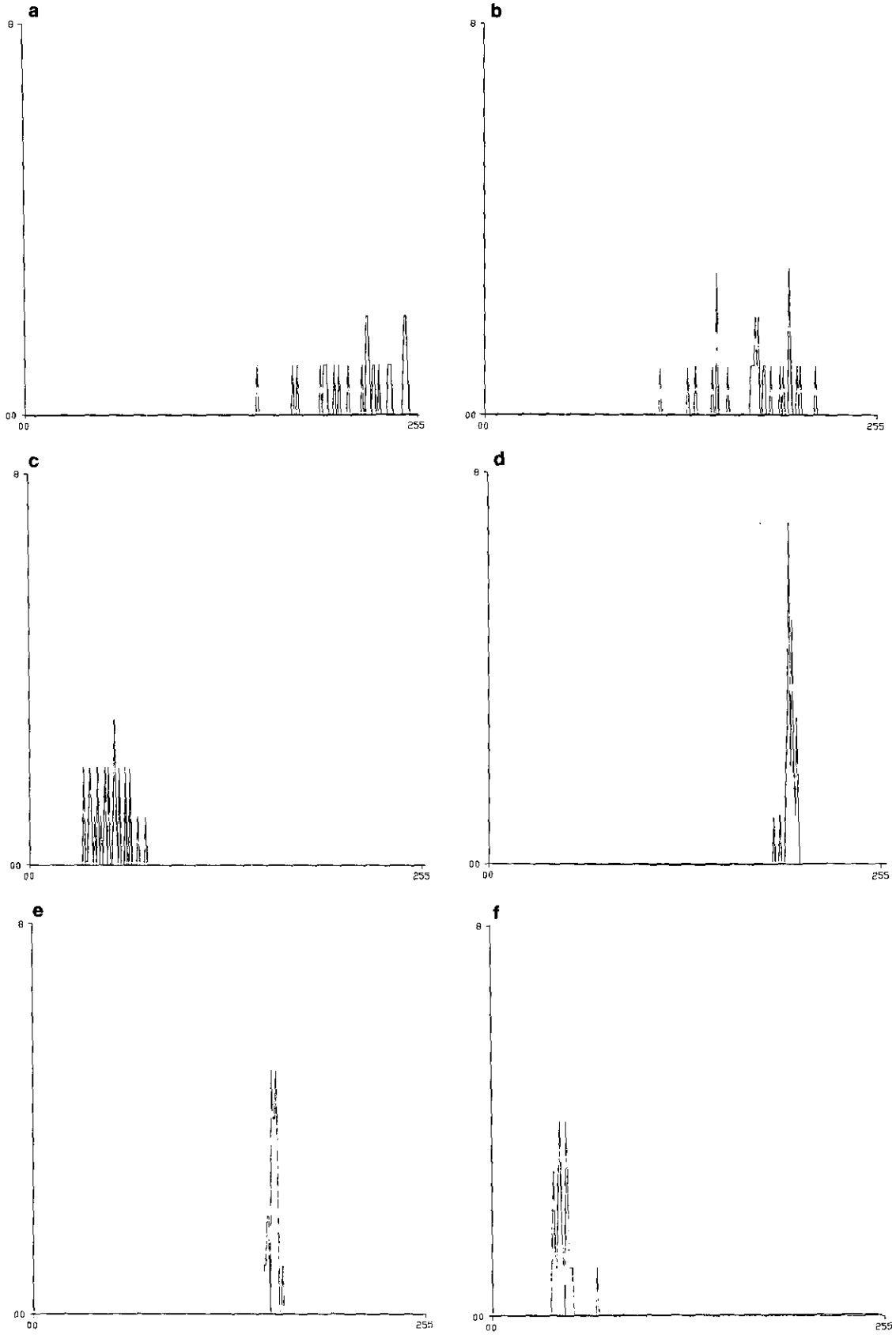


FIG. 10. This figure illustrates that normalized values, $[\hat{r}, \hat{g}, \hat{b}]$, possess smaller variation than the corresponding $[r, g, b]$ values. Plots are histograms of $[r, g, b]$ and normalized $[r, g, b]$ values for yellow surfaces at different orientations (x axes of all plots range from 0 to 255): (a) histogram of r component, (b) histogram of b component, (c) histogram of g component, (d) histogram of $\hat{r}$ component, (e) histogram of $\hat{g}$ component, and (f) histogram of $\hat{b}$ component.

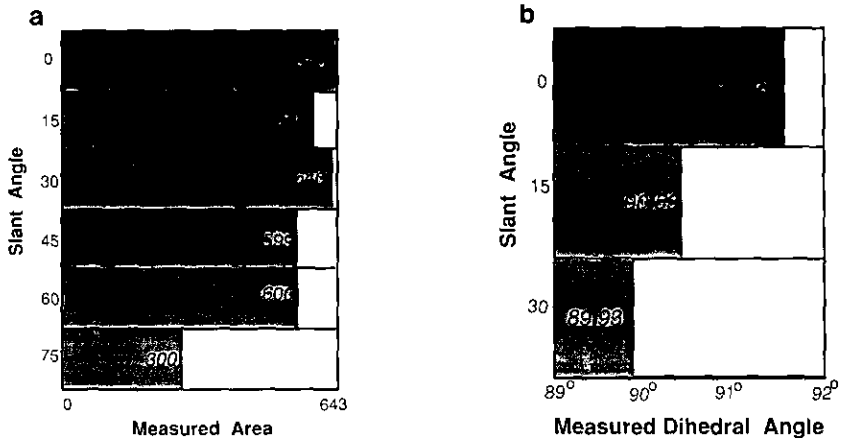


FIG. 11. (a) Area of surface 1 of the object in Fig. 1a as a function of the slant angle. (a) Dihedral angle between surface 1 and that of the object in Fig. 1a as a function of the slant angle, which is defined as the angle between the bisector of the dihedral angle and the direction of the illumination source.

of identically yellow colored surfaces taken at different viewpoints: in (a), (b), and (c) are shown the histograms for r, g, and b, as output by the camera, and shown in (d), (e), and (f) are the histograms for $\hat{r}$, $\hat{g}$, and $\hat{b}$. The heights of the histogram bars are the numbers of surfaces. Note that the span of the color values is much smaller for the normalized color than for the original r, g, and b output by our camera and in this sense the normalized color measurements achieve a greather degree of color constancy.

We will now discuss the uncertainty in the measurement of the attribute "area," again as a function of the slant angle of a surface in relation to the sensor. Figure 11a is a bar graph showing the measured area for surface 1 of the object in Fig. 2a. Area was measured by counting the number of range data points on the surface and dividing by the cosine of the slant angle. The division by the cosine accounts for the decreased number of stripes illuminating the same surface as the angle of the surface increases. As expected, the variation is relatively small for surface angles up to 60°, but then the measured value for the area changes rather quickly. The rapid change for large angles is owing to the discretization errors exacerbating the effects of the relatively few stripes that illuminate the surface at such angles. Of course, for a different set of scanning parameters, such as the number of light stripes per unit of translational movement of the sensor, the sampling rate along each stripe, etc., the dependence of the measurements on the orientation of a surface would be different.

As a final example of the uncertainty in attribute values, consider the dihedral angle between two surfaces. Figure 11b is a bar graph showing the measured values of the dihedral angle between surfaces 1 and 2 of the object in Fig. 1a as a function of the slant angle, which is the angle between the bisector of the dihedral angle and the illumination source. The variation of this attribute is relatively small.

In a manner similar to what was shown in Figs. 7 and 11, all attributes exhibit variations with respect to object pose, scanning parameters, etc. Note that the results shown in these figures were obtained by averaging the attribute values over ten samples in order to reduce random fluctuations. In the rest of this section, we will now address the issue of how these uncertainties can be modeled. Before attribute uncertainties can be incorporated into a hash table, they must be modeled in some manner, especially if the desire is to derive the structure of the table in some optimal manner with respect to the uncertainties.

Experimentally recorded variations, such as those shown in Figs. 7 and 11, sample some continuous function that describes the uncertainty distribution for a given database of objects. Assume, for example, that the measured attribute values for two classes of objects are as marked by X and O in Fig. 12. Presumably, the specific points marked in the figure sample an underlying continuous distributions depicted by the solid ellipses. If we assume that the underlying continuous distributions tell the true story and proceed to construct a hash table taking into account only the specific samples marked, bins V and VI will not contain pointers to class 'O' as they should. Therefore, MULTI-HASH fits Gaussian distributions—whose applicability is verified by using the Kolmogorov–Smirnov Goodness-of-Fit Test [10, 56]—to the training data and then as discussed below performs subsequent truncation of these distributions for practical reasons.

The uncertainty distribution for each attribute for each model LFS is modeled as a single-modal Gaussian func-

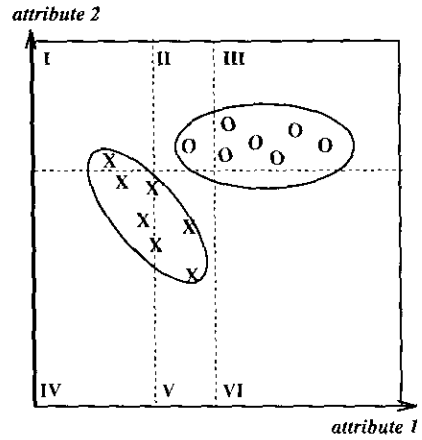


FIG. 12. Curves indicate true distributions and X's and O's mark the locations of training samples for two model LFSs. The partitioning shown corresponds to the training samples directly, as opposed to their continuous distributions.

tion. In other words, for the attribute a_i for model LFS_j, the density is expressed as

$$f_j^i(a_i) = \frac{1}{\sigma_i\sqrt{2\pi}} \exp\{-(a_i - \eta_i)^2/2\sigma_i^2\}, \tag{7}$$

where η_i is the mean of the values of attribute a_i for the model LFS_j's samples and σ_i is the standard deviation. These parameters are estimated by

$$\eta_i = \frac{1}{M}\sum_k x_k \tag{8}$$

$$\sigma_i^2 = \frac{1}{M-1}\sum_k x_k - \eta_i)^2, \tag{9}$$

where x_k is the value of attribute a_i for the kth training sample, and M the number of training samples collected for LFS_j. When an attribute takes on vector values, η_i will be a vector and σ_i^2 will become a covariance matrix.

If a single Gaussian function cannot be assumed to fit the data, methods can be applied to find the correct number of modes or components needed [35, 65, 70] and the parameters of these components [13, 22, 71]. Another benefit to fitting Gaussian functions to the data is that the outliers can be eliminated or their effect reduced [39]. We have tested the assumption that Gaussian fitting is appropriate for each of the system's attributes by applying the Kolmogorov–Smirnov Goodness-of-Fit test [10, 56] to the training data. This test showed that, for this data, for all but one object in the model-base the Gaussian assumption could be accepted at the .1 significance level. What that means is that the test that we applied to the data would reject a correct Gaussian assumption with a probability of .1. The one object whose LFSs failed the test at this level had some of its surfaces composed of different shades of wood.

The Gaussian distribution functions fit to the data span the entire attribute space. In fact, it has nonzero values at all points extending out to infinity in all directions. Because we are dealing with finite domains, these distribution functions must be truncated. A truncated Gaussian function for attribute a takes on the following form where x_l is the left (lower) truncation point and x_r is the right (upper) point:

$$\begin{aligned} f(a) &= \frac{1}{(\sigma_i\sqrt{2\pi} \times A(x_l, x_r))} \exp[-(a - \eta)^2/2\sigma^2] \\ &\quad \text{when } x_l \leq x \leq x_r \\ &= 0 \quad \text{elsewhere} \end{aligned} \tag{10}$$

and

$$A(x_l, x_r) = \int_{x_l}^{x_r} \frac{1}{\sigma\sqrt{2\pi}} \exp\{-(a - \eta)^2/2\sigma^2\}\, \delta a. \tag{11}$$

The truncation points can be chosen, as discussed in [68], by taking their maximum likelihood estimates. Suppose that we have for a given LFS a set of n training samples listed in increasing order with respect to their attribute a values: $(x_1, x_2, .. x_n)$. Given the truncated Gaussian function in Eq. (10), the following is the corresponding likelihood function when all of the training samples fall between x_l and x_r:

$$\prod_{i=1}^{i=n} f(x_i) = \frac{1}{(\sigma_i\sqrt{2\pi} \times A(x_l, x_r))^n} \exp\left[\sum_{i=1}^{n} -(x_i - \eta)^2/2\sigma^2\right]. \tag{12}$$

This is merely a product of the probabilities corresponding to each of the training samples, assuming that they are independent. If a training sample, say x_i, were to fall outside the interval defined by (x_l, x_r), this product of probabilities will go to zero since, in accordance with Eq. (10), the $f(x_i)$ would be zero.

The maximization of Eq. (12) with respect to x_l and x_r is achieved when the following is minimized:

$$A(x_l, x_r). \tag{13}$$

This is obviously minimum when $x_r = x_n$ and $x_l = x_1$.

To allow for greater freedom in modeling, the user can select x_r to be $\eta + k \times \sigma$ and x_l to be $\eta - k \times \sigma$ by specifying k. A safety check is implemented in the modeling program so that if the value of k would cause x_l to be

greater than x_1, then x_l is set to x_1 and also, and if k would cause x_r to be less than x_n then x_r is set to x_n.

For convenience, if the user does not know whether the uncertainty distributions are Gaussian or does not wish to apply goodness-of-fit tests, the user can opt to model the uncertainties by fitting intervals, meaning uniform distributions, to the training data for each model LFS along each attribute axis of the attribute space. In this case, for each model LFS the maximum and the minimum values along each attribute axis are extracted from the training data and are used as the limits of the interval being fit. In contrast to [19], here the uncertainties are modeled with intervals that change as a function of the model LFS being considered. The marginal density function for model LFS_j along the attribute a_i axis is described by

$$f_j^i(a_i) = \begin{cases} \dfrac{1}{a_{\max} - a_{\min}}, & a_{\min} \leq a_i \leq a_{\max} \\ 0, & \text{else.} \end{cases} \tag{14}$$

4. MULTIPLE ATTRIBUTE HASH TABLE CONSTRUCTION

For constructing a hash table, the system must reason over the attributes, taking into account their uncertainties, and figure out how to partition the attribute space so that, at least in the ideal case, each bin of the table contains a pointer to at most one model LFS. As discussed before, previous systems inappropriately partitioned the attribute space using fixed intervals and not as a function of how the space is populated by model LFSs.

One can think of many solutions to this problem of automatic discovery of partitions in the attribute space. We could use neural networks [46], symbolic reasoning [12, 52], statistical pattern classification (single shot classifiers) [16, 17, 21, 22, 71], decision trees [4, 6, 14, 18, 23, 47, 54, 57, 58–60, 67, 72], fuzzy logic [46], etc. In this paper, in what is to follow, we will explore the use of decision trees for discovering the partitions. An advantage of constructing a hash table from a decision tree is that the classification rules described by the decision tree can be printed out to give the user a clear explanation of the classification process.

In the next subsection, we first introduce the notion of an optimal hash table. Since the procedures that could be used to construct optimal hash tables directly tend to be computationally expensive, we will exploit the relationship between hash tables and decision trees and take advantage of efficient algorithms to induce decision trees. In Section 4.2, we then discuss some previous research on the induction of decision trees. In Section 4.3, we describe how MULTI-HASH combines uncertainty modeling and the learning of decision trees to solve the problem of efficiently constructing a multiple attribute hash table. Then, in Section 4.4, we discuss how a decision tree translates into a hash table.

4.1. *Optimal Hash Table*

Ideally, our goal ought to be to find an optimal hash table for hypothesis formation. Optimal means a table which on the average possesses "purer" bins and, at the same time, has a minimum number of bins. A bin is "pure" if it contains only one model LFS; meaning part or all of this model LFS's uncertainty distribution occupies the attribute space encompassed by the bin. A bin is "purer" than another bin if the number of model LFSs contained in the former is smaller than the number contained in the latter. If more than one table has the same "purity," the table that is simpler, meaning has fewer bins, should be chosen. Minimizing the number of bins not only reduces the amount of memory needed, but, as pointed out by Quinlan [58], given a choice between two correct tables it seems sensible to prefer the simpler one on the grounds that it is more likely to capture correctly more samples outside of the training set used to construct the table.

An optimal table can be found by searching over all possible partitions of the attribute space and choosing the set of partitions that are optimal in the sense of purity and, for this set, selecting a table with the minimum number of bins. In what follows, we will show that this notion of optimality can be expressed as the minimization of the average entropy associated with each bin.[1]

The entropy associated with each bin can be expressed as

$$H(bin_i) = -\sum_{j=1}^{J} \begin{cases} P(LFS_j|bin_i)\ log(P(LFS_j|bin_i)), & \text{if } P(\,) \neq 0, \\ 0, & \text{if } P(\,) = 0, \end{cases} \tag{15}$$

where J is the number of different model LFSs and $P(LFS_j|bin_i)$ is the probability that a sample will be from LFS_j given it has fallen into bin_i. Using Bayes' rule, $P(LFS_j|bin_i)$ is calculated as

[1] As the reader will recall, the entropy associated with a set of events is maximized if those events are all equiprobable. Clearly then the entropy would be minimized for the case when the probability distribution over the events is maximally nonuniform, meaning all of the probability is assigned to a single event. That is exactly what we wish to see in each bin of the hash table. In an optimal hash table, each bin will contain the samples from a single LFS.

$$P(LFS_j|bin_i) = \frac{P(bin_i|LFS_j)P(LFS_j)}{P(bin_i)} \quad (16)$$

$$P(bin_i) = \sum_{j=1}^{J} P(LFS_j, bin_i), \quad (17)$$

where $P(bin_i)$ is the probability that any sample randomly drawn from all the LFS distributions will fall in bin_i and $P(bin_i|LFS_j)$ is the probability that a sample drawn randomly from the distribution for LFS_j will fall in bin_i. $P(LFS_j)$ is the a priori probability that LFS_j will be observed in a scene and can be expressed as follows:

$$P(LFS_j) = \sum_{k=1}^{\#models} P(LFS_j|M_k)\, P(M_k)$$
$$\text{where } P(LFS_j|M_k) = 0 \text{ if } LFS_j \notin M_k \quad (18)$$

$$\sum_{k=1}^{\#models} P(M_k) = 1 \quad \textit{and} \quad \sum_{j=1}^{J} P(LFS_j|M_k) = 1. \quad (19)$$

$P(LFS_j|M_k)$ is to be interpreted as the relative frequency of the visibility of LFS_j for model M_k. If the object M_k were to be viewed from a large number of different viewpoints distributed uniformly over the viewing sphere, the number of times LFS_j would be visible divided by the total number of viewpoints would constitute an estimate for $P(LFS_j|M_k)$. $P(M_k)$ is the relative frequency with which the model M_k appears in the scene. If there is no prior knowledge, it is assumed that $P(LFS_j) = 1/J$ for all j, where recall that J is the number of LFSs in the modelbase.

$P(bin_i|LFS_j)$ needed in Eq. (16) can be estimated from the distributions fit to the training samples as

$$P(bin_i|LFS_j) = \int_{l_1}^{u_1}\int_{l_2}^{u_2}\cdots\int_{l_q}^{u_q} f_j(A)\, dA = \prod_{i=1}^{q}\int_{l_i}^{u_i} f_j^i(a_i)\, da_i, \quad (20)$$

where A is a vector composed of all of the attributes used for the table, f_j the distribution fit to the data of LFS_j and f_j^i the marginal distribution along the attribute axis a_i. The parameter q is the dimensionality of the attribute space and l_i the greatest lower bound for attribute a_i describing the underlying space occupied by the bin in question. Similarly, u_i is the least upper bound for attribute a_i.

Expressing $P(bin_i \mid LFS_j)$ as a product of the distributions along each attribute axis is based on the assumption that the uncertainty variations in attribute values are independent. That is, if for a given LFS_j there is a variation in the value of the dihedral angle caused by, say, the pose of the object in relation to the sensor, this variation would be independent of the variation in, say, the blue component of color. We believe that this assumption is justified because any correlations that might exist between the uncertainties in different attributes are at most of second order and, therefore, not very consequential.

Our discussion so far tells us how to compute the entropy associated with each bin, the formula given by Eq. (15). In order to construct a measure of entropy for the entire hash table, we must take into account the entropies associated with all its bins, weighting the contribution from each bin by $P(bin)$:

$$E = \sum_{i=1}^{\#bins} P(bin_i)\, H(bin_i). \quad (21)$$

The entropy associated with a bin will be maximum when all the LFS distributions overlapping with that bin are equiprobable. By the same token, the entropy associated with each bin will be a minimum—and the bin will be pure—when the bin overlaps the distribution from only one LFS, since in that case $H(bin)$ would be zero. As a result the average entropy, E, in Eq. (21) will be a minimum (zero) when each bin contains at most one of the model LFSs (i.e., when the bins are pure). If no zero average entropy table exists, then the table with minimum average entropy will possess bins which are on average as pure as possible.

As mentioned before, an optimal hash table can be constructed by searching all possible partitions of the attribute space and choosing a partition that minimizes the average entropy E. To elaborate, suppose the attribute a_i is quantized into n_i discrete levels for the purpose of measurement and analysis, then along this axis we can have a maximum of n_i partitions. The total number of different partitions that one can construct for the entire attribute space will then be

$$\text{Number of Possible Hash Tables} = \prod_{i=1}^{\#attributes} \sum_{k_i=1}^{n_i} \binom{n_i}{k_i}$$
$$= \prod_{i=1}^{\#attributes} O(2^n) = O(2^{n\times\#attributes}), \quad (22)$$

where n is of the same order of magnitude as the largest of the n_i's. Thus, the computational complexity of finding the optimal hash table by exhaustive search is exponential with respect to both the number of attributes used and the number of quantization levels along each attribute axis. This is demonstrated in Fig. 13, which shows plots of the log of the number of tables versus both the number of attributes and the number of quantization levels for each attribute. For the former plot, where we show the dependence on the dimensionality of the attribute space,

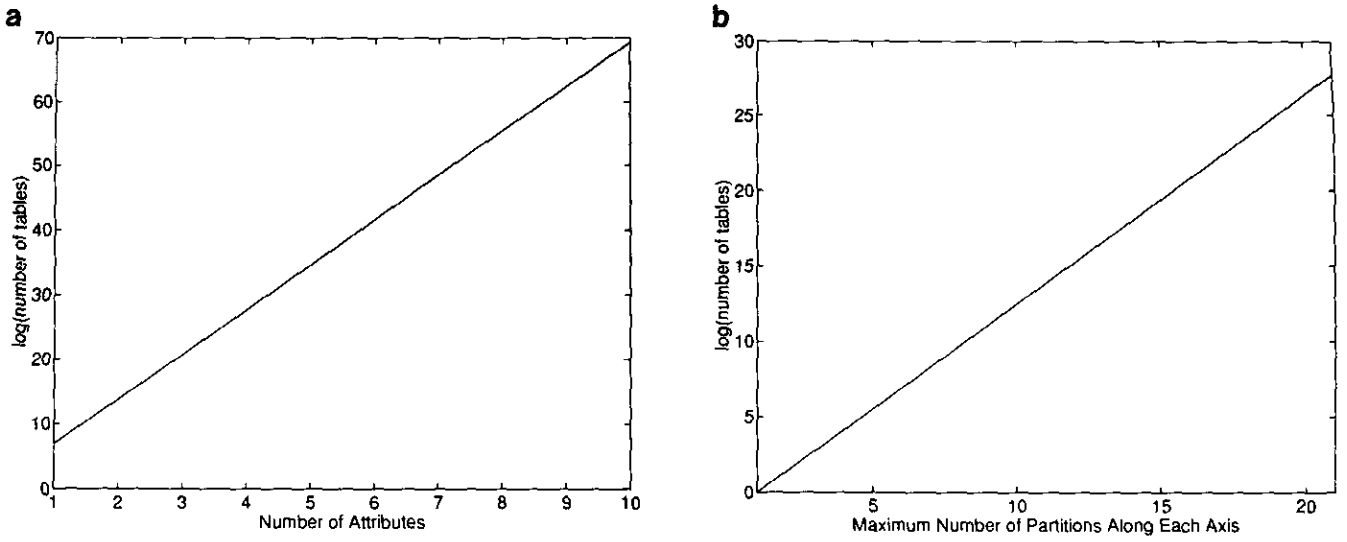


FIG. 13. Log plots of the number of possible tables versus and dimensionality of the attribute space in (a) and versus the number of quantization levels along each attribute axis in (b). For the plot in (a), we used 10 quantization levels for each attribute axis. For the plot in (b), we have a two-dimensional attribute space.

we used 10 quantization levels for each attribute axis. For the latter plot, a two-dimensional attribute space was assumed.

Since the combinatorics of this exhaustive search can make it too onerous for many applications, we will now discuss an approach that uses the induction of a decision tree to efficiently construct a hash table. First, we discuss how optimal tables are related to optimal trees.

The simple relationship between a decision tree and its corresponding hash table is described by the fact that the partitions on the various axes of a hash table correspond to tests at the different nodes of a decision tree. Therefore, since a direct approach to the construction of a hash table is too computationally cumbersome, an indirect approach to the construction of such a table would consist of first constructing an appropriate decision tree. Note that while, from the standpoint of ultimate utility, a hash table is equivalent to a decision tree, the former is more useful on account of constant time access to the contents of the bins. The time it would take to reach the leaves of a decision tree, on the other hand, is proportional to the depth of the tree. So the thing to do is to construct a decision tree and then translate it into an equivalent hash table. As an example, consider the decision tree in Fig. 14a. At each node we have shown a decision threshold for a two-dimensional attribute space. The dark lines shown in the table in Fig. 14b correspond to these decision thresholds. To make a table out of these thresholds, the dark lines must be extended by the dashed lines to create partitions as shown there. The partitions created lead to the table structure. It is interesting to note that there does not exist a unique decision tree for ascertaining the partitions needed for a hash table. For example, the two different decision trees shown in Figs. 15a and 15b yield the same hash table.

It has been proven in [40] that finding an optimal binary decision tree is NP-hard when the optimality criteria are the minimization of the classification error and the minimization of the expected number of tests required to identify an unknown sample. Therefore, strictly speaking, the problems of finding an optimum hash table and an optimum decision tree are both exponentially complex. Yet, due to the existence of already-published large body of work on heuristic algorithms for constructing decision trees, we shall follow the decision tree approach. Of course, as is true of all heuristic algorithms for solving hard combinatorial problems, these algorithms are not guaranteed to yield optimal decision trees, but, as we will show, the result can be very close to the optimum. The reader is referred to [4, 6, 14, 18, 23, 54, 57–60] for literature dealing with efficient heuristics for constructing decision trees. In what follows, we will first review some of the more salient developments reported in the literature, especially as they pertain to our own work. Subsequently, in Section 4.3, we will present a method that is especially suited to our vision domain.

4.2. *Decision Trees: A Brief Review*

As discussed in [60], the induction of a decision tree, meaning the learning of a decision tree from human-supplied examples, is one microcosm of machine learning. In [74], learning is defined as the process in which the attributes necessary for discriminating one concept from an-

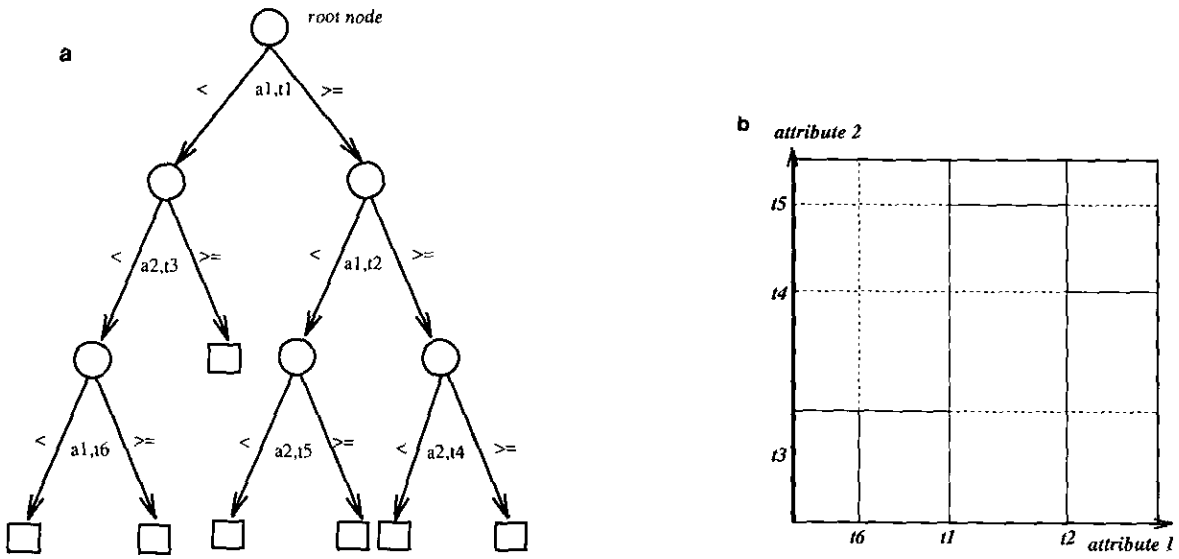


FIG. 14. (a) An example of a decision tree. At each node marked by a circle, a decision threshold is used. Each node marked by a box is a terminal node. The annotation (a, t) under a node means that the attribute a was subject to a decision threshold t, with all samples whose attribute values are less than t going to the left branch, and the rest to the right branch. (b) The corresponding hash table, where each test in the tree creates a partition for the appropriate attribute axis.

other are discovered. This is essentially the classification problem in which a concept is a model LFS in our case.

Unlike other methods of classification that use all of the attributes in a single decision step (often called single shot classifiers), decision trees classify by using a sequence of hierarchical decisions (tests). The interested reader is referred to [4, 6, 18, 23, 54, 57–60] for previous work in the area of top-down induction of decision trees, meaning arranging the tests in a decision tree in such a manner that the more "informative" a test the higher it will be in the tree. Various methods have been proposed to measure the information content of a test. Other aspects of algorithms for inducing decision trees are deciding when a node should no longer be split, meaning that it is a terminal leaf node, and also how to assign class labels (model LFSs) to terminal nodes.

We now present, in the form of steps delineated below, the core algorithm that is common to all methods that perform top-down induction of a decision tree.

1. Begin with a tree consisting of the root node containing all training samples.
2. At each nonterminal leaf node in the current tree, find the test {= (*attribute*, *threshold*)} that splits the training samples at this node into sets such that the *splitting criterion* is optimized.
3. If a node contains only elements from one class or can not be split any further based on the *stopping criterion* used, label it as a terminal node.

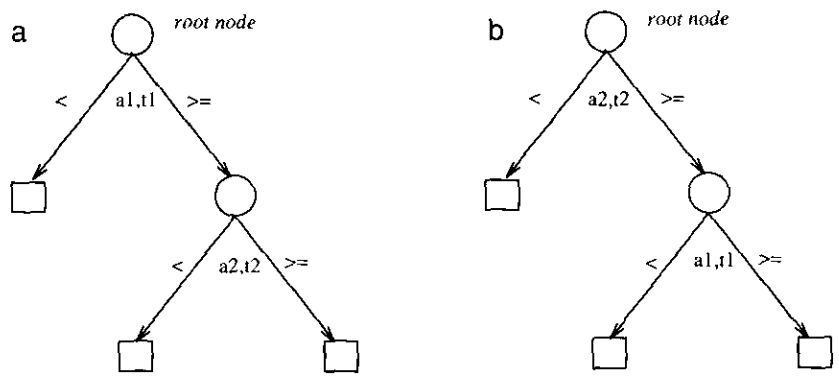


FIG. 15. Two possible trees that lead to the same table. In (a), the first test is applied to attribute $a1$, and in (b), the first test is applied to attribute $a2$.

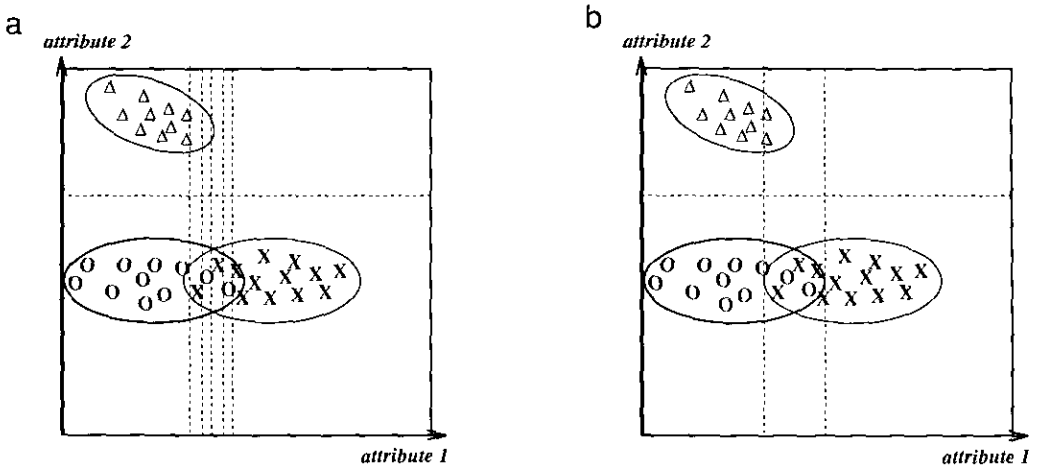


FIG. 16. When training samples are used directly for determining the decision thresholds, overfitting to the data is more likely to occur, as exemplified by too many decision thresholds in (a). On the contrary, when attribute uncertainty is modeled analytically, a better set of thresholds, as shown in (b), can be derived.

4. If any nonterminal leaf nodes remain, go to step 2.
5. Assign class labels to the terminal nodes.

Many splitting criteria have been proposed. Frequently the criteria involve the computation of entropy [4, 58]. Such a splitting criterion makes sense because the goal in growing a tree is to obtain nodes that, to the largest extent possible, contain a single class, meaning are pure, and, therefore, have minimal entropy.

In our approach to constructing decision trees, we use parametric models of uncertainty for the class distributions. The disadvantage of using the training samples directly is that one is more likely to overfit to the data, particularly when the number of training samples is small. By overfitting we mean the creation of a larger number of decision thresholds than are necessary. For example, Fig. 16 shows two different sets of decision thresholds for the two classes displayed there; the decision thresholds shown in (b) are clearly superior to those shown in (a), and therefore correspond to tests in a "right-sized" tree in the sense defined by Breiman *et al.* [4]. Further discussion on the adverse effects of a limited number of training samples on a decision tree is presented in [23, 58, 72]. As discussed in [58], overfitting can be reduced by using an appropriate stopping criterion. Another advantage of using parametric models of uncertainty is that at the lower levels of the tree the estimation of the probabilities that we use in our splitting criterion (discussed in the next subsection) is more accurate than for the case when training samples are used directly. This is a result of the fact that the nodes at lower levels of a tree will usually contain fewer samples. In a sense, the adverse effect of a limited number of training samples becomes magnified at these lower level nodes in a decision tree. Therefore, instead of using the training samples directly, we first fit parametric models to training data and then use these models for growing a decision tree.

4.3. *Growing a Decision Tree*

In this section, we discuss a method to efficiently construct a decision tree from which a multiple-attribute hash table can be directly created. Specifically, modeling of the uncertainty present in the training data and the efficient induction of a decision tree are combined in this method. Recall that attribute values for every class possess uncertainty and are thus described by a distribution. Because we utilize distribution models we no longer have to retain the training samples for the purpose of inducing a tree. Figure 17 shows a block diagram of our approach.

The search for the best test at each node as dictated by the splitting criterion involves simple integrations of the uncertainty distributions. At each node in our decision tree, only binary decisions are made for quantitative attributes; for qualitative attributes with v values the node is split into v children.[2] It is desirable to make a binary decision if possible because the best single partition will be found at each node. As pointed out by Quinlan [58], splitting a node into more than two modes introduces a bias into selecting optimum decision thresholds, in the sense that the entropy associated with each child node will, in general, become lower as the size of the attribute space associated with the child node decreases.

[2] If hierarchical relationships for a qualitative attribute are known—such as when rectangular and square shapes can be expressed more generally as parallelepiped shapes—then u-ary decisions, where $u < v$, can be made for these variables as discussed in [54]. Note that qualitative attributes are treated as having one-point distributions.

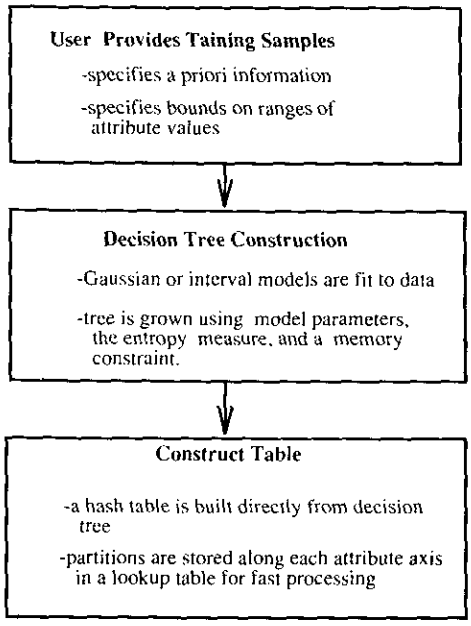


FIG. 17. From the training samples provided by the user to the construction of the hash table, this block diagram shows the overall organization of the processing steps.

In line with the discussion in Section 4.2, we will now present the core decision tree growing algorithm, shown in Fig. 18. Stored at each node n in the tree are

1. the test used to split the node into child nodes, the test described by the pair (*attribute*, *threshold*);

2. for all j, the joint probability $P(n, LFS_j)$ of node n and LFS_j;

3. $P(n)$, the probability that an LFS drawn randomly from the entire training set will reside at node n.

```
1   grow_tree_procedure(tree)
2   { Create tree with a root node
3     for each leaf node n ∈ tree that is not Terminal
4           { choose the test m such that
5         ∀ tests m'≠m  E(n,m) ≤ E(n,m')
6           if (stopping_procedure(n, m) == FALSE)
7                 { split n using the test m
8                   store m at node n
9             create children nodes n_l, and calculate P(n_l)
10              and P(n_l,LFS_j) for all j; store these at n_l.
11                  for each child n_l
12                    if n_l contains only one class
13                          { it is pure, label n_l as Terminal}
                    }
14            else
15          { label n as Terminal}
            }
    }
```

FIG. 18. Procedure to grow a decision tree.

As for the semantics of these probabilities, note that associated implicitly with each node is a portion of the attribute space. So, when, say, the root node is split into two child nodes, one of the child nodes inherits some portion of the attribute space, the rest going to the other. Keeping this in mind, the interpretations to be given to $P(n)$ and $P(n, LFS_j)$ parallel those given earlier to $P(bin)$ and $P(bin, LFS_j)$.

For the splitting criterion in the tree growing procedure, we associate two different entropies with each node. One, the entropy that reflects the various LFS distributions at that node; this, in common with the notation used earlier for the entropies of the bins, will be denoted by $H(n)$ at node n. Two, the average entropy calculated at node n by averaging over the entropies associated with all its child nodes; this entropy will be denoted by $E(n, test)$, where *test* is the test used to split the node n into its children. The splitting criterion at each node n selects a test that minimizes $E(n, test)$.

If a given test *test* splits a node n into child nodes $n_1, \ldots, n_m$, the expression for $E(n, test)$ is given by

$$E(n, test) = \sum_{i=1}^{number\ children} p(n_i \mid n) H(n_i) \tag{23}$$

and the expression for $H(n)$ by

$$H(n) = -\sum_{j=1}^{J} P(LFS_j \mid n) \log(P(LFS_j \mid n)), \tag{24}$$

where

$$P(LFS_j \mid n) = \frac{P(n \mid LFS_j) \times P(LFS_j)}{P(n)} \tag{25}$$

$$P(n) = \sum_{j=1}^{J} P(n, LFS_j). \tag{26}$$

In a manner parallel to Eq. (20), the conditional probability $P(n \mid LFS_j)$ is given by

$$P(n \mid LFS_j) = \int_{l_1}^{u_1} \cdots \int_{l_q}^{u_q} f_j(A)\, dA = \prod_{i=1 l_i}^{q} \int_{l_i}^{u_i} f_j^i(a_i)\, da_i, \tag{27}$$

where A is a vector composed of all of the attributes, f_j the joint distribution fit to the data of LFS_j, and f_j^i the marginal distribution along the attribute axis a_i. For each attribute a_i, l_i is the highest lower bound used in the tests along the path leading from the root down to the node n including the current test under consideration. In a similar sense, u_i is the lowest upper bound for attribute a_i.

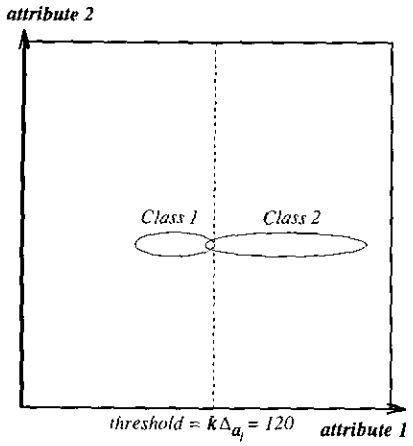


FIG. 19. Threshold selected to minimize entropy in Eq. (23).

These values for a node can be determined by tracing backward from the current node to the root, or for efficiency, the values could be stored at each node as the tree is grown. As part of the background knowledge necessary for any decision tree algorithm, it is also necessary to specify the range of values of each attribute and the quantization interval for the attribute. Specification of the latter is required since the search for the optimum threshold along an attribute axis takes place in steps of the quantization interval.

To give the reader some intuitive insights into how the above formulas work, assume that we have a two-attribute space and two model LFSs, whose attribute uncertainties are represented by truncated Gaussian functions labeled Class 1 and 2 shown in Fig. 19. The root of the tree represents the entire attribute space containing all of the model LFS distributions. Now we must decide what test to carry out in order to divide this space into two or more subspaces, each assigned to a different child node. The test could be on either of the attributes. Since each test is basically a thresholding operation on an attribute value, we must select a threshold on one of the attributes. The simplest approach to solving this problem consists of assuming a quantization of the attribute axes, meaning the existence of the smallest indivisible segment along each of the attributes, denoted Δa_i. The formulas shown above then call for testing for each threshold, as a multiple of Δa_i, for both attributes. Assume that we are testing the threshold

$$k \times \Delta a_i$$

on attribute a_i to figure out how the distributions at the root node should be divided up among the child nodes. In Fig. 19, we show this threshold as equal to 120.0 for attribute a_1. The means and standard deviations of each truncated Gaussian model fit to each model LFS class are

Class 1: $\eta_1 = 100.0, \sigma_1 = 10.0$ and $\eta_2 = 100.0, \sigma_2 = 3.0$;

Class 2: $\eta_1 = 155.0, \sigma_1 = 20.0$ and $\eta_2 = 100.0, \sigma_2 = 3.0$.

The Gaussian functions are truncated at $\pm 2\sigma$ around each mean, and we will assume that each model LFS is equally likely, $p(LFS_j) = 1/2, j = 1, 2$. We begin by estimating $p(root \mid LFS_j)$ using Eq. (27):

$$p(root \mid LFS_1) = p(root \mid LFS_2) = 1.0.$$

Thus,

$$p(root, LFS_1) = p(root, LFS_2) = 0.5 \text{ and } p(root) = 1.0$$

and

$$p(LFS_1 \mid root) = p(LFS_2 \mid root) = 0.5.$$

These estimates are used to compute the entropy associated with the root by using Eq. (24), and the resulting value is approximately 1.3863. Note that the more equal the probabilities $p(LFS_j \mid root)$, the larger the value of the entropy. Now, in order to compute E, we need to estimate the entropies $H(n_i)$ associated with each resulting child node using Eq. (24). Child n_1 covers the space to the left of the partition and child n_2 covers the space to the right. In our case, the threshold shown in Fig. 19 would result in the following:

$$p(n_1 \mid LFS_1) = \int_{80}^{120} f_1^1(a_1)\, da_1 \times \int_{94}^{106} f_1^2(a_2)\, da_2 = 1.0 \times 1.0 = 1.0$$

$$p(n_1 \mid LFS_2) = \int_{115}^{120} f_2^1(a_1)\, da_1 \times \int_{94}^{106} f_2^2(a_2)\, da_2 = 0.0181247 \times 1.0 = 0.0181247$$

$$p(n_2 \mid LFS_1) = \int_{120}^{120} f_1^1(a_1)\, da_1 \times \int_{94}^{106} f_1^2(a_2)\, da_2 = 0.0 \times 1.0 = 0.0$$

$$p(n_2 \mid LFS_2) = \int_{120}^{195} f_2^1(a_1)\, da_1 \times \int_{94}^{106} f_2^2(a_2)\, da_2 = 0.9818753 \times 1.0 = 0.9818753.$$

Here the values for the integrals of the Gaussians are obtained from standard published tables for such purposes. From these numbers, we get the following estimates:

$$p(n_1, LFS_1) = 0.5 \times 1.0 = 0.5$$

$$p(n_1, LFS_2) = 0.5 \times 0.0181247 = 0.0009062$$

```
1   stopping_procedure(node n, test)
2   { if ([E(n,test) - H(n)] < Zero_Difference)
3       { Stop Splitting
4         return(TRUE) }

5     if total bin count with test > MAX_BINS
6       { Stop Splitting
7         return(TRUE) }

8     if all of the LFS distributions falling in n totally overlap
9       { Stop Splitting
10        return(TRUE) }

11    else
12      { Make Split
13        return(FALSE) }
    }
```

FIG. 20. Procedure for implementing stopping criteria.

$$p(n_2, LFS_1) = 0.5 \times 0.0 = 0.0$$

$$p(n_2, LFS_2) = 0.5 \times 0.9818753 = 0.490933$$

$$p(n_1) = 0.5 \times 0.0009062 = 0.5009062$$

$$p(n_2) = 0.0 + 0.4909377 = 0.4909377$$

$$p(LFS_1 \mid n_1) = 0.9981909$$

$$p(LFS_2 \mid n_1) = 0.0018091$$

$$p(LFS_1 \mid n_2) = 0.0$$

$$p(LFS_2 \mid n_2) = 1.0.$$

Equation (24) then leads to the following estimates for $H(n_i)$:

$$H(n_1) = 0.0132318, \quad H(n_2) = 0.0.$$

Also

$$p(n_i \mid root) = p(n_i, root)/p(root) = p(n_i)/p(root)$$
$$= p(n_i)/1.0 = p(n_i).$$

These estimates lead to the following value for E:

$$E = 0.0066279.$$

Identical computations would be carried out at every quantization step along the a_1 and a_2 axes. Assuming, for the sake of argument, that every attribute is quantized into Q levels, we will have $2Q$ such calculations for the discovery of the best test. For the example here, as it turns out, the test $a_1 = 120$ yields optimal splitting at the root node since it results in the smallest value for the average entropy, E.

Besides the splitting criterion, also needed are stopping criteria that stop the growth of the tree at a node. The Stopping_Procedure shown Fig. 20 is based on the following three criteria:

1. If a node is pure, meaning that its bin intersects with the distribution of only one LFS, no further purpose would be served by splitting such a node. Enforcement of this criterion is effected automatically by our entropy-based node splitting procedure since the average entropy E computed at such a node for any split would not be less than the node entropy H. This manner of stopping the growth of a tree corresponds to line 2 in the procedure Stopping_Procedure shown in Fig. 20.

2. The other reason for not splitting a node is that if two or more LFS class distributions *completely* cover the region of the attribute space corresponding to this node. Consider the tree shown in Fig. 21, where the root node and node n_2 give rise to the tests t_1 and t_2, respectively. For the sake of illustrating this splitting criterion, we will assume that we have only two LFS classes and that only one attribute is involved. The tests t_1 and t_2 are assumed to be located as shown in Fig. 21a with respect to the distributions for the two classes. It is therefore clear that the portion of the attribute axis belonging to node n_3 will be the interval between the points t_1 and t_2 shown in Fig.

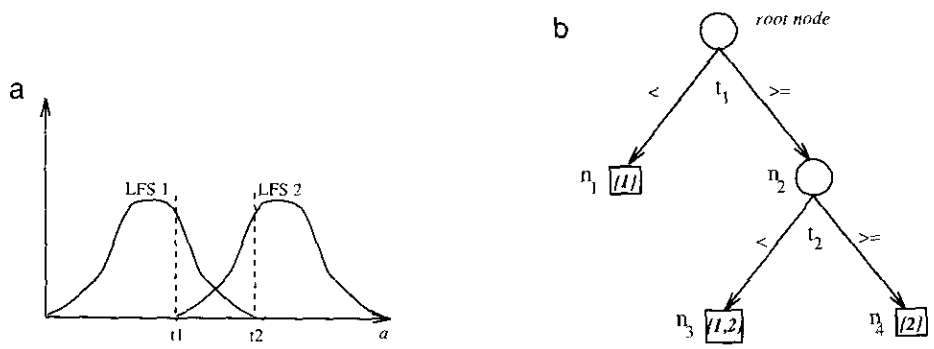


FIG. 21. (a) Distributions of model LFSs 1 and 2 over the attribute a axis. (b) A decision tree using tests on attribute a at thresholds t_1, and t_2. Boxes indicate terminal nodes, and brackets at each terminal node contain the labels of LFSs residing at that node.

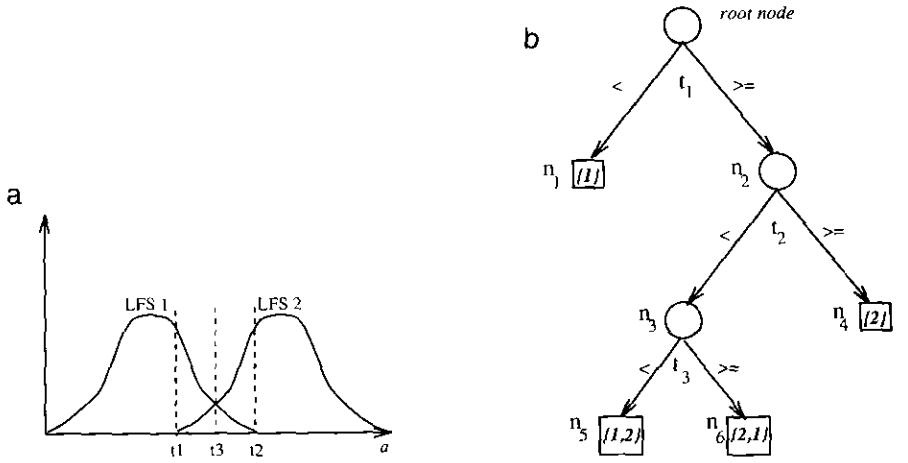


FIG. 22. (a) Distributions of model LFSs 1 and 2 over the attribute a axis. (b) A decision tree using tests on attribute a at thresholds t_1, t_2, and t_3. Boxes indicate terminal nodes, and brackets at each terminal node contain the labels of LFSs residing at that node.

21a. Since the two class distributions overlap this interval completely, it would be pointless to split node n_3 further because the children resulting from any possible split would still point to both LFSs.

3. Since the number of bins in the hash table grows exponentially with the depth of a decision tree, it is important to also add an additional stopping criterion that invokes a memory constraint. If the proposed test to split a node results in the number of bins in the corresponding table to exceed the maximum desired, the node is labeled terminal and not split further. The number of bins in a hash table corresponding to a tree can be calculated by

$$\prod_{i=1}^{\#attributes} (\#\ of\ unique\ tests\ in\ tree\ on\ a_i + 1). \quad (28)$$

These then are the stopping criteria we currently use in MULTI-HASH. It is possible to make the second stopping criterion more sophisticated by actually going ahead and splitting a node with completely overlapping LFS distributions to yield child nodes that point to the same LFSs but with different probabilities. To explain this idea in greater detail, we have shown in Fig. 22a and two LFS class distributions of Fig. 21a, except for the presence of one more test, t_3, for splitting node n_3. This test will be found by the same splitting procedure as in our tree growing procedure since the average entropy at node n_3 for t_3 will be less than the average entropy for any other choice of the test, assuming of course equiprobable LFSs. Shown in Fig. 22b is the resulting tree. At the child node n_5, since $P(LFS_1 \mid n_5) > P(LFS_2 \mid n_5)$, the LFS entries at node n_5 will be stored as an ordered list $\{LFS_1, LFS_2\}$. Similar arguments dictate that we store the ordered list $\{LFS_2, LFS_1\}$ at node n_6. The advantage of ranking the LFS entries at the terminal nodes is that the hypothesis verification can be carried out first with the highest-probability LFS. Shown in Fig. 23 is the extended Stopping_Procedure that includes this modification of the second stopping criterion.

4.4. *Translating a Decision Tree into a Hash Table*

The nodes in the decision tree grown give us a set of tests on the attributes. To construct a hash table, these tests are output into a file; the order in which the tests are listed for each attribute is not important. Subsequently,

```
1   stopping_procedure(node n, test)
2   { if ([E(n,test) - H(n)] < Zero_Difference)
3           { Stop Splitting
4             return(TRUE) }

5     if total bin count with test > MAX_BINS
6           { Stop Splitting
7             return(TRUE) }

8     if all of the LFS distributions falling in n totally overlap
9     { if ∃ child n_i and LFS_j where
            P(LFS_j | n_i) > Absolute_Thresh
          AND
            ∀k P(LFS_j | n_i) - p(LFS_k | n_i) ≥ Diff_Thresh
                  { Make Split
10                  return(FALSE) }
11      else
12                { Stop Splitting
13                  return(TRUE) }
            }

14      else
15          { Make Split
16            return(FALSE) }
    }
```

FIG. 23. Stopping procedure extended to include splitting of nodes containing totally overlapping model LFS distributions.

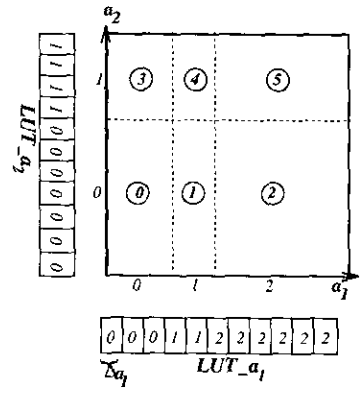


FIG. 24. Hash table and the corresponding look-up tables (LUT), one for each attribute used in the table. A look-up table (LUT) is a mapping from the quantization intervals to bin segments along a particular attribute axis.

the bins of the hash table are carved out by partioning the attribute space using these tests. While it is true that in the decision tree a test at a node partitioned only that part of the attribute space that corresponded to that node, in the hash table the same test partitions the entire attribute space. This point was made earlier with the help of Fig. 14, where the solid partitions correspond to the tests at the nodes and their continuations by dashed lines to the resulting bin boundaries for the hash table.

A hash table is represented by a one-dimensional array of bins, with the index of each bin calculated by a lexicographic ordering of all the bins with respect to all the attribute axes. The idea is illustrated very simply for the case of a two-dimensional attribute space. For this case, for which the table index of each bin is displayed by a circled number in Fig. 24, the index of a bin that contains the point (a_1, a_2) in the attribute space is given by

$$Table\ Index(a_1, a_2) = LUT\_a_2(a_2) \times Num\_Seg\_a_1 + LUT\_a_1(a_1) \quad (29)$$

where a_1 and a_2 are the LFS attributes used in the example table of Fig. 24. LUT_a_i is a look-up table which maps an attribute value into its appropriate segment along the attribute axis, where a segment is the interval between successive partitions along an attribute axis. Num_Seg_a is the number of segments along the attribute a axis. A LUT for an attribute is created by partitioning the range of values it can take into intervals of size Δ_a, where Δ_a is the interval between two successive quantization levels. This generalizes to N attributes in an obvious manner.

After the structure of the table has been set up, the bins must be filled with pointers to the model LFSs which fall in them. A model LFS is said to fall in a bin if its uncertainty distribution overlaps with the portion of the attribute space associated with the bin in question. In our current implementation, these pointers for each bin are computed by taking the intersection of the bin with the LFS distributions. These calculations are relatively trivial because of our assumption of truncated Gaussian or uniform distributions independently for each of the attributes. The intersection calculations therefore reduce to mere comparisons of the bin boundaries and the limits of the distributions.

4.5. *On the Optimality of the Hash Tables*

It is important to realize that no claim is being made that the hash tables constructed by growing decision trees, in the manner we have described in this section, are optimal. In fact, we do not believe that it is possible to construct a provably optimal hash table without running into exponential complexity.

Ours is a heuristic procedure that uses locally optimal decisions for constructing a decision tree, meaning that the decisions are the best they can be at each individual node. It goes without saying that this local optimality does not necessarily translate into global optimality. Despite this lack of guarantee about global optimality, we are convinced that MULTI-HASH is capable of generating good hash tables for all computer vision problems of practical interest. As we will show in Section 5.2, in comparison with a manually specified fixed-structure hash table described earlier in [26], MULTI-HASH's performance was much superior, in the sense that a smaller number of object hypotheses were generated by MULTI-HASH.

Since MULTI-HASH carries no guarantee of global optimality, one is of course curious about how close the bin partitions produced by MULTI-HASH might be to the ideal. In other words, how "far" is the hash table output by MULTI-HASH from the globally optimum hash table that one might construct by exhaustive search over all possible partitions of the attribute space. Since exhaustive search involves exponential complexity, for obvious reasons we did not seek a direct answer to this question. However, we did try to answer the question by altering the bin partitions produced by MULTI-HASH and computing the average entropy, our measure of global optimality. In Section 5.2, we will present an example for real data that resulted in the hash table shown in Fig. 31. The plot shown in Fig. 25 is of the average entropy associated with the different tables produced by displacing the bin partitions. At the displacement-parameter $k = 0$, the average entropy shown is for the original table constructed using the partitions of Fig. 31. At other values of k, each threshold partition in Fig. 31 is displaced as

$$New\ partition_j^i = partition_j^i + k \times span\_attribute_j, \quad (30)$$

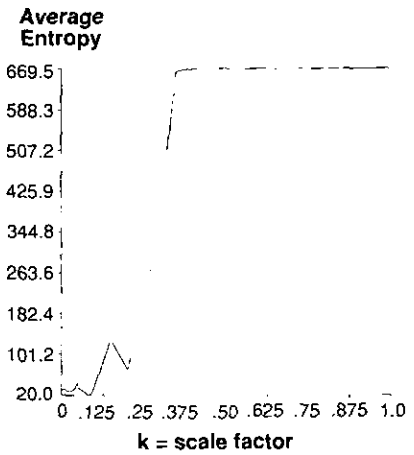


FIG. 25. Variation in the average entropy associated with a table as the partitions are moved by k times the range of each attribute. At $k = 0$, the partitions shown in Fig. 31 are used. Note at $k = 1$, the table consists of only one populated bin.

where $partition_j^i$ is the ith partition along the jth attribute axis and $span\_attribute_j$ is the entire span of values that the attribute a_j can take. The extreme case of $k = 1$, is the case where the table consists of one populated bin.

5. EXPERIMENTAL RESULTS

In Section 5.1, we will first present some of the more relevant details about the special structured-light scanner that was built for MULTI-HASH; this will be followed by a brief discussion of the low-level processing steps ending in a discussion of feature extraction. Subsequently, in Section 5.2, we will discuss the performance of MULTI-HASH on real scenes containing objects from the database of Fig. 1.

5.1. *Data Collection and Low Level Processing*

Because color must be extracted from a scene in addition to depth information, a special structured-light scanner was constructed that allows measurement of color at each point where the depth information is calculated. As is common in structured-light range imaging, the scene is scanned with a thin laser stripe for the calculation of range information. All the scene points illuminated with the laser stripe are then illuminated with a white light stripe. Since white light cannot be focused as sharply as laser light, the white light stripe is wider than the laser stripe.[3] A color camera then samples the white-light-illuminated part of the scene at exactly the points that were previously illuminated with the laser source. In this manner, registered range and color values are obtained. For the scene in Fig. 26a, the color-composite light-stripe image produced by our scanner is shown in Fig. 26b. Because the scanner is mounted high above the work space, we are dealing with relatively noisy data as illustrated by the 3D plot of points in Fig. 26c for the scene in Fig. 26a.

After computation of 3D depth values using triangulation formulas shown in [77], surface normals at each point are calculated using an adaptive window technique described in [8]. In this technique, the normal at a point is calculated by fitting a plane in a window surrounding the point in question. When the window, centered at the point in question, straddles a boundary between two surfaces, the resulting surface normal will be distorted which will cause a smooth rather than sharp transition in the normals when traveling across a surface edge. In [8], this distortion is significantly reduced by adaptive placement of the window when in the vicinity of an edge. The position of the window is moved so that it still contains the point in question but as little of the edge(s) as possible. The criterion minimized in the selection of the window's position is the product of the error of the resulting planar fit and a distance metric between the window center and the point in question.

The next processing step is that of segmentation. The goal of this step is to segment the range map into smooth surface patches separated by range and surface normal discontinuities. There exist many approaches to the general problem of 3D segmentation [15, 34, 36, 43, 64]. In MULTI-HASH, we use the same region-based approach as reported in [8]. Basically, the segmentation process recursively grows out a surface region starting from some pixel in the range map and stopping in a particular direction when a jump or crease edge is encountered. Detection of a jump edge in a range map is a fairly reliable process and, for two adjacent points, $p(i, j)$ and $p(k, l)$, is accomplished with the following test:

$$|p(i, u) - p(k, l)| > jump\_threshold. \qquad (31)$$

Here $p(i, j)$ is the vector $(x(i, j), y(i, j), z(i, j))$ of the three-dimensional coordinates of the (i, j)th point in the range map. (For those unfamiliar with structured-light imaging, the index i refers to the ith stripe in the scan and the index j refers to the jth camera scan line.) Detection of a crease edge is more difficult, especially when smoothly curving surfaces exist in a scene. A crease edge exists between two adjacent points of a range map when a relatively sharp change in the direction of the normal direction occurs. This is detected with the following test:

[3] If so desired, the relatively wide white-light stripe can be used to sample other photometric information such as texture surrounding each range point.

$$\frac{\cos^{-1}(n_{i,j} \cdot n_{k,l})}{|p(i,j) - p(k,l)|} > crease\_threshold. \quad (32)$$

Here $n_{i,j}$ is the normal vector at $p(i, j)$. The thresholds used in Eqs. (31) and (32) are functions of the surfaces in the model-base and the scanner resolution; these thresholds are chosen empirically. The value of the *crease_threshold* should be small enough so that a crease edge between two surfaces can be detected without a sharply rounded surface being erroneously declared a crease edge. Once a region can no longer be grown, a new region is started with the first point in the raster scan of the range map that does not already belong to a previously grown region, meaning that it is currently unlabeled in the segmentation map. For the scene in Fig. 26a, the resulting segmentation map is shown in Fig. 26d.

Vertices, edges and surfaces are the features extracted from the segmentation map. Each segmented region describes a surface whose vertices and edges are detected during a traversal around the surface boundaries. The features in a segmentation map—surfaces, edges, vertices—are represented in exactly the same manner as the features on model objects, that is, by appropriate attribute-value frames (See Section 2 for representation of objects.) Again, as was the case for objects, whereas scene surfaces and vertices are represented explicitly by attribute-value frames, the scene edges have an implicit representation via the adjacency pointers in the attribute-value frames for vertices. Shown below is the attribute-value frame for surface 8 in the segmentation map of Fig. 26d:

{
⟨Shape: *Planar*⟩
⟨Area: *346 pixels*⟩
⟨Color: *198.30148, 154.60211, 41.20289*⟩
⟨Principal Direction: *−0.09823, −0.75356, 0.65000*⟩
⟨Planar Parameter: *0.410341*⟩
⟨Adjacent Surfaces: *{14, 18}*⟩
⟨Angles: *between surface 8 and 14: 1.560425 radians, between surface 8 and 14: 1.560425 radians*⟩
}

In the frame shown above, the principal direction for a planar surface is defined as its normal vector, for a cylindrical surface the direction of its axis, and for a conical surface as its axis of revolution [7].

There are many approaches to classifying the shape of a surface [1, 36, 42, 66]. Currently, the system classifies the shape of a surface as either planar, cylindrical, conical, or unknown, and this classification has been successfully used by other 3D object recognition systems [7, 19]. Surface shape classification is performed as in [8] by first attempting to fit a plane to the data points of the surface. If the fitting error is less than an empirically chosen threshold, the surface shape is classified as planar. Otherwise, the surface is tested to see if it is cylindrical, conical or other through the use of an extended Gaussian image of the surface [37, 41] which is created by mapping the surface normals belonging to the surface in question onto a Gaussian sphere. Classification of surface shape into the categories of cylindrical, conical or other is made by examination of the distributions of the normals mapped onto the Gaussian sphere.

As discussed in [8], the surface normals of a planar region should ideally map to a point on a unit sphere, the radial vector to the point being in the same direction as the normal to the plane, as shown in Fig. 27a. And, the conical and cylindrical surface normals should ideally map to curves on a unit sphere. The curve created for a cylindrical surface should fall along a great circle of the unit sphere and the orientation of its axis should be parallel to the orientation of the cylinder's axis (see Fig 27b). Similarly, the curve created for a conical surface will fall along a minor circle (radius < 1) of the unit sphere such that its axis is parallel to the axis of revolution of the cone (Fig. 27c). Note that only part of a cylinder or cone will be visible in the scene and thus only an arc of the corresponding circular curve on the unit sphere will appear.

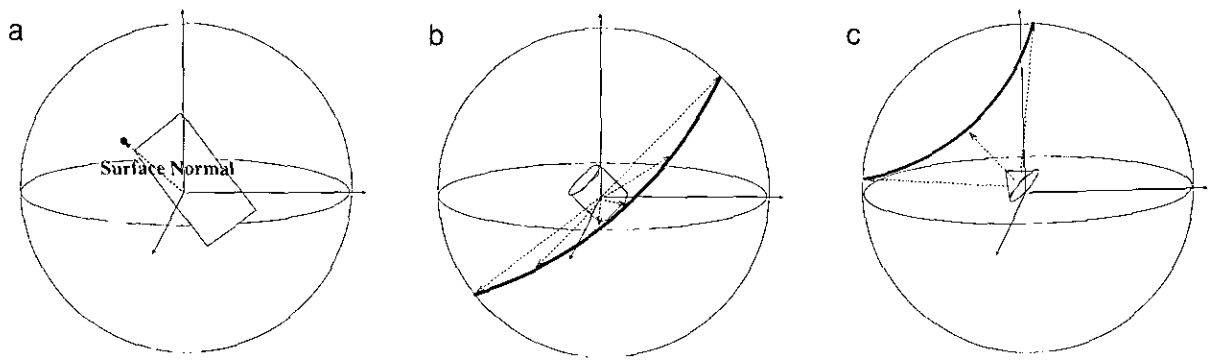


FIG. 27. Mappings of surface normals to an EGI for (a) a plane, (b) a cylinder, and (c) a cone.

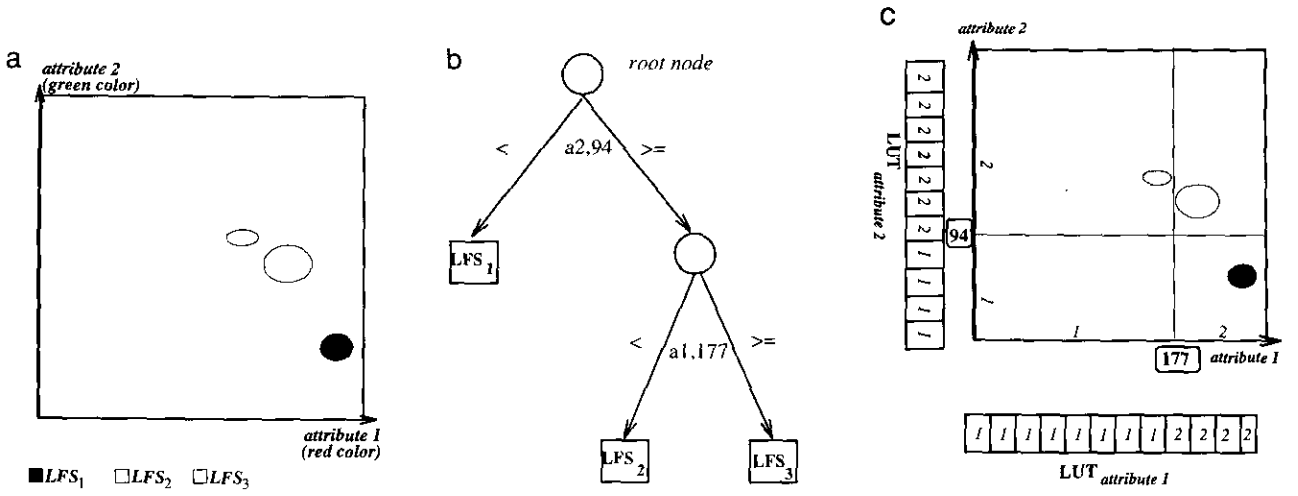


FIG. 28. (a) Gaussian distributions truncated at three standard deviations from the mean for three model LFSs. (b) Decision tree grown. (c) Corresponding table. LFS_1 is from a red object, LFS_2 from a white object, and LFS_3 from a wood-colored object.

5.2. *Performance of MULTI-HASH*

We will now present the results of modeling the uncertainties in the training data, the resulting tree grown, and the corresponding table. Because of space limitations, we will first go over a smaller example and then present the results on the model-base of Fig. 1.

For our smaller example, consider Fig. 28a, which depicts the truncated Gaussian distributions of three model LFSs over a space defined by two attributes, the red and green components of color. LFS_1 is an LFS from a real red-colored 3D object, LFS_2 an LFS from a white-colored object of the same shape, and LFS_3 an LFS from a wood-colored object, again of the same shape. Each attribute can take on values from 0 to 255, and the smallest quantization interval for each attribute axis is set to 1. The uncertainties are modeled by Gaussian distributions truncated at three standard deviations from the means. Figure 28b illustrates the tree grown and Fig. 28c shows the corresponding hash table. Note that no further partitions of the table would lead to "purer" bins and, for this example, the table produced is optimal, in the sense of minimizing average entropy, and, under the constraint of optimality, contains the smallest number of bins. Note that the look-up tables shown in Fig. 28c for each attribute are actually more finely partitioned but, are displayed here with a coarser partitioning simply for ease of illustration. Figure 29 illustrates similar results for the same attribute space, but this time LFS_1 is an LFS from a real red-colored 3-D object, LFS_2 an LFS from a yellow-colored object, and LFS_3 an LFS from a pink-colored object.

We will now present experimental results for the case of the model-base shown in Fig. 1. Training samples were

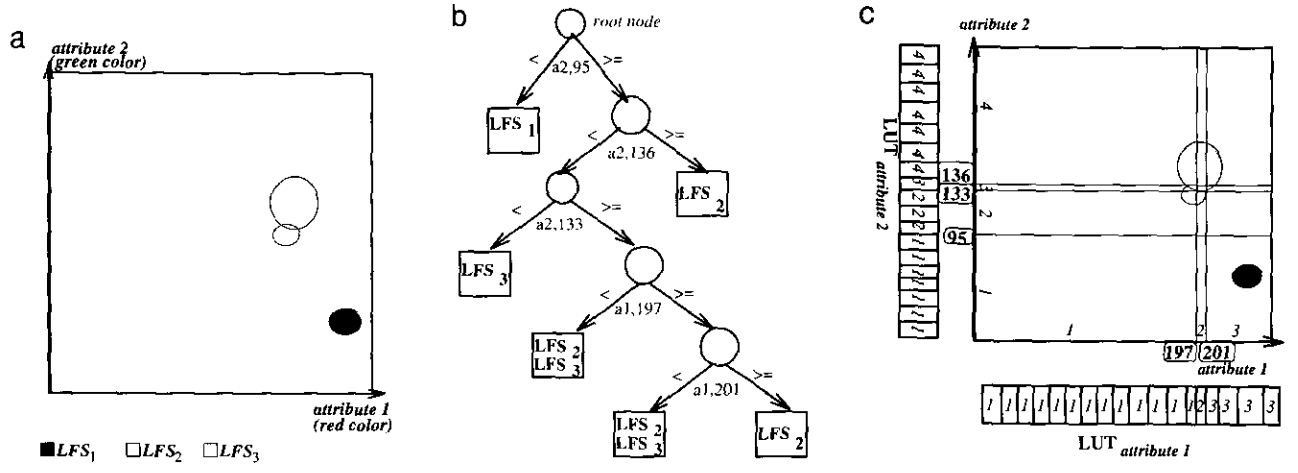


FIG. 29. Same as in Fig. 28, except that LFS_2 is now from a yellow object and LFS_3 from a pink object.

Attributes	red_round {7,10,9}				square_white {2,1,10}				square_pink {4,3,10}			
	mean	std	min	max	mean	std	min	max	mean	std	min	max
shape1	0	0	0	0	0	0	0	0	0	0	0	
r1	235.80	4.11	227.0	242.0	163.84	4.31	153.0	172.0	190.30	4.77	179	199
g1	61.54	4.01	55.0	71.0	145.98	1.94	140.0	151.0	124.85	3.14	117.0	131.0
b1	65.80	4.41	60.0	77.0	128.28	3.13	121.0	136.0	111.95	4.10	104.0	121.0
shape2	0	0	0	0	0	0	0	0	0	0	0	0
r2	235.02	4.00	227.0	242.0	158.50	7.03	148.0	171.0	185.56	3.53	178.0	195.0
g2	61.75	3.65	53.0	68.0	148.17	2.31	143.0	153.0	128.96	2.66	121.0	135.0
b2	66.57	3.27	60.0	73.0	132.21	6.21	121.0	144.0	115.84	4.25	104.0	123.0
shape3	0	0	0	0	0	0	0	0	0	0	0	0
r3	231.99	8.29	208.0	242.0	162.31	5.53	148.0	168.0	189.64	3.66	181.0	198.0
g3	65.94	8.16	54.0	87.0	147.26	1.91	143.0	153.0	127.05	3.09	120.0	132.0
b3	72.84	11.07	60.0	100.0	128.69	5.40	121.0	144.0	111.82	3.68	101.0	121.0
angle1	1.58	0.03	1.3	1.8	1.57	0.02	1.4	1.8	0.76	0.02	0.6	1.0
angle2	1.57	0.04	1.3	1.8	1.57	0.01	1.4	1.8	1.57	0.01	1.4	1.8

FIG. 30. Gaussian parameters (mean, std) and uniform distribution parameters (min, max) fix to the training data collected for a few of the LFS in the model-base of Fig. 1. The left column is the LFS involving surfaces 7, 10, and 9 for the red object of Fig. 1b. The middle column is the LFS involving surfaces 2, 1, and 10 for the white object of Fig. 1a. The right column is the LFS involving surfaces 4, 3, and 10 for the pink object of Fig. 1a. Note that shape is a qualitative attribute where 0 indicates a planar surface, 1 indicates a cylindrical surface, and 2 indicates other.

collected for the LFSs of the objects in the model-base. The number of samples collected for each LFS ranged from 14 to 44. The different samples corresponded to different poses. Figure 30 shows the parameters for the two cases of using Gaussian and uniform distributions for the uncertainties associated with a few of the model LFSs. In the table, the attributes *shapei*, *ri*, *gi*, *bi* refers to the *i*th surface in the LFS. For example, *shape*1, *r*1, *g*1, *b*1 are for the surface marked 7 in the LFS characterized by the surfaces (7, 10, 9) in Fig. 30; *shape*2, *r*2, *g*2, *b*2 are for the surface marked 10, and so on. Figure 31 lists the unique tests along each attribute axis used in the decision tree grown from the training data modeled with Gaussian distributions. Note that the entry *none* means that no test was specified for that attribute; in other words, that attribute was found to be irrelevant for the desired discriminations. The actual tree is not shown due to space limitations.

Attribute	Range	Tests
shape1	0,1,2	none
r1	0.0-255.0	173.0
g1	0.0-255.0	102.0
b1	0.0-255.0	73.0, 104.0
shape2	0,1,2	none
r2	0.0-255.0	193.0
g2	0.0-255.0	68.0, 136.0, 140.0
b2	0.0-255.0	40.0, 73.0, 134.0
shape2	0,1,2	none
r3	0.0-255.0	none
g3	0.0-255.0	135.0
b3	0.0-255.0	73.0, 104.0
angle1	0.0-3.0	1.1
angle2	0.0-3.0	1.1

FIG. 31. Unique tests used in decision tree grown using the Gaussian distributions derived from the training data collected for the model-base in Fig. 1. A memory constraint requiring the resultant table to possess less than 10,000 bins was used. The entry "none" indicates no test was specified along this attribute axis.

The hash table constructed from the tests in Fig. 31 was used for object recognition on a set of 10 test scenes composed of objects in random placements in a bin. Randomization was achieved by tossing the objects into the bin or shaking the bin vigorously. Figure 32 shows the composition of a typical test scene by displaying its composite colored structured-light image and the resulting segmentation map.

From the segmentation map, the flow of control for object recognition is as shown in Fig. 2. This flow of control was reviewed in Section 2. The recognition scheme is set up so that the first object recognized in the bin is picked up by a Puma robot. Figure 33 shows the robot picking up the first object recognized in a scene. Out of the 10 test scenes, an object was recognized and picked up in 7. In the three scenes where no objects were recognized, failure was due to either improper segmentation or due to the fact no objects presented a vertex to the sensor. The reader will recall that we use a vertex-centered definition of LFS. So if all the visible objects in a pile are such that none presents a vertex to the sensor, an LFS will not be constructed. Figure 34 shows for each scene the recognition result and a comparison of the number of hypotheses processed by the current system in comparison with the earlier, less sophisticated system of [26]. For these test scenes, every scene LFS processed retrieved from the table its corresponding model LFS. It is important to note that in and of itself MULTI-HASH does not depend on vertices for the formation of LFS; the

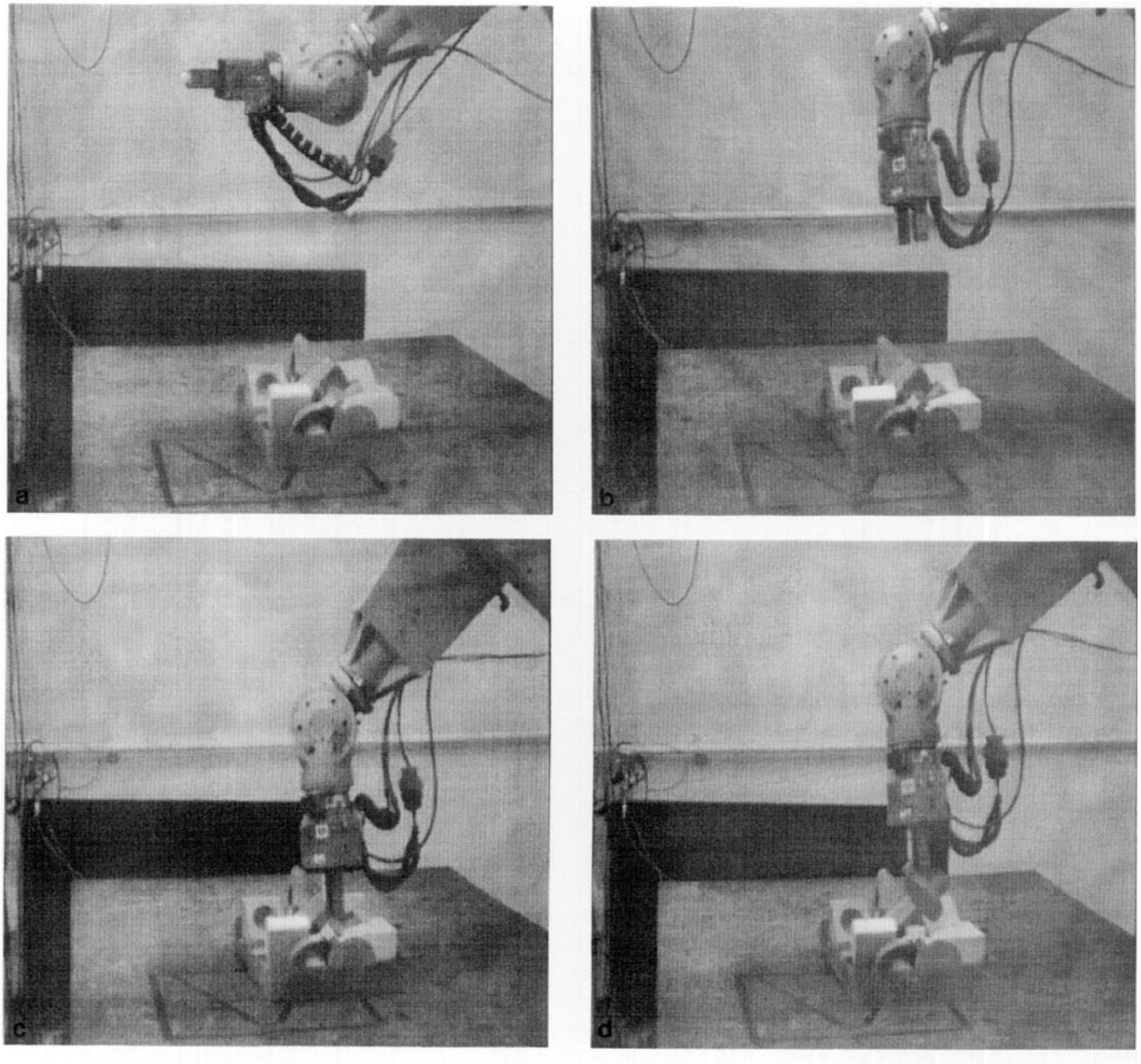


FIG. 33. Robot picking up the first recognized object, a red, round object. The sequence is (a), (b), (c), (d).

use of vertices for this purpose is merely an implementation choice at this time.

6. CONCLUSION

Model-based vision is made complicated by the fact that it can be difficult for a human to specify all the strategies needed for discriminating between the objects in a model-base. These strategies, even when they can all be laid down for a given model-base, may require extensive and onerous revisions if further objects are added to the model-base. Therefore, it is not surprising that there is great interest at this time in systems that are capable of synthesizing, under supervised training of course, all the recognition strategies. The role of a human in such systems would be to merely show different object instances to the vision system and to delineate the correspondences between, say, the different features on the shown objects and those on the models.

An example of such a system was presented in this paper. The human operator shows different objects in different poses to the sensory system attached to MULTI-HASH. Before this supervised training takes place, MULTI-HASH is provided with a list of objects in the model-base, together with the different local feature sets for each object. During training, the human operator

Scene	>= 1 object recognized and picked up by robot	#LFSs processed in Hypothesis Generation current system	#LFSs processed in Hypothesis Generation system [26]
scene 1	yes	8	8
scene 2	yes	4	15
scene 3	yes	367	537
scene 4	yes	197	407
scene 5	yes	857	2352
scene 6	yes	83	168
scene 7	yes	20	47
scene 8	no	628	1380
scene 9	no	775	2460
scene 10	no	545	1425

FIG. 34. Experimental results presented on 10 random scenes of objects from the model-base (see Fig. 1) and others thrown into a pile.

also declares correspondences between the local feature sets extracted from the scene and the local feature sets on the models. MULTI-HASH uses all this information to construct models of uncertainty for the values of the attributes of object surfaces. Using these uncertainty models, MULTI-HASH constructs a decision tree that eventually gets transformed into a hash table for fast object recognition.

MULTI-HASH is written is C and consists of over 11,000 lines of code, not including the code for low-level processing and robotic manipulation. MULTI-HASH is used routinely in the Robot Vision Lab for bin picking of colored objects of the kind discussed in the paper using a PUMA 762 robot.

REFERENCES

1. P. Besl and R. Jain, Invariant surface characteristics for 3D object recognition in range images, *Comput. Vision Graphics Image Process.* **33,** 1986, 33–80.
2. R. Bolles and R. Cain, Recognizing and locating partially visible objects: The local-feature-focus method, *Int. J. Robotics Res.* **1**(3), 1982, 57–82.
3. R. Bolles and P. Horaud, 3DPO: A three-dimensional part orientation system, *Int. J. Robotics Res.* **5**(3), 1986, 3–26.
4. L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees,* Wadsworth, Belmont, CA, 1984.
5. A. Califano and R. Mohan, Multidimensional indexing and recognizing visual shapes, *IEEE Conference on Computer Vision and Pattern Recognition, 1991,* pp. 28–34.
6. B. Cestnik, I. Kononenko, and I. Bratko, ASSISTANT 86: A knowledge elicitation tool for sophisticated users, *in Progress in Machine Learning* (I. Bratko and N. Lavarc, Eds.), Sigma, Wilmslow, 1986.
7. C. Chen, and A. Kak, A robot vision system for recognizing 3-D objects in low-order polynomial time, *IEEE Trans. Systems Man Cybernet.* **19**(6), 1989, 1535–1563.
8. C. Chen and A. Kak, *A Robot Vision System for Recognizing 3-D Objects in Low-Order Polynomial Time,* Technical Report, TR-EE 88-48, Purdue University, December 1988.
9. D. Clemens and D. Jacobs, Space and time bounds on indexing 3-D models from 2-D images, *IEEE Trans. Pattern Anal. Mach. Intell.*
10. W. Conover, *Practical Nonparametric Statistics,* **13,** 1991, 1007–1017. Wiley, New York, 1971.
11. M. Costa, R. Haralick, and L. Shapiro, Optimal affine-invariant point matching, in *Proceedings of the 10th Intl. Conf. on Pattern Recognition, 1990,* pp. 233–236.
12. R. Cromwell, and A. Kak, Automatic generation of object class descriptions using symbolic learning techniques, in *Proceedings of AAAI-91, July 1991,* pp. 710–717.
13. A. Dempster, N. Laird, and D. Rubin, Maximum likelihood estimation from incomplete data via EM algorithm, *J. R. Statist. Soc. B, 1977,* 1–38.
14. T. Elomaa, Extending the learnability of decision trees, in *3rd International Conference on Tools for Artificial Intelligence, 1991,* pp. 504–505.
15. T. Fan, G. Medioni, and R. Nevatia, Segmented descriptions of 3-D surfaces, *IEEE J. Robotics Automat.* **3**(6), 1987, 527–538.
16. W. Feller, *An Introduction to Probability Theory and Its Applications,* Vol. 2, Wiley, New York, 1966.
17. T. Ferguson, Bayesian density estimation by mixtures of normal distributions, in *Recent Advances in Statistics* (M. Rizvi, J. Rustagi and D. Siegmund (Eds.), Academic Press, New York, 1983.
18. N. Findler, T. Bickmore, L. Ihrig, and W. Tsang, A note on the comparison of five heuristic optimization techniques of a certain class of decision-trees, *Inform. Sci.* **53**(1–2), 1991, 89–100.
19. P. Flynn and A. Jain, 3D object recognition using invariant feature indexing of interpretation tables, *CVGIP: Image Understand.* **55**(2), 1992, 119–129.
20. P. Flynn, Saliencies and symmetries: Toward 3D object recognition from large model databases, in *IEEE Conference on Computer Vision and Pattern Recognition, 1992,* pp. 322–327.
21. K. Fukunaga, *Introduction to Statistical Pattern Recognition,* Academic Press, Boston, 1990.
22. K. Fukunaga, Estimation of the parameters of a Gaussian mixture using the method of moments, *IEEE Trans. Pattern Anal. Mach. Intell.* **5**(4), 1983, 410–416.
23. S. Gelfand, C. Ravishankar, and E. Delp, An iterative growing and pruning algorithm for classification tree design, *IEEE Trans. Pattern Anal. Mach. Intell.* **13**(2), 1991, 163–174.
24. R. Gershon, A. Jepson and J. Tsotsos, Highlight identification using chromatic information, in *1st International Conference on Computer Vision, 1987,* pp. 161–170.
25. R. Gershon, A. Jepson, and J. Tsotsos, Ambient illumination and the determination of material changes, *J. Opt. Soc. Am.* **3**(10), 1986, 1700–1707.
26. L. Grewe, and A. Kak, Integration of geometric and non-geometric attributes for fat object recognition, *Appl. Artif. Intell. SPIE* **1964,** 1993, 13–28.
27. L. Grewe and A. Kak, Integration of geometric and non-geometric attributes in a fast object recognition scheme, *Int. J. Pattern Recognition Artif. Intell.,* to appear.
28. L. Grewe, and A. Kak, Interactive learning of multiple attribute hash table for fast 3D object recognition, in *IEEE 2nd CAD-Based Vision Workshop, Feb. 1994,* pp. 17–27.
29. W. E. Grimson, On the sensitivity of geometric hashing, in *Proceeding of the 3rd Intl. Conference on Computer Vision (ICCV '90), pp. 334–338.*
30. G. Healey, Using color for geometric-insensitive segmentation, *J. Opt. Soc. Am. A* Vol. **6**(6), 1989, 920–937.

31. G. Healey, Segmenting images using normalized color, *IEEE Trans. Systems Man Cybernet* **22**(1), 64–73, 1992.
32. G. Healey and T. Binford, A color metric for computer vision, in *IEEE Conference on Computer Vision and Pattern Recognition, 1988*, pp. 10–17.
33. G. Healey and T. Binford, The role and use of color in a general vision system, in *Proceedings of the DARPA Image Understanding Workshop, USC, 1987*, pp. 599–613.
34. M. Hebert and J. Ponce, A new method for segmenting 3-D scenes into primitives, in *IEEE Workshop on Computer Vision, 1982*, pp. 836–838.
35. J. Henna, On estimating the number of constituents of a finite mixture of continuous distributions, *Ann. Inst. Statist. Math.* **37**(A), 1985, 235–240.
36. R. Hoffman and A. Jain, Segmentation and classification of range images, *IEEE Trans. Pattern Anal. Mach. Intell.* **9** 5, 1987, 608–620.
37. B. Horn, Extended Gaussian images, *Proc. IEEE* **72**(12), 1984, 1671–1686.
38. B. Horn, *Robot Vision*, MIT Press, Cambridge, MA, 1986.
39. P. Huber, *Robust Statistics*, Wiley, New York, 1981.
40. L. Hyafil, and R. Rivest, Constructing optimal binary decision trees is NP-complete, *Inform. Process. Lett.* **5**(1), 1976, 15–17.
41. K. Ikeuchi, Recognition of 3-D objects using the extended Gaussian image, *IJCAI*, 1981, 595–600.
42. D. Ittner and A. Jain, 3-D surface discrimination for local curvature measures, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 1985*, pp. 119–123.
43. A. Jain and S. Nadabar, MRF model-based segmentation of range images, in *Proceedings of the 3rd Intl. Conference on Computer Vision (ICCV '90)*, pp. 667–671.
44. W. Kim and A. Kak, 3-D object recognition using bipartite matching embedded in discrete relaxation, *IEEE Trans. Pattern Anal. Mach. Intell.* **13**(3), 1991, 224–251.
45. G. Klinker, S. Shafer, and T. Kanade, Using a color reflection model to separate highlights from object color, in *1st Intl. Conf. on Computer Vision, 1987*, pp. 145–150.
46. B. Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Prentice–Hall, Englewood Cliffs, NJ, 1992.
47. D. Landgrebe, The development of a spectral-spatial classifier for earth observational data," *Pattern Recognition* **12**, 1980, 165–175.
48. Y. Lamdan, J. Schwartz, and H. Wolfson, Affine invariant model-based object recognition, *IEEE Trans. Robotics Automat.* **6**(5), 1990, 579–589.
49. Y. Lamdan and H. Wolfson, Geometric hashing: A general and efficient model-based recognition scheme, in *Proc. IEEE Int. Conf. Robotics and Automation, Philadelphia, PA, Apr. 1988*, pp. 1407–1413.
50. Y. Lamdan and H. Wolfson, On the error analysis of geometric hashing, in *IEEE Conference on Computer Vision and Pattern Recognition, 1991*, pp. 22–27.
51. L. Maloney and B. Wandell, Color constancy: a method for recovering surface spectral reflectance, *J. Opt. Soc. Am.* **3**(1), 1986, 29–33.
52. R. Michalski, Pattern recognition as rule-guided inductive inference, *IEEE Trans. Pattern Anal. Mach. Intell.* **2**(4), 1980, 349–361.
53. C. Novak and S. Shafer, "Supervised color constancy for machine vision, *SPIE Human Vision Visual Process. Digital Display II* **1453**, 1991, 353–368.
54. M. Nunez, The use of background knowledge in decision tree induction, *Machine Learning* **6**(3), 1991, 231–250.
55. Y. Ohta, T. Kanade, T. Sakai, Color information for region segmentation, *Comput. Graphics Image Process.* **13**, 1980, 222–241.
56. D. Phillips, *Applied Goodness of Fit Testing*, American Institute of Industrial Engineers, 1972.
57. S. Qing-Yun and K. Fu, A method for the design of binary tree classifiers, *Pattern Recognition* **16**(6), 1983, 593–603.
58. J. Quinlan, Induction of decision trees, *Machine Learning* **1**(1), 1986, 81–106.
59. J. Quinlan, Simplifying decision trees, *Int. J. Man-Mach. Stud.* **27**(3), 1987, 221–234.
60. J. Quinlan and R. Rivest, Inferring decision trees using the minimum description length principle, *Inform. and Comput.* **80** 3, 1989, 227–248.
61. J. Reichman, Determination of absorption and scattering coefficients for nonhomogeneous media 1 : Theory, *Appl. Opt.* **12**, 1973, *1811–1815*.
62. I. Rigoutsos and R. Hummel, Robust similarity invariant matching in the presence of noise, in *Proceedings of the 8th Israeli Conference on Artificial Intelligence and Computer Vision, Dec. 1991*.
63. I. Rigoutsos and R. Hummel, Implementation of geometric hashing on the Connection Machine, *Directions in Automated CAD-Based Vision*, Dec. 1991.
64. R. Rimey and F. Cohen, A maximum-likelihood approach to segmenting range data, *IEEE J. Robotics Automat.* **4**(3), 1988, 277–286.
65. J. Rissanen, Modeling by shortest data description, *Automatica* **14**, 1978, 465–471.
66. B. Sabata, F. Arman, and J. Aggarwal, Segmentation of 3-D range images using pyramidal data structures, in *Proceedings of the 3rd Intl. Conference on Computer Vision (ICCV '90)*, pp. 662–666.
67. S. R. Safavian and D. Landgrebe, A survey of decision tree classifier methodology, *IEEE Trans. Systems Man Cybernet.* **21**(3), 1991, 660–674.
68. H. Schneider, *Truncated and Censored Samples from Normal Populations*, Dekker, New York, 1986.
69. M. Schwarz, L. Grewe, and A. Kak, *Representation of Color and Segmentation of Color Images*, Technical Report, TR-EE 94-2, Purdue University, Jan. 1994.
70. G. Schwarz, Estimating the dimension of a model, *Ann. Statist.* **6**(2), 1978, 461–464.
71. B. Shahshahani, and D. Landgrebe, An algorithm for classification of multi-spectral data and its implementation on a massively parallel computer, *Neural Stochastic Methods in Image Signal Process. II, SPIE* **2032**, July 1993.
72. B. Shepherd, An appraisal of a decision tree approach to image classification, in *Proceedings of the 8th International Joint Conference on Artificial Intelligence, Aug. 1983*, pp. 473–475.
73. F. Stein and G. Medioni, Structural indexing: Efficient 3-D object recognition, *IEEE Trans. Pattern Anal. Mach. Intell.* **14**(2), 1992, 125–145.
74. Y. Suganuma, Learning structures of visual patterns from single instances, *Artif. Intell.* **50**, 1991, 1–36.
75. M. Swain and D. Ballard, Indexing via color histograms, in *IEEE Proc. 3rd Intl. Conf. Computer Vision, Osaka, Japan, 1992*, pp. 390–393.
76. K. Torrance and E. Sparrow, Theory of off-specular reflection from roughened surfaces, *J. Opt. Soc. Am.* **57**, 1967, 1105–114.
77. H. Yang and A. Kak, Edge Extraction and labeling from structured-light 3-D vision data, in *Selected Topics in Signal Processing*, S. Haykin (Ed.), pp. 148–193. Prentice-Hall, New York, 1989.