PERGAMON

Pattern Recognition 32 (1999) 1175–1201

PATTERN RECOGNITION

THE JOURNAL OF THE PATTERN RECOGNITION SOCIETY

Curvature scale-space-driven object recognition with an indexing scheme based on artificial neural networks

George Bebis[a,*], George Papadourakis[b], Stelios Orphanoudakis[c]

[a] *Department of Computer Science, University of Nevada, Reno, NV 89557, USA*
[b] *Department of Science, Technological Educational Institute, Heraklion, Crete, Greece*
[c] *Department of Computer Science, University of Crete, Heraklion, Crete, Greece*

Received 20 June 1997; received in revised form 29 October 1998; accepted 29 October 1998

Abstract

This paper addresses the problem of recognizing real flat objects from two-dimensional images. In particular, a new object recognition technique which performs under occlusion and geometric transformations is presented. The method has mainly been designed to handle complex objects and incorporates two main ideas. First, matching operates hierarchically, guided by a curvature scale space segmentation scheme, and takes advantage of important object features, that is, features which distinguish an object from other objects. This is different from many classical approaches which employ a rather large number of very local features. Second, the model database is built by using artificial neural networks (ANNs). This is also different from traditional approaches where classical indexing schemes, such as hashing, are utilized to organize and search the model database. Important object features are obtained in two steps: first, by segmenting the object boundary at multiple scales using its resampled curvature scale space (RCSS) and second, by concentrating at each scale separately, searching for groups of segments which distinguish an object from other objects. These groups of segments are then used to build a model database which stores associations between segments and models. The model database is implemented using a set of ANNs which provide the essential mechanism not only for establishing correct associations between groups of segments and models but also for enabling efficient searching and robust retrieval. The method has been tested using both artificial and real data illustrating good performance.

Keywords: Object recognition; Curvature scale space; Artificial neural networks; Centroidal profile; Moments

1. Introduction

Object recognition is an essential part of any high-level robotic system. During the last two decades, there has been a variety of approaches to tackle the problem of object recognition. The most successful approach is probably in the context of *model-based* object recognition [1, 2], where the environment is rather constrained and recognition relies upon the existence of a set of predefined object models. A successful model-based object recognition system must be able to handle complex image scenes. This includes recognizing various types of objects in

* Corresponding author. Tel.: +1-702-784-6463; fax: +1-702-784-1877; e-mail: bebis@cs.unr.edu

0031-3203/99/$20.00
PII: S0031-3203(98)00159-9

cluttered environments assuming noise and occlusion and determining their position and orientation. The majority of model-based object recognition systems have two separate phases of operation: *training* and *recognition*. During the training phase, a model database is built by establishing proper associations between features and models. During the recognition phase, scene features are used to retrieve appropriate associations stored in the model database. Usually, there are two steps in the matching process: the *hypothesis generation* step and the *hypothesis verification* step. In the hypothesis generation step, the identities of one or more objects present in the scene are hypothesized. In the hypothesis verification step, tests are performed to check whether a given hypothesis is acceptable or not, by seeking further supporting evidence.

Two major approaches have emerged in order to deal with the issue of efficiently generating hypotheses during matching: *search-based* and *indexing-based*. Search-based approaches proceed by first extracting a collection of image features. Then, a correspondence between these features and a set of model features is hypothesized. The position and orientation of the model object are determined by this hypothesis. Two well known paradigms from this category are the alignment technique [3] and the interpretation tree [4]. Indexing is an alternative paradigm which tries to speedup searching by trading space. In the training phase, features which remain unchanged under geometric transformations (*invariants*) are extracted from each model. Then, information related to the models from which the invariants were extracted is stored into an index table, using the invariants as indexes. Efficient indexing schemes, such as hashing or k-d trees, are utilized in this stage [5–12]. In the recognition phase, groups of features are chosen from the scene and invariants based on these features are computed and used as indexes to the same index table. As a result, only these groups of model features that might feasible match the scene features whose invariants used to access the table are considered for verification.

During the last few years, much of the interest in indexing-based object recognition has been triggered by the development of a technique known as geometric hashing [5], although the fundamental ideas behind geometric hashing were known earlier [6–8]. Specifically, geometric hashing uses invariants based on points or lines. The idea is to describe the object in a new coordinate system which is invariant to geometric transformations. Point or line features, represented in this system, are used as indexes to a hash table where information about the new coordinate system and the models is stored. In [6], invariants based on boundary segments are used. First, the boundary of the object is decomposed into subsections and then, the footprints of these subsections (i.e., the first few coefficients of the Fourier expansion of a function denoting the angular direction of the boundary subsection) are computed and used as indexes to a hash table where information about the models is stored. In [7], invariants based on fixed length boundary segments which are common to several model objects are used along with information about the location and orientation of the segments. In another approach [9], consecutive segments are grouped together to form more powerful features, called super-segments. Each super-segment is encoded using information about the angles formed by the individual segments in the super-segment as well as information about the eccentricity of the super-segment. The encoded super-segments are then used as an index to a hash table where information about the super-segment's location, length and orientation is stored.

In this paper, a new indexing-based object recognition method is proposed. Some of its key characteristics are the following: First, emphasis is given on the detection and utilization of important object features, that is, features which can distinguish a model object from other model objects. These features are rich in information and small in number. This is opposed to many classical techniques where a large number of very local features is employed. Our approach for obtaining these features is by using a multi-scale segmentation scheme based on the resampled curvature scale space (RCSS) of the object contour [13]. Second, matching operates hierarchically, guided by the multi-scale segmentation scheme, and takes advantage of important object differences first. As a result, matching can be established faster and more accurately. This is different from classical approaches where a significant amount of time is spent on matching features with very little discriminative power. Very local features generate a large number of false hypotheses, which must be rejected by applying the time consuming verification step. Taking advantage of important object features that might be present in the scene yields less hypotheses and passes the discriminating power of the algorithm from the expensive verification step to the hypothesis generation step. Finally, the organization of the model database is based on artificial neural networks (ANNs) [14], which provide the essential mechanism not only for implementing the appropriate associations between features and models but also for enabling fast searching and robust retrieval, especially when noisy or distorted data is considered. This is also opposed to many classical approaches which use sensitive to noise and distortions indexing schemes.

The proposed method has been mainly designed to handle complex objects. It is assumed that the object are described by their boundaries which do not cross over themselves and have a number of inflection points. This means that it is not appropriate for objects whose contour can be described by a small number of parameters such as circles or rectangles. Objects appearing in the scene may have undergone similarity transformations,

that is, rotation, translation and scaling. This is the case where the viewing angle of the camera is the same both for the model and scene objects. Similarity invariants are used for storing and retrieving information to and from the model database. Extending the method to more general transformations such as affine transformations (weak perspective) is straightforward assuming that affine invariants are used.

The organization of the paper is as follows: Section 2 discusses the motivations for developing the proposed approach and Section 3 presents a brief overview. The curvature scale space approach and a description of the hierarchic segmentation scheme are given in Section 4. The implementation of the model database using ANNs is presented in Section 5. The recognition phase and the verification procedure are presented in Section 6. Experiments and system's performance are presented in Section 7. Finally, our conclusions are provided in Section 8.

2. Motivations

An important issue in the implementation of a model-based object recognition system is how to extract important object features and how to utilize them efficiently during matching. A common approach is to split the object contour into a number of segments and then use them for matching [7–12]. However, because a large number of segments is usually obtained and because individual segments are usually too local to be appropriate for matching, recognition can become quite slow. Several methods [9–12] try to alleviate this problem by grouping together a number of adjacent segments in order to create more descriptive segments. This approach, however, requires the a priori choice of a parameter which determines the number of segments to be grouped together, a parameter which is rather data dependent. Our approach for obtaining descriptive segments is based on the use of a multi-scale segmentation scheme and does not require such a parameter. The multi-scale segmentation scheme yields segments extracted from various scale levels, localized down to the original scale. In this way, the descriptive power of each segment is very naturally related to the scale it is located.

Segments located at relatively high scales are usually very descriptive and distinct. This makes them quite attractive for matching. Moreover, these segments are usually very long, which makes them more effective for localizing the objects in the scene more accurate. Unfortunately, their main drawback is that they are very sensitive to occlusions. On the other hand, segments detected at lower scales are less descriptive and distinct, but they are more resistant to occlusions since they are usually shorter. Thus, there is a tradeoff between high and low scale segments. The approach taken in this paper is to consider both kinds of segments by using an efficient hierarchic matching procedure, driven by the multi-scale segmentation scheme. The main idea is that when high scale segments are present in the scene, it is advantageous to detect and use them for matching because both recognition and localization can become faster and more accurately. However, if high scale segments are not available because of occlusions, lower scale segments must be considered. In general, the proposed matching procedure attempts first to match segments detected at higher scales; if this is not possible, matching gradually utilizes segments detected at lower scales.

Another important issue in the design of a model based object recognition system is the issue of organizing and searching the model database. Most of the current approaches use efficient indexing schemes in order to store and retrieve information to and from the model database. Hashing is a popular indexing scheme. In various applications, it is important for the hash function to distribute the data randomly over the hash table. In object recognition, however, it is important for the hash function to be proximity preserving, that is, to map similar data to close by locations in the hash table. Choosing an appropriate hash function is not easy and is data dependent. Also, hashing is quite effective in noise-free environments but its performance deteriorates in the presence of noise. In other words, it is very unlikely that the correct item will be retrieved in the presence of noisy keys, unless a noise tolerant scheme is incorporated in the retrieval procedure. Thus, it is desirable to consider alternative schemes which are not data dependent and demonstrate robustness to noise.

The multi-layer ANN poses a number of properties which make them an attractive alternative choice. A multi-layer ANN implements a nonlinear mapping of the from $u = G(x)$. The mapping function G is established during a training phase where the network learns to correctly associate input patterns x to output patterns u. The powerful capabilities of multi-layer ANNs to implement complex mappings are supported by some very significant theoretical results. The most important of these results states that single hidden layer feed-forward network with arbitrary sigmoid hidden layer activation functions can approximate arbitrarily well an arbitrary mapping from one finite-dimensional space to another [15]. This implies that feed-forward networks can approximate virtually any function of interest to any desired degree of accuracy, provided sufficiently many hidden units are available. In the context of indexing, G represents the hash function, x represents the information used to search the database, and u represents the retrieved information.

Implementing the model database using multi-layer ANNs requires the establishment of appropriate associations between features and models first. Then, the mapping (hash function) required to implement these associations can be found by training an ANN. Since the choice

of an appropriate hash function is problem dependent, a scheme which finds the hash function through learning is highly desirable. In addition, assuming that the hash keys are unique, the computed hash function will be collision free. ANNs offer fast information retrieval because they are able to examine multiple competing hypothesis simultaneously. Increasing the number of feature-model associations might increase training time but it will not have a serious impact on the retrieval time. It should be emphasized that ANNs are not to be considered as simple lookup tables. Actually, the most important property of ANNs is their ability to generalize: given noisy and distorted inputs, it is possible that the correct feature-model associations can still be retrieved. Proximity can be enforced by choosing a proximity-preserving representation scheme for the outputs of the ANNs.

3. An overview of the proposed method

Following the well defined framework of model-based object recognition, the proposed method operates in two different phases: *training* and *recognition*. During the training phase, the model database is built by training a number of properly structured ANNs to learn associations between important groups of segments and models. By important groups of segments we mean groups which characterize model objects uniquely. These groups are extracted during preprocessing, using the RCSS of the model contours [13]. In particular, the curvature of each object contour is computed at multiple scales and the curvature zero-crossing points at each scale are detected. Then, a hierarchy of contour segmentations is obtained by concentrating at each scale separately and breaking the original contour at points which correspond to the curvature zero-crossings detected at this scale (localized, however, down to the original scale). Each segment from the hierarchy is normalized in order to become invariant to similarity transformations. This is performed by computing the centroidal profile of each segment [16,17]. Then, we represent each segment by using four moments of its centroidal profile [18,19]. This representation is invariant to the starting point selection in the computation of the centroidal profile and is also economical and tolerant to noise [19]. Segments represented by the moments of their centroidal profile will be referred to as *encoded segments*.

The model database has been partitioned into two parts. The first part is implemented using a single ANN called, *segment classifier*. The goal of the segment classifier is to assign segments to appropriate segment classes. For reasons to be explained shortly, the outputs of the segment classifier (i.e., segment classes) have been encoded using the Gray code scheme [27]. The encoded representation of a segment class will be referred to as *encoded segment classification*. The second part of the model database is implemented using a number of hierarchically structured ANNs called, *object recognizers*. The object recognizers use the outputs of the segment classifier (i.e., encoded segment classifications) to recognize the model objects. The reason we have used gray coding to represent the outputs of the segment classifier is to increase the noise tolerance of the object recognizers. Each object recognizer uses combinations of encoded segment classifications to perform its task. These combinations correspond to groups of segments which characterize the model objects uniquely. Important groups of segments vary in the number of segments they contain and different object recognizers perform recognition using groups of different sizes. Specifically, there exists an object recognizer in the top of the hierarchy which performs recognition using groups of size one. At the next level, there exists another recognizer which considers groups of size two, and so on.

The important groups of segments are extracted from the hierarchies of contour segmentations using a *selective search* procedure. This procedure returns important groups of segments, detected at various levels of the hierarchy, starting from a very high scale level and proceeding to the lower ones. The importance of a group of segments is related to the degree it can distinguish a model from other models. An important group of segments might consist of one, two, or more adjacent segments which may not be important by themselves, however, their identity along with their order in the group provide sufficient information for distinguishing a model from other models.

The recognition phase operates in a *hypothesis generation-verification* manner. Initially, a closed, maybe composite, contour is extracted from the scene and a hierarchy of segmentations is obtained using the multi-scale segmentation scheme. Then, segments are extracted from different scales and are encoded. Matching is performed from higher to lower scales and in a local to a global basis, that is, from matching one segment to matching groups of segments. Segments obtained at a specific level of the hierarchy of segmentations are forwarded first, one at a time, to the segment classifier. The output segment classifications are then forwarded to the object recognizers. Initially, the top level object recognizer attempts to perform recognition using groups of size one. If recognition cannot be established, pairs of segment classifications, corresponding to adjacent segments, are formed and forwarded to the object recognizer located at the next level. This process is repeated until either a correct recognition is performed or the lowest level in the hierarchy of contour segmentations is reached, without having recognized all the objects present in the scene successfully. In case of a possible match at some step of the matching procedure, verification takes place to evaluate it.

4. The preprocessing step

The role of the preprocessing step is twofold: to compute the hierarchy of segmentations for each model contour and to encode the obtained contour segments. The multi-scale segmentation scheme requires the computation of the RCSS for each model contour. Fig. 1 illustrates the various steps in the preprocessing step. Before we describe each step in detail, we review the curvature scale space approach.

4.1. Curvature scale space

The curvature scale space (CSS) approach was introduced in [20] as a shape representation for planar curves. Object boundaries are usually represented as planar curves that do not cross over themselves. Specifically, this method is based on finding points of inflection (i.e, curvature zero-crossings) on the curve at varying levels of detail. The curvature k of a planar curve, at a point on the curve, is defined as the instantaneous rate of change of the slope of the tangent at that point with respect to arc length, and it can be expressed as follows:

$$k(t) = \frac{\dot{x}(t)\,\ddot{y}(t) - \dot{y}(t)\,\ddot{x}(t)}{(\dot{x}(t)^2 + \dot{y}(t)^2)^{3/2}}, \tag{1}$$

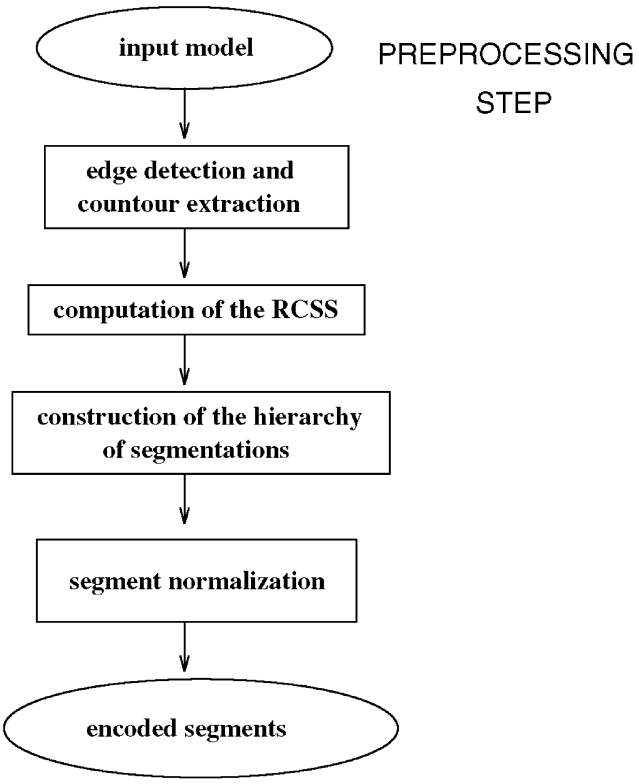

Fig. 1. The steps involved during preprocessing.

where $\dot{x}(t)$, $\ddot{x}(t)$, $\dot{y}(t)$, and $\ddot{y}(t)$ are the first and second derivatives of $x(t)$ and $y(t)$, respectively, and $(x(t), y(t))$ is the parametric representation of the curve.

In order to compute the curvature of the curve at varying levels of detail, $x(t)$ and $y(t)$ are convolved with a Gaussian function $g(t, \sigma)$. Defining $X(t, \sigma)$ and $Y(t, \sigma)$ as the convolution of $x(t)$ and $y(t)$ respectively with the Gaussian function, the smoothed curve curvature $k(t, \sigma)$ can be expressed as follows:

$$k(t, \sigma) = \frac{X_t(t, \sigma)\,Y_{tt}(t, \sigma) - X_{tt}(t, \sigma)\,Y_t(t, \sigma)}{(X_t(t, \sigma)^2 + Y_t(t, \sigma)^2)^{3/2}}, \tag{2}$$

where $X_t(t, \sigma)$ and $X_{tt}(t, \sigma)$ are defined as

$$X_t(t, \sigma) = x(t) * \frac{\partial g(t, \sigma)}{\partial t}, \tag{3}$$

$$X_{tt}(t, \sigma) = x(t) * \frac{\partial^2 g(t, \sigma)}{\partial t^2}. \tag{4}$$

$Y_t(t, \sigma)$ and $Y_{tt}(t, \sigma)$ are defined in a similar manner. The function defined implicitly by $k(t, \sigma) = 0$ is the CSS image of the curve.

To demonstrate the CSS approach, the object contour shown in Fig. 2a was convolved with a Gaussian filter, using successively doubled values of σ. Fig. 3a–h show some of the convolved curves and their corresponding curvature functions. The object's contour was divided into 256 equally sized subintervals for this task. The locations at which $k = 0$ are marked on each curve. The CSS representation of the object contour is shown in Fig. 4a.

The RCSS representation was introduced in [13], and was shown to be suitable when there is a high-intensity non-uniform noise or local shape differences exist. Also, it can be computed for open curves. In general, as a planar curve is convolved according to the process described earlier, the parameterization of its coordinate functions $x(t)$ and $y(t)$ does not change. In other words, the function which maps values of the parameter t of the original coordinate functions $x(t)$ and $y(t)$ to values of the parameter t of the smoothed coordinate functions $X(t, \sigma)$ and $Y(t, \sigma)$ is the identity function. Assuming that the mapping function is different from the identity function, the RCSS of the curve is the function defined implicitly by $k(T, \sigma) = 0$, where $T = T(t, \sigma)$, is a monotonic function of t. A simple way to compute the RCSS is given in [13]. First, a Gaussian filter based on a small value of the standard deviation is computed. Next, the curve is parameterized by the normalized arc length parameter and it is convolved with the filter. The resulting curve is then convolved again with the same filter. This process is repeated until the curve is convex and no longer has any curvature zero-crossing points. The curvature zero-crossings of each curve are marked in the RCSS image. It is important to note, however, that after the original curve

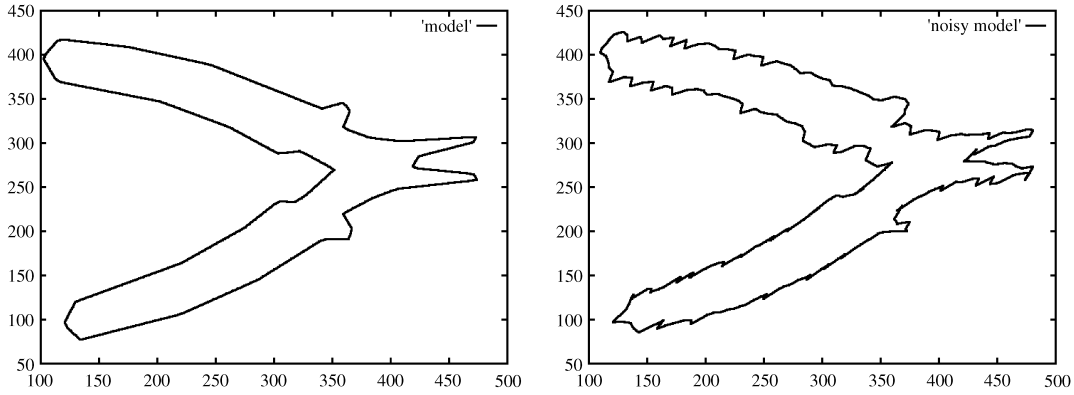

Fig. 2. A model object and the same object corrupted by noise. The procedure used to corrupt the object with noise is given in Section 7.4.

has been convolved with the Gaussian filter, the resulting curve is no longer parameterized by the normalized arc length parameter. Thus, the resulting curve must first be re-parameterized by the normalized arc length parameter before it is convolved again. Fig. 4b shows the RCSS of the object shown in Fig. 2a. To demonstrate the tolerance of the RCSS in presence of noise, we have considered the noisy object shown in Fig. 2b (see Section 7.4 for a description of the procedure used to generate the noisy object). Fig. 5a shows its CSS image while Fig. 5b shows its RCSS image. Comparing Figs. 4 and 5 we can clearly see the extra curvature zero-crossing that were introduced in Fig. 5 due to noise. Obviously, the RCSS image is less noisy than the CSS image.

4.2. The hierarchy of contour segmentations

The first step in the construction of the hierarchy of contour segmentations involves the computation of the RCSS image. Then, the hierarchy of segmentations is obtained by concentrating at each scale separately and breaking the original contour at points which correspond to the curvature zero-crossing points detected at this scale. It should be mentioned that curvature extrema can also be used in the segmentation of the object contours since the number of zero-crossings is usually small which limits the kind of objects that can be handled by our system. However, our decision to use curvature zero-crossings only has been based on the assumption that the model database consists of rather complex objects as discussed in the introduction. Segmenting the object contour using curvature zero-crossing points only, keeps the number of segments relatively small. Also, curvature zero-crossings are robust over different viewpoints [21], which implies that the proposed method can be extended to more general cases where the viewpoint is arbitrary (e.g., affine transformations).

Segmentation of the original contour using curvature zero-crossing points detected at high scales, requires the projection of zero-crossings down to the original scale (localization). This is because convolving a curve with an averaging filter such as the Gaussian has the effect of shrinking the size of the contour [13]. The amount of shrinking is directly related to the curvature of the contour and the degree of smoothing. As a result, it alters the location of curvature zero-crossing points, as we move from one scale to the other. In order to determine the location of a zero-crossing point detected at a higher scale down to a lower scale, scale space tracking is required. During scale space tracking, the location of higher scale curvature zero-crossing points is projected down to a lower scale. This is performed by following the location of the curvature zero-crossing points, from scale to scale, until the desired lower scale is reached [22]. Fig. 6 illustrates the hierarchical contour segmentation of the object contour shown in Fig. 2a. In this example, the contour has been segmented into 2, 4, and 6 segments.

The segmentation procedure can now be described as follows: Initially, the RCSS of the object contour is computed by convolving the object contour repeatedly, until no curvature zero-crossing points exist any more. Then, the coarsest apex points in the RCSS are detected. An apex point is the location where a pair of curvature zero-crossing point disappears, as we move from lower to higher scales. In other words, apex points represent the maxima of the RCSS contours. Each apex point introduces a new pair of zero-crossing points which can be used to further sub-split the object contour. To ensure that the endpoints of the segments are determined accurately, the zero-crossings are localized down to the original scale. Fig. 6a shows the segmentation of the object contour into two segments using the zero-crossings corresponding to the coarsest apex point. Fig. 6b shows the projected zero-crossing locations, down to the original scale. Fig. 6c shows the segmentation of the object contour into four segments, using the next coarsest apex point and Fig. 6d shows the localized segments. Finally, Fig. 6e shows the segmentation of the object contour into

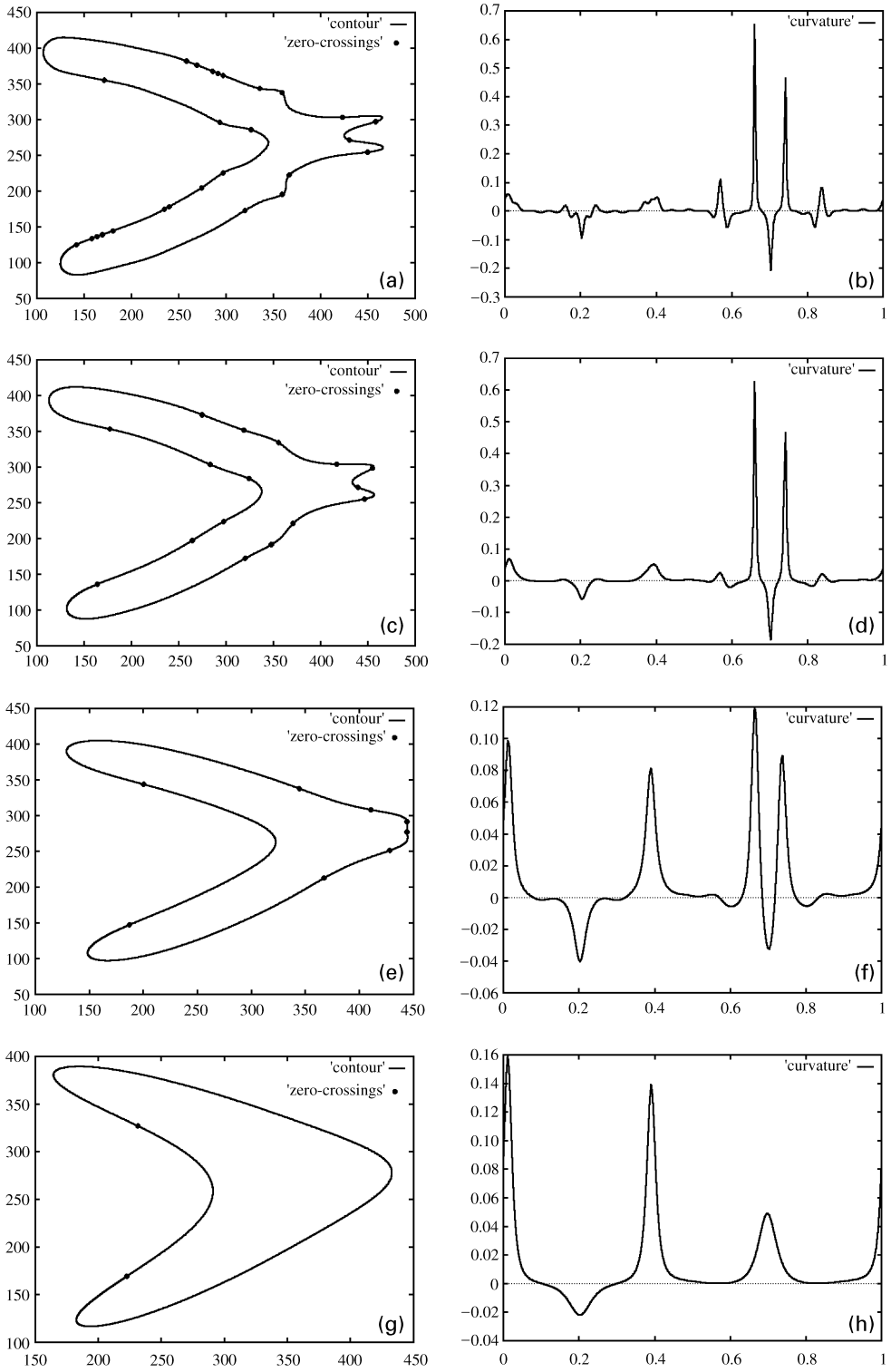

Fig. 3. The object contour and its curvature at various scale levels: (a), (b) $\sigma = 2$, (c), (d) $\sigma = 4$, (e), (f) $\sigma = 8$, (g), (h) $\sigma = 16$.

six segments, using the next apex point, and Fig. 6f shows the segmentation at the original scale.

A few practical issues need to be discussed at this point. First of all, the hierarchy of segmentations is built in a top-down fashion, that is, breaking the contour at the highest scale first, then breaking the contour at a lower scale, until the original scale is reached. In our implementation, however, we do not start from the highest possible scale and we do not reach the original scale either. This is because segments located at high scales are not very

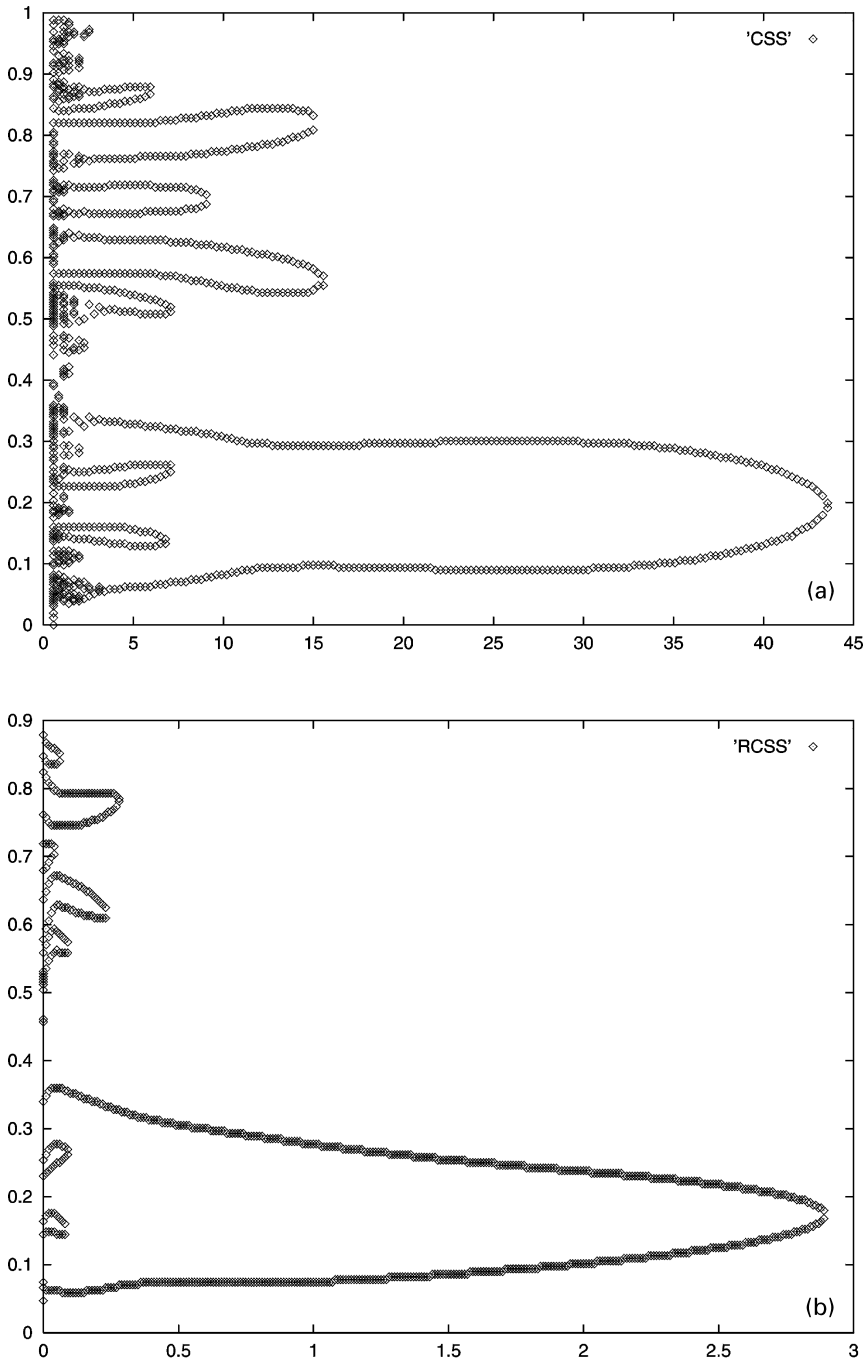

Fig. 4. (a) The curvature scale space, (b) the resampled curvature scale space.

reasonable (see Fig. 6b) while segments located at very low scales are large in number and have low descriptive power. Specifically, we start splitting the contour at a scale where no less than 4 zero-crossing points exist and we stop splitting at a scale where no more than 8 zero-crossing points exist. This means that the minimum number of segments we allow at a particular segmentation level is 4 while the maximum number is 8. Second, the localization of the zero-crossings is not performed at the original scale but a slight higher scale to provide noise tolerance. Finally, it should be expected that the segmentation of a contour remains unchanged over a range of scales. This is because no new curvature zero-crossing points are introduced in every scale. This means that there is no need to consider segmentations at these scales because they will be essentially the same given that zero-crossings are localized down to the same scale. Thus, the hierarchy of contour segmentations we compute includes

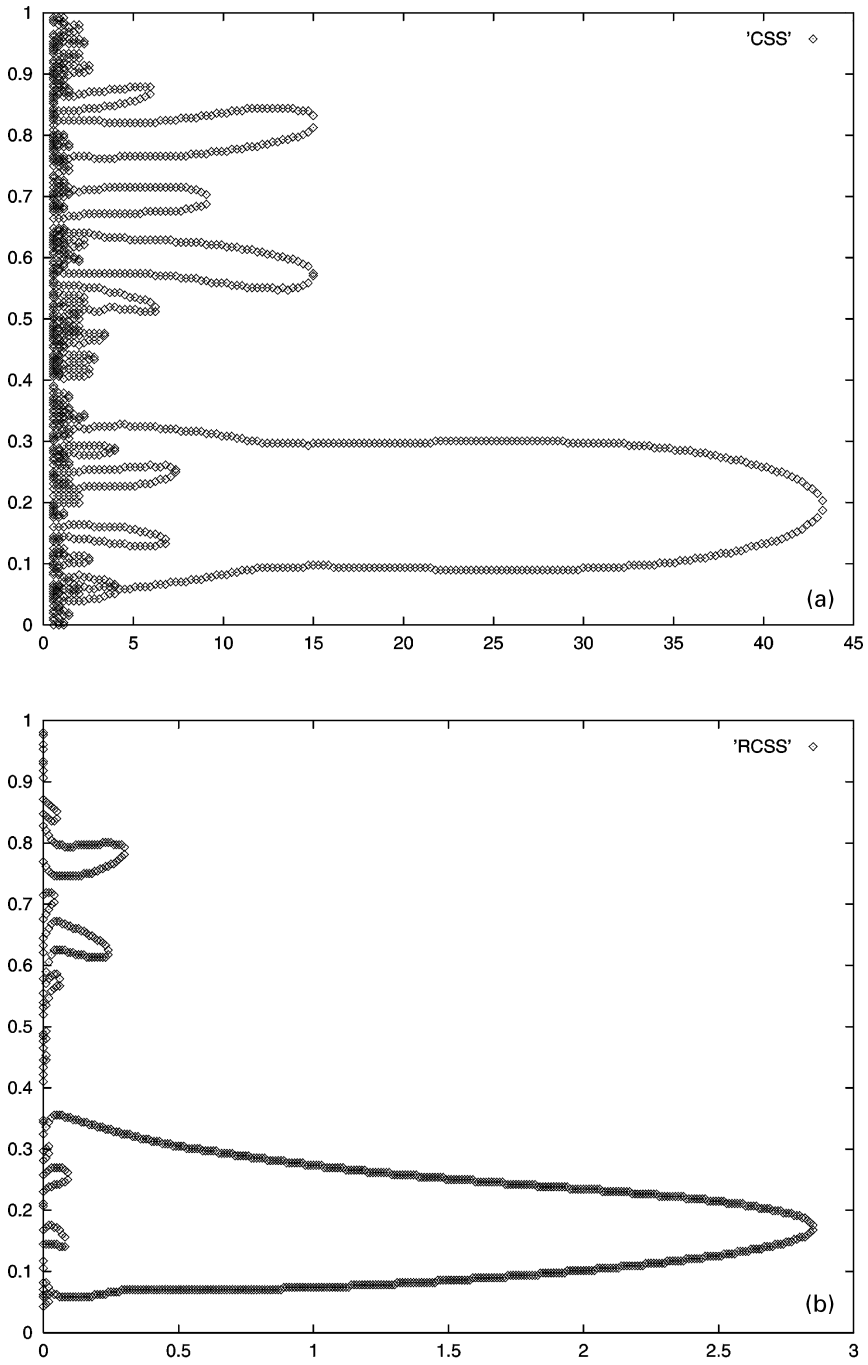

Fig. 5. (a) The noisy curvature scale space, (b) the noisy resampled curvature scale space. Note that more curvature zero-crossings have been introduced due to noise.

only segmentations corresponding to scales where new zero-crossing points are introduced.

4.3. Encoding the contour segments

The first step in the encoding of the segments is the computation of the centroidal profile of each segment. Then, each segment is represented by four moments of its centroidal profile. The centroidal profile is a well known 1-D representation of the object contour [16], which has been extensively used in the past and has also been successfully utilized in object recognition using ANNs [17]. It is characterized by an ordered sequence which represents the Euclidean distance from the digitized contour of the object to its centroid as a function of distance

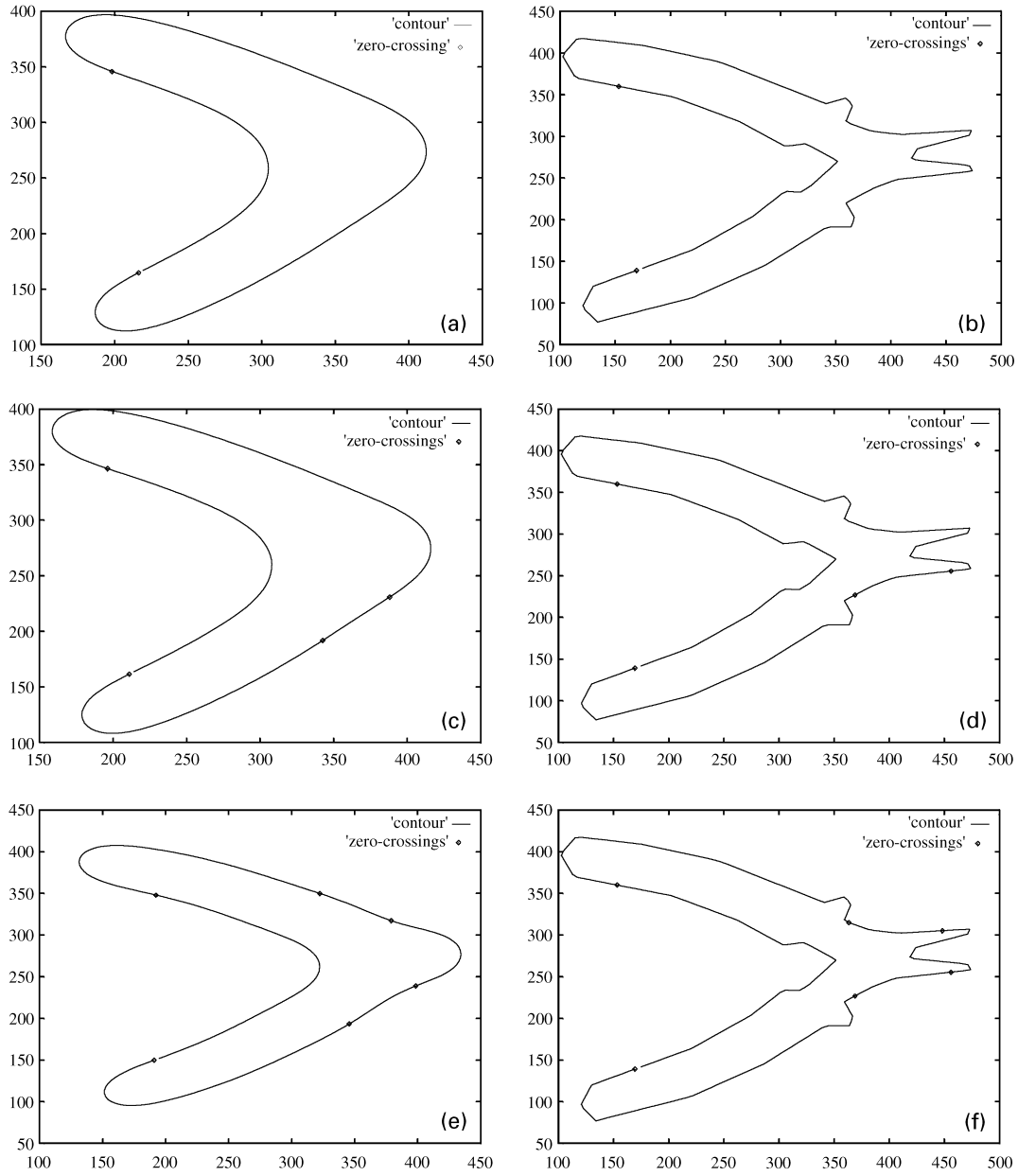

Fig. 6. Segmentation at a coarse scale (left column) and localization of the segments down to the original scale (right column).

along the contour. In a similar fashion, a segment's centroidal profile represents the distance between each contour point in the segment and the centroid of the region formed by connecting the end points of the segment by a straight line. Fig. 7 illustrates a simple contour segment and its centroidal profile.

Gupta and Srinath [18] have proposed the use of statistical moment functions, derived from the centroidal profile of the object contour, to represent the shape of the object. Here, we use statistical moment functions to represent the centroidal profile of segment. The low order-moments used in our study are the following:

(a) *normalized amplitude variation*:

$$\frac{(1/N)[\sum_{i=1}^{N}[d(i)-m_1]^2]^{1/2}}{m_1},$$

(b) *coefficient of skewness*:

$$\frac{(1/N)\sum_{i=1}^{N}[d(i)-m_1]^3]}{(1/N)[\sum_{i=1}^{N}[d(i)-m_1]^2]^{3/2}},$$

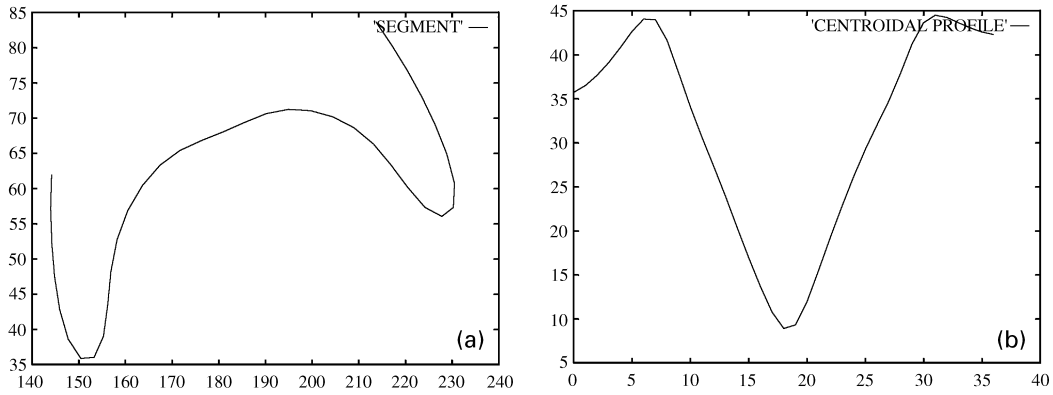

Fig. 7. (a) a segment from a model's contour (b) the centroidal profile of the segment.

(c) *coefficient of kurtosis*:

$$\frac{(1/N)\sum_{i=1}^{N}[d(i)-m_1]^4]}{(1/N)[\sum_{i=1}^{N}[d(i)-m_1]^2]^2},$$

(d) *Fifth-order moment*:

$$\frac{(1/N)\sum_{i=1}^{N}[d(i)-m_1]^5}{(1/N)[\sum_{i=1}^{N}[d(i)-m_1]^2]^{5/2}},$$

where

$$m_1 = \frac{1}{N}\sum_{i=1}^{N} d(i)$$

and $d(i)$ is the centroidal profile representation of a segment with N points, $i = 1, 2, \ldots, N$. The first three moments are the same used in [18]. Here, we use an additional moment to increase discrimination. The advantages of using moments to represent the centroidal profile of a segment are: (i) they are invariant to the starting point used to compute the centroidal profile, (ii) they represent the centroidal profile economically, and (iii) they are quite robust to noise and distortions (see [18]).

5. The training phase

During the training phase, the model database is built using appropriate segment-model associations. Building the model database involves training the segment and object recognizers. In the next subsection, we briefly review the multilayer neural network architecture used to implement the segment classifier and the object recognizers.

5.1. The multilayer neural network architecture

ANNs are specified by the topology of the network, the characteristics of the nodes and the training algorithm [14,23,24]. The topology of a multilayer ANN is a structured hierarchical layered network as shown in Fig. 8. It consists of several distinct layers of nodes including an input layer and an output layer. Between the input and the output layer we have one or more layers of nodes which are called hidden. Generally, each node in one layer is interconnected with all the nodes in adjacent layers with connections. Each connection is associated with a weight which measures the degree of interaction between the corresponding nodes. Nodes are relatively simple processing elements and the capabilities of multilayer ANNs stem from the nonlinearities used within them and the dense interconnectivity amongst them.

The algorithms for multilayer ANN processing can be divided in two phases: performance and training. In the performance phase of the algorithm, information flows from the input layer through the hidden layers to the output layer. In this phase, the nodes update their own activation values based on the system dynamics. In the training phase, an adaptation of the weights corresponding to the connection nodes takes place. One of the well-known training algorithms is the back-propagation rule [14]. It is an iterative algorithm which in each step adjusts the connection weights in the network, minimizing an error function. This is achieved using a gradient search which corresponds to a steepest descent on an error surface representing the weight space. In this study, the back-propagation with momentum algorithm has been used [14].

5.2. The segment classifier

The purpose of the segment classifier is to assign encoded segments to appropriate segment classes. The way segment classes are determined is by applying a clustering procedure to the entire set of encoded segments. The clustering algorithm used is a variation of a simple cluster seeking algorithm [26]. The segment classifier is

implemented by a multilayer ANN. The inputs to the ANN are the moments of the segment's centroidal profile, while the output represents the segment class to which the input segment belongs to. The steps involved in the training of the segment classifier are illustrated through a block diagram in Fig. 9. The next subsection describes the procedure for obtaining the segment classes.

5.2.1. Clustering the encoded segments

Let $\{S_1, S_2, \ldots, S_M\}$ represent M segments to be clustered. Each segment is represented using four moments of its centroidal profile $\{m_{i1}, m_{i2}, m_{i3}, m_{i4}\}$, for $1 \leqslant i \leqslant M$. Before clustering, it is necessary to define a measure of similarity which will establish a rule for assigning patterns (encoded segments), to the domain of a particular cluster center (segment class). The Euclidean distance measure [26] has been used for computing the similarity between two clusters.

The steps of the clustering algorithm can now be summarized as follows: Initially, a segment S_i is chosen to represent the first segment class Z_1. Without loss of generality, $Z_1 = S_1$ has been chosen. Next, a nonnegative threshold T is specified. The distance e_{12} between the second segment S_2 and Z_1 is computed next. If this distance exceeds T, a new segment class $Z_2 = S_2$, is created, otherwise, S_2 is assigned to the domain of segment class Z_1. Suppose that $e_{12} > T$, so that Z_2 is established. In the next step, the S_3 pattern is presented and the distances e_{13} and e_{23} are computed. The smaller of them is compared to T. If it is less than T, then S_3 is assigned to the corresponding class, otherwise, a new class $Z_3 = S_3$, is created. In a similar manner, the distances from each new encoded segment to every established segment class are computed. A new segment class is created if all of these distances exceed T. Otherwise, the segment is assigned to the domain of the segment class to which it is closest. Obviously, the number of classes formed dependent on the choice of the threshold parameter T.

5.2.2. Training the segment classifier

The training of the segment classifier is performed using training examples of the form (segment, segment class). Before training, however, we must decide how to represent the segment classes at the outputs of the network. The inputs of the network are simply the four moments of the segment's centroidal profile. The outputs of the network must be an appropriate representation of the segment class. Choosing a proper encoding scheme for representing the outputs of the segment classifier is very important for our approach. Such a scheme must satisfy two requirements: First, it must be economical and second, it must be proximate. Non-economical representation schemes are not desirable because they will increase the size of the networks. The proximity constraint assures that elements from this scheme, representing similar classes, will be similar. Proximity is very desirable for increasing the robustness of the object recognizers. Recalling our discussion in Section 3, the inputs of the object recognizers are provided by the segment classifier. A proximity-preserving scheme will reduce the possibility that the object recognizers make incorrect decisions, although some of the segment classifier's decisions might be incorrect.

In order to represent the segment classes in a way that satisfies both of the above requirements, first we order the classes in a list, based on their similarity, and then we encode them using an encoding scheme which preserves proximity. Two classes are considered similar if their cluster centers are similar. The Euclidean distance measure is used as a measure of similarity. The segment classes are ordered in a list as follows: Initially, a segment class is randomly chosen and is placed first in the list. Next, its distance from all other classes is computed and the most similar segment class is placed second in the list. Following the same procedure, the distances between the second-ordered segment class and the remaining unordered classes are computed and the most similar segment class is placed third in the list. This process is repeated, until all the segment classes have been ordered. Obviously, there are many different ordered sequences which can be obtained by following the above methodology, since the class placed first in the list is chosen randomly. However, all of them have the desired property that adjacent segment classes in the list are similar.

Next, the ordered classes are encoded in an economical and proximity-preserving way. This encoding must also be appropriate for the output nodes of the segment classifier. One of the most commonly used encoding schemes in ANN classification tasks is the local scheme. According to this scheme, each class is assigned to a distinct output node. Although this scheme satisfies the requirement of proximity, it is not economical since it requires M output nodes to express M different classes. Another possibility is to use binary coding. This scheme is more economical since it requires $\log_2(M+1)$ output nodes to express M different classes but, despite the economy in the number of digits used, this scheme does not satisfy the proximity requirement. An appropriate scheme, satisfying both of the requirements, is the Gray code scheme [27]. Successive elements of the Gray code scheme differ by only one bit and it is equally economical as the binary code scheme. Thus, segment classes have been represented using gray coding. It should be mentioned that the performance of the segment classifier itself does not seem to depend on the particular output encoding used. For example, we performed experiments using a different encoding scheme (binary scheme) and the classification results were very similar to those using gray coding.

The training of the segment classifier is performed by presenting to the input nodes of the network the moments of the centroidal profile of the segments while and

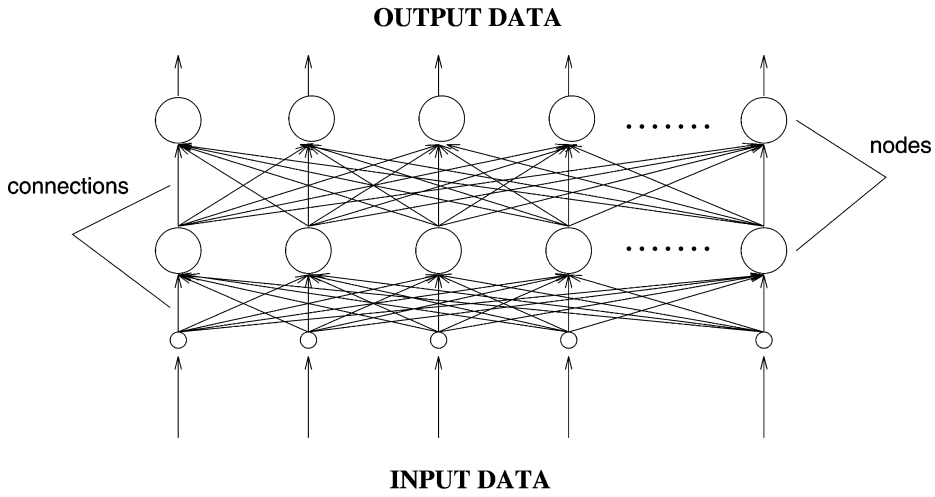

Fig. 8. A two-layer, feed-forward, artificial neural network.

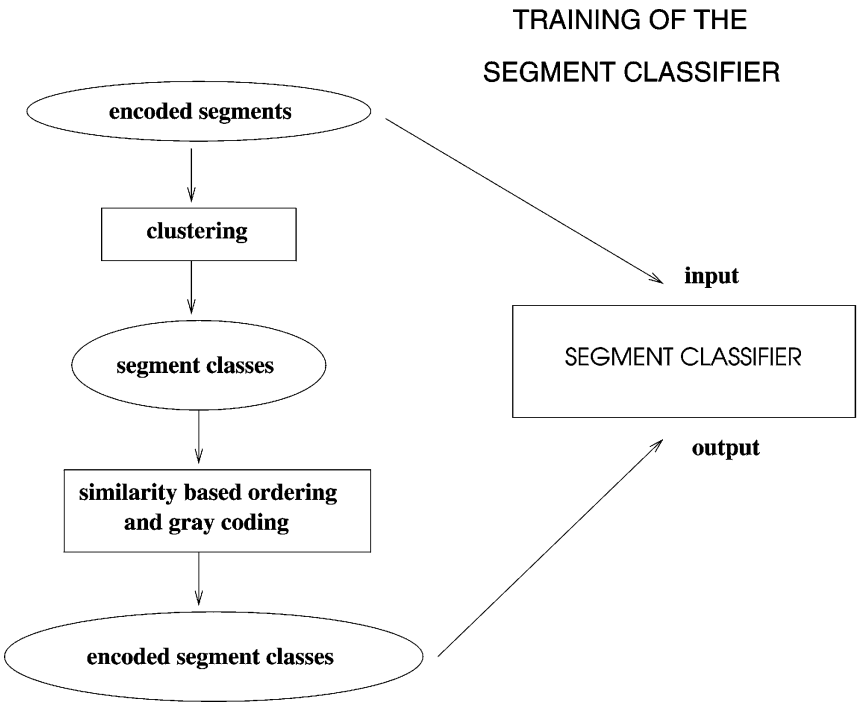

Fig. 9. The steps involved in the training of the segment classifier. First, the segments are clustered into clusters. Then, the segment clusters are ordered based on the similarity of their cluster centers and finally encoded using gray coding. The moments of the segments and the gray-coded segment clusters are then used to train the segment classifier.

the gray-coded segment classes at the output nodes. Training continues until the segment classifier is able to classify all segments correctly.

5.3. The object recognizers

The purpose of the object recognizers is to perform recognition based on segment classifications made by the segment classifier. Each object recognizer is trained to learn associations between important groups of segments and models. For each model, important groups of segments are extracted by applying a selective search procedure on the hierarchy of contour segmentations associated with the model. The segments obtained are classified by the segment classifier and the obtained segment classifications are used to form the inputs of the object recognizers. Each object recognizer has different capabilities which mainly depend on the size of important

groups utilized by the recognizer for recognition. Fig. 10 shows a block diagram of the steps involved in the training of the object recognizers. Next, we describe in detail the selective search procedure.

5.3.1. Finding important groups of segments

The objective of the selective search procedure is to identify groups of segments with high discrimination power, that is, groups of segments which can be used to distinguish a model from other models. These groups are referred to as *important groups*. First, the selective procedure looks for important groups of size one, that is, single segments which characterize models uniquely. Such segments are usually found at the high levels of the segmentation hierarchies. More specifically, the selective search procedure works as follows: initially, one of the hierarchies of contour segmentations is randomly chosen and designated as the "reference hierarchy". Then, we concentrate at each level of the reference hierarchy, starting from the highest possible level, comparing segments at this level with segments coming from any level of the other hierarchies. The reason we compare segments from one level of the reference hierarchy with segments from every level of the other hierarchies is because although some objects might have common segments, these segments might appear at different scales. If a segment from the highest level of the reference hierarchy does not match with any segment from the other hierarchies, it is considered to be an important segment. After important segments have been identified at the highest level of the reference hierarchy, the procedure continues at the next lower level of the reference hierarchy, until all the levels of the reference hierarchy have been considered. Then, one of the remaining hierarchies is randomly chosen to become a new reference hierarchy and the same procedure is repeated until all the hierarchies have been considered as reference hierarchies.

Obviously, non-important segments have no discriminatory power by themselves. This does not mean, however, that they are completely useless. For example, consider objects which consist of exactly the same segments, but the order in which the segments appear is different from one object to another. Using the identity of the segments only in order to distinguish them is impossible in this case. However, if the order of the segments is also taken into consideration, then we might be able to distinguish them. Following this idea, the selective search procedure described above extracts not only important segments, but also important groups of segments. An important group of segments consists of a number of adjacent segments which may not be important by themselves, however, their identity along with their order in the group makes possible the discrimination of one object from another. Initially, the search procedure looks for groups with a minimum number of elements (i.e., two). Then, it looks for larger size groups. In our implementation, we use groups of size one, two, three, and four. It should be mentioned that the maximum size of groups we consider at a particular segmentation level depends on the maximum number of segments available at this segmentation level and the maximum group size we allow (i.e., it is the minimum of the two).

As we move from higher to lower segmentation levels, certain segments remain unchanged unless new zero-crossings are introduced which further sub-split them. This implies that some of the comparisons that take place at higher levels, do not necessarily have to take place at lower levels. Also, it should be mentioned that the number of important groups is not very big since we do not consider every group of segments but only groups of adjacent segments which keeps the number of combinations small.

5.3.2. Training the object recognizers

The important groups of segments obtained by applying the selective search procedure are used to train the object recognizers. During training, the object recognizers learn to associate important groups of segments with the models uniquely characterized by these groups. The object recognizers have been structured hierarchically, according to the size of the groups they utilize in order to perform their recognition task. The object recognizer located at the top of the hierarchy is trained to recognize model objects using important groups of size one. The object recognizer located at the next level is trained to recognize model objects using important groups of size two. In a similar fashion, the object recognizer located at the bottom the hierarchy is trained to recognize model objects using important groups of a maximum size. In the context of indexing, each object recognizer implements a different hash function. Since important groups of segments characterize model objects uniquely (i.e., it is a one to one mapping), the computed hash functions will be collision free.

To train the object recognizers, we must first determine an appropriate representation for the outputs of the networks. The inputs to the object recognizers are simply the encoded segment classifications associated with the important groups. For each important group, we find the encoded segment classification of the segments in the group and we concatenate them to form the input to the object recognizers. The output of the object recognizers is an appropriate representation of the model characterized by the important group of segments. The local coding scheme, mentioned earlier in Section 5.2.2, has been used to represent model objects. According to this scheme, the number of output nodes is equal to the number of model objects, with each node representing a different model. Although this scheme is not economical, it is very suitable for our task because it allows us to distinguish between *ambiguous* and *unambiguous* recognition. In the case of an unambiguous recognition, the

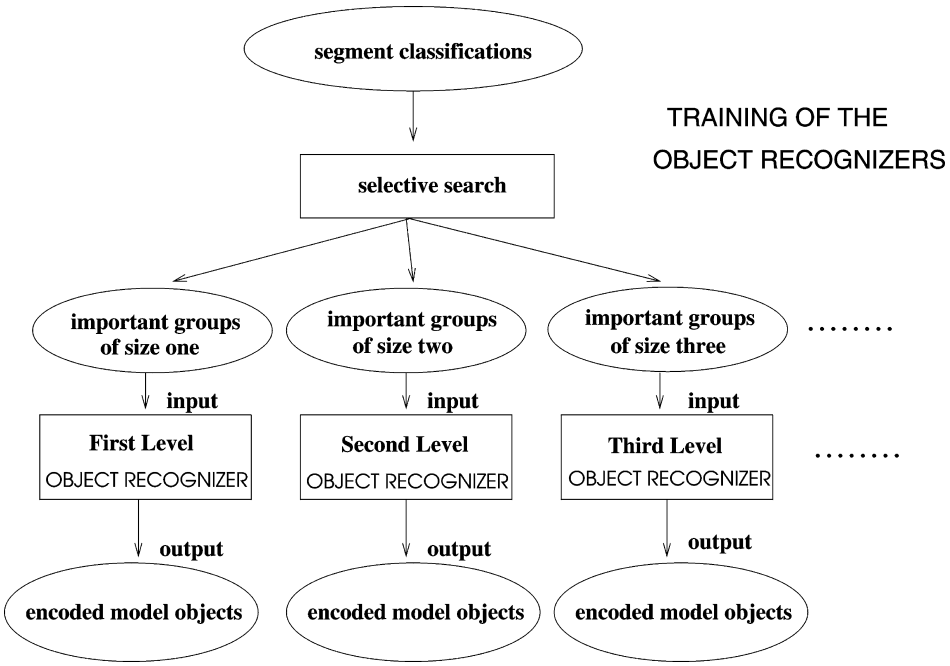

Fig. 10. The steps involved in the training of the object recognizers. Each recognizer is trained using groups of segment classifications (i.e., outputs of the segment classifier).

output of an object recognizer contains only one 1 at a certain location and 0's everywhere else. Outputs with more than one 1's are considered to be ambiguous because they do not correspond to a particular model object representation. Ambiguous recognition results are rejected immediately while unambiguous recognition results are considered for verification.

The training sets of the object recognizers include two kinds of training examples: *positive* and *negative*. Positive training examples are of the type (*important group*, *model*), where we assume that the important group characterizes the model uniquely. Negative training examples correspond to classifications of non-important groups of segments, that is, groups that do not characterize a model uniquely. The need to augment the training set with negative examples will become more obvious in the next subsection where a simple example is presented. The basic idea is that negative training examples will force the network to respond in a predictable way when non-important groups are presented to its inputs. Non-important groups are extracted at the same time that we search for important groups. When a non-important group is presented to the inputs of an object recognizer, we require that the object recognizer responds in a predictable way. Specifically, we require that the output consists of 0's only (i.e., none of the nodes is allowed to respond with a high value). To be consistent with our notation, we call this special output the representation of the "null" model. Thus, all negative training examples are of the type (*non-important group*, *null-model*).

5.4. An example

In this subsection, a simple example is presented to demonstrate the training step. Let us assume that the model database contains two different models, shown in Fig. 11a and b. Initially, the hierarchical segmentation takes place which partitions the models into two (Fig. 11c and d), four (Fig. 11e and f), and six (Fig. 11g and h) segments. Note that in practice, we do not consider segmentations with 2 segments as we have already discussed in Section 4.2. Here, we use them just for illustration purposes. The total number of segments is 24. Let us assume that the resulted segments have been normalized and clustered into 10 classes $S_0, S_1, \ldots, S_9$, by applying the clustering scheme described in Section 5.2.1. Fig. 11 shows the segment class assigned to each segment. The training set of the segment classifier will include examples of the form:

$$(m_{i1}\, m_{i2}\, m_{i3}\, m_{i4}) \rightarrow S_j,$$

where $i = 0, 1, \ldots, 23$ and $j = 0, 1, \ldots, 9$, and $(m_{i1}\, m_{i2}\, m_{i3}\, m_{i4})$ are the moments of the centroidal profile associated with the i segment.

Denoting by $O_{j(k)}$ the jth model contour, partitioned in k segments, then each object contour can be represented in terms of its segment classifications as follows:

$$O_{1(2)} = (S_0 S_1), \qquad O_{2(2)} = (S_0 S_2).$$

$$O_{1(4)} = (S_0 S_5 S_4 S_3) \qquad O_{2(4)} = (S_0 S_8 S_7 S_6),$$

$$O_{1(6)} = (S_0 S_3 S_4 S_9 S_4 S_3), \quad O_{2(6)} = (S_0 S_6 S_7 S_9 S_7 S_6).$$

These are actually the hierarchies of segmentations of the model objects. It should be noted that S_0 and S_9 are common in both models.

The object recognizer located at the top of the hierarchy is trained to recognize a model using important groups of size one, that is, important segments only. In this example, its training set includes the following training examples:

$S_1 \to O_1, \quad S_2 \to O_2$ (1st segmentation level, Fig. 11c, d)

$S_5 \to O_1, \quad S_8 \to O_2$ (2nd segmentation level, Fig. 11e, f)

$S_4 \to O_1, \quad S_7 \to O_2$ (2nd segmentation level, Fig. 11e, f)

$S_3 \to O_1, \quad S_6 \to O_2$ (2nd segmentation level, Fig. 11e, f)

where $S_i \to O_j$ indicates that S_i characterizes O_j uniquely. The reason that S_0 and S_9 have not been included in the training set is because they correspond to non-important segments, that is, they cannot be used to distinguish between the two models. However, it is important for the object recognizer to learn to produce an ambiguous recognition when any of these two segment classifications is presented to its inputs. These segment classifications will actually give rise to the negative training examples discussed in the previous section. To illustrate the necessity of the negative examples, let us suppose that S_0 is represented by 0000, S_1 by 0001, S_4 by 0010 and S_8 by 0100 (gray codes). Also, suppose that S_0 has not been used in the training of the object classifier and that after training has been completed, we present S_0 to its inputs. The question is how the network is going to respond. Ideally, we would like the network to yield an ambiguous recognition. However, given that ANNs are tolerant to noise and distortions, it is very likely that the recognizer will perceive the representation of S_0 as a noisy version of one of the representations associated with the positive training examples (for example, note that S_0's representation is very close to the representations of S_1, S_4, and S_8 – they differ only by one bit). As a result, it might yield an unambiguous recognition. To prevent situations like these, it is important to augment the training set of the object recognizer by including the following two negative examples:

$$S_0 \to O_{\text{NULL}}, \qquad S_9 \to O_{\text{NULL}}.$$

O_{NULL} denotes the null object and can be represented at the output nodes of the object recognizer as a sequence of zeros (ambiguous representation). The same methodology is followed during the training of the rest of the object recognizers: whenever a combination of segment classifications does not characterize a model unambiguously, it is added to the training set of the appropriate object recognizer as negative training example.

The training set of the object recognizer located below the next level, contains training examples corresponding to important groups of size two. These are formed by concatenating the encoded segment classifications corresponding to the segments of the important groups. In our example, the training set of the object recognizer will include the following training examples:

$S_0S_1 \to O_1, S_0S_2 \to O_2$ (1st segmentation level, Fig. 11c, d)

$S_0S_5 \to O_1, S_0S_8 \to O_2$ (2nd segmentation level, Fig. 11e, f)

$S_5S_4 \to O_1, S_8S_7 \to O_2$ (2nd segmentation level, Fig. 11e, f)

$S_4S_3 \to O_1, S_7S_6 \to O_2$ (2nd segmentation level, Fig. 11e, f)

$S_3S_0 \to O_1, S_6S_0 \to O_2$ (2nd segmentation level, Fig. 11e, f)

$S_4S_9 \to O_1, S_7S_9 \to O_2$ (3rd segmentation level, Fig. 11g, h)

$S_9S_4 \to O_1, S_9S_7 \to O_2$ (3rd segmentation level, Fig. 11g, h)

No negative examples exist for this training set. It should be noted, that although S_0 and S_9 are not useful by themselves for distinguishing between the two models, their combination with other segments provides groups of segments which can be used for discrimination purposes.

In a similar way, the training sets of the other object recognizers can be constructed. The training set of the bottom level object recognizer will include training examples based on six segment classifications.

6. The recognition phase and the verification procedure

The major characteristic of the recognition phase is that it operates hierarchically. In particular, recognition starts by selecting segments located at high levels of the hierarchy of segmentations of the scene contour. This is because segments located at high levels are rich in information and small in number. If recognition cannot be established using information from these levels, probably because the segments present at these levels have been affected by occlusion, segments located at lower levels are considered. Concentrating at lower scales yields more segments, which might be less descriptive, however, it is more likely that most of them will be unaffected by occlusions. The general strategy of the matching procedure is that, when high level segments are present in the scene, it is worth using them for matching because recognition can become faster and verification more accurate. However, when high level segments are partially visible, due to occlusions, segments from lower levels must be considered.

Let us now describe the recognition procedure in more detail. Initially, a closed, possibly, composite contour is extracted from the scene and its RCSS image is computed. Then, a hierarchy of segmentations is obtained and the centroidal profiles of the segments are computed.

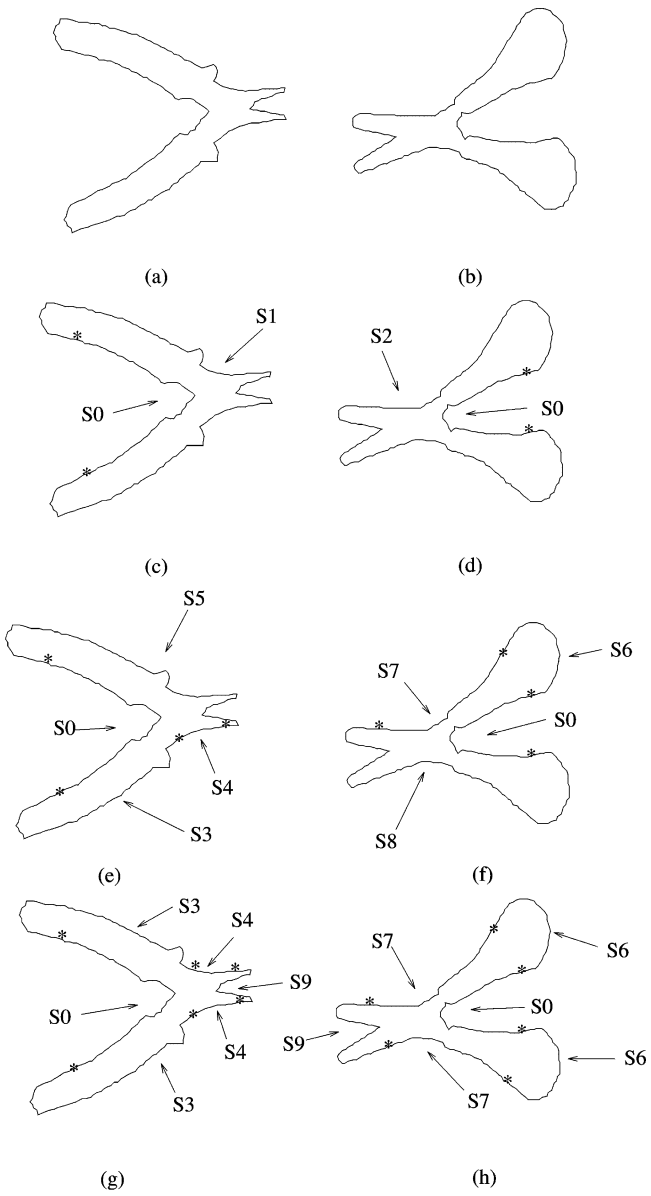

Fig. 11. An illustration of the hierarchical segmentation and representation of object contours using their segment classifications. At the higher levels of the hierarchy, each object is represented by a small number of segment classifications. As we move down to a lower level, more segment classifications are used to represent objects.

Next, each segment is represented by four moments of its centroidal profile. Matching is performed from higher to lower scales and in a local to global basis. First, recognition is attempted using segments located at a high scale. Each segment is classified by the segment classifier and the obtained encoded segment classifications are fed, one at a time, to the top level object recognizer. If the object recognizer fails to yield an unambiguous recognition, pairs of encoded segment classifications, corresponding to adjacent segments, are fed, one at a time, to the next level object recognizer. If this object recognizer is also unable to perform an unambiguous recognition, encoded segment classifications corresponding to triplets of adjacent segments are chosen and fed to the object recognizer located at the next lower level. The same procedure is repeated for all other groups at this level of the hierarchy. It should be reminded that the size of a group can be between one and four as we discussed in Section 5.3.1. If recognition is still impossible, the immediately lower level of the hierarchy of segmentations is chosen and the same procedure is repeated. The procedure continues until all the levels of the hierarchy of segmentations have been considered or all the objects present in the scene have been recognized. Fig. 12 shows the major steps of the recognition procedure.

In case of an unambiguous recognition at some step of the recognition procedure, verification follows to evaluate its validity. During verification, we need to find which group of segments from the candidate model corresponds to the group of segments chosen from the scene and compute the transformation that brings them into alignment. Then, the same transformation is applied to all the points of the model in order to back-project it on the scene and compute its overlap with the scene. The only information available to us at the time that an unambiguous recognition occurs is the classification of the scene segments and the identity of the candidate model. The object recognizer's decision limits the search to a specific model only. Also, the segment classifier limits the search to these model segments which have the same classification with the ones chosen from the scene. To implement this step efficiently, we keep a table with information about the groups of segments extracted during preprocessing. In particular, we store information regarding the model object, the coordinates of the endpoints, and its classification. To find the correct match between model and scene segments, a number of hypotheses must be tested. In particular, a new hypothesis is formed for every group found in the table which belongs to the same model and has the same segment classifications with the group chosen from the scene. Each of the hypotheses requires the computation of a similarity transformation. Then, the candidate model is back-projected onto the scene and the fraction of the model accounted for by the scene data is computed. If this fraction is grater than a threshold, the verification is considered to be successful and the contour of the recognized model is removed from the scene. Recognition continues until no more objects have left in the scene. Fig. 13 illustrates the verification procedure.

A number of issues should be clarified at this point. First of all, as was the case in preprocessing, segmentation does not start at the highest possible scale level and does not end at the lowest possible level. As we discussed in Section 5.3.1, each segmentation level contains between 4 and 8 segments. During recognition, however, we

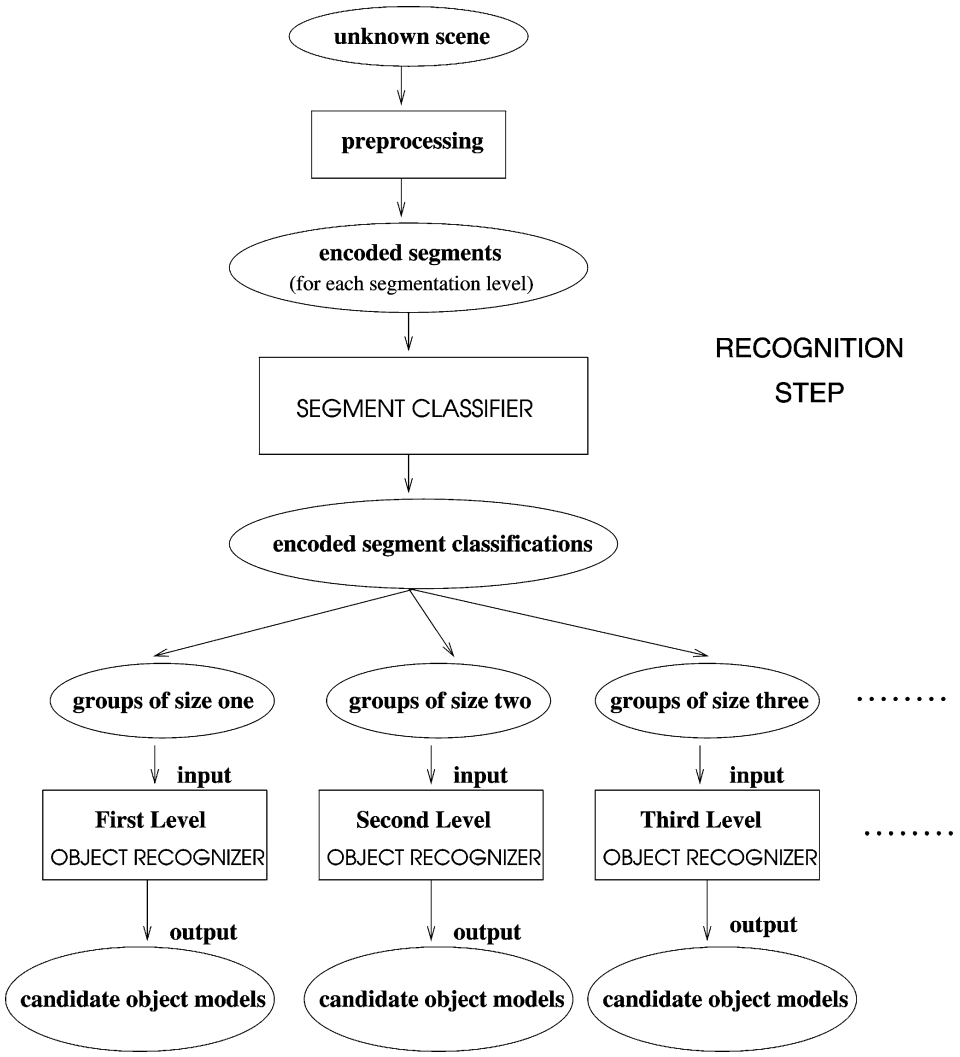

Fig. 12. The steps involved in the recognition phase. First, the preprocessing step is applied. Then, the segment classifier is applied to obtain the segment classifications. Finally, groups of segment classifications are forwarded to the object recognizers which attempt to recognize the unknown object(s). The details are given in the text.

should take into consideration that the scene contour might be composite. This implies that in order for us to obtain non-composite segments, we should consider splitting the contour into more segments. In our experiments, we have considered segmentations yielding between 6 and 12 segments. In other words, the segmentation procedure does not consider segmentations at scale levels with less than 6 and more than 12 zero-crossings. Another issue is the redundancy that might appear to exist during recognition. In particular, it might appear redundant that combinations of segments are tried. The reason is that some of these groups correspond to segments which exist at a higher segmentation level. If recognition is impossible at the higher level, then is seems that there is no benefit for considering combinations of segments from the lower levels. However, this is not absolutely true. In fact, the redundancy involved during recognition can be proven very beneficial in case of occlusion or non-uniform noise. In these cases, recognition at a high segmentation level might be impossible but recognition at a low level quite feasible. This is because many of the segments in the group might be classified correctly which will allow the object recognizer to reach a correct decision.

7. Implementation issues and simulation results

In this section, experimental results are presented to illustrate the efficiency of the proposed technique. The performance of the recognition system was tested using a library of 20 real objects. This library contains fairly complex objects with a large variety of features. Initially, the objects were positioned on a light table resulting in

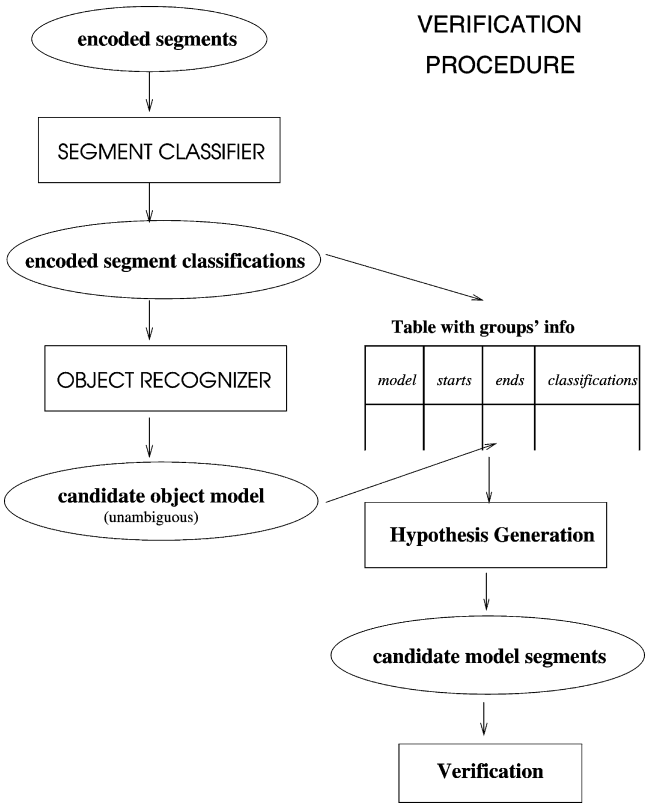

Fig. 13. The steps involved during verification. The data structure used for the formulation of the hypotheses is also shown.

high contrast images. A Laplacian edge detector separated the objects from the background and a boundary following routine extracted their boundaries. The twenty simple closed planar shape boundaries thus obtained are shown in Fig. 14. All boundaries were approximated by 256 contour pixels. Inner boundary information was not used in this study.

7.1. Applying the preprocessing step

Once the boundaries of the object models were extracted, their RCSS images were computed. Thus, each contour was convolved with a Gaussian filter for a continuous range of scales. The standard deviation σ of the Gaussian was set equal to 1 and the filter's kernel size was chosen to be 5σ (this size can still provide a good approximation for the second derivative of the Gaussian function). Then, we computed the hierarchy of segmentations for each model, as described in Section 4.2. Localization was performed at $\sigma = 1$. As discussed, the highest level of each hierarchy contains no less than 4 segments while the lowest level contains no more than 8 segments. The total number of segments obtained was 351. Note that segments which appear at multiple segmentation levels were counted only once. For each segment obtained, we computed its centroidal profile and then represented it in terms of four moments of its centroidal profile (encoded segments).

7.2. Training the segment classifier

The training of the segment classifier requires first the clustering of the segments obtained from the previous step, the ordering of the segment clusters according to their similarity, and the gray-coding of the segment classes. The threshold parameter T, mentioned in Section 5.2.1, was set to 0.05. The total number of segment clusters obtained was 159. The segment clusters were represented using an 8-bit Gray code. This determined also the number of output nodes of the segment classifier (8 nodes). The network used was a two layer network (one hidden layer and one output layer), with 80 nodes in the hidden layer. The learning rate and momentum term were set to 0.2 and 0.4 correspondingly. The stopping criterion used was that the mean square error of the outputs was close to 0.001. The number of training epochs required for convergence was 100 000.

7.3. Training of the object recognizers

The inputs to the object recognizers were appropriate combinations of encoded segment classifications. The object recognizers were structured in a hierarchy and each one of them was trained with groups of encoded segment classifications of a specific size. These segment classifications correspond to important segments of important groups of segments which were extracted by applying the selective search procedure described in Section 5.3.1. Two-layer ANNs were used for implementing the object recognizers. The object recognizer located at the top of the hierarchy was trained to recognize model objects using important groups of size one (i.e., important segments). Thus, the number of input nodes for this object recognizer was set to 8. The number of positive training examples used to train this object recognizer, which actually represents the number of important segments found by the selective search procedure, was equal to 107 while the number of negative examples was equal to 14. The ANN located at the next lower level of the hierarchy was trained to recognize model objects using important groups of size two. Thus, the number of its input nodes was set to $2 \times 8 = 16$. Its training set consisted of 150 positive training examples and 8 negative training examples. In a similar manner, the number of input nodes and training examples of the rest object recognizers were determined. Table 1 shows these numbers as well as the number of nodes chosen in the hidden layers. The local encoding scheme was used to represent the identity of the model objects at the output nodes of the object recognizers. The number of output nodes was set to 20 (equal to the number of model objects in our

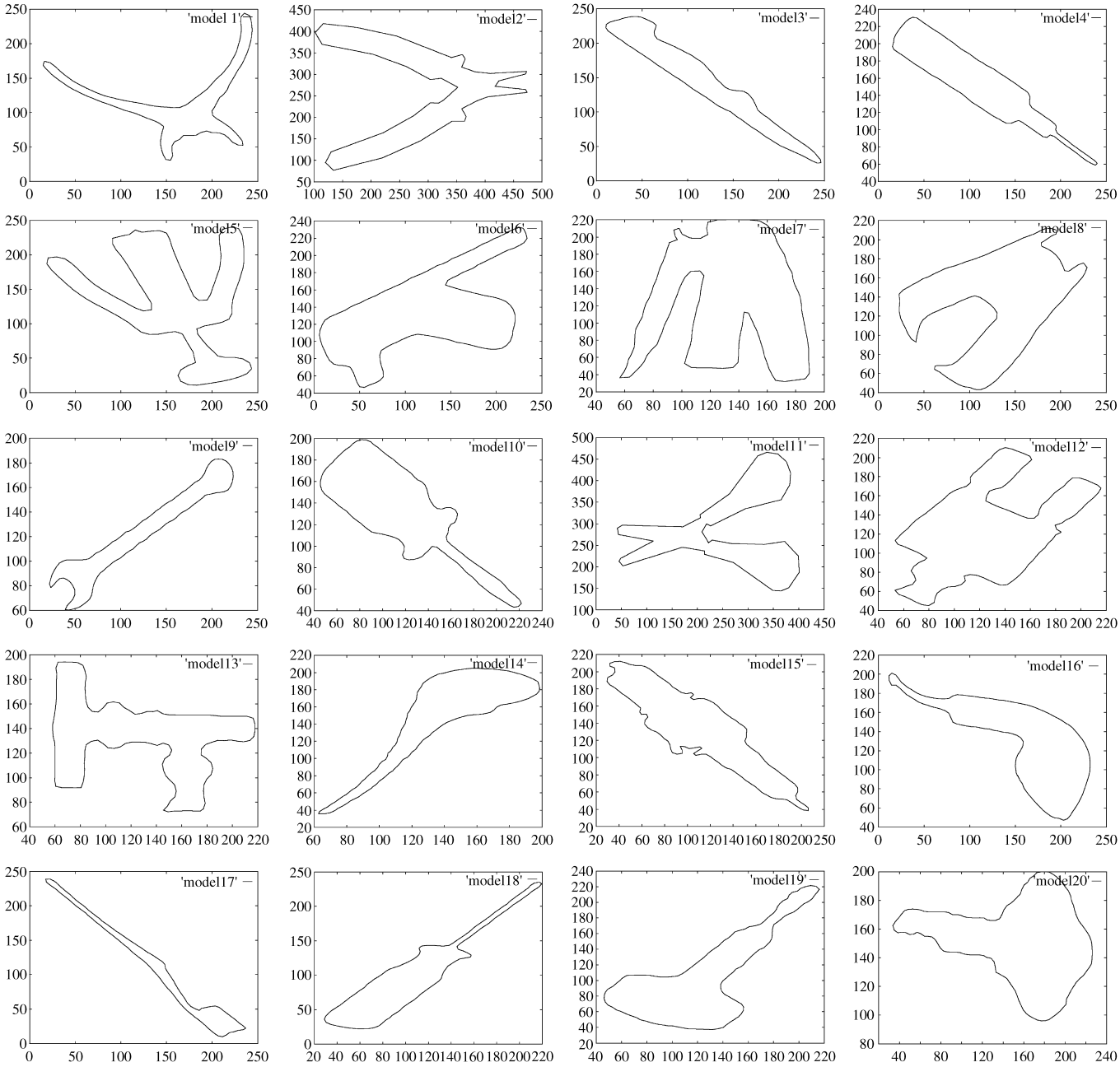

Fig. 14. The model objects contained in the model database.

database). The same learning rate, momentum term, and stopping criterion used to train the segment classifier were also used to train the object recognizers. Table 1 shows the number of training epochs required for convergence by each object recognizer. Note that the training time of each object recognizer was significantly shorter than that of the segment classifier.

7.4. Experiments and results

The performance of the proposed approach was studied through a number of experiments. First, we performed experiments to test the robustness of the method in presence of noise and distortion. No occlusion was allowed at this time. For each model object, 100 test objects were generated. These test objects consisted of samples of different sizes of the model objects in various translational and rotational positions, with noise and distortion. The objects were rotated over the range from 0 to 2π radians and translated to random positions. The size of each object was varied from 0.5 to 1.5 times the size of the original object. Noise and distortion effects were introduced by adding random noise to the boundary points using the approach of You and Jain [28].

Table 1
Parameters used for training the object recognizers

			Complex objects			
Object recognizer	Training examples	Negative examples	Input layer	Hidden layer	Output layer	Epochs
First	107	14	8	64	20	391
Second	150	8	16	32	20	607
Third	158	0	24	16	20	291
Fourth	158	0	32	8	20	253

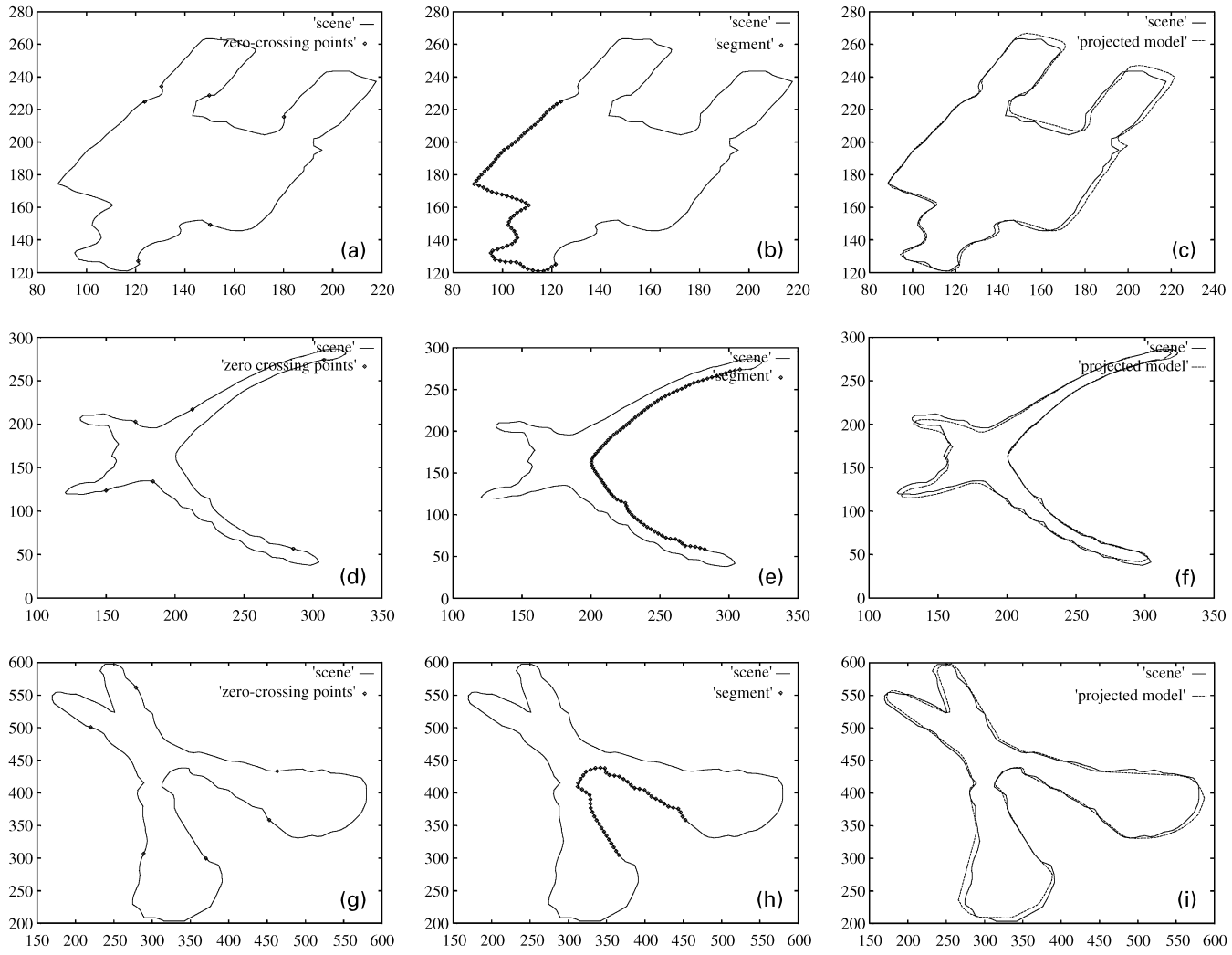

Fig. 15. Recognition results (proposed method). The high-lighted segments which led to correct recognition are shown in the second column. The correctly recognized model, back-projected on the original scene (dotted line) is shown in the third column.

Specifically, if the coordinates of the kth boundary point are $(x(k), y(k))$ then the coordinates of the corresponding point on the noisy boundary $(x_n(k), y_n(k))$ are given by

$$x_n(k) = x(k) + drc\cos(\theta(k)),$$

$$y_n(k) = y(k) + drc\sin(\theta(k))$$

where d is the distance of boundary point k to point $k+1$, r is a sample from Gaussian distribution N(0,1), c is a parameter which controls the amount of distortion and $\theta(k)$ is the angle from the x-axis to the normal direction of the boundary at point k. In our simulations, c was set to 0.5. The points that caused crossover on the boundary

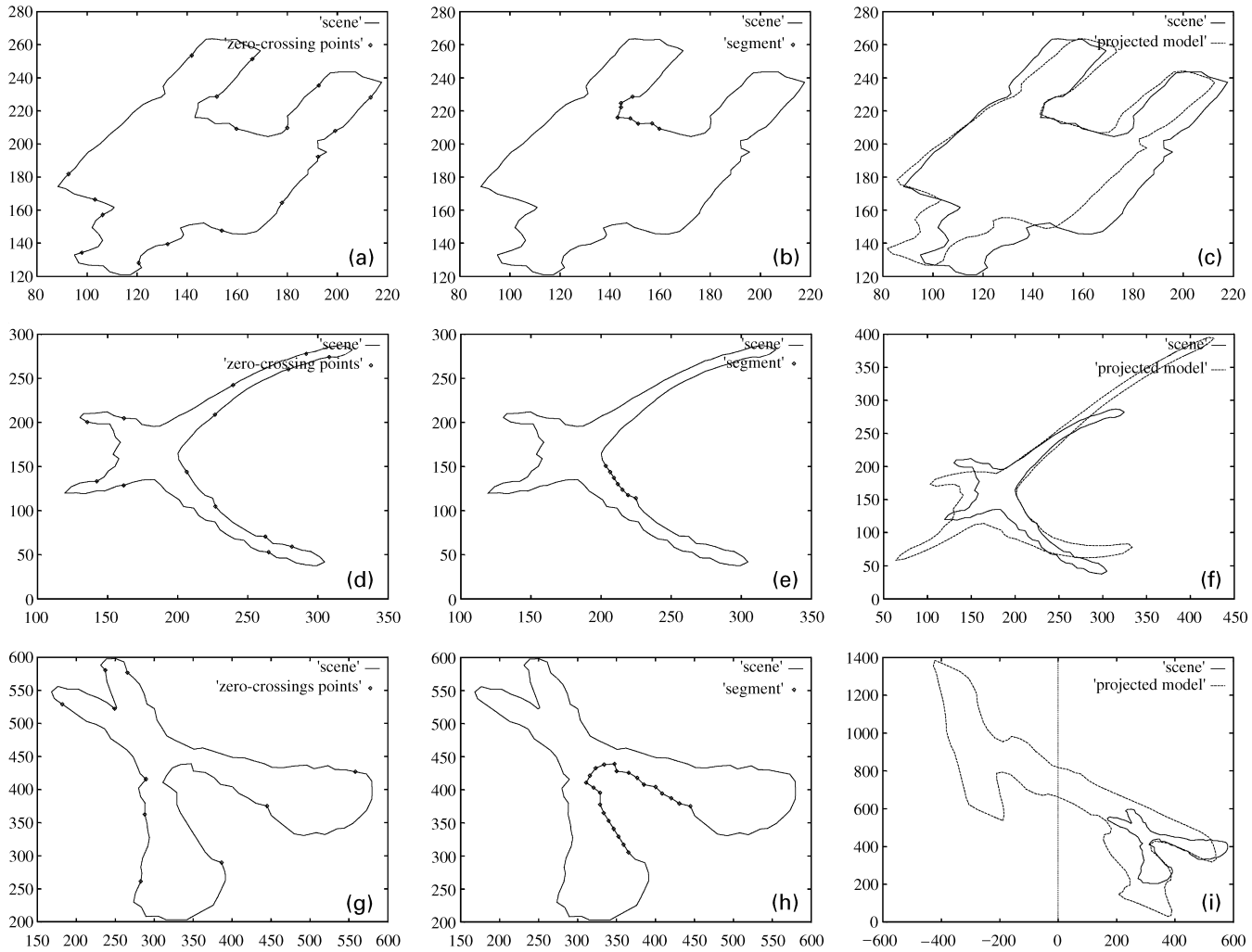

Fig. 16. Recognition results (variation). The high-lighted segments which led to correct recognition are shown in the second column. The correctly recognized model (dotted line), superimposed on the original scene (solid line) is shown in the third column.

were omitted. The amount of noise in an object contour was determined by the percentage of contour points corrupted with noise. The accuracy of the proposed was tested using objects corrupted with various amounts of noise (0–50% noise corruption). Fig. 15 shows some of the test objects (assuming 30% noise corruption) and the recognition results (60% or more of the model points were required to match with the scene).

For comparison purposes, we also implemented another method which is actually a variation of [6]. In [6], the segments are represented by the Fourier descriptors of the object's contour segments. Here, we are using moments of the centroidal profiles of the segments, as is the case with our approach. Also, the procedure for extracting the contour segments in [6] is different than the one used here. In [6], segments are obtained by moving a window of a specific size over the object's contour. Each time the center of the window is moved, the part of the object's contour enclosed within the window represents a new segment. Obviously, the segments obtained by this procedure have a high degree of overlap, making the system tolerant to occlusions. In our implementation, the segments are determined by the curvature zero-crossings of the object's contour. A Gaussian filter with σ equal to 1.0 was used for the detection of the zero-crossings. Of course, the number of segments obtained using this method is much less and the system is less tolerant to occlusions. However, our objective here is to demonstrate the problems that traditional indexing schemes are facing when noise and distortion are present. Also, we want to demonstrate the benefits of using segments coming from high scales first for better recognition and localization. Fig. 16 shows the recognition results in the case of the variation, using the same test objects (Fig. 15). Some very interesting comments can be made by observing these results. First, both methods were able to recognize the instance of the object shown in Figs. 15 and 16, however, the proposed method has localized the

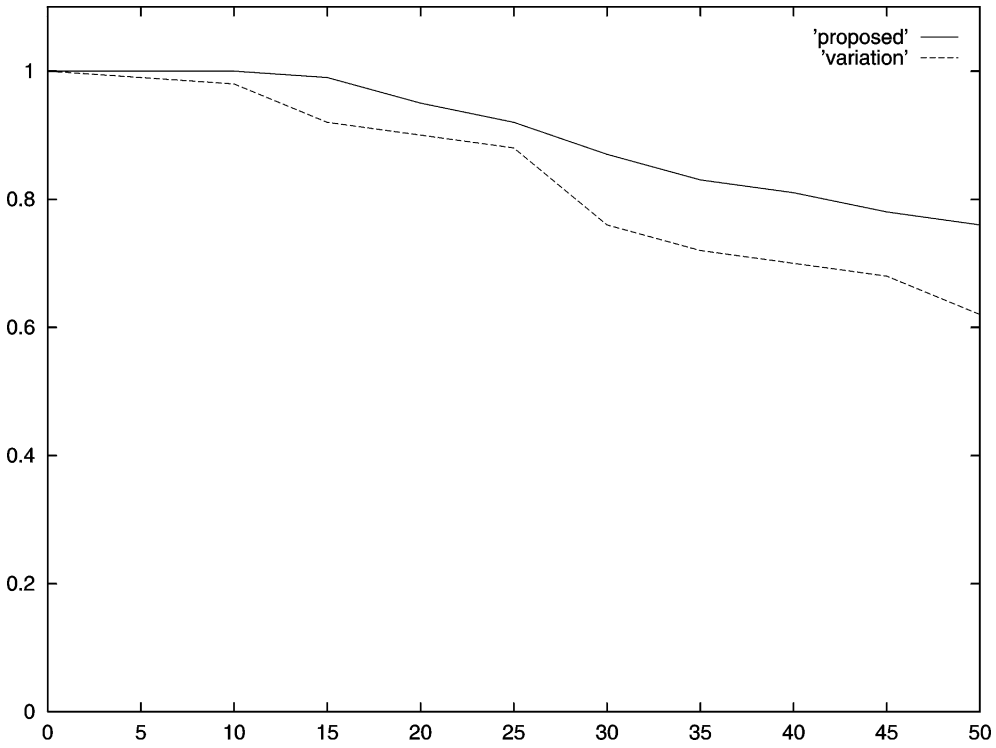

Fig. 17. The recognition accuracy of the two methods. The solid line corresponds to the proposed method while the dotted one corresponds to the variation. The horizontal axis corresponds to the percent of noise while the vertical one corresponds to recognition accuracy.

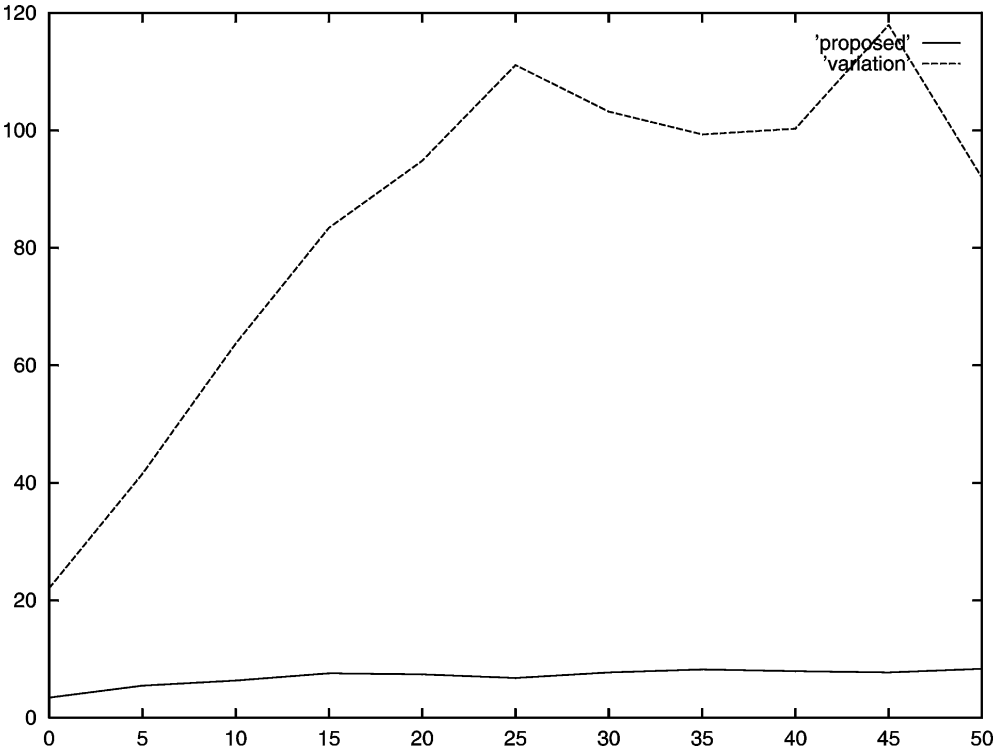

Fig. 18. The average number of hypotheses verified by the two methods. The solid line corresponds to the proposed method while the dotted one corresponds to the variation. The horizontal axis corresponds to the percent of noise while the vertical one corresponds to number of hypotheses.

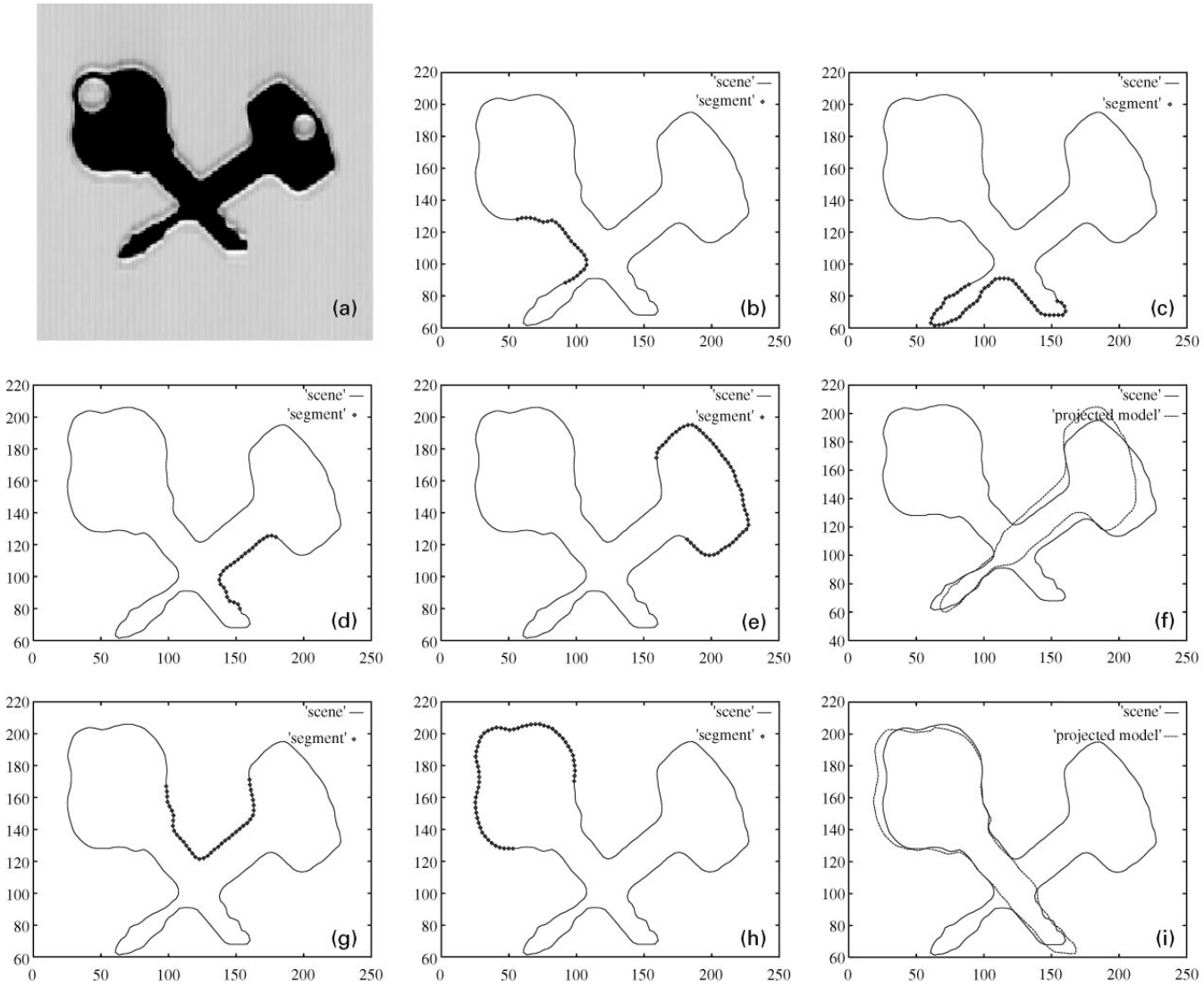

Fig. 19. Recognition results using a real scene. The high-lighted segments represent the segments chosen by the recognition algorithm (in that order). The last graph in this figure shows the model (dotted line) superimposed onto the scene (solid line).

object more accurately as can be seen by observing Figs. 15c and 16c. The reason is that the proposed method utilized a segment extracted from a higher scale (Fig. 15b) which was much longer than the segment used by the variation (Fig. 16b). In addition, our method verified only two hypotheses while the variation verified 46. Figs. 15d and 16d show an instance of another object. In this case, the variation was able to hypothesize the correct model but recognition was unsuccessful because verification failed due to the poor alignment of the model with the scene (Fig. 16f). On the other hand, the proposed method recognized the model correctly and localized it quite accurately as can be seen from Fig. 15f. In this example, the proposed method verified only two hypotheses while the variation verified 86 hypotheses. Finally, Fig. 16i shows an incorrect recognition. Although the same segment chosen by our method (Fig. 15h) was also chosen by the variation (Fig. 16h), noise did not allow the variation to access the correct hash table location and form the correct hypotheses. Fig. 15i shows the recognition results of the proposed method. The number of hypotheses verified by our method in this case was 6 while the variation verified 376 hypotheses before it reports the unsuccessful recognition.

Fig. 17 shows the overall recognition accuracy of the two methods, assuming various levels of noise (0–50%). The average number of hypotheses verified by the two methods are also shown in Fig. 18. Obviously, the proposed method is more accurate and faster. This is due to two reasons: first, due to the RCSS driven matching approach and second, due to the ANN-based indexing scheme which does not allow for collisions. In most

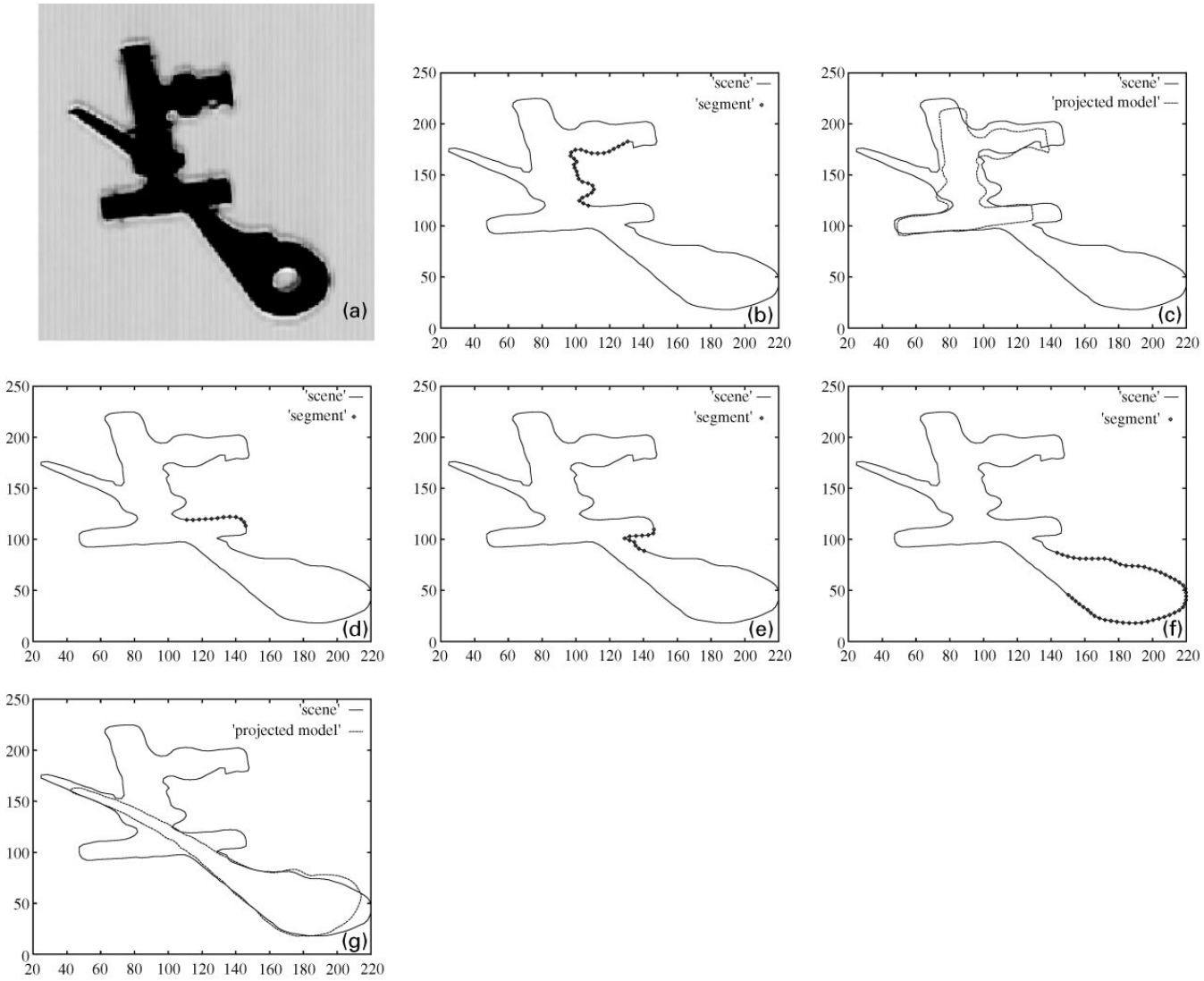

Fig. 20. Recognition results using a different real scene. The high-lighted segments represent the segments chosen by the recognition algorithm (in that order). The last graph in this figure shows the model (dotted line) superimposed onto the scene (solid line).

of the recognition experiments, recognition was performed by the object recognizer located at the top of the hierarchy and using segments from the two highest segmentation levels. On the other hand, the variation verified many hypotheses due to the fact that very local segments were used for recognition. Also, noise affected the indexing scheme, giving rise to many false positive hypotheses.

Next, we tested the proposed approach using objects corrupted by occlusion. Occlusion was introduced in a random way. The amount of occlusion in an object contour was determined by the percentage of absent contour points. The test objects included in this experiment were also rotated, translated, and scaled. Our results show that recognition accuracy and number of hypotheses were not significantly affected with atmost 50 accuracy was 2–5% less in certain cases and that the number of hypotheses was increased by 5–10% approximately. With more than 50% occlusion, however, we observed that the recognition accuracy started to deteriorate rather fast and the number of hypotheses to increase considerably.

Finally, we tested our approach using a number of real scenes containing overlapping objects. Figs. 19a and 20a show two of the scenes used in our experimentation. In each case, we show the segments chosen by the method during recognition. In the case of scene shown in Fig. 19a, for example, the method chose some composite segments first (Fig. 19b–d). In most of the cases, ambiguous recognition results were reported by the object classifiers. In the rest of the cases, verification rejected the candidate matches. Fig. 19e shows the segment which yielded the successful recognition shown in Fig. 19f. Also, Fig. 19h shows the segment which yielded the recognition of the

model shown in Fig. 19i. The total number of hypotheses tested for the recognition of both models was 8. In the case of the scene shown in Fig. 20a, segments from the second segmentation level yielded correct recognition. This is because the zero-crossings detected at the first segmentation level did not form segments that were similar to these formed in the preprocessing step for the models shown in the scene. Thus, the number of hypotheses tested in this case was higher (42 hypotheses).

8. Conclusions

A new method for the recognition of two-dimensional objects has been presented in this paper. Recognition by the proposed method is invariant to object size, position, and orientation, it is insensitive to noise, and it is quite accurate even when objects are partially occluded. The method consists of a preprocessing step, a training phase and a recognition phase. The preprocessing step generates the hierarchy of contour segmentations for each model object, using the RCSSs of the object contours. In the training phase, a selective search procedure extracts important groups of segments from the various levels of the hierarchies. The model database is then built. It consists of two main components, both implemented using multilayer ANNs. The first component, called the segment classifier, assigns segments to an appropriate class. The second component consists of a group of ANNs which are structured hierarchically and called object recognizers. Each recognizer performs recognition using groups of segment classifications. Recognition operates hierarchically, driven by the RCSS based contour segmentation scheme. First, recognition is attempted using important groups extracted from high levels of the hierarchy of segmentations. If recognition is not possible at these levels, information from lower levels is utilized.

The proposed method has been tested using both artificial and real data illustrating good performance. Some comments are the following: first, although the hierarchi cal segmentation scheme is quite effective, it is also quite time consuming. In the present implementation, we have not tried to optimize it. However, faster implementations are possible using a number of heuristics [20]. Second, maintaining the model database is quite important. Assuming that the model database needs to be expanded by adding new models, our present implementation requires retraining of all the ANNs used. This is quite inefficient. However, there are ANN models which have the ability to store new knowledge without significantly destroying previous one. This problem is known as the "stability plasticity" problem and some architectures which have shown to deal with this problem are the Fuzzy-ARTMAP [30], and Cascade Correlation [29] architectures.

Acknowledgements

This research was supported by a grant (JFA) from the Research Advisory Board of the University of Nevada, Reno.

References

[1] T. Binford, Survey of model based image analysis systems, Int. J. Robotics Res. 1 (1) (1982) 18–63.
[2] R. Chin, C. Dyer, Model-based recognition in robot vision, Comput. Surveys 18 (1) (1986) 67–108.
[3] D. Huttenlocher, S. Ullman, Recognizing solid objects by alignment with an image. Int. J. Comput. Vision 5 (2) (1990) 195–212.
[4] E. Grimson, T. Lozano-Perez, Localizing overlapping parts by searching the interpretation tree, IEEE Trans. Pattern Anal. Machine Intell. 9 (4) (1987) 469–482.
[5] Y. Lamdan, J. Schwartz, H. Wolfson, Affine invariant model-based object recognition, IEEE Trans. Robotics Automation 6 (5) (1990) 578–589.
[6] A. Kalvin, E. Schonberg, J. Schwartz, M. Sharir, Two dimensional model based boundary matching using footprints. Int. J. Robotics Res. 5 (4) (1986) 38–55.
[7] T. Knoll, R. Jain, Recognizing partially visible objects using feature indexed hypotheses, IEEE J. Robotics Automation, RA-2 (1986) 3–13.
[8] R. Mehrotra, W. Grosky, Shape matching utilizing indexed hypotheses generation and testing, IEEE Trans. Robotics Automation 5 (1) (1989).
[9] F. Stein, G. Medioni, Structural indexing: efficient 2-D object recognition, IEEE Trans. Pattern Anal. Machine Intell. 14 (12) (1992) 1198–1204.
[10] I. Sethi, N. Ramesh, Local association based recognition of two-dimensional objects, Machine Vision Appl. 5 (1992) 265–276.
[11] J. Gorman, O. Mitchell, F. Kuhl, Partial shape recognition using dynamic programming, IEEE Trans. Pattern Anal. Machine Intell. 10 (2) (1988) 257–266.
[12] R. Mehrotra, F. Kung, W. Grosky, Industrial part recognition using a component-index, Image Vision Comput. 8 (3) (1990) 225–232.
[13] F. Mokhtarian, A. Mackworth, A theory of multiscale, curvature-based shape representation for planar curves. IEEE Trans. Pattern Anal. Machine Intell. 14 (8) (1992) 789–805.
[14] J. Hertz, A. Krogh, R. Palmer, Introduction to the Theory of Neural Computation, Addison-Wesley. Reading MA, 1991.
[15] K. Hornik, M. Stinchombe, Multilayer feed-forward networks are universal approximators, in: H. White et. al. (Eds.), Artificial Neural Networks: Approximation and Learning Theory, Blackwell Press, Oxford, 1992.
[16] H. Freeman, shape description via the use of critical points, Pattern Recognition 10 (1978) 159–166.
[17] G. Bebis, G. Papadourakis, Model based object recognition using invariant boundary contour representations and artificial neural networks, Pattern Recognition 25 (1) (1992) 25–44.

[18] L. Gupta, M. Srinath, Contour sequence moments for the classification of closed planar shapes, Pattern Recognition 20 (3) (1987) 267–272.
[19] L. Gupta, K. Malakapalli, Robust partial shape classifications by using invariant breakpoints and dynamic alignment, Pattern Recognition 23 (10) (1990) 1103–1111.
[20] F. Mokhtarian, A. Mackworth, Scale-based description and recognition of planar curves and two-dimensional shapes, IEEE Trans. Pattern Anal. Machine Intell. 8 (1) (1986) 34–43.
[21] D. Huttenlocher, S. Ullman, Object recognition using alignment, Proc. 1st Int. Conf. on Computer Vision, England 1987, pp. 102–111.
[22] S. Pei, C. Lin, The detection of dominant points on digital curves by scale-space filtering, Pattern Recognition, 25 (1) (1992) 1307–1314.
[23] G. Carpenter, Neural network models for pattern recognition and associative memory, Neural Networks 2 (1989) 243–257.
[24] G. Hinton, Connectionist learning procedures, Artificial Intell. 40 (1989) 187–233.
[25] D. Hush, B. Horne, Progress in supervised neural networks, IEEE Signal Process. Mag. (1993) 8–39.
[26] J. Tou, R. Gonzalez, Pattern Recognition Principles, Addison-Wesley Reading, MP, 1974.
[27] W. Chambers, Basics of Communications and Coding, Clarendon Press, Oxford, 1985.
[28] Z. You, A. Jain, Performance evaluation of shape matching via chord length distribution, Computer Vision Graphics Image Process. 28 (1984) 129–142.
[29] S. Fahlman, C. Lebiere, The cascade-correlation learning architecture, In: D. Touretzky (Ed.), Advances in Neural Information Processing Systems II, Morgan Kaufmann, San Mateo, 1990, pp. 524–532.
[30] G. Carpenter, S. Grossberg, N. Markuzon, J. Reynolds, B. Rosen, Fuzzy ARTMAP: a neural network architecture for incremental supervised learning of analog multidimensional maps, IEEE Trans. Neural Networks 3 (5) (1992) 698–713.

About the Author—GEORGE BEBIS received the B.S. degre in Mathematics and the M.S. degree in Computer Science from the University of Crete, Greece, in 1987 and 1991, respectively, and the Ph.D. degree in Electrical and Computer Engineering from the University of Central Florida, Orlando, in 1996. Currently, he is an Assistant Professor in the Department of Computer Science at the University of Nevada, Reno (UNR) and the director of the Computer Vision and Robotics Laboratoy (CVRL) at UNR. From 1996 until 1997 he was a Visiting Assistant Professor in the Department of Mathematics and Computer Science at the University of Missouri, St. Louis while from June 1998 to August 1998 he was a summer faculty in the Center for Applied Scientific Research (CASC) at Lawrence Livermore National Laboratory (LLNL). His research intrests include computer vision, image processing, artificial neural networks, and genetic algorithms. Dr Bebis has served on the program committees of several national and international conferences and has organized and chaired several conference sesstion. He is a member of the IEEE Society and the Program Chair of his IEEE local section.

About the Author—GEORGE M. PAPADOURAKIS was born in Patras Greece in 1959. He received the B.Sc. degree from the Michigan Technological University in 1978, his Master's degree from the University of Cincinnati in 1981, and his Ph.D. degree from the University of Florida in 1986, all in Electrical Engineering. From 1986 until 1988 he was an Assistant Professor in the Department of Computer Engineering at the University of Central Florida, Orlando, Florida. From 1988 until 1992 he was a Visiting Professor in the Department of Computer Science at the University of Crete in Heraklion, Greece. Since 1990 he has also been a researcher at the Institute of Computer Science-FORTH in Heraklion, Greece. In 1994 he became a Professor in the Computer Group of the Science Department at the Technological Education Institute of Heraklion in Greece. His current research interests include neural network applications, VLSI design and computer architecture. He has authored over 14 Journal and 40 International Conference publications. He is a member of IEEE, the International Neural Network Society, the European Neural Network Society, Technical Chamber of Greece and the Computer Society of Greece.

About the Author—STELIOS ORPHANOUDAKIS received the B.A. degree in engineering scienes from Dartmouth College, Hanover, NH, in 1971, the M.S. degree in electrical engineering from M.I.T. Cambridge, MA, in 1973, and the Ph.D. degree in electrical engineering from Dartmouth College in 1976. He is Director of the Institute of Computer Science, Foundation for Research and Technology-Hellas, and Professor of Computer Science, University of Crete, Greece. He held a faculty appointment in the Departments of Diagnostic Radiology and Electrical Engineering at Yale University, USA, from 1975 until 1991. Prof. Orphanoudakis has many years of academic and research experience in the fields of computer vision and robotics, intelligent image management and retrieval by content, and medical imaging. He has served on various committees and working groups of the European Commission and has been active in European RD programs. He currently serves on the Board of Directors and is Vice President of the European Research Consortium for Informatics and Mathematics (ERCIM). He is also a member of the National Telecommunications Commission and the National Advisory Research Council of Greece.