COMPUTER VISION AND IMAGE UNDERSTANDING
Vol. 72, No. 3, December, pp. 360–378, 1998
ARTICLE NO. IV980679

Indexing Based on Algebraic Functions of Views

George Bebis

Department of Computer Science, University of Nevada, Reno, Nevada 89557

Michael Georgiopoulos

Department of Electrical & Computer Engineering, University of Central Florida, Orlando, Florida 32816

and

Mubarak Shah and Niels da Vitoria Lobo

Department of Computer Science, University of Central Florida, Orlando, Florida 32816

Received January 10, 1997; accepted January 16, 1998

In this paper, we propose the use of algebraic functions of views for indexing-based object recognition. During indexing, we consider groups of model points and we represent all the views (i.e., images) that they can produce in a hash table. The images that a group of model points can produce are computed by combining a small number of reference views which contain the group using algebraic functions of views. Fundamental to this procedure is a methodology, based on Singular Value Decomposition and Interval Arithmetic, for estimating the allowable ranges of values that the parameters of algebraic functions can assume. During recognition, scene groups are used to retrieve from the hash table the most feasible model groups that might have produced the scene groups. The use of algebraic functions of views for indexing-based recognition offers a number of advantages. First of all, the hash table can be built using a small number of reference views per object. This is in contrast to current approaches which build the hash table using either a large number of reference views or 3D models. Most importantly, recognition does not rely on the similarity between reference views and novel views; all that is required for the novel views is to contain common groups of points with a small number of reference views. Second, verification becomes simpler. This is because candidate models can now be back-projected onto the scene by applying a linear transformation on a small number of reference views of the candidate model. Finally, the proposed approach is more general and extendible. This is because algebraic functions of views have been shown to exist over a wide range of transformations and projections. The recognition performance of the proposed approach is demonstrated using both artificial and real data. © 1998 Academic Press

1. INTRODUCTION

Recognizing objects from images has been a challenging task in computer vision. This is because objects may look very different from different viewing positions. The most successful approach is in the context of *model-based* object recognition [1], where the environment is rather constrained and recognition relies upon the existence of a set of predefined model objects. Given an unknown scene, recognition implies: (i) the identification of a set of features from the unknown scene which approximately match a set of features from a known view of a model object, (ii) the recovery of the geometric transformation that the model object has undergone (pose recovering), and (iii) verification that other features coincide with predictions. Since usually there is no *a priori* knowledge of which model points correspond to which scene points, recognition can be computationally too expensive, even for a moderate number of models. Various approaches have been proposed in the literature for dealing with this issue.

One approach to limit the possible number of matches is by using geometric constraints [2]. Another approach is to establish hypothetical matches using the minimum possible number of model-scene feature correspondences [3]. Indexing is an alternative approach which has been given considerable attention lately [4–14]. It is based on the idea of using *a priori* stored information about the models in order to quickly eliminate noncompatible model-scene feature matches during recognition. Hence, only the most feasible matches are considered, that is, the matches where the model features could have projected to the scene features. Indexing-based methods usually employ a hash scheme to efficiently store and retrieve information about the models into a hash table. There are two different phases of operation: *preprocessing* and *recognition*. During preprocessing, groups of model features are considered and a description for each one of them is computed. These descriptions are then used to access the hash table. Appropriate information about the group of model features is stored in the indexed location. During recognition, groups of scene points are considered and their descriptions are used to access the hash table.

In the noiseless case, each indexed location will contain exactly the set of model groups compatible with the group of scene features used to access the table. Ideally, one would like the index computed from a group of model features to remain the same, regardless of changes in the appearance of the model when it is observed from different viewpoints. Such an index is said to be *invariant*. The main advantage of invariant indices is that a single entry for each group of model features needs to be stored, regardless of changes in the viewpoint. Geometric hashing [4] is an example of a method which uses affine invariants for the recognition of planar objects. Projective invariants of some special case, two-dimensional, algebraic curves have been also utilized in another study [8].

Building indexing schemes to recognize general 3D objects using invariants is not possible in general, since it has been shown that no general-case invariants exist for single views of general three-dimensional point sets [12]. As a result, model-based invariants have been proposed for indexing [10]. These invariants can be learned from several images of the object. The basic idea is that a function can be constructed for each group of model features that, given a group of image features, evaluates to zero if and only if the model group could project to the image group. Another approach is to take advantage of the fact that the angles and distances of image features change little (i.e., remain invariant) over a substantial range of viewing directions (probabilistic peaking effect [15]). Probabilistic indexing [11] is based on this idea. Quite common are also approaches which consider a separate model for each view of a 3D object [5], obtained by taking pictures of the object from different viewing directions. Then, an indexing scheme based on invariants is employed for each model view. Alternatively, other methods assume that the 3D structure of the model objects is available (i.e., CAD models). The viewing sphere is then sampled and a description about the images that groups of model features produce, from each point on the viewing sphere, is stored in a hash table. During recognition, groups of features are chosen from the scene and the hash table is accessed to find the most feasible three-dimensional model groups that might have produced them. A system based on this idea has been implemented in [12], assuming orthographic projection. This system has been improved in the case of 3D linear transformations so that the hash table is built using analytical formulas, without having to sample the viewing sphere [13,14]. In particular, it was shown in [14] that the images of groups of 3D points can be represented as a pair of 1D lines in two high-dimensional spaces. During preprocessing, each group of model points is represented by a line in each of the two spaces. During recognition, groups of scene points are used to retrieve sets of model features indexed in both spaces. The intersection of the two sets corresponds to the possibly matching groups of model points.

In this paper, a new indexing-based object recognition approach is proposed based on algebraic functions of views [16–22]. Algebraic functions of views are functions which express a relationship among a number of views of the same object in terms of their image coordinates alone. For example, in the case of orthographic projection, the image coordinates of any three views of an object satisfy a linear function [16]. The key idea in using algebraic functions of views for indexing is that they allow us to compute all possible views (i.e., images) that a group of model points can produce using a small number of views which contain the group. Thus, 3D models are not required. We will be referring to the space of views that a group of points can produce as the *space of transformed views* of the group. During indexing, the space of transformed views is sampled and the sampled views are represented in a hash table. During recognition, image groups are used to retrieve from the hash table the model groups that might have produced them. To construct the space of transformed views of a group, we apply the algebraic functions of views on the reference views of the group. To estimate the allowable ranges of values that the parameters of algebraic functions can assume, we use a methodology based on Singular Value Decomposition (SVD) [24] and Interval Arithmetic (IA) [25].

Our approach is different from [5] which requires a large number of reference views to ensure that new views are similar to at least one of the reference views. In our case, new views can be constructed by combining a small number of reference views. Furthermore, our approach for generating the images that a model group can produce during preprocessing is more practical since it does not require 3D models. In [13,14] for example, the lines which represent the images of a model group can be found easily only if the 3D structure of the object is known. Since this information is not always available, a set of different 2D images, containing the group, is used instead [13,14]. Each image defines a point in each of the two representational spaces and a line must be fitted to these points, in each space, to approximate the actual lines. This procedure requires more effort and time since edges must be extracted, interest points must be detected, and point correspondences across the images must be established. On the other hand, our approach is based on a small number of images per model and makes on approximations in computing the images that a group of model points can produce. Another advantage of using algebraic functions of views is that verification becomes simpler. This is because candidate models can be back-projected onto the scene by combining a small number of their reference views only. Finally, the availability of algebraic functions of views over a wide range of transformations and projections [16–22] makes the proposed approach more general and extendible.

The paper is organized as follows: In Section 2 we present an overview of the algebraic functions of views. A general framework for employing algebraic functions of views for indexing-based object recognition is presented in Section 3. In Section 4 we present a method for estimating the allowable ranges of values that the parameters of algebraic functions can assume, and in Section 5 we introduce a procedure, called “preconditioning,” for obtaining tighter ranges of values. Sections 6 through 9 deal with a number of practical issues and in Section 10 we consider various issues related to the performance of the method. Section 11 presents recognition results using both artificial and real 3D objects, assuming orthographic projection and 3D linear transformations. Finally, Section 12 includes our conclusions.

2. BACKGROUND ON ALGEBRAIC FUNCTIONS OF VIEWS

In this section, we summarize a number of theoretical results regarding algebraic functions of views. First, we introduce some terminology that will be useful throughout this paper. We assume that the database contains M models and that each model is represented by a number of aspects $A_m, m = 1, 2, \ldots, M$. In the case of planar objects, one aspect per object is enough, while in the case of general 3D objects, more aspects are necessary to represent the object from different viewing directions. We assume that each aspect is represented by V different views which we call reference views. The number of reference views V per aspect depends on the transformations and projection under consideration and will be specified in the next paragraph. For each aspect, we assume a number of "interest" points N (e.g., corners, junctions, etc.), which are common in all the views associated with the aspect. We also assume that the point correspondences across the views have been established.

Algebraic functions of views were first introduced, in the case of scaled orthographic projection (weak perspective), by Ullman and Basri [16]. In particular, it was shown in [16] that if we let an object undergo 3D rigid transformations (i.e., rotations and translations in space) and we assume that the images of an object are obtained by orthographic projection followed by a uniform scaling, then any novel view of an object can be expressed as a linear combination of three other views of the same object. Specifically, let us consider three reference views of the same object V_1, V_2, and V_3, which have been obtained by applying different rigid transformations, and three points $p' = (x', y')$, $p'' = (x'', y'')$, and $p''' = (x''', y''')$, one from each view, which are in correspondence. If V is a novel view of the same object, obtained by applying a different rigid transformation, and $p = (x, y)$ is a point which is in correspondence with p', p'', and p''', then the coordinates of p can be expressed in terms of the coordinates of p', p'', and p''' as

$$x = a_1x' + a_2x'' + a_3x''' + a_4 \tag{1}$$

$$y = b_1y' + b_2y'' + b_3y''' + b_4, \tag{2}$$

where the parameters a_j, b_j, $j = 1, \ldots, 4$, are the same for all the points which are in correspondence across the four views.

The above result can be simplified if we generalize the orthographic projection by removing the orthonormality constraint associated with the rotation matrix. In this case, the object undergoes a 3D linear transformation in space. Linear combinations correspond to scaled orthographic projection followed by a 2D affine transformation and they characterize also the images that can be produced by a photograph of an object [26]. In this case, the algebraic functions of views are simpler and they involve only two reference views. Let us consider two reference views V_1 and V_2 of the same object which have been obtained by applying different linear transformations, and two points $p' = (x', y')$, $p'' = (x'', y'')$, one from each view, which are in correspondence. Then, given a novel view V of the same object which has been obtained by applying another linear transformation and a point $p = (x, y)$ which is in correspondence with points p' and p'', the coordinates of p can be expressed as a linear combination of the coordinates of p' and p'' as

$$x = a_1x' + a_2y' + a_3x'' + a_4 \tag{3}$$

$$y = b_1x' + b_2y' + b_3x'' + b_4, \tag{4}$$

where the parameters a_j, b_j, $j = 1, \ldots, 4$, are the same for all the points which are in correspondence across the three views. It is worth mentioning that not all the information from the second reference view is used but only "half" of it (i.e., only the x coordinates). Of course, (3) and (4) can be rewritten using the y coordinates of the second reference view instead.

The extension of algebraic functions of views in the case of perspective projection was carried out in [20–23]. In particular, it was shown that three perspective views of an object satisfy a trilinear function. Moreover, it was shown that a simpler and more practical pair of algebraic functions exist when the reference views are orthographic [20, 21]. This is useful for realistic object recognition applications. In this paper, we consider the case of orthographic projection assuming 3D linear transformations only.

3. A FRAMEWORK FOR INDEXING USING ALGEBRAIC FUNCTIONS OF VIEWS

Algebraic functions of views can be used to predict the image coordinates of points in a novel view by appropriately combining the image coordinates of the same points across a number of reference views. This idea can be used for recognizing unknown views of an object [17, 23]. There are two main problems with this approach: first, we need to find which points from the reference views correspond to which points from the unknown view and, second, we need to find the correct values for the parameters of the algebraic functions (i.e., a_j's, b_j's). Both problems are difficult to deal with. First of all, the number of possible point correspondences between reference and novel views increases exponentially with the number of points. Second, searching for the appropriate parameter values might be prohibitive since the domain of parameters might be very large [16]. Here, we propose the *coupling* the algebraic functions of views with indexing. The idea is to use algebraic functions of views to predict all the views (i.e., images) that a group of model points can produce and represent the predictions in a hash table. During recognition, groups of points are chosen from the scene and the hash table is accessed to find all the model groups that might have produced them along with information related to the point correspondences and the parameters of the algebraic functions.

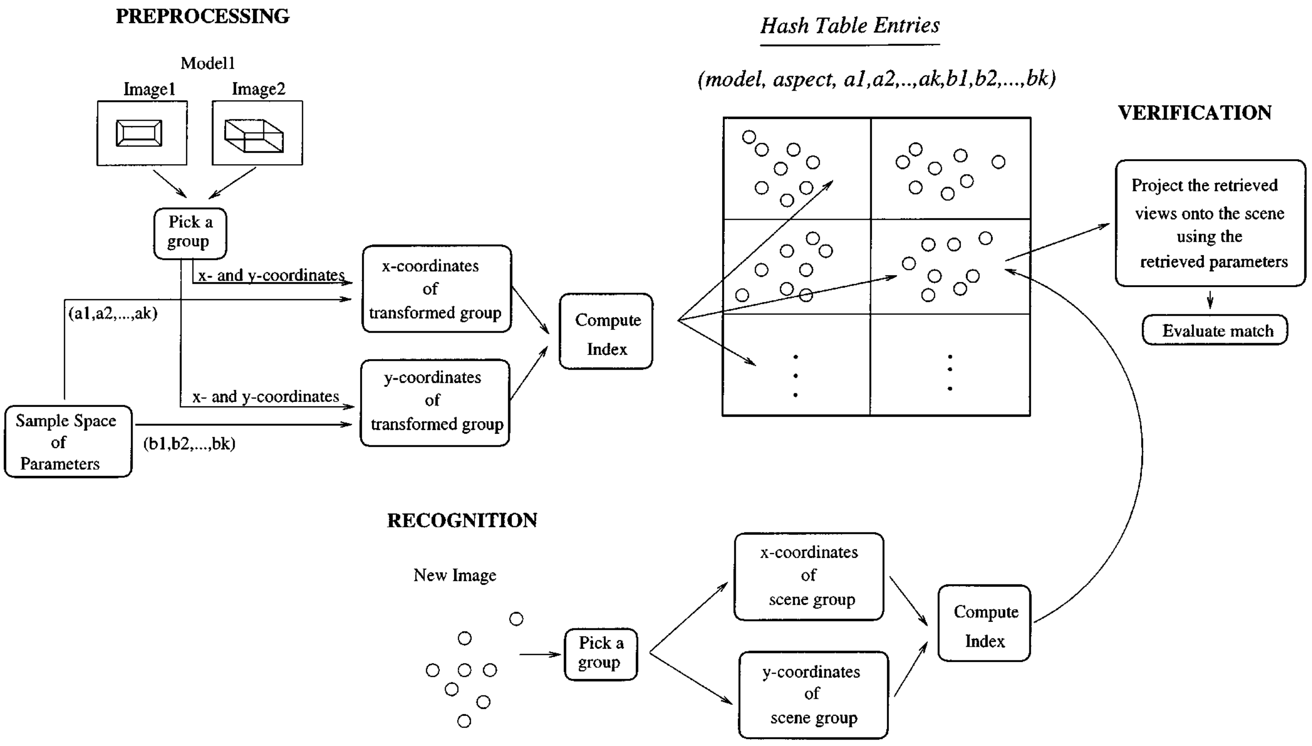
PREPROCESSING
Model1
Image1
Image2
Pick a group
x- and y-coordinates
(a1,a2,...,ak)
x-coordinates of transformed group
x- and y-coordinates
Sample Space of Parameters
(b1,b2,...,bk)
y-coordinates of transformed group
Compute Index
Hash Table Entries
(model, aspect, a1,a2,..,ak,b1,b2,...,bk)
VERIFICATION
Project the retrieved views onto the scene using the retrieved parameters
Evaluate match
RECOGNITION
New Image
Pick a group
x-coordinates of scene group
y-coordinates of scene group
Compute Index

FIG. 1. A framework for indexing using algebraic functions of views.

Given a model, a set of aspects, a set of reference views per aspect, and the point correspondences across the views of each aspect, the first step is to compute the allowable ranges of values that the parameters of algebraic functions can assume. Then, the views that a model group can produce (space of transformed views) can be computed by combining the reference views of the model group using algebraic functions of views. From a practical point of view, it is impossible to consider all possible combinations, that is, to assume all possible values for the parameters of the algebraic functions, since this will generate an infinite number of transformed model views. As a result, each parameter's range is actually sampled into a finite number of points and a finite number of transformed model groups is generated only. The coordinates of the transformed model groups are then used to generate an index to a hash table where information about the model, the aspect, the group, and the set of parameter values used to generate the transformed model group are stored. During recognition, we consider groups of scene points and we use their image coordinates to generate an index to the hash table. The entries stored at the indexed location identify a model, an aspect, a model group, and a set of parameter values that might have produced the scene group. A verification step follows to reject or accept candidate matches. Figure 1 illustrates these steps.

There are some important issues that must be considered during the implementation of the proposed scheme. One of them is how to compute the range of values that the parameters of the algebraic functions can assume. Here, we propose a method based on SVD [24] and IA [25]. Another important issue has to do with the space requirements of the method. We are dealing with this issue (i) by *preconditioning* the reference views in order to compute narrow ranges of values for the parameters of algebraic functions, (ii) by generating and storing information about only the x or y coordinates of the transformed model groups, and (iii) by considering only *well-conditioned* groups, that is, groups which are tolerant to noise. Finally, we consider the issue of predicting the parameters of the algebraic functions accurately during recognition. A scheme based on neural networks is employed for this.

4. ESTIMATING THE RANGES OF VALUES FOR THE PARAMETERS

Under the assumption of orthographic projection, two reference views V_1 and V_2 must be combined in order to obtain a new view V, as Eqs. (3) and (4) illustrate. Given the point correspondences across the three views, the following system of equations must be satisfied,

$$\begin{bmatrix} x_1' & y_1' & x_1'' & 1 \\ x_2' & y_2' & x_2'' & 1 \\ \cdots & \cdots & \cdots & \cdots \\ x_N' & y_N' & x_N'' & 1 \end{bmatrix} \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \\ a_4 & b_4 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \cdots & \cdots \\ x_N & y_N \end{bmatrix}, \qquad (5)$$

where $(x_1', y_1'), (x_2', y_2'), \ldots (x_N', y_N')$ and $(x_1'', y_1''), (x_2'', y_2''), \ldots$

(x''_N, y''_N) are the coordinates of the points of the reference views V_1 and V_2, respectively, and $(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N)$ are the coordinates of the points of the novel view V. Splitting the above system of equations into two subsystems, one involving the a_j parameters and one involving the b_j parameters, we have

$$Pc_1 = p_x \tag{6}$$

$$Pc_2 = p_y, \tag{7}$$

where P is the matrix formed by the x and y coordinates of the reference views (plus a column of 1's), c_1 and c_2 are vectors corresponding to a_j's and b_j's (the parameters of the algebraic functions), and p_x, p_y are vectors corresponding to the x and y coordinates of the novel view. Both (6) and (7) are overdetermined which means that they can be solved using a least-squares approach such as SVD [24]. Since SVD is very important for the estimation of the parameters' ranges, we briefly present its main steps here. Using SVD, P can be factorized as $P = U_P W_P V_P^T$ where both U_P and V_P are orthonormal matrices, while W_P is a diagonal matrix whose elements w_{ii}^P are always nonnegative (the singular values of P). The solutions of the above two systems are $c_1 = P^+ p_x$ and $c_2 = P^+ p_y$ where P^+ is the pseudoinverse of P. Assuming that P has been factorized, its pseudoinverse is $P^+ = V_P W_P^+ U_P^T$ where W_P^+ is also a diagonal matrix with elements $1/w_{ii}^P$ if w_{ii}^P greater than zero (or a very small threshold in practice) and zero otherwise. In specific, the solutions of (6) and (7) are given by the equations [24]

$$c_1 = \sum_{i=1}^{k} \left(\frac{u_i^P p_x}{w_{ii}^P} \right) v_i^P \tag{8}$$

$$c_2 = \sum_{i=1}^{k} \left(\frac{u_i^P p_y}{w_{ii}^P} \right) v_i^P, \tag{9}$$

where u_i^P denotes the ith column of matrix U_P, v_i^P denotes the ith column of matrix V_P, and $k = 4$.

To determine the range of values for c_1 and c_2, we assume first that the novel views has been scaled such that the x and y coordinates belong within a specific interval. This can be done, for example, by mapping the novel view to the unit square. In this way, its x and y image coordinates will be mapped to the interval [0, 1]. To determine the range of values for c_1 and c_2, we need to consider all possible solutions of (6) and (7), assuming that p_x and p_y belong to [0,1]. We have used IA [25] in order to solve this problem. In IA, each variable is represented as an interval of possible values. Given two interval variables $t = [t_1, t_2]$ and $r = [r_1, r_2]$, then the sum and the product of these two interval variables is defined as [25]

$$t + r = [t_1 + r_1, t_2 + r_2] \tag{10}$$

$$t * r = [\min(t_1 r_1, t_1 r_2, t_2 r_1, t_2 r_2),\ \max(t_1 r_1, t_1 r_2, t_2 r_1, t_2 r_2)]. \tag{11}$$

Applying the interval arithmetic operators to (8) and (9), instead of standard arithmetic operators, we can compute interval solutions for c_1 and c_2 by setting $p_x = [0,1]$ and $p_y = [0,1]$. In interval notation, we want to solve the systems $Pc_1 = p_x^I$ and $Pc_2 = p_y^I$, where the superscript I denotes an interval vector. The solutions c_1^I and c_2^I should be understood to mean $c_1^I = [c_1\colon Pc_1 = p_x,\ p_x \in p_x^I]$ and $c_2^I = [c_2\colon Pc_2 = p_y,\ p_y \in p_y^I]$.

Significant research has been performed in the area of interval linear systems [27]. In general, the matrix of a system of interval equations is also an interval matrix, that is, a matrix whose components are interval variables. In our case, things are simpler since the elements of P are the x and y coordinates of the reference view of the object which are always fixed. When interval solutions are computed, not every solution in c_1^I and c_2^I corresponds to p_x and p_y that belong to p_x^I and p_y^I [27, 28]. In other words, $p_x^I \subseteq Pc_1^I$ and $p_y^I \subseteq Pc_2^I$. In the context of our approach, if new views are generated by choosing the values for the parameters of the algebraic functions from the interval solutions obtained, then some of the generated views might not lie completely within the unit square. We will be referring to these views as "invalid views" and to the solutions which generate the invalid views as "invalid solutions." Clearly, invalid views can be rejected easily by testing whether the coordinates of a view lies within the unit square.

An interval solution is called "sharp" if it does not contain many invalid solutions. Within our context, it is important to compute sharp interval solutions since this will save time and space. However, if we merely apply the interval arithmetic operators on (8) and (9), then it is very likely that we will obtain solutions that will not be very sharp. There are various factors that affect the sharpness of an interval solution. One of them is the participation of a given interval quantity to the computations of a solution more than once [28]. As a matter of fact, this is the case with (8) and (9). To make it clear, let us rewrite the solution for the ith component of c_1, $1 \le i \le k$, more analytically:

$$\begin{aligned} c_{i1} = {} & \frac{v_{i1}^P}{w_{11}^P}\left(u_{11}^P x_1 + u_{21}^P x_2 + \cdots + u_{N1}^P x_N\right) \\ & + \frac{v_{i2}^P}{w_{22}^P}\left(u_{12}^P x_1 + u_{22}^P x_2 + \cdots + u_{N2}^P x_N\right) \\ & + \cdots \frac{v_{ik}^P}{w_{kk}^P}\left(u_{1k}^P x_1 + u_{2k}^P x_2 + \cdots + u_{Nk}^P x_N\right). \end{aligned}$$

Clearly, each x_j $(1 \le j \le N)$ enters in the computation of c_{i1} more than once. To avoid this, we can factor out the x_j's and rewrite the above equation as

$$c_{i1} = \sum_{j=1}^{N} x_j \left(\sum_{r=1}^{k} \frac{v_{ir}^P u_{jr}^P}{w_{rr}^P} \right).$$

The interval solution c_{i1}^I can now be obtained by applying the interval arithmetic operations on the equation above. Similarly,

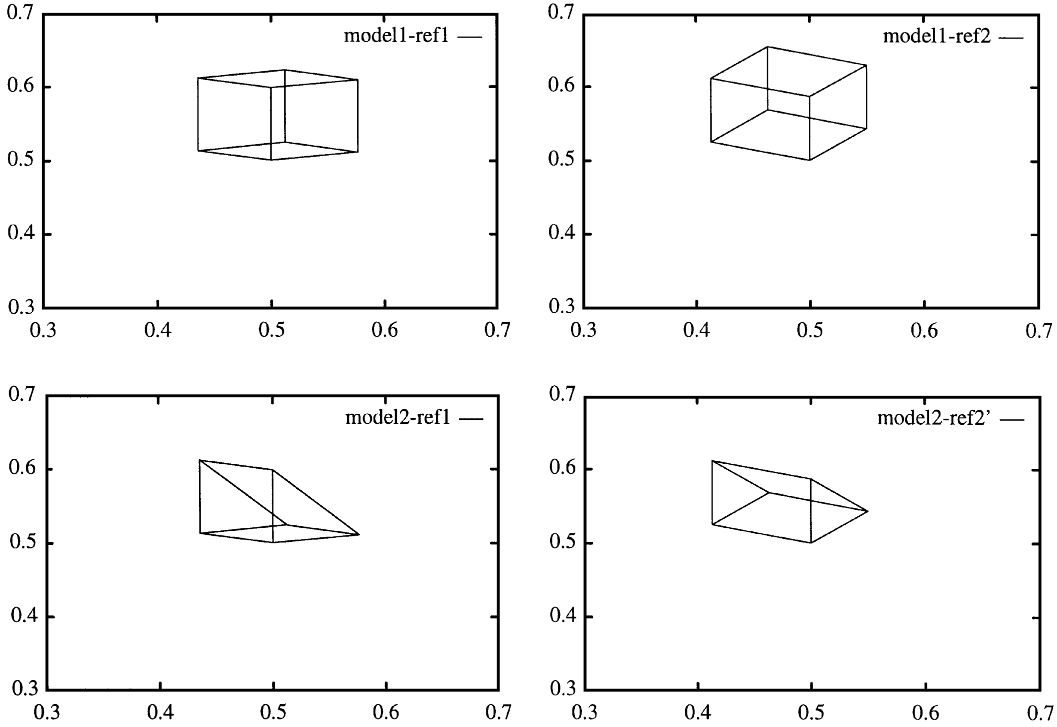

FIG. 2. Some artificial 3D objects.

we can obtain interval solutions for the rest elements of c_1^I as well as for c_2^I. It should be noted that since both (8) and (9) involve the same matrix P and p_x^I, p_y^I assume values form the same interval, the interval solutions c_1^I and c_2^I will be the same.

As an example, let us consider the 3D objects shown in Fig. 2 (two different reference views are shown per object). The interest points used in this experiment correspond to corner points. Table 1 shows the range of values computed for c_1 (which are the same with those computed for c_2).

5. PRECONDITIONING THE REFERENCE VIEWS

As Table 1 illustrates, the width of the range of values varies from parameter to parameter. Wide ranges are not desirable because more sets of values must be considered. It is thus important to consider ways to compute narrower ranges. In this section, we present a methodology called *preconditioning* for optimizing the parameters' ranges of values. By preconditioning we imply a transformation that transforms the original reference views to new reference views, yielding very narrow ranges. Before we describe the steps involved in this transformation, let us first investigate how the width of the ranges is affected.

In the previous section, we considered the interval solutions of $Pc_1 = p_x$ and $Pc_2 = p_y$. Alternatively, we could have considered the solutions of $P(c_1 + \delta c_1) = (p_x + \delta p_x)$ and $P(c_2 + \delta c_2) = (p_y + \delta p_y)$, assuming all possible δp_x and δp_y with $(p_x + \delta p_x)$ and $(p_y + \delta p_y)$ in [0,1] (p_x and p_y can be assumed fixed, for example, $p_x = p_y = 0.5$). Obviously, the width of the computed interval solutions c_1^I and c_2^I will depend on the magnitude of δc_1 and δc_2. It is well known that the relative error in the solution of a system of equations depends on the condition number of P [29]. In other words, if we consider the system $P(c_1 + \delta c_1) = (p_x + \delta p_x)$, the following inequality is known to be true,

$$\frac{\|\delta c_1\|}{\|c_1\|} \leq cond(P)\frac{\|\delta p_x\|}{\|p_x\|},$$

TABLE 1
The Computed Ranges Using the Original Views

	Ranges of values			
	range of a1	range of a2	range of a3	range of a4
model1	[−25.321 25.321]	[−10.154 10.154]	[−23.173 23.173]	[−5.943 6.943]
model2	[−27.771 27.771]	[−10.154 10.154]	[−24.328 24.328]	[−8.496 9.496]

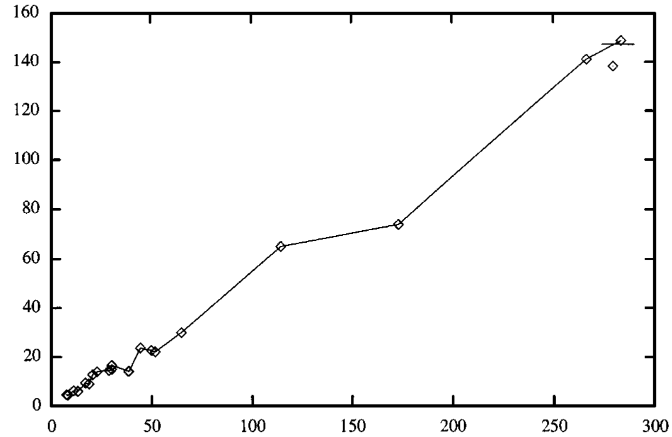

FIG. 3. The width of a_1's range versus the condition of the reference views.

where *cond*(P) is the condition number of P defined as $\|P\|\,\|P^{-1}\|$. If the condition number of P is large, then the relative error will also be large which implies that the width of c_1's range will also be large. The same holds true for c_2.

We define the "condition" of a reference view as the ratio of the maximum singular value over the minimum singular values of P. This ratio can be regarded as the condition number of P and while it is not exactly equal to the actual condition number, they usually have about the same order of magnitude numerically [24, 29]. We have performed a number of experiments to demonstrate the dependence of the width of the parameters' ranges on the condition of the reference views. First, we generated a number of random views per object and we computed each parameter's range. Then, for each parameter, we plotted the condition of the view (horizontal axis) versus the width of the computed ranges (vertical axis). Figure 3 shows one of the plots, assuming 20 random views, *model*3, and a_1. Clearly, large condition numbers imply wide ranges.

It is thus reasonable to ask whether it is possible to choose reference views having the best possible condition. Here, we propose a procedure (*preconditioning*) to transform the original reference views to new reference views having better condition. A transformation to obtain new reference views from the old reference views can be obtained using (5)

$$\begin{bmatrix} x_1' & y_1' & x_1'' & 1 \\ x_2' & y_2' & x_2'' & 1 \\ \cdots & \cdots & \cdots & \cdots \\ x_N' & y_N' & x_N'' & 1 \end{bmatrix} \begin{bmatrix} a_1 & b_1 & a_5 & 0 \\ a_2 & b_2 & a_6 & 0 \\ a_3 & b_3 & a_7 & 0 \\ a_4 & b_4 & a_8 & 1 \end{bmatrix} = \begin{bmatrix} x_1'^n & y_1'^n & x_1''^n & 1 \\ x_2'^n & y_2'^n & x_2''^n & 1 \\ \cdots & \cdots & \cdots & \cdots \\ x_N'^n & y_{N_m(a)}'^n & x_N''^n & 1 \end{bmatrix} \tag{12}$$

or

$$PC = P^n, \tag{13}$$

where C is a transformation matrix, P is the matrix corresponding to the old reference views, and P^n is the matrix corresponding to the new reference views. The idea is to find a matrix C which yields new reference views having better condition. Let us consider the singular value decomposition of P, C, and P^n: $P = U_P W_P V_P^T$, $C = U_C W_C V_C^T$, and $P^n = U_{P^n} W_{P^n} V_{P^n}^T$. Substituting these expressions in (13) we have

$$\left(U_P W_P V_P^T\right)\left(U_C W_C V_C^T\right) = \left(U_{P^n} W_{P^n} V_{P^n}^T\right). \tag{14}$$

In order for the new matrix P^n to have good condition, its singular values, that is, the elements of W_{P^n}, must have similar magnitudes. Observing (14) we find it rather difficult to draw any conclusions about the condition of the new view P^n. However, this would be much easier if we could relate the singular values of P^n to the singular values of P and C. Without making any assumptions about the transformation matrix C, it is difficult to establish such as relationship. However, since we have freedom in choosing the elements of C, (14) can be simplified if we choose $U_C = V_P$. Then, (14) can be written as $(U_P W_P V_P^T)(V_P W_C V_C^T) = (U_{P^n} W_{P^n} V_{P^n}^T)$ or $(U_P W_P W_C V_C^T) = (U_{P^n} W_{P^n} V_{P^n}^T)$, since $V_P^T V_P = I$. According to the above equation, the singular values of P^n are now equal to the product of the singular values of P and C, that is, $W_{P^n} = W_P W_C$. The key idea is then to choose the singular values of C in a way such all that the singular values of P^n have the same magnitude. Obviously, we must choose W_C as

$$W_C = \lambda W_P^{-1}, \tag{15}$$

where λ is a positive constant. As a result, $W_{P^n} = \lambda I$, which means that all the singular values of P^n will be equal to λ and the condition of the new view will be the best possible (one). The details involved in the calculation of λ as well as in the calculation of the elements of matrix C can be found in Appendix A.

We have performed a number of experiments to demonstrate the preconditioning procedure, using the objects shown in Fig. 2. Figure 4 shows the preconditioned views (only the first preconditioned reference view is shown per object since only the x coordinates of the second reference view are used). One comment we can make by observing Fig. 4 is that preconditioning seems to spread the views around origin. Table 2 shows the ranges of the parameters in the case of the preconditioned reference views. Comparing them with the ranges obtained using the original views (Table 1) we can conclude that preconditioning yields very tight ranges.

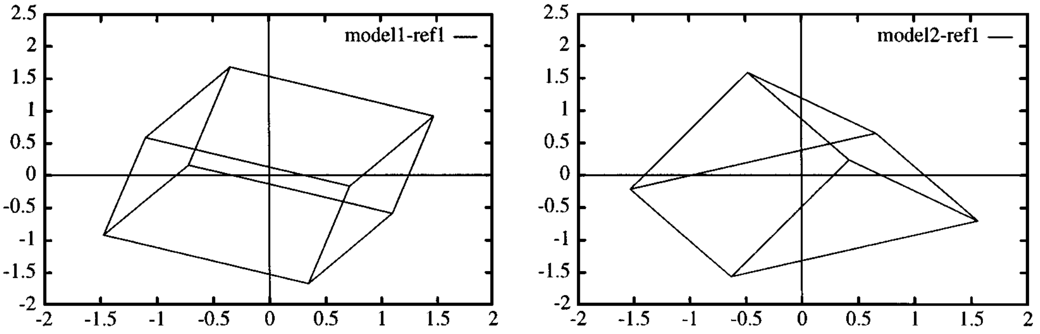

FIG. 4. The "preconditioned" reference views.

6. DECOUPLING THE IMAGE COORDINATES

In this section, we consider again the preprocessing step of the proposed approach and we show that there is significant redundancy in the information stored in the hash table. According to our discussion in Section 3, for each model group, we compute first the images that the model group can produce (i.e., transformed model groups). Then, the coordinates of the points in the transformed model group are used to store appropriate information in the hash table. The computation of the coordinates of the points in a transformed model group is performed using Eqs. (3) and (4). There are two observations to be made at this point. First, let us recall that both a_j's and b_j's assume values from the same ranges (see our discussion in Section 4). Second, the same basis vector (i.e., (x', y', x'')) is involved in the computation of both the x and y coordinates of the groups. Based on these two observations it can be easily concluded that the transformation which generates the x coordinates is exactly the same as the transformation which generates the y coordinates. As a result, it is not necessary to represent the same transformation twice over the hash table and only one of the two coordinates (the x coordinates here) can be used for indexing. This simplification offers significant time and space savings; however, recognition becomes slightly more complicated. Specifically, the hash table must be accessed twice per scene group during recognition: first, the x coordinates of the scene group are used to give rise to hypotheses which predict the a_j parameters and, second, the y coordinates of the scene group which will give rise to hypotheses which predict the b_j parameters. Then, the intersection of the hypotheses needs to be found. Figure 5 shows the revised scheme where only the x coordinates of the transformed model groups are utilized.

TABLE 2
The Computed Ranges Using the Preconditioned Views

	Ranges of values			
	range of a1	range of a2	range of a3	range of a4
model1	[−0.454 0.454]	[−0.417 0.417]	[−0.392 0.392]	[0.000 1.000]
model2	[−0.439 0.439]	[−0.413 0.413]	[−0.423 0.423]	[0.000 1.000]

7. PREDICTING THE PARAMETERS OF THE ALGEBRAIC FUNCTIONS DURING RECOGNITION

Given a scene group, the goal of recognition is to predict the model group and the parameters of the algebraic functions that have produced the scene group. However, it is important to understand that there will be errors in the recovery of the parameters mainly because the hash table is built by sampling the space of parameters into a finite number of points. As a result, if an actual image group is not very similar to one of the transformed model groups computed during preprocessing, then the predicted parameters might not be very close to the actual ones. Of course, the error depends on the sampling step used to sample the space of parameters. This error can be made small by choosing a small sampling step but this is not desirable since it will increase space requirements. Since errors in the prediction of the parameters will have a great impact on the performance of the verification step (i.e., the predicted model might not be back-projected onto the scene accurately), it is important to consider approaches which will allow us to predict the parameters accurately.

In a recent paper [31], we studied the problem of learning to predict the correct pose of a planar object, undergoing 2D affine transformations. The idea was to train a neural network with a number of affine transformed views of the object in order for it to learn to predict the parameters of the affine transformation between the training views and a reference view of the object. To demonstrate our approach, we performed experiments using several objects. A separate neural network was assigned to each object which was trained with views of this object only (*object-specific networks*). Our experimental results showed that training was extremely fast and that only a small number of training views was sufficient for the networks to generalize well. By generalization we mean the ability of the networks to predict the correct affine transformation even for views that were never exposed to them during training. We also considered issues related to the discrimination power and noise tolerance of the networks.

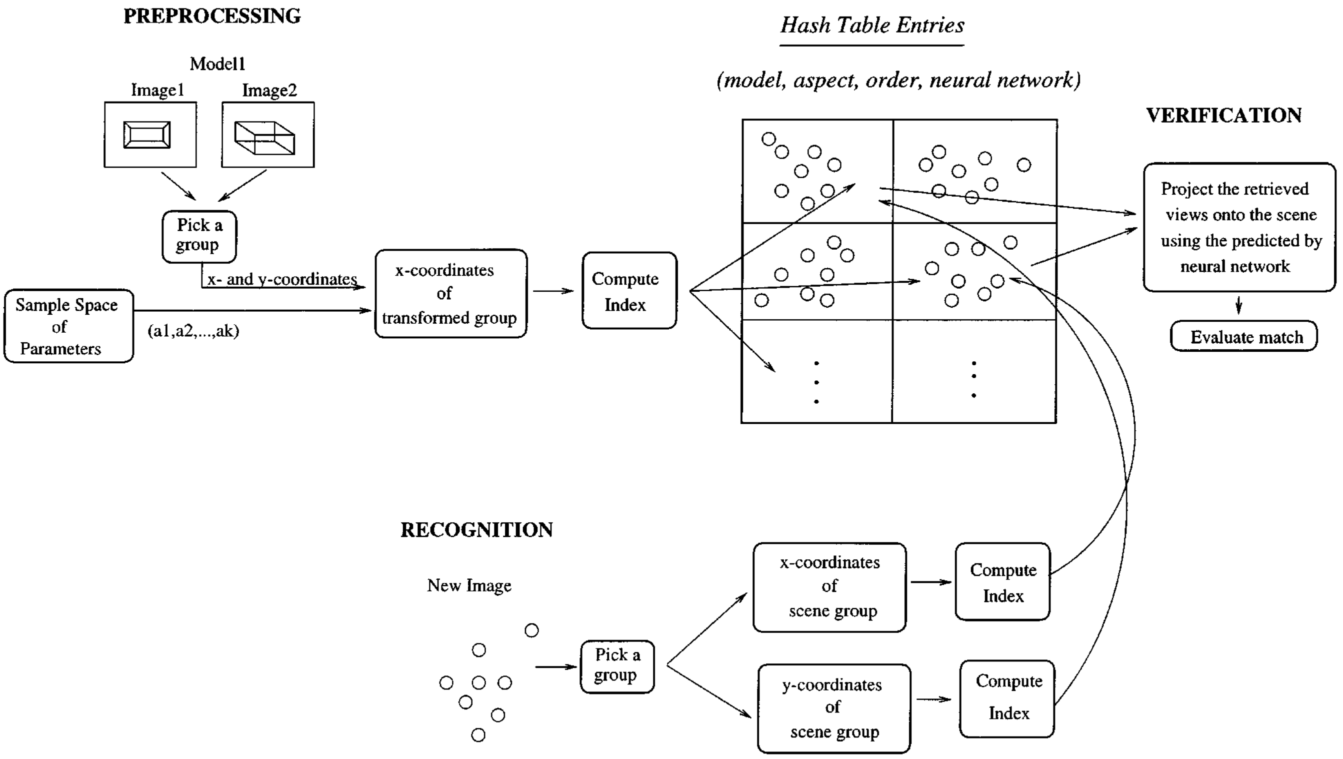
PREPROCESSING
Model1
Image1
Image2
Pick a group
x- and y-coordinates
Sample Space of Parameters
(a1,a2,...,ak)
x-coordinates of transformed group
Compute Index
Hash Table Entries
(model, aspect, order, neural network)
VERIFICATION
Project the retrieved views onto the scene using the predicted by neural network
Evaluate match
RECOGNITION
New Image
Pick a group
x-coordinates of scene group
y-coordinates of scene group
Compute Index
Compute Index

FIG. 5. A revised framework for indexing using algebraic functions of views.

Our results showed that the discrimination power of the networks was excellent. Their noise tolerance was not very good initially; however, it was dramatically improved by applying a preprocessing to the inputs based on PCA [24].

Motivated by this work, we have decided to use the same approach for model groups. The idea is assign a different neural network for each model group (*group-specific neural networks*). To train the networks, we generate a number of training views which contain the groups. In fact, the training views can be chosen from the views we generate during the hash table construction step. Then, when an entry is stored into the hash table, instead of storing the parameters of the algebraic function we store a pointer to the neural network associated with the model group. Figure 5 shows the hash entries. During recognition, the coordinates of the scene group are used to retrieve from the hash table appropriate hash entries. Then, the parameters of the algebraic functions are estimated by presenting the coordinates of the scene group to the neural network whose pointer is part of the hash entries retrieved. Figure 6a illustrates the neural network approach. Figure 6b illustrates the simplified neural network scheme (only the x coordinates are used).

It should be noted that the neural network approach is not the only alternative approach to recover the parameters of the algebraic functions. For example, we could have stored the coordinates of the transformed model groups in the hash table during preprocessing. Then, when a scene group is matched to a model group during recognition, the parameters can be recovered by solving two systems of equations (Eqs. (6) and (7)). Since the systems are overdetermined, a least-squares approach, such as SVD [24], can be used. However, the neural network scheme has the advantage that it is faster and has less space requirements. To see this, let us assume that the decomposition of P is computed offline, as the training of the neural networks is also performed offline. Assuming that each model group contains G points and that the number of parameters is $2k$ ($k = 4$ in our case), the neural network approach requires $2kG$ multiplications and $2kG$ additions to predict the parameters of the transformation (linear networks). On the other hand, SVD requires $2k(G + 2k)$ multiplications, $2kG$ divisions, and $2k(G + 2k)$ additions (see Eqs. (8) and (9)). Given that these computations must be repeated many times during recognition, the neural network approach is obviously less time consuming. In terms of space requirements, the neural network approach requires the storage of $2kG$ values per network (i.e., weights), while SVD requires the storage $2kG + 2k + (2k)^2$ values (U, W, and V matrices). Given again that this information must be stored for many groups, the neural network approach has less space requirements.

8. GROUP SIZE, WELL-CONDITIONED GROUPS, AND ORDERING

An important issue when we consider groups of points is how to chose the group size G, that is, the number of points in a group. Obviously, in order for the groups to be useful for

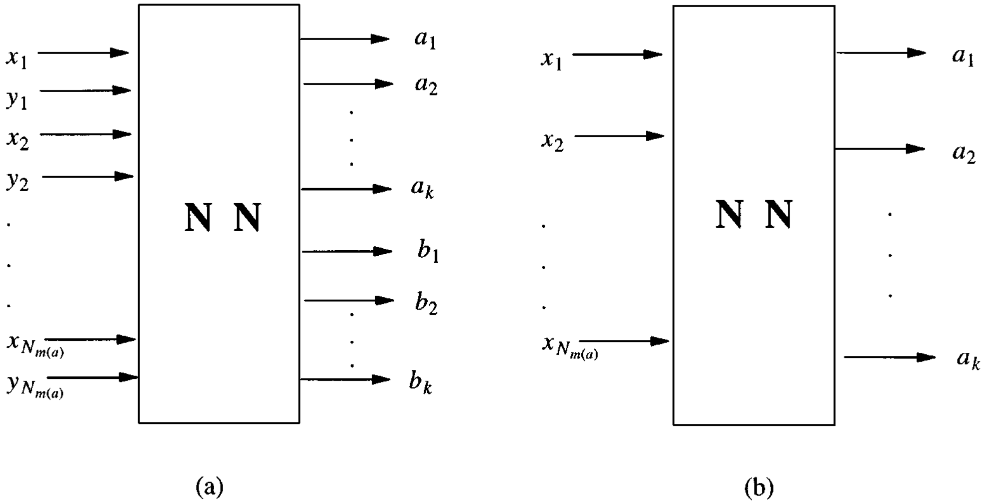

FIG. 6. (a) The neural network scheme; (b) The simplified neural network scheme.

matching, they must provide enough discriminating power. In fact, it is desirable to choose G in a way such that every group of image points may have been produced only by one group of model points. For 3D linear transformations, the algebraic functions of views involve eight parameters. This means that we need to match at least four image points to four model points in order to determine the parameters. As a result, the minimum group size which provides some discrimination is five.

It has been shown that the likelihood a particular group of image points matches a particular group of model points shrinks exponentially with the size of the group [12]. However, it is not possible to consider large size groups because they are more vulnerable to occlusions. Also, the discriminatory improvement offered by large groups diminishes rapidly beyond some point [7]. In this paper, we have chosen to demonstrate our approach using groups of size five ($G = 5$). It should be noted, however, that this choice is somewhat subjective, and other approaches might be more appropriate, for example, using multiple group sizes, an adaptive group size, or grouping [14].

Considering all possible model groups for a given size is not practical since this would require too much space. Here, we consider only *well-conditioned* model groups. The definition of the condition of a model group is similar to the definition of the condition of a model view: it is the condition of the $G \times k$ matrix (denoted as P_{g_m}), formed by considering the x and y coordinates of the points in the group, across the reference views, plus a column of 1's (see Eq. (5)). Assuming noise in the location of the image points, the solution of (5) will include some error which is related to the condition of the matrix P_{g_m} [29]. As a result, even though a model group might have been correctly matched to an image group, it will be very difficult for the verification procedure to find additional matches to support this hypothesis. This will waste recognition time and it is better to avoid such hypothetical matches from the beginning by disqualifying bad-conditioned model groups during preprocessing.

To choose groups with good condition, we simply compute the condition of each model group and then we reject groups having a condition greater than a threshold t. It is important, however, to ensure that most model points are represented in the groups chosen and that the same model point does not appear in every model group chosen. This is to ensure that recognition does not depend on a few model points which might not be always available anyway due to occlusions. To find the number of times a model point appears in the groups chosen we construct a histogram. If there are many model points that are not sufficiently represented in the groups chosen, we choose new model groups by increasing the threshold t. If a model point appears in many groups (e.g., in half or more), we start removing groups which contain this point, updating the histogram at the same time, until a certain criterion is met (e.g., at most half of the groups contain the point). If this procedure eliminates many groups, then we increase t and we repeat the same steps.

Another important issue is the order of the points in a group. If we do not make any assumptions about the order, either all possible orders must be considered during preprocessing or all possible orders must be considered during recognition. Since the second approach will increase recognition time, we consider the first approach only. To avoid considering all possible orders during preprocessing, we apply a canonical ordering to the points of the model groups. During recognition, the same canonical ordering is applied to the scene groups. Information about the ordering is stored in the hash table during preprocessing. The canonical ordering procedure employed here is very simple: we just sort the x coordinates of the points within a group in increasing order. During recognition, we sort both the x and y coordinates of the scene groups before we compute the indices to access the hash table. A different canonical ordering procedure has been proposed in [12]; however, it is not applicable here because only the x coordinates of the model groups are used during preprocessing.

9. ELIMINATING INVALID HYPOTHESES

By considering only well-conditioned groups during preprocessing, we have restricted ourselves to a much smaller set of model groups. This saves space during preprocessing but the probability of selecting a scene group which matches one of the model groups is now much smaller. Hence, most of the hypotheses that will be established during recognition will be incorrect and must be ruled out quickly. If the unknown scene contains more than one object, it will be very beneficial to apply some kind of grouping in order to identify groups of scene points that might belong to the same object. Then, we can select subgroups of size G from these groups instead of selecting them randomly. This approach will eliminate many matches, but there will be still many invalid matches left. To speed up recognition, it is important to keep the number of hypotheses low. Our approach is to reject as many invalid matches as possible without having to verify them first. This is performed by evaluating each hypothesis before verification, using a number of simple tests. If a hypothesis passes all the tests, then it is passed to the verification step; otherwise, it is rejected.

The way hypotheses are formed during recognition is by combining every hash entry retrieved using the x coordinates of the scene group with every hash entry retrieved using the y coordinates of the group. This will produce many hypotheses but not all of them need to be verified. In specific, let us consider a hypothesis formed by combining the entry ($model_x$, a_x, nn_x, $order_x$) retrieved using the x coordinates of the scene group with the entry ($model_y$, a_y, nn_y, $order_y$) retrieved using the y coordinates of the same group. This hypothesis will be considered for verification, only if all of the following five conditions are satisfied: (1) $model_x = model_y$, (2) $a_x = a_y$, (3) $nn_x = nn_y$, (4) the parameters predicted by nn_x and nn_y are within the ranges computed during preprocessing, and (5) the predicted model group is well-conditioned. Figure 7 illustrates the procedure.

The first two constraints are straightforward to understand: both the x and y coordinates of the scene group should predict the same model and aspect. The third constraint implies that both the x and y coordinates of the scene group should predict the same model group. The identity of a model group is implicitly implied by the identity of the neural network associated with it. The fourth constraint exploits the discriminating power of the neural networks. If some points in the image group do not belong to the same object, it is expected that the parameters predicted by the neural network will not be within the ranges computed during preprocessing. Experimental results obtained

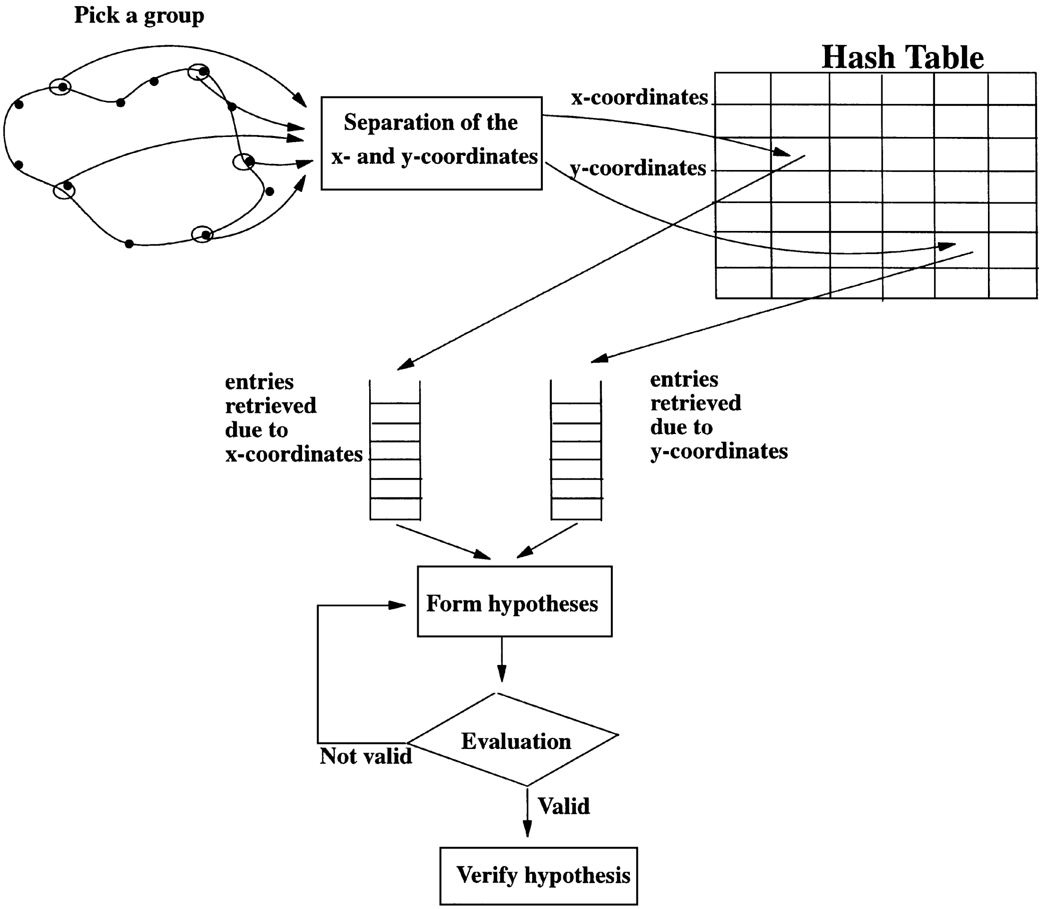
Pick a group
Separation of the x- and y-coordinates
x-coordinates
y-coordinates
Hash Table
entries retrieved due to x-coordinates
entries retrieved due to y-coordinates
Form hypotheses
Evaluation
Not valid
Valid
Verify hypothesis

FIG. 7. Evaluation of hypotheses.

here as well as in [31] have shown that the discriminating power of the neural networks is very good.

The purpose of the last constraint is to rule out scene groups that have not been produced by well-conditioned model groups. Assuming that P_{g_m} is the matrix formed by the coordinates of the points in the model group and P_{g_s} is the matrix formed by the coordinates of the points in the image group, then $P_{g_m} C = P_{g_s}$ (see Eq. (13)). P_{g_s} consists of four columns, the first two of which are the x and y coordinates of the scene group, and the last one is just a column of 1's. The third column corresponds to the transformed x coordinates of the second reference view. Since objects are recognized from a single unknown image, the elements of this column are chosen to be the same as the x coordinates of the group in the second reference view. The matrix C consists of four columns as well. The first two are set to the parameters predicted by the neural networks, and the fourth one is just the vector $[0001]^T$. The third column must be set equal to the vector $[0010]^T$ so that the third column of P_{g_s} is the same as the third column of P_{g_m}. P_{g_m} can be computed by multiplying P_{g_s} by C^{-1}. The last constraint simply checks whether the condition of $P_{g_m} = C^{-1} P_{g_s}$ is less than a threshold (the same threshold used during preprocessing).

10. SPACE REQUIREMENTS, SAMPLING EFFECT, AND NOISE TOLERANCE

In this section, we consider several important issues with regard to the performance of the method: space requirements, effect of sampling of parameters, and noise tolerance. If we assume N points per reference view and a group size G, there are $N_G = \binom{N}{G}$ possible model groups (without considering different orderings since we use canonical ordering). Let us denote the number of well-conditioned groups as $\tilde{N}_G$ (with $\tilde{N}_G \ll N_G$). If the sampling step used to sample the range of parameter a_j is s_{a_j}, $j = 1, 2, \ldots, k$ $(k = 4)$, each parameter can assume $n_{a_j} = ((\max_{a_j} - \min_{a_j})/s_{a_j} + 1)$ values $(\min_{a_j}, \max_{a_j}$ correspond to the min and max values of $a_j)$. Thus, the number of transformed views we need to be generate per aspect is $N_V = \prod_{j=1}^{k} n_{a_j}$. Not all of these views have to be considered during preprocessing since some of them correspond to "invalid" views (i.e., they do not lie entirely within the unit square). Let us denote the number of valid views as $\tilde{N}_V$ $(\tilde{N}_V \ll N_V)$. Each time a valid view is generated, a hash entry is made for each well-conditioned group contained in the view. Thus, the total number of entries that must be stored per model is $A_m \tilde{N}_V \tilde{N}_G$ where A_m is the number of aspects associated with model m. This is the number of entries to be stored without accounting for noise. If we choose to account for noise during preprocessing, additional entries must be stored as we discuss later in this section. For each group, we also need to store the weights of the neural network associated with the group. Each network has k inputs and k outputs, that is, k^2 weights must be stored.

Let us now turn our attention to the sampling of the parameters. If the sampling step s_{a_j} is very large, then the sampling of the space of transformed views will be very coarse. This, however, will affect recognition since the point coordinates of the transformed model groups will be very different from the point coordinates of the actual image groups. As a result, it is very likely that the actual image groups will access wrong hash bins during recognition. To investigate this issue more carefully, we need to be more specific about the function *index*( ) which returns the hash table address. In this paper, the index space considered is the space of image coordinates. This is different from other approaches where the space of affine coordinates is considered instead [4, 13, 14]. The main reason that the affine space has been considered in other approaches is because it yields a minimal representation. Although this is indeed true, an analysis of the effect of sensor noise is more complicated in this case. Also, the affine space does not allow for accounting for sensor noise during preprocessing [32]. Representing the model groups in the space of image coordinates does not yield a minimal representation but the analysis for the effect of sensor noise is easier and it allows to account for sensor noise during preprocessing. In our implementation, the dimensionality of the hash table is equal to the group size G, that is, *index*( ) accepts as input the x or y coordinates of a group and returns a G-dimensional index.

To demonstrate the indexing procedure, let us assume that $G = 1$. In this case, *index*( ) implies a quantization of the interval [0,1] plus a linear scaling to ensure that the computed index fits the dimensions of the hash table (see Fig. 8). Specifically, the interval [0,1] is partitioned into a number of subintervals, and a hash bin is assigned to each one of them. The number of subintervals is determined by the size of the hash table which is denoted as H. The width of each subinterval will be $h = 1/H$, and the knot points q_j, which define the subintervals, will be $q_j = j * h$, $j = 1, 2, \ldots, H - 1$. If x_k is the x coordinate of a point, then *index*$(x_k) = Q(x_k) = j$ if $x_k \in [q_j^* - h/2, q_j^* + h/2]$, with q_j^* being the middle point of $[q_j, q_{j+1}]$. In general $(G \neq 1)$, *index*( ) implies a quantization of a hypercube with sides equal to [0,1]. Assuming that $(x_1, \ldots, x_G)$ are the x coordinates of the points in a group, a G-dimensional index is computed by following the previous procedure for each x coordinate of the group, that is, *index*$(x_1, x_2, \ldots, x_G) = (Q(x_1), Q(x_2), \ldots, Q(x_G))$.

Let us now assume that the x coordinates of an actual image group are $(x_1, x_2, \ldots, x_G)$ while the x coordinates of a predicted image group, corresponding to the above actual image group, are $(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_G)$. The coordinates of the predicted image group have been obtained by applying the transformation

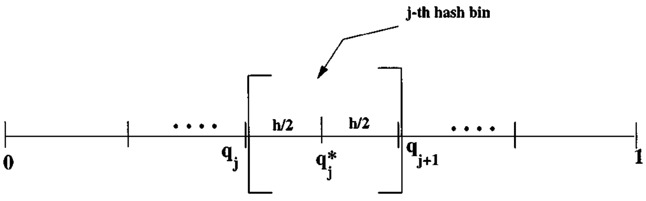

FIG. 8. Demonstration of the index-generation process.

$\hat{x}_i = \hat{a}_1 x'_i + \hat{a}_2 y'_i + \cdots + \hat{a}_k$, where $x'_i, y'_i, \ldots,$ are the coordinates of the model group which has produced the image group and $\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_k$ are the predicted parameters of the algebraic functions. Let us assume that the actual parameters of the algebraic functions are $a_1, \ldots, a_k$. Then, the x coordinates of the actual image group are given by $x_i = a_1 x'_i + a_2 y'_i + \cdots + a_k$. The question is whether the actual and predicted groups access the same hash bin; that is, whether $index(\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_G) = index(x_1, x_2, \ldots, x_G)$. This will be true if $Q(\hat{x}_i) = Q(x_i)$, $k = 1, 2, \ldots, G$. Let us assume that $Q(x_i) = j$. In order for $Q(\hat{x}_i) = j$, we must have $|\hat{x}_i - q^*_j| \le h/2$. Let us rewrite $|\hat{x}_i - q^*_j|$ as

$$|\hat{x}_i - q^*_j| = |(\hat{x}_i - x_i) + (x_i - q^*_j)| \le |\hat{x}_i - x_i| + |x_i - q^*_j|. \quad (26)$$

Considering the first term only we have

$$\begin{aligned} &|\hat{x}_i - x_i| \\ &\quad = |(\hat{a}_1 x'_i + \hat{a}_2 y'_i + \cdots + \hat{a}_k)) - (a_1 x'_i + a_2 y'_i + \cdots + a_k)| \\ &\quad = |(\hat{a}_1 - a_1)x'_i + (\hat{a}_2 - a_2)y'_i + \cdots + (\hat{a}_k - a_k)| \\ &\quad \le |\hat{a}_1 - a_1||x'_i| + |\hat{a}_2 - a_2||y'_i| + \cdots + |\hat{a}_k - a_k| \\ &\quad \le \frac{s_{a_1}}{2}|x'_i| + \frac{s_{a_2}}{2}|y'_i| + \cdots + \frac{s_{a_k}}{2}. \qquad (27) \end{aligned}$$

Taking into consideration that $|\hat{x}_i - q^*_j| \le h/2$, we have from (26) and (27)

$$|\hat{x}_i - q^*_j| \le \frac{s_{a_1}}{2}|x'_i| + \frac{s_{a_2}}{2}|y'_i| + \cdots + \frac{s_{a_k}}{2} + h/2. \quad (28)$$

The above inequality implies that in order to ensure that the correct hash bin will not be missed during recognition, we must account for the sampling error during preprocessing. In particular, each time an entry is stored in a hash bin, we must also store a pointer to this entry in the hash bins located around a neighborhood of the indexed hash bin. The size of the neighborhood can be computed using (28) and depends on the sampling steps (s_{a_j}'s) and the size of the hash table. Equation (27) allows us to find the neighborhood for each of the dimensions of the hash table. Then, we need to consider the union of neighborhoods over all the dimensions. It should be mentioned that our experimental results have shown that the upper bound given by (28) is not tight and much smaller bounds (for example, 1/2 of it) have worked well in our experiments.

Let us now consider the effect of sensor noise. For this, we assume that there is an uncertainty in the location of the model points which is at most n_e pixels. This means that the true image point must lie within n_e pixels of the actual (noisy) image point. Taken into consideration that each image is mapped (scaled) to the unit square before recognition, the maximum distance corresponding to the scaled image will be $\tilde{e} = n_e/N_i$, assuming images of size $N_i \times N_i$. The question is whether the noisy coordinates will access the correct hash bin, that is, whether $index(x_1, x_2, \ldots, x_G) = index(\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_G)$, where $\tilde{x}_i = x_i + \tilde{e}$. In order for this to be true, we must have $Q(x_i) = Q(\tilde{x}_i)$. Let us consider the difference $|\tilde{x}_i - q^*_j|$:

$$|\tilde{x}_i - q^*_j| = |(x_i - q^*_j) + \tilde{e}| \le |x_i - q^*_j| + \tilde{e} \le h/2 + \tilde{e}.$$

To account for noise, we need to follow a similar procedure. In particular, every time an entry is made in the table during preprocessing, we must also store a pointer to this entry in the

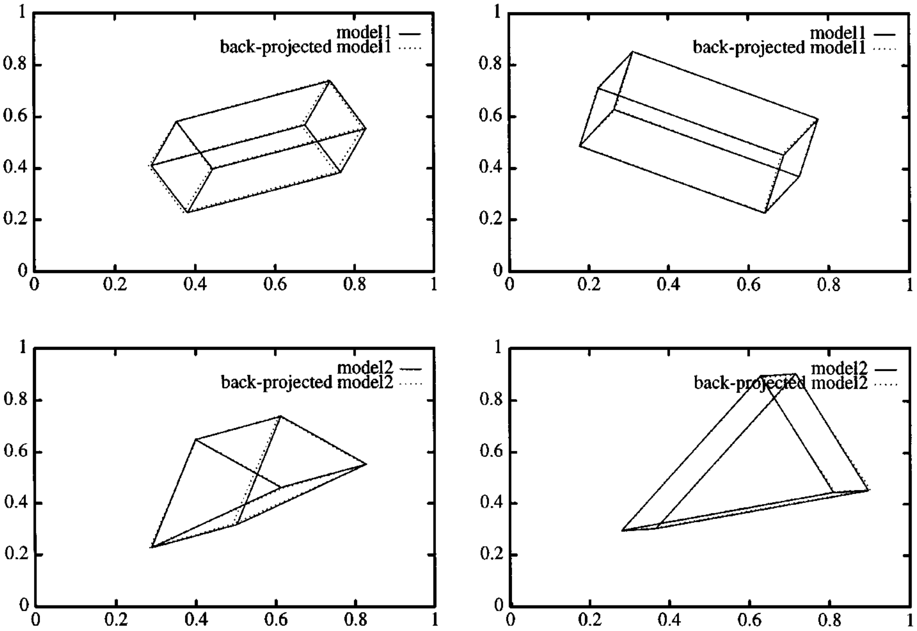

FIG. 9. Some artificial test views.

TABLE 3
Actual and Predicted Parameters (Artificial Data)

	Parameters	
	Actual parameters (Figure 9(a))	Predicted parameters (Figure 9(a))
a_1, a_2, a_3, a_4	-0.08248 -0.15374 0.09463 0.55224	-0.08222 -0.15323 0.09412 0.55223
b_1, b_2, b_3, b_4	-0.05775 0.02241 0.13408 0.48342	-0.05732 0.02222 0.13413 0.48313
	Actual parameters (Figure 9(b))	Predicted parameters (Figure 9(b))
a_1, a_2, a_3, a_4	-0.01866 -0.14224 0.00280 0.46931	-0.01863 -0.14223 0.00280 0.46932
b_1, b_2, b_3, b_4	0.07291 0.04679 0.19096 0.44305	0.07288 0.04675 0.19093 0.44299
	Actual parameters (Figure 9(c))	Predicted parameters (Figure 9(c))
a_1, a_2, a_3, a_4	-0.06960 -0.13060 0.08588 0.53691	-0.07512 -0.13366 0.08039 0.53807
b_1, b_2, b_3, b_4	-0.06665 0.02818 0.16177 0.49026	-0.06666 0.02814 0.16176 0.49025
	Actual parameters (Figure 9(d))	Predicted parameters (Figure 9(d))
a_1, a_2, a_3, a_4	-0.14990 0.08872 0.14231 0.61792	-0.14990 0.08868 0.14231 0.61792
b_1, b_2, b_3, b_4	-0.21708 -0.12707 0.01205 0.55004	-0.21713 -0.12714 0.01200 0.54999

hash bins located in a neighborhood around the indexed hash bin. It should be mentioned, however, that some of these pointers might have been already stored in the appropriate hash bins during the previous step which accounts for sampling error.

11. RECOGNIZING 3D OBJECTS

In this section, we demonstrate the proposed approach using both artificial and real 3D objects. The group size used in the experiments reported here is $G = 5$. A five-dimensional hash table of size 10 × 10 × 10 × 10 × 10 was utilized ($h = 1/10$). The step size used to sample the ranges of the parameters was $s_{a_j} = 0.05$. First, we performed a number of experiments using the artificial objects shown in Fig. 2. For each object, we generated a number of test views by choosing the parameters of the algebraic functions randomly. The test views were normalized so that their x and y coordinates were in the interval [0, 1]. This was performed by choosing a random subsquare within the unit square and by mapping the square enclosing the view (defined by its minimum and maximum x and y coordinates) to the randomly chosen subsquare. We also added some random noise in the location of the points to simulate sensor noise. Figure 9 shows some of the test views considered (solid line). In all cases, recognition was successful and the parameters of the algebraic functions were recovered very accurately as Table 3 illustrates. Figure 9 shows the predicted models (dashed line) back-projected onto the test views. The number of hypotheses verified in each case is shown in Table 4. To demonstrate the significance of the hypothesis evaluation procedure discussed in Section 9, Table 4 shows the number of hypotheses verified using (third column) and without using (second column) the five conditions mentioned in Section 9.

Next, we performed a number of experiments using the real 3D objects shown in Figs. 10a–10f. For each object, we considered a particular aspect and we captured two different pictures of the object (reference views). Both reference views have many features in common; however, the first view is different from the second in that the object has undergone translation and rotation. Next, we applied a corner and junction detector [33] in order to extract the interest points of the views. Figures 10g–10l show the common interest points considered in each case (the lines connecting the corners have been added to enable visualization).

Then, the reference views were preconditioned and the range of values for the parameters of the algebraic functions were computed. Table 5 shows the ranges computed for each object. Table 6 shows the number of well-conditioned groups chosen, the number of points represented in the groups, and the number of views $\tilde{N}_V$ considered during preprocessing.

Some of the scenes used in our recognition experiments are shown in Fig. 11. First, the interest points were detected using the same corner detector [33]. Nonimportant interest points were removed manually. Then, scene groups are chosen and used to access the hash table and establish hypotheses. In the current implementation, the scene groups are selected randomly during recognition. This is of course very inefficient. However, our main objective here is to demonstrate the usefulness of algebraic functions of views within indexing-based object recognition. There is no doubt that some kind of grouping will be very

TABLE 4
Verified Hypotheses

	Hypotheses	
	Without using (1)-(5)	Using (1)–(5)
Fig. 9(a)	12888	45
Fig. 9(b)	50907	65
Fig. 9(c)	107603	186
Fig. 9(d)	20257	61

TABLE 5
The Computed Ranges for 3D Objects (Preconditioned Views)

	Ranges of values			
	range of a1	range of a2	range of a3	range of a4
model1	[−0.41933 0.41933]	[−0.36234 0.36234]	[−0.42926 0.42926]	[0.0 1.0]
model2	[−0.44177 0.44177]	[−0.45138 0.45138]	[−0.43368 0.43368]	[0.0 1.0]
model3	[−0.42321 0.42321]	[−0.41114 0.41114]	[−0.37975 0.37975]	[0.0 1.0]

crucial to the performance of our method or to the performance of indexing-based approaches in general. In all cases, the models present in the scene were recognized correctly. Figure 11 shows the recognized models back-projected on the test scenes. Also, Table 6 shows the actual and predicted parameters of the algebraic functions in each case.

12. CONCLUSIONS

In this paper, we proposed a new approach for indexing-based object recognition using algebraic functions of views. The proposed approach has a number of advantages. First, it requires a small number of reference views. Most importantly, recognition does not depend on the similarity between novel and reference views. Second, verification becomes simpler. This is because candidate models can now be back-projected onto the scene by applying a linear combination on a small number of reference views of the models. Finally, the approach is more general and extendible. This is because algebraic functions of views exist over a wide range of transformations and projections.

To understand more clearly the strengths and weaknesses of the proposed approach, we discuss next a number of important issues. The first issue has to do with the camera model being used in this work. From our discussion in Section 2, the camera model being used here is based on the assumption of orthographic projection which is only an approximation of perspective projection. Although we have not performed a careful analysis on the sensitivity of our method to the assumption of orthographic projection, we believe that the performance of the method will degrade gradually as soon as more and more

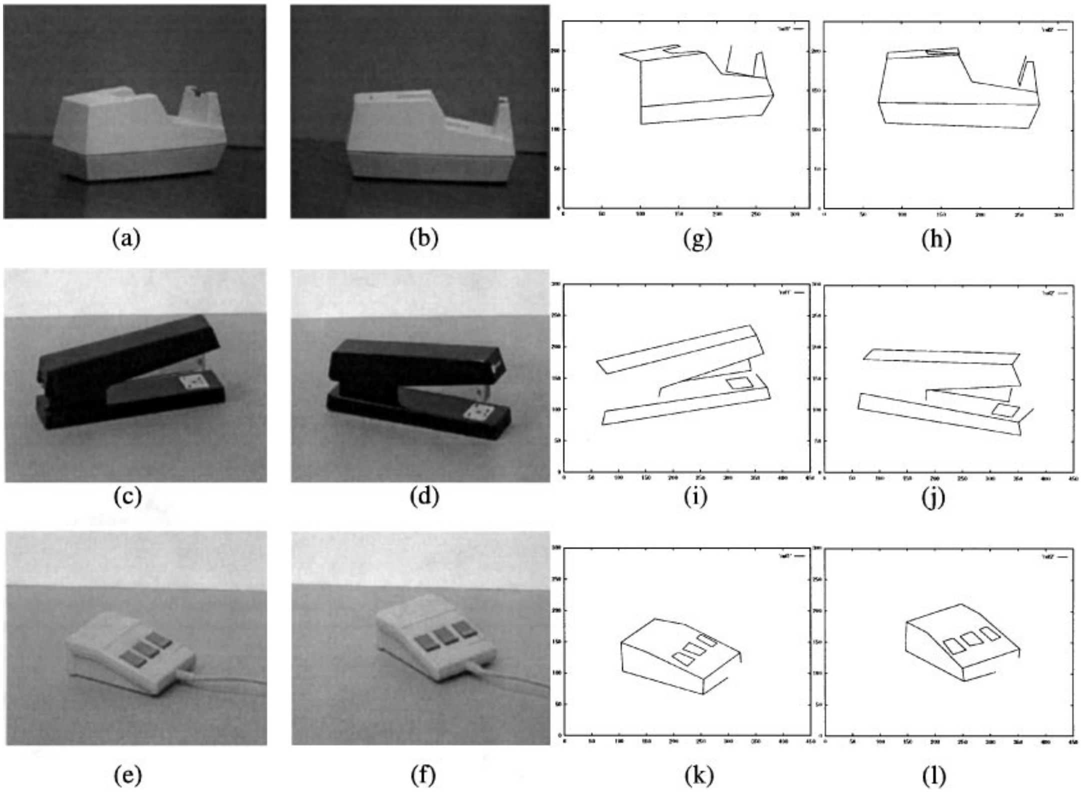

FIG. 10. Real model objects.

TABLE 6
Well-Conditioned Groups and Valid Transformed Views

	Example data					
	N_G	$\tilde{N}_G$	N_V	$\tilde{N}_V$	$\tilde{N}_V\tilde{N}_G$	points represented
model1	11628	16	102060	3991	63856	16/19
model2	11628	16	136458	3344	53504	17/19
model3	26334	14	102816	2851	39914	20/22

perspective distortions are introduced. In fact, some comments on the effect of perspectivity can be found in [16] (page 1002). It is reported in [16] that the effect of perspectivity appears to be quite limited. Specifically, the linear combination scheme was tried to objects with ratio of distance-to-camera to object-size down to 4 : 1 with only minor effects on the results.

Another issue is the issue of self-occlusion. Indeed, the combination of views method assumes that the objects are transparent [17–19]. To deal with self-occlusion, the hash table must be built using reference views from different viewpoints. However, it should be emphasized that this is not the same as using reference view from every possible viewpoint [5]. The key issue in using algebraic functions of views is that correct recognition can be established as long as there are scene groups which are contained in at least two reference views. In other words, recognition does not depend on the similarity between novel and reference views. As a result, a small number of reference views should be sufficient. In this paper, we have demonstrated the proposed approach using reference views associated with a specific aspect of the model objects only. In other words, we have made sure that the views to be recognized contain common groups of points with the reference views. The question of course is how to choose a smaller number of reference views which allow viewpoint-independent recognition. One idea is to capture a large number of reference views and then apply some kind of an elimination procedure. This problem is by no means a simple one and more effort is required to deal with it.

Finally, it is important to consider the issue of generating realistic views both during preprocessing and recognition. By realistic view we mean a view that can be obtained using a practical camera-object setting. In the current implementation, every transformed view is considered during preprocessing as long as it is valid, that is, as long as it lies within the unit square. Disregarding views which are not realistic has two advantages: first of all, space will be saved during preprocessing and, second, time will be saved during recognition. Although we are not dealing

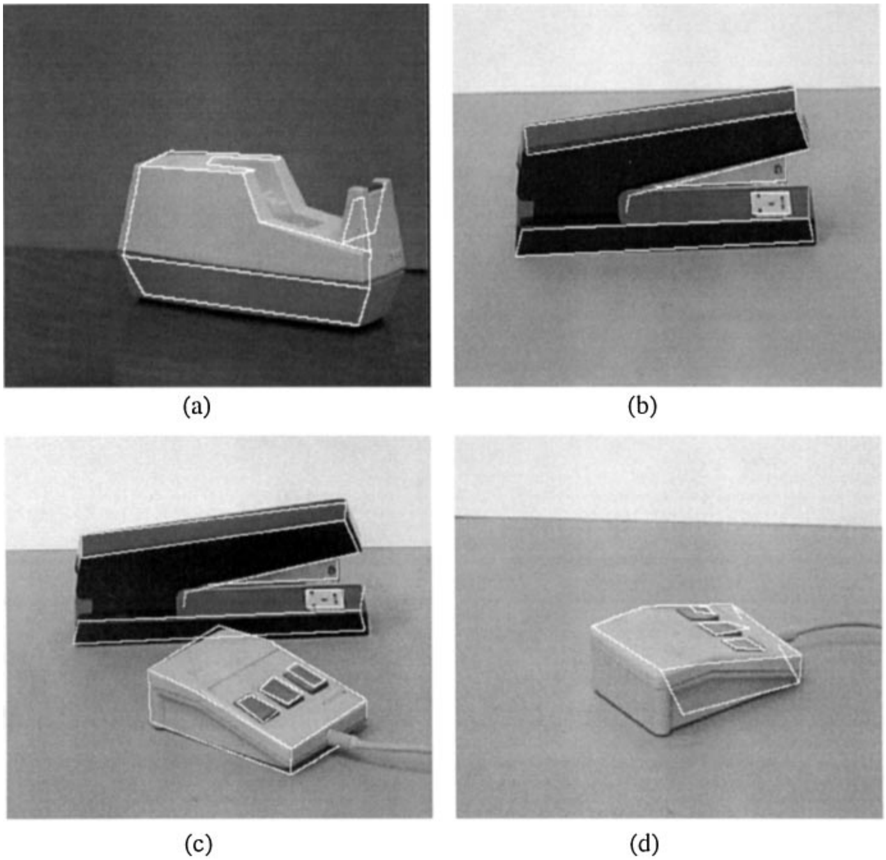

FIG. 11. Real scenes and recognition results.

TABLE 7
Actual and Predicted Parameters (Real Data)

	Parameters	
	Actual parameters (Figure 11(a))	Predicted parameters (Figure 11(a))
a_1, a_2, a_3, a_4	0.03704 0.19696 0.04488 0.63449	0.03700 0.19692 0.04485 0.63446
b_1, b_2, b_3, b_4	−0.12358 0.05752 0.01046 0.53638	−0.12353 0.05757 0.01051 0.53644
	Actual parameters (Figure 11(b))	Predicted parameters (Figure 11(b))
a_1, a_2, a_3, a_4	−0.05899 0.25588 0.00348 0.60012	−0.05899 0.25586 0.00345 0.60009
b_1, b_2, b_3, b_4	0.11146 0.00472 0.00298 0.47189	0.11190 0.00550 0.00161 0.47198
	Actual parameters (Figure 11(c))	Predicted parameters (Figure 11(c))
a_1, a_2, a_3, a_4	−0.05758 0.25394 0.00120 0.60300	−0.05636 0.25418 −0.00063 0.60274
b_1, b_2, b_3, b_4	0.09407 0.00413 0.00021 0.38442	0.09410 0.00416 0.00025 0.38447
	Actual parameters (Figure 11(c))	Predicted parameters (Figure 11(c))
a_1, a_2, a_3, a_4	0.01682 0.13225 −0.00602 0.62491	0.01682 0.13228 −0.00601 0.62492
b_1, b_2, b_3, b_4	−0.08168 0.03142 0.000150 0.68023	−0.08168 0.03146 0.00018 0.68026
	Actual parameters (Figure 11(d))	Predicted parameters (Figure 11(d))
a_1, a_2, a_3, a_4	−0.08481 0.06358 −0.03732 0.61571	−0.08480 0.06357 −0.03733 0.61572
b_1, b_2, b_3, b_4	−0.05666 −0.04054 0.01670 0.52448	−0.05660 −0.04049 0.01673 0.52454

with this issue here, we believe that one way to deal with this problem is by imposing certain constraints on the parameters of the algebraic functions. Note that since we assume general 3D linear transformations, these constraints might have to be object specific.

For future research, we plan to extend the proposed approach to the case of perspective projection. For this, we plan to use the algebraic functions proposed in [20, 21]. In specific, Shashua [20, 21] has shown that perspective views of an object can be expressed as a nonlinear combination of two orthographic views of the object. The extension will be carried out along the lines of the current approach. There are some important differences between the orthographic and perspective case. The most important is that the algebraic functions of views involve eight parameters in the case of perspective projection. This means that the space of transformed views will contain many more views in this case. In this case, it will be of fundamental importance to consider realistic views only. Another approach might be to find new algebraic functions involving more views but less parameters. It might be also worth experimenting with the algebraic functions proposed in the case of paraperspective projection [13]. In this case, the algebraic functions look essentialy the same as those in the orthographic case; however, the parameters must satisfy now certain contraints. Also, different neural network models must be used for the prediction of the parameters of the algebraic functions. In the case of orthographic projection, one-layer models were used since the combination of views is linear in this case. However, in the case of perspective projection, two-layer models must be used since the combination of views is nonlinear.

APPENDIX A

We have shown in Section 5 that $W_C = \lambda W_P^{-1}$. In this case, C has the form $\lambda V_P W_P^{-1} V_C^T$. It is important now to choose λ in a way such that the resulting transformation matrix C is a valid transformation matrix. Since the last column of C is equal to $[0\ \ 0\ \ 0\ \ 1]^T$, the following equation should be satisfied: $V_P W_C (v_4^C)^T = [0\ \ 0\ \ 0\ \ 1]^T$. We proceed by splitting the above problem into two subproblems:

$$W_C \left(v_k^C\right)^T = z, \tag{16}$$

and

$$V_P z = [0 \quad 0 \quad 0 \quad 1]^T. \tag{17}$$

V_P is known from the SVD analysis of P. The idea is to solve for z first (Eq. (17)) and then solve for $(v_k^C)^T$ (Eq. (16)). In solving (16), we need to consider an additional constraint: the magnitude of the solution vector $(v_k^C)^T$ must be equal to 1:

$$\left(v_{k1}^C\right)^2 + \left(v_{k2}^C\right)^2 + \cdots + \left(v_{kk}^C\right)^2 = 1. \tag{18}$$

This is a direct consequence of the orthonormality of V_C. Assuming that the elements of W_C are w_{ii}^C, $i = 1, 2, \ldots, k$, the solutions of (16) are

$$v_{k1}^C = \frac{z_1}{w_{11}^C}, v_{k2}^C = \frac{z_2}{w_{22}^C}, \ldots, v_{kk}^C = \frac{z_k}{w_{kk}^C}. \tag{19}$$

Substituting these expressions into (18) we have

$$\left(\frac{z_1}{w_{11}^C}\right)^2 + \left(\frac{z_2}{w_{22}^C}\right)^2 + \cdots + \left(\frac{z_k}{w_{kk}^C}\right)^2 = 1$$

or

$$\left(z_1 \prod_{i\neq 1} w_{ii}^C\right)^2 + \left(z_2 \prod_{i\neq 2} w_{ii}^C\right)^2 + \cdots + \left(z_k \prod_{i\neq k} w_{ii}^C\right)^2 = \left(\prod_{i=1}^{k} w_{ii}^C\right)^2. \tag{20}$$

Thus, the singular values of C must satisfy (20). From (15), $w_{ii}^C = \lambda / w_{ii}^P$, $i = 1, 2, \ldots, k$. Substituting w_{ii}^C in (20) and solving for λ we have

$$\left(z_1 \prod_{i\neq 1} \lambda / w_{ii}^P\right)^2 + \left(z_2 \prod_{i\neq 2} \lambda / w_{ii}^P\right)^2 + \cdots + \left(z_k \prod_{i\neq k} \lambda / w_{ii}^P\right)^2 = \left(\lambda \Big/ \prod_{i=1}^{k} w_{ii}^P\right)^2$$

or

$$\lambda = \left(\prod_{i=1}^{k} w_{ii}^P\right) \times \sqrt{\left(z_1 \Big/ \prod_{i\neq 1} w_{ii}^P\right)^2 + \left(z_2 \Big/ \prod_{i\neq 2} w_{ii}^P\right)^2 + \cdots + \left(z_k \Big/ \prod_{i\neq k} w_{ii}^P\right)^2}, \tag{21}$$

where only the positive values of λ have been considered since the sign of λ affects the sign of the singular values which must be positive.

Next, the rest elements of V_C need to be determined. A simple way to determine them is by assuming that V_C is a matrix from a class of matrices which are known to be orthogonal. For example, we can assume that V_C is a Householder matrix [30]. A Householder matrix H is defined as

$$H = \begin{bmatrix} 1 - 2d_1^2 & -2d_1 d_2 & \cdots & -2d_1 d_k \\ -2d_2 d_1 & 1 - 2d_2^2 & \cdots & -2d_2 d_k \\ \cdots & \cdots & \cdots & \cdots \\ -2d_k d_1 & -2d_k d_2 & \cdots & 1 - 2d_k^2 \end{bmatrix}, \tag{22}$$

where d is a unit vector and d_j, $j = 1, 2, \ldots, k$, are its components. The elements of H are fully determined by the elements of d. The components of d can be determined by setting the elements of the last row of H equal to the elements of the last row of V_C, (i.e., v_k^C). In specific, d_j's can be determined as

$$1 - 2d_k^2 = v_{kk}^C \quad \text{or} \quad d_k = \pm\sqrt{\frac{1 - v_{kk}^C}{2}} \tag{23}$$

$$-2d_k d_{k-1} = v_{k(k-1)}^C \quad \text{or} \quad d_{k-1} = -\frac{v_{k(k-1)}^C}{2d_k} \tag{24}$$

$$\text{''}$$

$$\text{''}$$

$$\text{''}$$

$$-2d_k d_1 = v_{k1}^C \quad \text{or} \quad d_1 = -\frac{v_{k1}^C}{2d_k}. \tag{25}$$

Either the positive or negative d_k value can be used (the positive value has been considered here). It can be easily verified that the vector d, whose components are determined by (23)–(25), is a unit vector.

REFERENCES

1. R. Chin and C. Dyer, Model-based recognition in robot vision, *Computing Surveys* **18**(1), 1986, 67–108.
2. E. Grimson and T. Lozano-Perez, Localizing overlapping parts by searching the interpretation tree, *IEEE Pattern Anal. Mach. Intelligence* **9**(4), 1987, 469–482.
3. D. Huttenlocher and S. Ullman, Recognizing solid objects by alignment with an image, *Int. J. Computer Vision* **5**(2), 1990, 195–212.
4. Y. Lamdan, J. Schwartz, and H. Wolfson, Affine invariant model-based object recognition, *IEEE Trans. Robotics Automation* **6**(5), 1990, 578–589.
5. Y. Lamdan, J. Schwartz, and H. Wolfson, On recognition of 3D objects from 2D images, in *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1407–1413, 1988.
6. G. Bebis, M. Georgiopoulos, and N. da Vitoria Lobo, Learning geometric hashing functions for model based object recognition, in *Fifth International Conference on Computer Vision (ICCV-95), June 1995*, pp. 543–548, M.I.T., Cambridge, MA.
7. T. Breuel, Indexing of visual recognition from large model base, *AI Memo 1108*, MIT, Artificial Intelligence Lab., August 1990.
8. C. Rothwell, A. Zisserman, D. Forsyth, and J. Mundy, Planar object recognition using projective shape representation, *Int. J. Computer Vision* **16**, 1995, 57–99.
9. A. Califano and R. Mohan, Multidimensional indexing for recognizing visual shapes, *IEEE Pattern Anal. Mach. Intelligence* **16**(4), 1994, 373–392.
10. D. Weinshall, Model-based invariants for 3D vision, *Int. J. Computer Vision* **10**(1), 1993, 27–42.
11. C. Olsen, Probabilistic indexing for object recognition, *IEEE Pattern Anal. Mach. Intelligence* **17**(5), 1995, 518–522.
12. D. Clemens and D. Jacobs, Space and time bounds on indexing 3D models from 2D images, *IEEE Pattern Anal. Mach. Intelligence* **13**(10), 1991, 1007–1017.
13. D. Jacobs, Space efficient 3D model indexing, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 439–444, 1992.
14. D. Jacobs, Matching 3D models to 2D images, *Int. J. Computer Vision*, 1997.
15. J. Ben-Arie, The probabilistic peaking effect of viewed angles and distances with application to 3D object recognition, *IEEE Pattern Anal. Mach. Intelligence* **12**(8), 1990, 760–774.

16. S. Ullman and R. Basri, Recognition by linear combination of models, *IEEE Pattern Anal. Mach. Intelligence* **13**(10), 1991, 992–1006.

17. R. Basri, Recognition by combinations of model views: alignment and invariance, in *Applications of Invariance in Computer Vision* (J. Mundy, A. Zisserman, and D. Forsyth, Eds.), pp. 435–450, 1994.

18. R. Basri and S. Ullman, The alignment of objects with smooth surfaces, *Computer Vision, Graphics, Image Processing: Image Understand.* **57**(3), 1993, 331–345.

19. R. Basri, Paraperspective $\equiv$ affine, Technical report, Dept. of Applied Mathematics, The Weizmann Institute of Science.

20. A. Shashua, Algebraic functions for recognition, *IEEE Trans. Pattern Anal. Mach. Intelligence* **17**(8), 1995, 779–789.

21. A. Shashua, Trilinearity in visual recognition by alignment, in *Third European Conference on Computer Vision (ECCV'94)*, pp. 479–484, 1994.

22. O. Faugeras and L. Robert, What can two images tell us about a third one?, in *Third European Conference on Computer Vision (ECCV'94)*, pp. 485–492, 1994.

23. A. Sugimoto and K. Murota, 3-D object recognition by combination of perspective images, *SPIE Image Modeling* **1904**, 1993.

24. W. Press *et al.*, *Numerical Recipes in C: The Art of Scientific Programming*, Cambridge University Press, Cambridge, UK, 1990.

25. R. Moore, *Interval Analysis*, Prentice-Hall, New York, 1966.

26. D. Jacobs, *Recognizing 3D Objects Using 2D Images*, Ph.D thesis, Dept. of Electrical Engineering and Computer Science, MIT, 1992.

27. A. Neumaier, *Interval Methods for Systems of Equations*, Cambridge Univ. Press, Cambridge, UK, 1990.

28. E. Hansen and R. Smith, Interval arithmetic in matrix computations: Part II, *SIAM J. Numerical Anal.* **4**(1), 1967.

29. G. Forsythe, M. Malcolm, and C. Moler, *Computer Methods for Mathematical Computations*, Chap. 9, Prentice Hall, New York, 1977.

30. G. Golub and C. Van Loan, *Matrix Computations*, Johns Hopkins Univ. Press, Baltimore, 1989.

31. G. Bebis, M. Georgiopoulos, N. da Vitoria Lobo, and M. Shah, Learning affine transformations of the plane for model based object recognition, in *13th International Conference on Pattern Recognition (ICPR-96)*, Vienna, Austria, August 1996.

32. W. Grimson, D. Huttenlocher, and D. Jacobs, A study of affine matching with bounded sensor error, *Int. J. Computer Vision* **13**(1), 1994, 7–32.

33. S. Smith and J. Brady, SUSAN: A new approach to low level image processing, DRA technical report TR95SMS1, Department of Engineering Science, Oxford University, 1995.