Closed-Loop Object Recognition Using Reinforcement Learning

Jing Peng and Bir Bhanu, *Fellow, IEEE*

Abstract—Current computer vision systems whose basic methodology is open-loop or *filter* type typically use image segmentation followed by object recognition algorithms. These systems are not robust for most real-world applications. In contrast, the system presented here achieves robust performance by using reinforcement learning to induce a mapping from input images to corresponding segmentation parameters. This is accomplished by using the confidence level of model matching as a reinforcement signal for a *team of learning automata* to search for segmentation parameters during training. The use of the recognition algorithm as part of the evaluation function for image segmentation gives rise to significant improvement of the system performance by automatic generation of recognition strategies. The system is verified through experiments on sequences of indoor and outdoor color images with varying external conditions.

Index Terms—Adaptive color image segmentation, function optimization, generalized learning automata, learning in computer vision, model-based object recognition, multiscenario recognition, parameter learning, recognition feedback, segmentation evaluation.

1 Introduction

IMAGE segmentation, feature extraction, and model matching are the key building blocks of a computer vision system for model-based object recognition [8], [23]. The tasks performed by these building blocks are characterized as the low (segmentation), intermediate (feature extraction), and high (model matching) levels of computer vision. The goal of image segmentation is to extract meaningful objects from an image. It is essentially a pixel-based processing. Model matching uses a representation such as shape features obtained at the intermediate level for recognition. It requires explicit shape models of the object to be recognized. There is an abstraction of image information as we move from low to high levels and the processing becomes more knowledge-based or goal-directed.

Although there is an abundance of proposed computer vision algorithms for object recognition, there have been few systems that achieve good performance for practical applications, for most such systems do not adapt to changing environments [2]. The main difficulties, typically associated with systems that are mostly open-loop or *filter* type, can be characterized as follows.

1) The fixed set of parameters used in various vision algorithms often leads to ungraceful degradation in performance.
2) The image segmentation, feature extraction, and selection are generally carried out as preprocessing steps to object recognition algorithms for model matching. These steps totally ignore the effects of the earlier results (image segmentation and feature extraction) on the future performance of the recognition algorithm.
3) Generally, the criteria used for segmentation and feature extraction require elaborate human designs. When the conditions for which they are designed are changed slightly, these algorithms fail. Furthermore, the criteria themselves can be a subject of debate [3].
4) Object recognition is a process of making sequences of decisions, i.e., applying various image analysis algorithms. Often the usefulness of a decision or the results of an individual algorithm can *only* be determined by the final outcome (e.g., matching confidence) of the recognition process. This is also known as "vision-complete" problem [7], i.e., one cannot really assign labels to the image without the knowledge of which parts of the image correspond to what objects.

This paper presents a learning-based vision framework in which the above problems can be adequately addressed. The underlying theory is that any recognition system whose decision criteria for image segmentation and feature extraction, etc., are developed autonomously from the outcome of the final recognition might transcend all these problems. A direct result of the theory is that the low- and high-level components of a vision system must interact to achieve robust performance under changing environmental conditions. Our system accomplishes this by incorporating a reinforcement learning mechanism to control the interactions of different levels within it. Specifically, the system takes the output of the recognition algorithm and uses it as a feedback to influence the performance of the segmentation process. As a result, the recognition performance can be significantly improved over time with this method.

• *The authors are with the College of Engineering, University of California, Riverside, CA 92521. E-mail: {jp, bhanu}@vislab.ucr.edu.*

Manuscript received 11 Apr. 1996; revised 1 Dec. 1997. Recommended for acceptance by L. Shapiro.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 106038.

One attractive feature of the approach is that it includes the matching or recognition component as part of the evaluation function for image segmentation in a systematic way. An additional strength is that the system develops its independent decision criteria (segmentation parameters) to best serve the underlying recognition task. It should be emphasized that our interest is not in a simple mixture of learning and computer vision, but rather in the principled integration of the two fields at the algorithmic level. Note that the goal here is to seek a general mapping from images to parameter settings of various algorithms based on recognition results. To our knowledge, however, no such approach exists in the computer vision field. Also, there is no work in the neural network field (e.g., application of Neocognition [10]) for parameter adaptation of segmentation algorithms [3].

This work is most closely related to the work by Bhanu et al. [3], [5], [6], where they describe a system that uses genetic and hybrid algorithms for learning segmentation parameters. However, the recognition algorithm is not part of the evaluation function for segmentation in their system. The genetic or hybrid algorithms simply search for a set of parameters that optimizes a prespecified evaluation function (based on global and local segmentation evaluation) that may not best serve the overall goal of robust object recognition. Furthermore, the papers assume that the location of the object in the image is known for specific photointerpretation application. In our work, we do not make such an assumption. We use explicit geometric model of an object, represented by its polygonal approximation, to recognize it in the image.

In addition, Wang and Binford [24] and Ramesh [21] have investigated statistical methods for performance evaluation and tuning free parameters of an algorithm. Wang and Binford [24] presented a theoretical analysis for edge estimation and showed how one can select the gradient threshold (tuning parameter) for edge detection. Ramesh [21] has developed a methodology for the analysis of computer vision algorithms and systems using system engineering principles. To characterize the performance of an algorithm he developed statistical models for ideal image features (such as edges and corners) and random perturbations at input/output of an algorithm. Additionally, prior distributions for image features are also obtained. Using these models and a criterion function, he can characterize the performance of a given algorithm as a function of tuning parameters and determine these parameters automatically. Our approach presented in this paper differs significantly from Ramesh's [21] approach.

1) Ramesh's approach is open loop, while our approach is closed loop. In our approach, recognition results determine how the segmentation parameters should be changed.
2) Ramesh is tuning the parameters of an individual algorithm—it is known that the optimization of individual components does not necessarily give the optimal results for the system. We are working with a complete recognition system (segmentation, feature extraction, and model matching components) and improving the performance of the complete system.
3) Ramesh builds elaborate statistical models (using the training data) that require complex processes of annotation and approximating the measured distributions with mathematical functions to be used later. Our learning approach does not build explicit statistical models. It uses geometrical models during model matching.
4) It is relatively easier to build statistical models for algorithms like edge and corner detection. For *complex* algorithms like *Phoenix*, it is difficult to model the "perfect" algorithm behavior analytically, since the performance of segmentation depends nonlinearly with the changes in parameter values, and there are some heuristics used in the algorithm. Considering the above factors, our approach is more general for the problem that we are trying to solve. We have developed a learning-based approach presented in this paper.

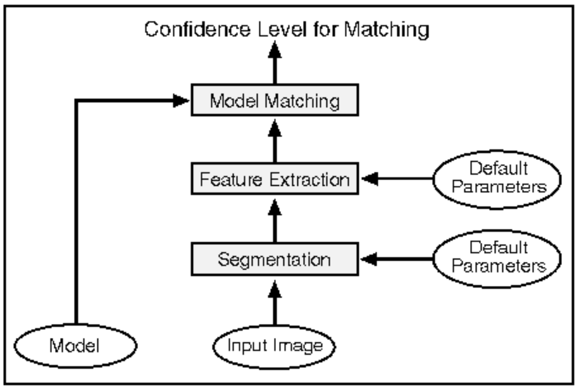


Fig. 1. Conventional multilevel system for object recognition.

Section 2 describes a general framework for reinforcement learning-based adaptive image segmentation. Section 3 describes the reinforcement learning paradigm and the particular reinforcement learning algorithm employed in our system. Section 4 presents the experimental results evaluating the system, and Section 5 concludes the paper. Two appendices describe the basic segmentation and model matching algorithms used to perform experiments for closed-loop object recognition using reinforcement learning.

2 Reinforcement Learning System for Segmentation Parameter Estimation

2.1 The Problem

Consider the problem of recognizing an object in an input image, assuming that the model of the object is given, and that the precise location of the object in the image is unknown. The conventional method for the recognition problem, shown in Fig. 1, is to first segment the input image, then extract and select appropriate features from the segmented image, and finally perform model matching using these features. If we assume that the matching algorithm produces a real valued output indicating the degree of success upon its completion, then it is natural to use this real valued output as feedback to influence the performance of segmentation and feature extraction, so as to bring about system's earlier decisions favorable for more accurate model matching. The rest of the paper describes a reinforcement learning-based vision system to achieve just that.

2.2 Learning to Segment Images

Our current investigation into reinforcement learning-based vision systems is focused on the problem of learning to segment images. An important characteristic of our approach is that the segmentation process takes into account the biases of the recognition algorithm to develop its own decision strategies. A consequence of this is that the effective search space of segmentation parameters can be dramatically reduced. As a result, more accurate and efficient segmentation and recognition performance can be expected.

2.2.1 Image Segmentation

We begin with image segmentation [13], because it is an extremely important and difficult low-level task. All subsequent interpretation tasks including object detection, feature extraction, object recognition, and classification rely heavily on the quality of the segmentation process. The difficulty arises for image segmentation when only local image properties are used to define the region-of-interest for each individual object. It is known [2], [9] that correct localization may not always be possible. Thus, a good image segmentation cannot be done by grouping parts with similar image properties in a purely bottom-up fashion. Difficulties also arise when segmentation performance needs to be adapted to the changes in image quality, which is affected by variations in environmental conditions, imaging devices, lighting, etc. The following are the key characteristics [3] of the image segmentation problem:

1) When presented with a new image, selecting the appropriate set of algorithm parameters is the key to effectively segmenting the image.
2) The parameters within most segmentation algorithms typically interact in a complex, nonlinear fashion, which makes it difficult to model the parameters' behavior analytically.
3) The variations between images cause changes in the segmentation results, the objective function that represents segmentation quality varies from image to image. Also, there may not be a consensus on segmentation quality measures.

2.2.2 Our Approach

Each combination of segmentation parameters produces, for a given input, a unique segmentation image from which a confidence level of model matching can be computed. The simplest way to acquire high payoff parameter combinations is through trial and error. That is, generate a combination of parameters, compute the matching confidence, generate another combination of parameters, and so on, until the confidence level has exceeded a given threshold. Better yet, if a well-defined evaluation function over the segmentation parameter space is available, then local gradient methods, such as hill-climbers, suffice. While the trial-and-error methods suffer from excessive demand for computational resources, such as time and space, the gradient methods suffer from the unrealistic requirement for an evaluation function. In contrast, reinforcement learning performs trials and errors, yet does not demand excessive computational resources; it performs hill-climbing in a statistical sense, yet does not require an evaluation function. In addition, it can generalize over unseen images as we shall see later. Furthermore, it can be easily adapted to multilevel computer vision systems. It is also feasible to construct fast, parallel devices to implement this technique for real-time applications. It thus fits our goal nicely here.

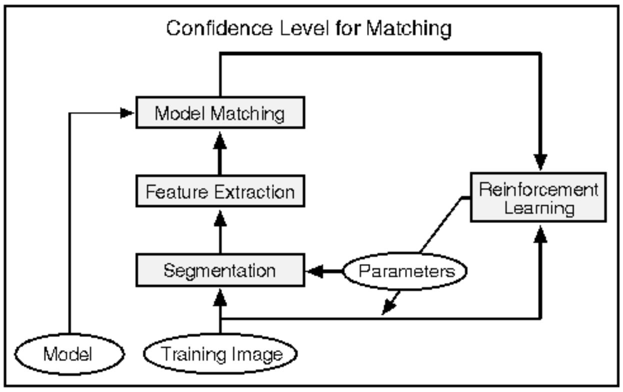


Fig. 2. Reinforcement learning-based multilevel system for object recognition.

```
• LOOP:
    1. $rr = 0$ ($rr$: average matching confidence)
    2. For each image $i$ in the training set do
      (a) Segment image $i$ using current segmentation
          parameters
      (b) Perform noise clean up
      (c) Get segmented regions (also called blobs or con-
          nected components)
      (d) Perform feature extraction for each blob to obtain
          token sets
      (e) Compute the matching of each token set against
          stored model and return the highest confidence
          level, $r$
      (f) $rr = rr + r$
      (g) Obtain new parameters for the segmentation algo-
          rithm using $r$ as reinforcement for the reinforcement
          learning algorithm
• UNTIL number of iterations is equal to N or $rr/n \geq R_{th}$
```

Fig. 3. Main steps of the reinforcement learning-based object recognition algorithm.

Fig. 2 depicts the conceptual diagram of our reinforcement learning-based object recognition system that addresses the parameter selection problem encountered in the image segmentation task by using the recognition algorithm itself as part of the evaluation function for image segmentation. Note that the reinforcement learning component employs a particular reinforcement learning algorithm that will be described in the next section. Fig. 3 shows the main steps of the algorithm we use, where the algorithm terminates when either the number of iterations reaches a prespecified value (N) or the average matching confidence over entire training data (denoted by rr) has exceeded a given threshold, called R_{th}. Note that n denotes the number of images in the training set. In the event that the number of iterations has exceeded N, we will say that the object is not present in the image. Also, for simplicity, we assume that only one instance of the model is present in the image. Multiple instances of the model can be recognized by slight modification of the algorithm.

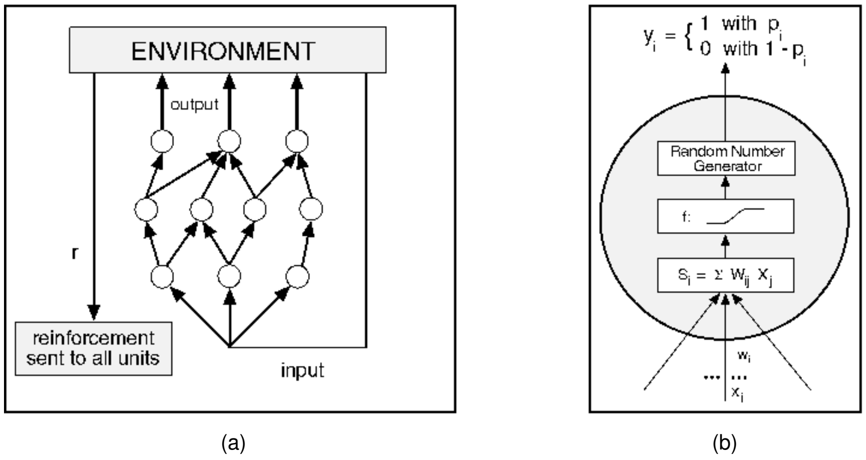

ENVIRONMENT
output
r
reinforcement sent to all units
input
(a)
$y_i = \begin{cases} 1 \text{ with } p_i \\ 0 \text{ with } 1 - p_i \end{cases}$
Random Number Generator
f:
$S_i = \Sigma\, W_{ij}\, X_j$
w_i
x_i
(b)

Fig. 4. (a) Connectionist reinforcement learning system. (b) Bernoulli quasilinear unit.

3 Reinforcement Learning

In this section, we begin with a brief overview of the reinforcement learning technique. We then describe reinforcement learning algorithms applicable to our task and the modifications of these algorithms to effectively solve the problem identified in Section 2.1.

Reinforcement learning is an important machine learning paradigm. It is a framework for learning to make sequences of decisions in an environment [1]. It is distinct from supervised learning, like the popular backpropagation algorithm, in that feedback it receives is evaluative instead of instructive. That is, for supervised learning, the system is presented with the correct output for each input instance, while for reinforcement learning, the system produces a response that is then evaluated using a scalar indicating the appropriateness of the response. As an example, a chess playing computer program that uses the outcome of a game to improve its performance is a reinforcement learning system. Knowledge about an outcome is useful for evaluating the total system's performance, but it says nothing about which actions were instrumental for the ultimate win or loss. In general, reinforcement learning is more widely applicable than supervised learning, since any supervised learning problem can be treated as a reinforcement learning problem.

In the reinforcement learning framework, a learning system is given, at each time step, inputs describing its environment. The system then makes a decision based on these inputs, thereby causing the environment to deliver to the system a reinforcement. The value of this reinforcement depends on the environmental state, the system's decision, and possibly random disturbances. In general, reinforcement measuring the consequences of a decision can emerge at a multitude of times after a decision is made. A distinction can be made between associative and nonassociative reinforcement learning. In the nonassociative paradigm, reinforcement is the only information the system receives from its environment. Whereas, in the associative paradigm, the system receives input information that indicates the state of its environment as well as reinforcement. In such learning systems, a "state" is a unique representation of all previous inputs to a system. In computer vision, this state information corresponds to current input image. Our object recognition applications require us to take into account the changes appearing in the input images. The objective of the system is to select sequences of decisions to maximize the sum of future reinforcement (possibly discounted) over time. It is interesting to note that, for a given state, an associative reinforcement learning problem becomes a nonassociative learning problem.

As noted above, a complication to reinforcement learning is the timing of reinforcement. In simple tasks, the system receives, after each decision, reinforcement indicating the goodness of that decision. Immediate reinforcement occurs commonly in function optimization problems. In more complex tasks, however, reinforcement is often temporally delayed, occurring only after the execution of a sequence of decisions. Delayed reinforcement learning is important because, in many problem domains, immediate reinforcement regarding the value of a decision may not always be available. For example, in object recognition, the goodness of segmentation is not known until the recognition decision has been made. Delayed reinforcement learning is attractive and can play an important role in computer vision [20]. Because delayed reinforcement learning does not concern us here, we do not discuss this subject further.

In this paper, we instead concentrate on the immediate reinforcement learning paradigm, for it provides a simple, yet principled framework within which the main problems identified above can be properly addressed. It also serves as a stepping stone for better understanding of the issues involved in computer vision that need to be addressed by delayed reinforcement learning [20]. A well-understood method in immediate reinforcement learning is the REINFORCE algorithm [25], a class of connectionist reinforcement learning algorithms, that performs stochastic hill-climbing, and which is the subject of our paper.

3.1 Connectionist Reinforcement Learning

The particular class of reinforcement learning algorithms employed in our object recognition system is the connectionist REINFORCE algorithm [25], where units in such a network (depicted in Fig. 4a) are *Bernoulli quasilinear units*, in that the output of such a unit is either 0 or 1, determined

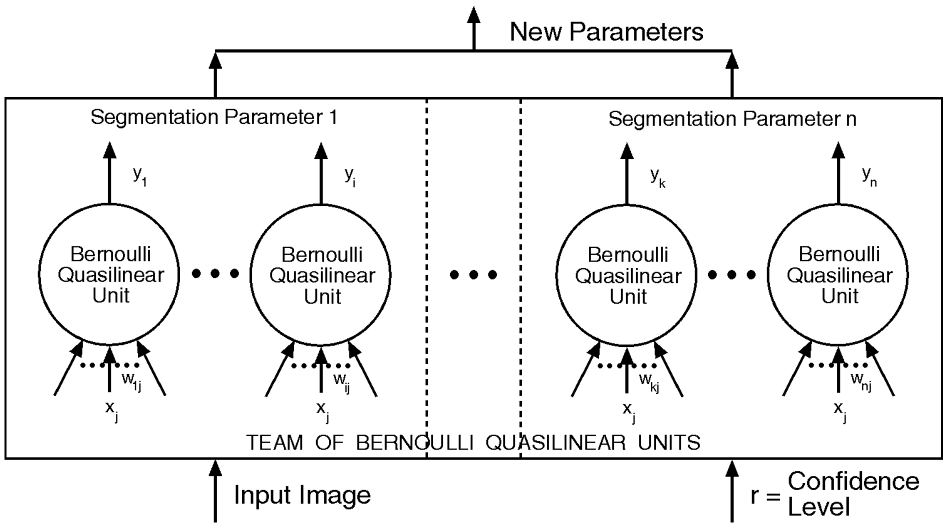


Fig. 5. Team of Bernoulli units for learning segmentation parameters.

stochastically using the Bernoulli distribution with parameter $p = f(s)$, where f is the logistic function,

$$f(s) = 1/(1 + exp(-s)) \tag{1}$$

and $s = \sum_i w_i x_i$ is the usual weighted summation of input values to that unit. For such a unit, p represents its probability of choosing one as its output value. Fig. 4b depicts the ith unit.

In the general reinforcement learning paradigm, the network generates an output pattern and the environment responds by providing the reinforcement r as its evaluation of that output pattern, which is then used to drive the weight changes according to the particular reinforcement learning algorithm being used by the network. For the Bernoulli quasilinear units used in this research, the REINFORCE algorithm prescribes weight increments equal to

$$\Delta w_{ij} = \alpha(r - b)(y_i - p_i)x_j \tag{2}$$

where α is a positive learning rate, b serves as a *reinforcement baseline*, x_j is the input to each Bernoulli unit, y_i is the output of the ith Bernoulli unit, and p_i is an internal parameter to a Bernoulli random number generator (see (1)). Note that i takes values from one to n and j from one to m, where n and m are the number of the units in the network and the number of input features, respectively.

It can be shown [25] that, regardless of how b is computed, whenever it does not depend on the immediately received reinforcement value r, and when r is sent to all the units in the network, such an algorithm satisfies

$$E\{\Delta \mathbf{W} \mid \mathbf{W}\} = \alpha \nabla_{\mathbf{W}} E\{r \mid \mathbf{W}\} \tag{3}$$

where E denotes the expectation operator, $\mathbf{W}$ represents the weight matrix ($n \times (m + 1)$, $m + 1$ because of m inputs plus a bias) of the network, and $\Delta\mathbf{W}$ is the change of the weight matrix. A reinforcement learning algorithm satisfying (3) has the property that the algorithm statistically climbs the gradient of expected reinforcement in weight space. That is, the algorithm is guaranteed to converge to a local optimum. For adapting parameters of the segmentation algorithm, it means that the segmentation parameters change in the direction along which the expected matching confidence increases. The next two subsections describe the particular network and the algorithm used in this paper.

3.2 The Team Architecture

We use a form of trial generating network in which all of the units are output units and there are no interconnections between them. This degenerate class of network corresponds to what is called a *team* of automata in the literature on stochastic learning automata [18]. We, therefore, call these networks as *teams of Bernoulli quasilinear units*. The main thrust of the architecture is its simplicity and its generality as a function approximator. Fig. 5 depicts the team network used here, which corresponds directly to the reinforcement learning component in Fig. 2. Each segmentation parameter is represented by a set of Bernoulli quasilinear units, and the output of each unit is binary as we have described earlier.

For any Bernoulli quasilinear unit, the probability that it produces a 1 on any particular trial given the value of the weight matrix $\mathbf{W}$ is

$$\Pr\{y_i = 1 | \mathbf{W}\} = p_i = f(s_i) = \frac{1}{1 + e^{-s_i}}$$

where $s_i = \sum_j w_{ij} x_j$. Because all units pick their outputs independently, it follows that, for such a team of Bernoulli quasilinear units, the probability of any particular output vector $\mathbf{y}(t)$, corresponding to an instance of segmentation parameters, conditioned on the current value of the weight matrix $\mathbf{W}$ is given by

$$\Pr\{\mathbf{y} \mid \mathbf{W}\} = \prod_{i \in \{1,\cdots,n\}} p_i^{y_i}(1 - p_i)^{1-y_i}. \tag{4}$$

The weights w_{ij} are adjusted according to the particular learning algorithm used. We note that when $s_i = 0$ and, hence, $p_i = 0.5$, the unit is equally likely to pick y_i either 0 or 1, while increasing s_i makes a 1 more likely. Adjusting the weights in a team of Bernoulli quasilinear units is thus tantamount to adjusting the probabilities (p_is) for individual units.

Note that, except bias terms, there are no input connections in the team networks experimented in [26]. In contrast, the team network used in this paper does have input weights that play the role of long-term memory in associative learning tasks.

3.3 The Team Algorithm

The specific algorithm we used with the team architecture has the following form: At the *t*th time step, after generating output $\mathbf{y}(t)$ and receiving reinforcement $r(t)$, i.e., the confidence level indicating the matching result, increment each weight w_{ij} by

$$\Delta w_{ij}(t) = \alpha\big(r(t) - \bar{r}(t-1)\big)\big(y_i(t) - \bar{y}_i(t-1)\big)x_j - \delta w_{ij}(t) \tag{5}$$

where α, the learning rate, and δ, the weight decay rate, are parameters of the algorithm. The term $\big(r(t) - \bar{r}(t-1)\big)$ is called the *reinforcement factor* and $\big(y_i(t) - \bar{y}_i(t-1)\big)$ the *eligibility* of the weight w_{ij} [25]. Generally, the eligibility of a weight indicates the extent to which the activity at the input of the weight was connected in the past with unit output activity. Note that this algorithm is a variant of the one described in (2), where b is replaced by $\bar{r}$ and p_i by $\bar{y}_i$.

$\bar{r}(t)$ is the exponentially weighted average, or *trace*, of prior reinforcement values

$$\bar{r}(t) = \gamma\,\bar{r}(t-1) + (1-\gamma)r(t) \tag{6}$$

with $\bar{r}(0) = 0$. The trace parameter γ was set equal to 0.9 for all the experiments reported here. Similarly, $\bar{y}_i(t)$ is an average of past values of y_i computed by the same exponential weighting scheme used for $\bar{r}$. That is,

$$\bar{y}_i(t) = \gamma\,\bar{y}_i(t-1) + (1-\gamma)y_i(t). \tag{7}$$

Note that (3) does not depend on the eligibility. However, empirical study shows superior performance with this form of eligibility for function optimization [26].

The use of weight decay is chosen as a simple heuristic method to force sustained exploration of the weight space since it was found that REINFORCE algorithms without weight decay always seemed to converge prematurely. It is argued in [26] that having weight decay (the second term $\delta w_{ij}(t)$ in (5)) is very closely related to having a nonzero mutation rate at a particular allele (feature value) in a genetic algorithm [11]. The size of the weight decay rate δ was chosen to be 0.01 in all our experiments. Note that there are other ways to force sustained exploration. One possibility is to maximize a linear combination of system's entropy and reinforcement. We omit here the detailed analysis of the method except commenting that such a strategy seeks not only a particular region of the space having high reinforcement values, but also a variety of such high value regions.

3.4 Implementation of the Algorithm

A different training strategy from that described in Fig. 3 was used in the experiments reported here. Instead of looping through every image in the training set, the training procedure samples images proportional to the level of matching confidence the current system achieves. That is, the lower the matching confidence the system gets on an image, the more likely the image will be sampled. In this way training is focused on those images having the lowest matching confidence, and thus faster performance improvement can be achieved. Fig. 6 shows the main steps of the proportional training algorithm, where *MAXCONFID* (= 1 in this paper) is the maximum confidence level the system can achieve, i.e., when a perfect matching occurs, n is the number of images in the training set, and N and R_{th} are input parameters to the algorithm.

```
• LOOP:
    1. For each image $i$ in the training set do
    (a) Compute matching confidence for image $i$: $CONFID_i$
    (b) $n_i = MAXCONFID - CONFID_i$
    (c) If $\sum_i n_i$ is 0, then terminate.
    (d) $proportion_i = \frac{n_i}{\sum_i n_i}$
    2. $rr = 0$ ($rr$: average matching confidence)
    3. For $k = 1$ to $n$ do
    (a) Sample image $i$ according to $proportion_i$
    (b) Segment image $i$ using current segmentation parameters
    (c) Perform noise clean up
    (d) Get segmented regions (also called blobs or connected components)
    (e) Perform feature extraction for each blob to obtain token sets
    (f) Compute the matching of each token set against stored model and return the highest confidence level, $r$
    (g) Obtain new parameters for the segmentation algorithm using $r$ as reinforcement for the team REINFORCE algorithm
    (h) $rr = rr + r$
• UNTIL number of iterations is equal to N or $rr/n \geq R_{th}$
```

Fig. 6. Main steps of the proportional training algorithm.

4 EXPERIMENTAL RESULTS

This section describes experimental results evaluating the performance of our system on a variety of data, including a set of synthetic images, two sets of color images, one of which is indoor and the other is outdoor, and a large set of simulated data. The system has been implemented on a SUN Ultra-1 workstation. For the real images the segmentation algorithm takes about one quarter of per iteration time. Programming optimizations can reduce the expense per iteration further.

4.1 Evaluation on Synthetic Images

In order to give insights into our approach this section uses synthetic images with controlled statistics to demonstrate that the proposed technique is indeed capable of learning correct segmentation parameters. An 80 × 80 image consisting of a target region against a background is generated. The size of the target is 40 × 40 pixels. Both the background and the target are generated by two Gaussian distributions with $\mu_b = 130$ and $\mu_t = 145$, and standard deviations $\sigma_b = \sigma_t = \sigma = 2$, respectively. In this case, it is possible to analytically compute a theoretical threshold [12] that minimizes a total pixel misclassification error according to

$$T_{th} = \frac{\mu_b + \mu_t}{2} + \frac{\sigma^2}{\mu_b - \mu_t}\ln\frac{P_t}{P_b}, \tag{8}$$

where P_b and P_t are the a priori probabilities of background and target pixels, respectively. For this image, $T_{th} = 138$. To

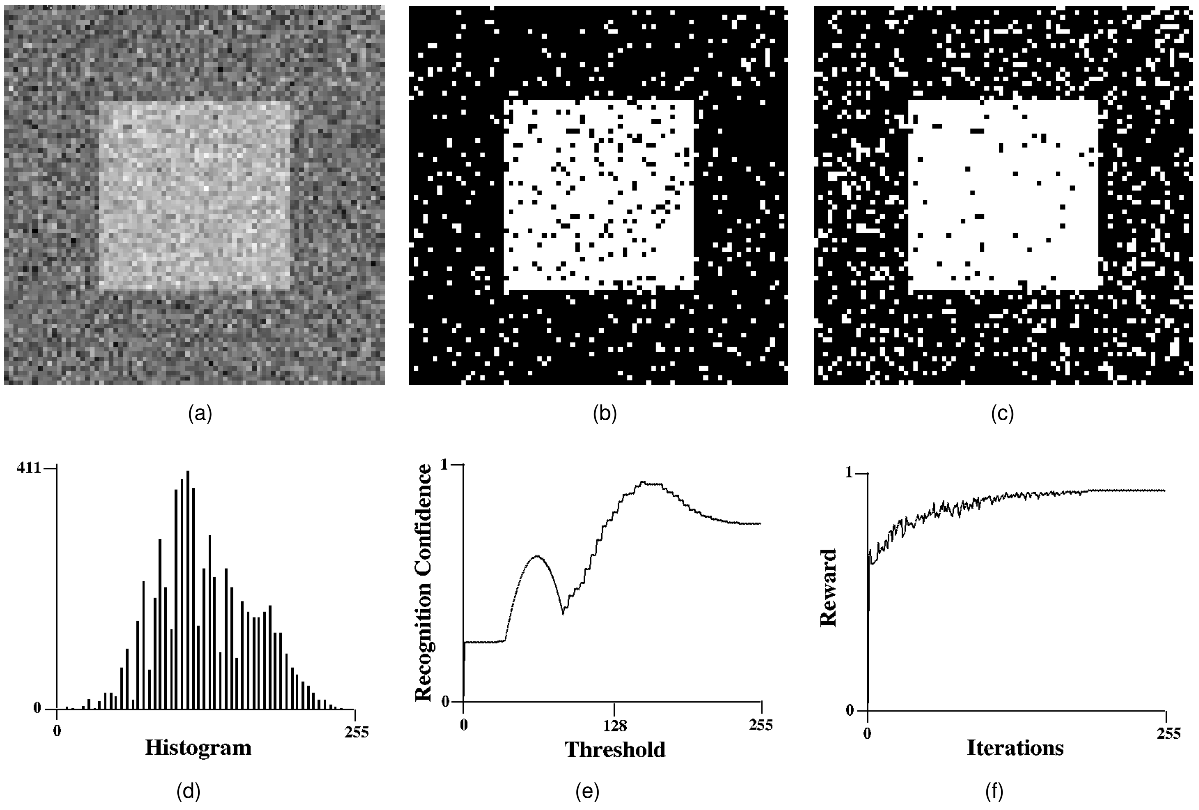


Fig. 7. (a) A noisy synthetic image (*SNR* = 10). (b) Learned segmentation. (c) Theoretical segmentation. (d) Histogram of image in (a). (e) Recognition confidence as a function of threshold. (f) Recognition confidence received over time.

complicate the matter, however, Gaussian random noise $N(0, \sigma_n)$ with a signal-to-noise ratio ($SNR = ((\mu_s - \mu_b)/\sigma_n)^2$) 10 is generated and added to this image. Pixel values are then linearly scaled from 0 to 255. The goal is to recognize the target. Fig. 7a shows the image and Fig. 7d its histogram.

The segmentation algorithm used in this experiment has one parameter $T \in [0, 255]$ and does the following to an input image: For each pixel, if its value is greater than or equal to T, it is set to 1, otherwise to 0. It is the parameter T that has to be learned by the system. The "recognition algorithm" we use in this subsection computes a confidence value according to

$$r = \frac{n(I) - \left(\left(n(G) - n(G \cap R)\right) + \left(n(R) - n(R \cap G)\right)\right)}{n(I)} \qquad (9)$$

where G is the region of the ground truth (target), R the region of pixels having a value of 1, and I is the input image. Function $n(\cdot)$ returns the number of pixels of its argument. r is then used as reinforcement to drive learning. Note that (9) happens to be one (pixel classification) of possible criteria for image segmentation [3]. Fig. 7e plots r (9) against the threshold. A local maximum is added to make the problem more interesting.

Eight Bernoulli units are used to encode the threshold (T_l) to be learned. It took on average (over 50 runs) fewer than 100 iterations (*alpha* = 0.2) to learn a threshold (152) that achieves the optimal recognition confidence of 0.92. Fig. 7b shows the segmentation result and Fig. 7f the average r value received over time. As a comparison, Fig. 7c shows the segmentation using the theoretical threshold (138) that achieves a confidence value of 0.87, which is far from the optimal one. It is important to note that optimality is determined by (9), i.e., the pixel classification. When the Gaussian random noise is removed, the learned threshold and the theoretical one are identical for the image in Fig. 7a.

To further evaluate our technique, 50 images were generated as training data having the following statistics: μ_t varied uniformly from 145 to 155, the size of the square target varied from 25 × 25 to 35 × 35 and the *SNR* from 10 to 15. An additional set of 500 images was generated independently as testing data, whose statistics are as follows. μ_t varied from 140 to 160 uniformly, the size of the target from 20 × 20 to 40 × 40, and the *SNR* from three to 20. The best achievable average r value (experimentally determined) on the training data is 0.971, whereas this value is 0.941 on the testing data.

Each unit in the team network has a total of four input weights, each of which takes an average pixel value of input on a 40 × 40 quadrant on the input image plane. Learning consists of repeated sweeps through the training set until the average value of r has reached 0.95 (approximately 100 sweeps). The network is then applied to the testing data. The result shows that the system achieved the nearly optimal average r value of 0.939 over the testing data. In contrast, the theoretical threshold[1] achieved an average r value of 0.924, which is worse than that using the learned threshold.

1. Note that known statistics from each testing image is used to compute the threshold based on (8). The learning system, however, does not have access to such knowledge.

These results show convincingly that the system can indeed learn correct segmentation parameters and that when images are far from being ideal the learning system can actually outperform the theoretical method, at least for those images presented here.

4.2 Evaluation on Real Images

For the color images the *Phoenix* algorithm [15] was chosen as the image segmentation component in our system because it is a well-known method for the segmentation of color images with a number of adjustable parameters. It has been the subject of several PhD theses [19], [22]. *Phoenix* works by splitting regions using histogram for color features. Appendix A provides a brief overview of the algorithm. Note that any segmentation algorithm with adjustable parameters can be used in our approach.

The *Phoenix* algorithm has a total of fourteen adjustable parameters. The four most critical ones that affect the overall results of the segmentation process are used in learning. These parameters are *Hsmooth*, *Maxmin*, *Splitmin*, and *Height*. *Hsmooth* is the width of the histogram smoothing window, where smoothing is performed with a uniformly weighted moving average. *Maxmin* defines the peak-to-valley height ratio threshold. Any interval whose peak height to higher shoulder ratio is less than this threshold is merged with the neighbor on the side of the higher shoulder. *Splitmin* defines the minimum size for a region to be automatically considered for splitting. This is an absolute value, not a percentage of the image area. *Height* is the minimum acceptable peak height as a percentage of the second highest peak. The team algorithm searches for a combination of these parameters that will give rise to a segmentation from which the best recognition can be achieved. The ranges for each of these parameters are the same as those used in [3]. Table 1 shows sample ranges for each of these parameters. The resulting search space is about one million sample points.

Each of the *Phoenix* parameters is represented using five-bit Gray code that has the advantage over simple binary code in that only one bit changes between representations of two consecutive numbers. One reason for using the binary representation is its usefulness as a model of certain types of distributed adaptive decision-making [25]. Another reason is that it offers a combinatorially advantageous way of approaching learning problems having a large search space. While the same task could be learned in the original parameter space, for many types of problems, including image segmentation, the binary representation can be expected to learn much faster. Since there are four parameters, we have a total of 20 Bernoulli quasilinear units and each parameter corresponds to the outputs of five units.

The feature extraction consists of finding polygon approximation tokens for each of the regions obtained after image segmentation. The polygon approximation is obtained using a split and merge technique [4] that has a fixed set of parameters.

Object recognition employs a cluster-structure matching algorithm [4] that is based on the clustering of translational and rotational transformations between the object and the model for recognizing 2D and 3D objects. A brief description of the algorithm is given in Appendix B. The algorithm takes as input two sets of tokens, one of which represents the stored model and the other represents the input region to be recognized. It then performs topological matching between the two token sets and computes a real number that indicates the confidence level of the matching process. This confidence level is then used as a reinforcement signal to drive the team algorithm.

TABLE 1
SAMPLE RANGES FOR SELECTED PHOENIX PARAMETERS

Parameter	Sampling Formula	Test Range
Hsmooth hsindex $\in$ [0 : 31]	hsmooth = 1 + 2 * hsindex	1 – 63

Maxmin:
mmindex $\in$ [0 : 31]
	ep = ln(100) + 0.05 * mmindex
	maxmin = exp(ep) + 0.5
		100 – 471

Splitmin: smindex $\in$ [0 : 31]	splitmin=9 + 2 * smindex	9 – 71
Height: htindex $\in$ [0 : 31]	height=1 + 2 * htindex	1 – 63

It is important to note that, in the current implementation of the system, the cluster-structure matching algorithm does not have the knowledge of actual object location in the image. It simply attempts to match the stored model against the polygonal approximation of each blob in the segmented image whose size is at least 80 percent of the size of the model, and at the same time does not exceed it by more than 20 percent. The confidence level returned is the highest value ever obtained during matching.

It is worth pointing out that, during learning, the weights are updated after each presentation of an input image. This is in direct analogy to the typical weight update procedure in connectionist networks where weights are updated according to the stochastic gradient or incremental procedure instead of the total gradient rule [16]. That is, updates take place after each presentation of a single exampler without averaging over the whole training set. Both empirical and theoretical studies show that the stochastic gradient rule converges significantly faster than the total gradient rule, especially when training set contains redundant information.

Parameters (α, γ, and δ) used in reinforcement learning are determined empirically, and they are kept constant for all images. It is interesting to note that in theory the convergence of the algorithm to a local optimum does not depend on γ and δ. In practice, however, these learning parameters do affect the speed of convergence, as shown by various empirical studies conducted by several researchers [25], [26], including us. Likewise, α has to be chosen sufficiently small to prevent oscillation and ensure convergence. The experimental tests performed by us showed that once the algorithm has achieved convergence many of these parameter values give rise to good segmentation performance, as verified by us visually. The initial parameter values for the *Phoenix* algorithm are chosen at random. We expect, however, that the good starting values of the segmentation parameters affect the convergence rate. Finally, as a comparison, the segmentation results with the *Phoenix* algorithm using default parameters [15] are also obtained for feature extraction and recognition on the same tasks.

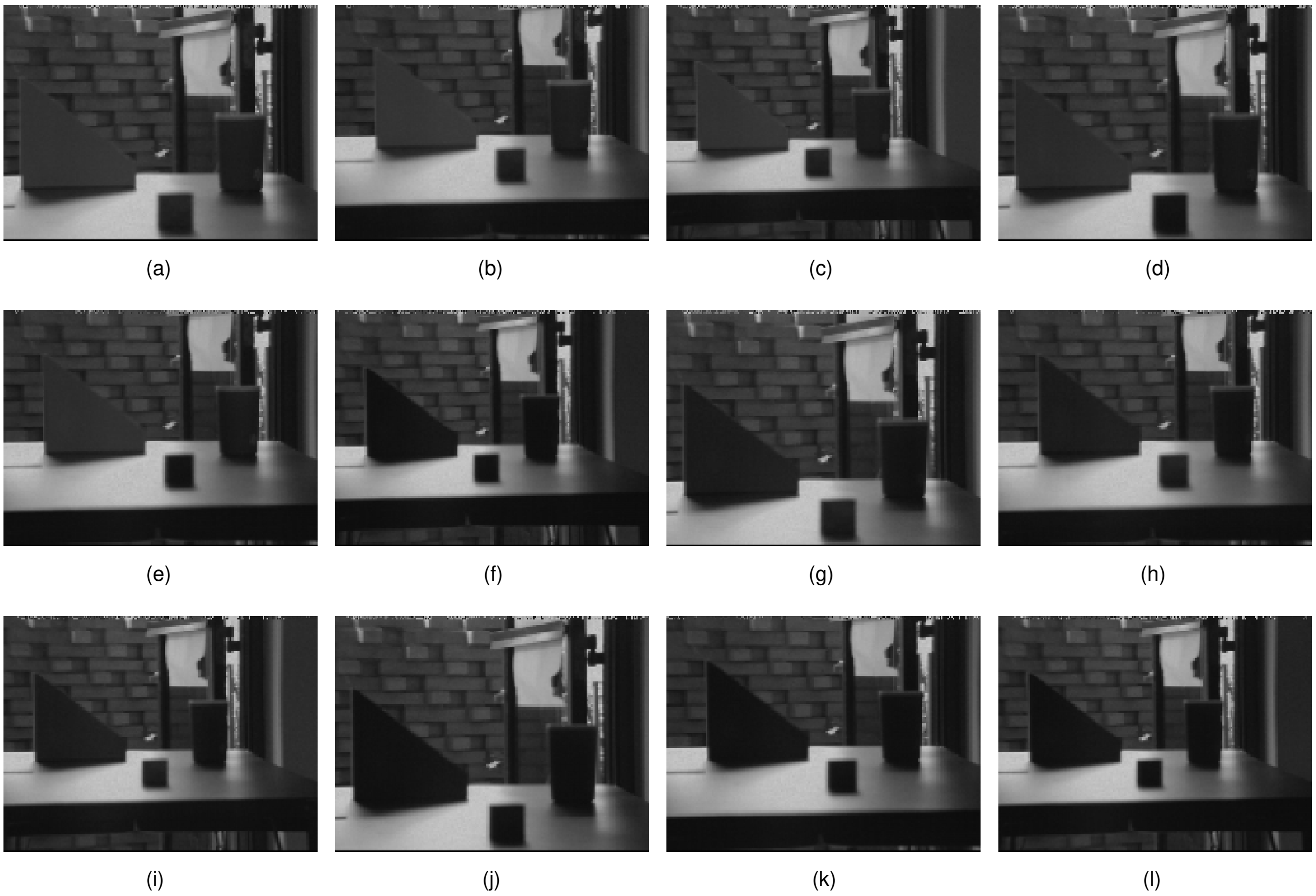


Fig. 8. Twelve color images having simple geometric objects.

4.2.1 Results on Indoor Images

The first segmentation task whose experimental results we report here is a sequence of indoor color images (160 × 120 pixels) having simple geometric objects with varying lighting and motion conditions. These images are (shown in Fig. 8) divided into four groups, where images a, b, and c are in the first group, and images d, e, and f are in the second group, and so on. Within each group, images are moving away from the camera, and from group 1 to group 4 lighting conditions deteriorate. The training set consists of the images c, h, k, and l (randomly selected), whereas the testing data come from the rest of the images (eight images). The objective of the task is to find a set of *Phoenix*'s parameters that give rise to a segmentation of the input image that, after appropriate feature extraction, will result in the recognition of the triangular object. The model of the triangular object is represented by a polygonal approximation of its shape. The threshold for matching confidence in this case was set to 0.8. The learning rate parameter α was set to 0.008 in all the experiments. Note that, unlike previous work on image segmentation, the criteria measuring image segmentation quality here are completely determined by the matching algorithm itself.

Each unit in the team network has a total of eight input weights, which is also the total number of effective weights of the entire network, since units in the network are independent. In the first experiment each of the input weights takes an average gray value of input on a 60 × 40 neighborhood on the input image plane of 120 × 160 pixels. This input image is the luminance image of the corresponding color image. Note that, in this experiment, the average is normalized to lie between −1 and 1. For weights that are adjacent in a unit, their receptive fields are at least 40 pixels apart in the input image. Thus, the input image is undersampled, which, in turn, greatly reduces the number of weights in the network. The motivation is that variations in lighting need not be adapted with high resolution.

In the second experiment, each input image is projected onto the subspace spanned by the eight eigenvectors corresponding to eight largest eigenvalues of the original (luminance) image vector space (120 × 160 pixels). More specifically, the sample mean vector, μ, is computed as $\mu = (1/n)\sum_{i=1}^{n} \mathbf{x}_i$, where n is the number of sample vectors (in this paper n equals 12) and $\mathbf{x}$ denotes $m \times 1$ column vectors of input images. Note that here m equals 19,200. A centered input matrix $\mathbf{X}$ is constructed according to $\mathbf{X} = (\mathbf{x}_1 - \mu, \mathbf{x}_2 - \mu, \ldots, \mathbf{x}_2 - \mu)$.

Then the sample covariance matrix is obtained $\mathbf{C} = \frac{1}{n-1}\mathbf{X}\mathbf{X}^t$ and its eigensystem is computed, yielding eigenvalues λ_i, $i = 1, 2, \ldots, m$, of $\mathbf{C}$ in descending order so that $\lambda_j \geq \lambda_{j+1}$ for $j = 1, 2, \ldots, m - 1$. Let $\mathbf{A}$ be a $8 \times m$ matrix

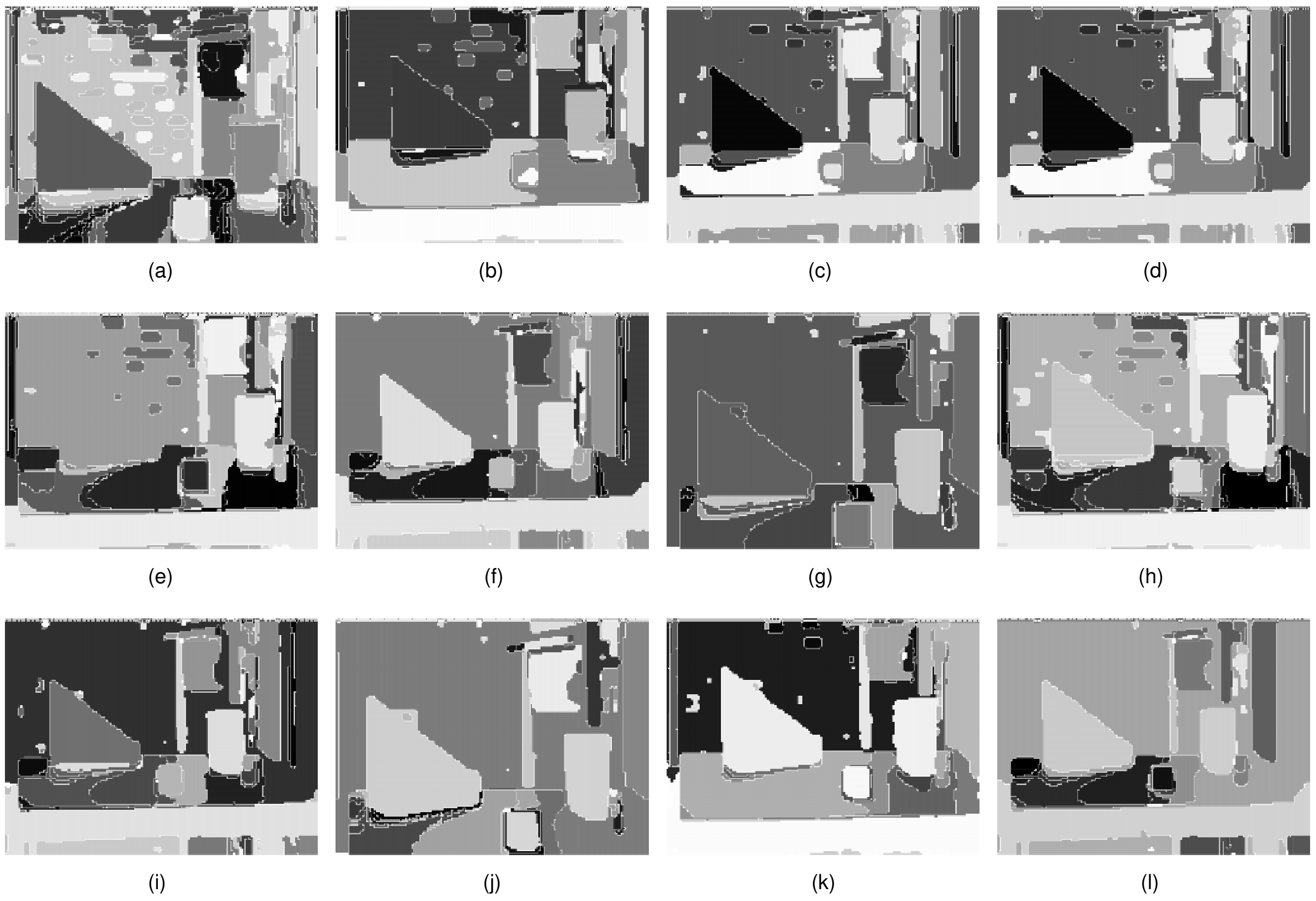


Fig. 9. Segmentation performance of the *Phoenix* algorithm with learned parameters.

whose rows are formed from the eigenvectors of **C**, ordered so that the first row of **A** is the eigenvector corresponding to the largest eigenvalue, and the last row is the eigenvector corresponding to the eighth largest eigenvalue. Then new inputs are computed according to $\mathbf{z} = \mathbf{A}\mathbf{x}$ where $\mathbf{z}$ denotes 8×1 column vectors. These inputs are normalized to lie between –1 and 1. Our goal is to see which experiment can offer better performance. It turns out that the second experiment performed slightly better than the first one, as can be seen below (Figs. 9 and 10). Note that, unless stated otherwise, all the figures in this section are obtained under the condition that the system takes inputs from the subspace spanned by the first eight eigenvectors (major axes) corresponding to the eight largest eigenvalues of **C**.

Fig. 9 shows the segmentation performance (both training and testing) of the *Phoenix* algorithm with learned parameters on the images shown in Fig. 8. The training results in Fig. 9 are obtained after a mean value (over five runs) of 250 passes through the training data. Fig. 10 shows the average confidence (over five runs) received by the two experiments (eigen input and mean input) over time during training (hillclimber results are explained below under Computational Efficiency in Section 4.4). Each run consists of a sequence of trials until the average confidence level has exceeded 0.8. The threshold (0.8) serves our purpose well here since it is sufficient to demonstrate the effect of learning for object recognition.

Fig. 11 shows the trajectory of each of the four *Hsmooth*, *Maxmin*, *Splitmin*, and *Height* parameters during training in a typical run on a particular image (in this case it is the image of Fig. 8c). Note that no attempt was made to determine if the set of parameters giving rise to the final recognition is unique.

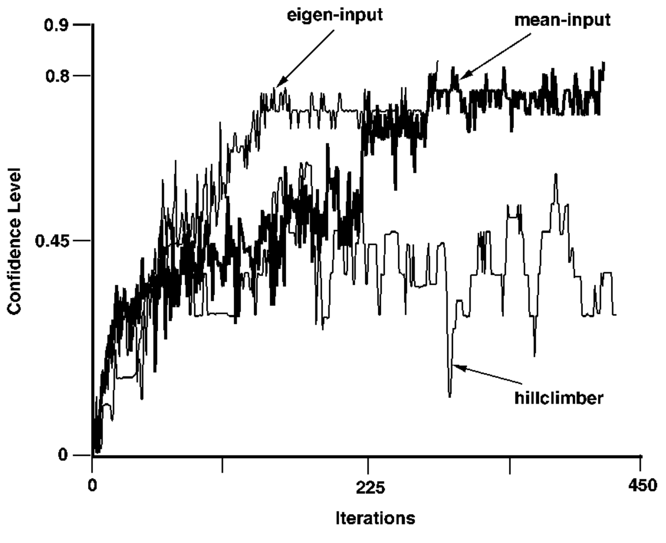


Fig. 10. Average confidence received by the three methods (mean input, eigeninput, and hill climber) over time during training.

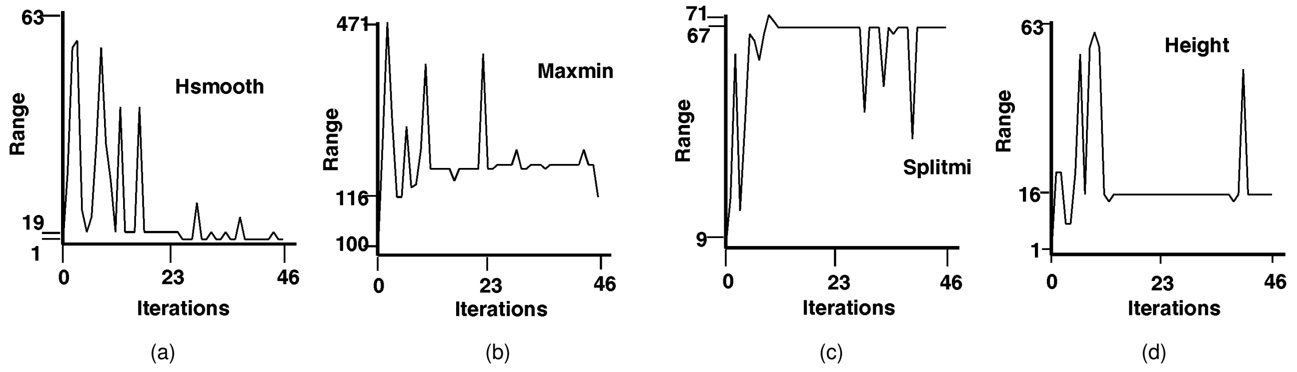


Fig. 11. Trajectories for a particular run for each of the four parameters *Hsmooth*, *Maxmin*, *Splitmin*, and *Height* during training on a particular image (Fig. 8g).

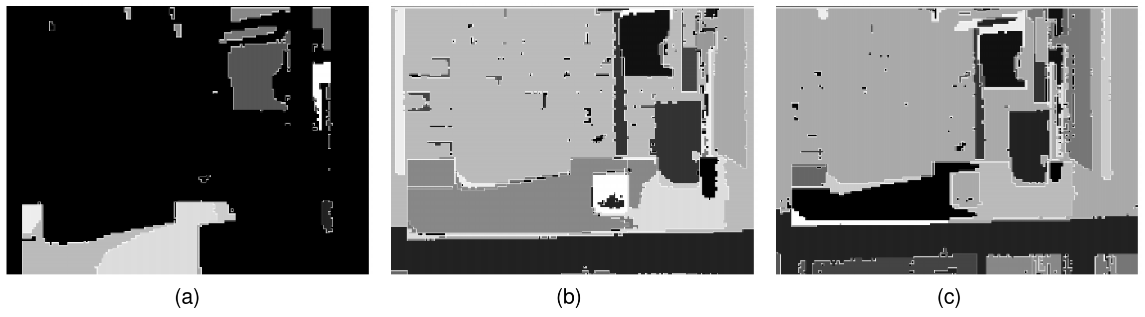


Fig. 12. Samples of segmentation performance of the *Phoenix* algorithm with default parameters on indoor color images (Figs. 8a, 8b, and 8c, respectively).

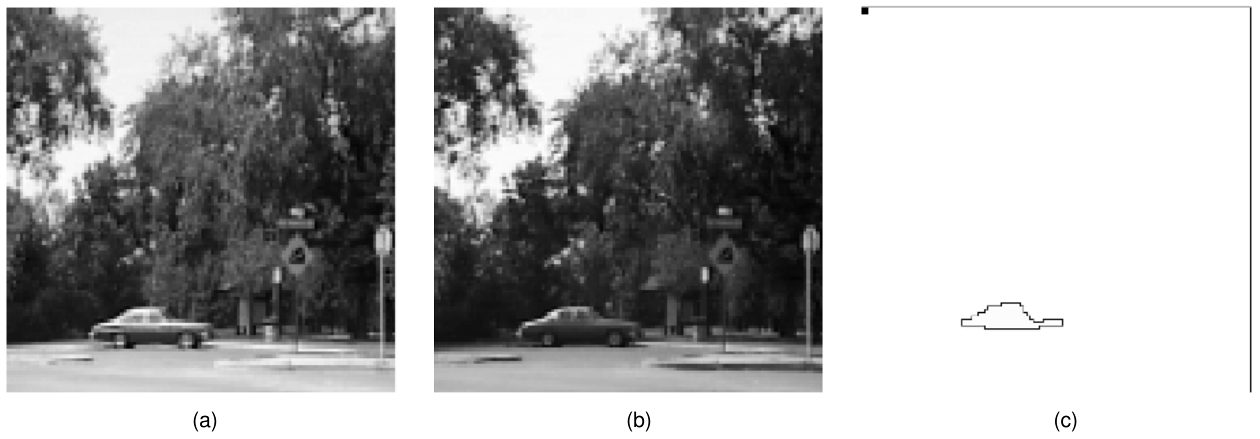


Fig. 13. (a) and (b) Samples of outdoor color images with varying environmental conditions. (c) Polygon approximation of the car used in the matching algorithm.

When the segmentation parameters obtained after training were applied to the images in the testing set, recognition results for all the images, but Fig. 8e, are acceptable. However, if we include image Fig. 8e in the training set and allow learning to continue, experiments have been performed that show that successful recognition can be achieved for all testing images in much reduced time (less than 50 percent) compared to the time taken for training on the original training data.

In comparison, the *Phoenix* algorithm with default parameter setting was also run on the same images. Fig. 12 shows the samples of the segmentation performance of the *Phoenix* algorithm with default parameters on the images in Figs. 8a, 8b, and 8c. These default parameters were obtained after extensive tests [15]. This default parameter setting resulted in a total matching failure.

4.2.2 Results on Outdoor Images

The second segmentation task involves a sequence of 10 outdoor color images obtained under varying environmental conditions, two of which are shown in Figs. 13a and b. These images are collected approximately every 15 minutes over approximately a 2 and 1/2 hour period [3]. The images exhibit varying shadow and reflection on the car as the position

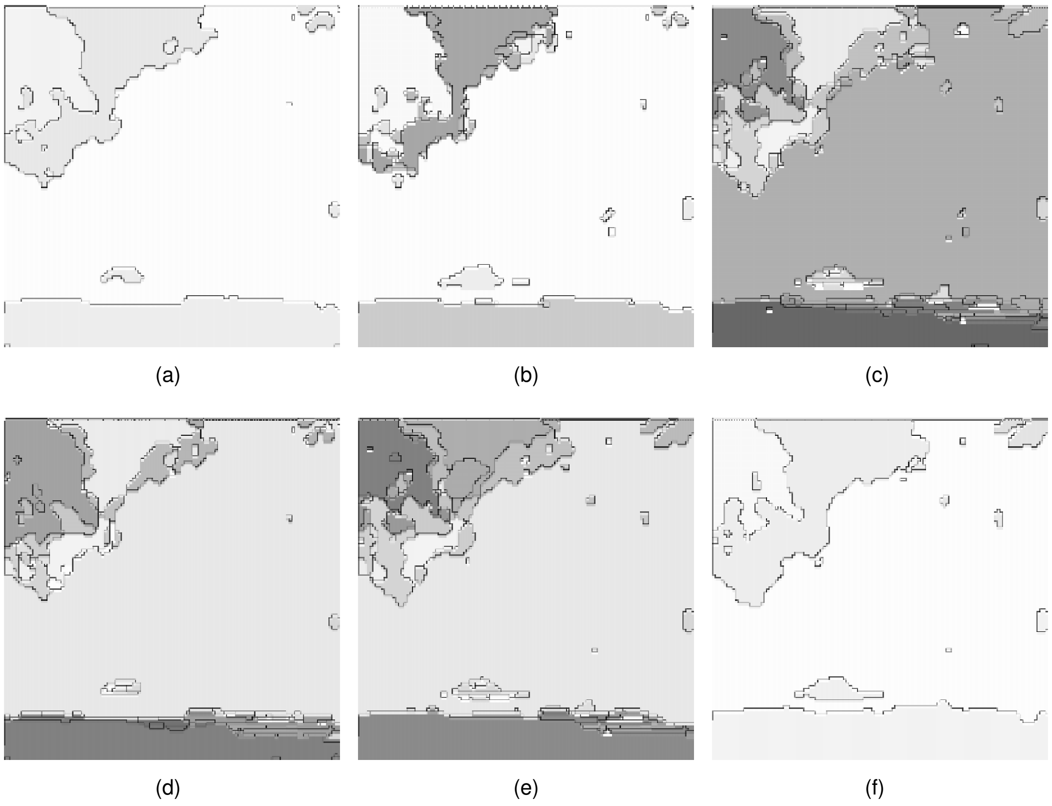


Fig. 14. Sequence of segmentations of the first frame during training.

of the sun changed and clouds came in and out the field of view of the camera that had auto iris adjustment turned on. The overall goal is to recognize the car in the image. The original images are digitized at 480 × 480 pixels in size and are then subsampled to produce 120 × 120 pixel images. Five of these odd-numbered images are used as training data and five even-numbered images as testing data.

Similar to the team network for the indoor images, each unit here has a total of nine input weights, each of which takes an average gray value of input on a 40 × 40 neighborhood on the input image plane of 120 × 120 pixels. These averages are normalized to lie between –1 and 1. Polygonal approximation of the car shown in Fig. 13c is used as the model in the cluster-structure matching algorithm. It was extracted manually in an interactive session from the first frame in the sequence.

Fig. 14 shows a sequence of segmentations for frame 1 with *Phoenix's* parameters sampled at iterations 20, 30, 40, 50, 60, and 74 in a particular run during training, and corresponding parameter values at each of these intervals are shown in Table 2. Note that Fig. 14f shows the final segmentation result when the highest confidence matching has been achieved. The threshold for acceptable matching confidence is set at 80 percent.

Figs. 15a and b show the *Phoenix* segmentation performance on two testing images (frames 2 and 4) with learned parameters obtained after training on frames 1, 3, 5, 7, and 9. For frame 2, the matching is acceptable. However, for frame 4, the result is not acceptable, and learning is to be performed similar to the indoor examples for the adaptation of parameters.

Finally, Figs. 15c and d show the samples of performance of *Phoenix* with default parameters on the outdoor color images shown in Fig. 13. Note that these segmentation results are totally unacceptable.

4.3 Evaluation on a Large Simulated Data Set

The simulated data experiment allows us to examine how the system will behave with a large data set. We assume the function, F, representing segmentation, feature extraction and model matching components shown in Fig. 2, is given by

$$F_{\mathbf{p}}(\mathbf{x}) = \sum_{k=0}^{3} F_{\mathbf{p}}^{k}(\mathbf{x}), \tag{10}$$

and

$$F_{\mathbf{p}}^{k}(\mathbf{x}) = 2.5n \prod_{i=kn/4+1}^{(k+1)n/4} \left(1 - \left|x_i - p_i\right|\right). \tag{11}$$

where $\mathbf{p} \in \{0, 1\}^n$ is a constant. F is a mapping from the n-dimensional hypercube $\{0, 1\}^n$ into the real numbers, where $n = 20$. Each point $\mathbf{x}$ in its domain is an n-dimensional bit vector.

TABLE 2
CHANGES OF PARAMETER VALUES DURING TRAINING

Iteration	Hsmooth	Maxmin	Splitmin	Height
20	53	135	55	58
30	17	142	39	42
40	21	105	43	24
50	1	165	51	42
60	1	135	19	62
74	1	300	55	64

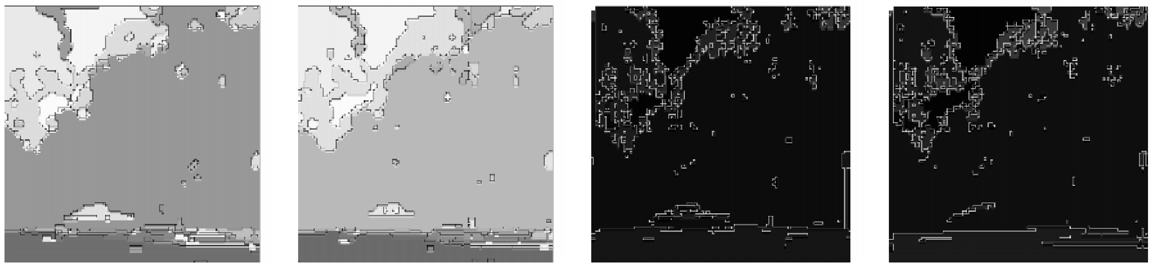

(a) (b) (c) (d)

Fig. 15. (a) and (b) Segmentation performance of the *Phoenix* algorithm on two testing images (frames 2 and 4) with learned parameters. (c) and (d) Samples of segmentation performance of the *Phoenix* algorithm with default parameters on the two images shown in Fig. 13.

The function $F_{\mathbf{p}}(\mathbf{x})$ is computed as follows: Divide the 20 bits into four equal-sized groups. For each group, compute a score which is 2.5 n if all the bits in that group are the same as those in $\mathbf{p}$ and is 0 otherwise. Then $F_{\mathbf{p}}(\mathbf{x})$ is the sum of these four scores. This function has a global maximum of 200 at $\mathbf{p}$. It also has very large plateaus over which the function is constant. These plateaus will confound any myopic hillclimber.

In terms of the vision system described in the paper, $\mathbf{x}$ corresponds to the encoding of segmentation parameters and $F_{\mathbf{p}}$ represents in abstract terms the matching confidence resulting from applying Phoenix with $\mathbf{x}$ to a given input image $\mathbf{p}$.

Note that since the precise nature of the function (10) to be optimized is known, we can more reliably predict the strengths and limitations of the system. In this experiment, $\mathbf{p}$ is randomly generated uniformly from $\{0, 1\}^n$. Then, 2,000 data points whose Hamming distance to $\mathbf{p}$ is at most four are randomly generated from a distribution such that 80 percent of the data points are produced by perturbing the first 10 bits of $\mathbf{p}$, 10 percent by the first 15 bits, and the remaining 10 percent by the entire 20 bits. Conceptually, each of these data points may be viewed to simulate the segmentation parameter values for an image that will give rise to the best possible recognition result for the image.

Out of these 2,000 data points, 500 are randomly selected as training data. The remaining 1,500 data points as testing data. As in the real data experiments described above (Section 4.2), 15 normalized eigenfeatures are computed to represent these data. Thus, there are 20 Bernoulli units, each of which has 15 input lines that encode a particular pattern to be searched for.

Training consists of repeated sweeps through the training set until the average value of F has reached 190, which is about 95 percent of the optimal value of 200 (see (10)). An added benefit is that it prevents the system from overfitting the data, resulting in better generalization. The result shows that after about 5,000 sweeps through the training data, the system achieved an average value of 180 over 90 percent of the testing data and an average value of 170 over the entire testing data. Further examination revealed that the majority of those testing data whose value is less than 180 come from 20 bit perturbation to $\mathbf{p}$. These data were least represented, and, therefore, resulted in relatively not-so-good performance. This generalization characteristic is typical in connectionist networks. These results demonstrate that the algorithm can be expected to perform reasonably well on large data sets in large problem domains.

4.4 Computational Efficiency

The computational efficiency of the system should be evaluated against other systems having similar operating characteristics. Currently there is no similar system in the computer vision field that directly uses recognition result as a feedback to drive learning for image segmentation. Thus, as a comparison we applied a stochastic hillclimber to the same indoor images used for the experiments described in the above (Section 4.2). We first applied the K-Means algorithm [17] to the eigenfeatures to determine K centers, where K = 4 in this experiment. Then four images that are closest to the four centers are used as training data. There are, therefore, four sets of *Phoenix* parameters, each of which is associated with a particular center. Again, 20 bits are used to encode these *Phoenix* parameters. For a given image, generalization is made by searching for the nearest cluster center and then applying the set of *Phoenix* parameters associated with the cluster.

In the beginning, the hillclimber occasionally moves along directions that are not very promising. However, as search continues the probability of downhill movement is reduced. The annealing schedule (a schedule that reduces the probability of downhill movement) used in this experiment is an inverse function of the number of iterations. It is important to note that if each dimension of the 20 dimension input space at every iteration has to be examined to estimate the gradient, the amount of computation required would be prohibitive. Instead, we randomly perturb three dimensions (where each dimension is equally likely to be selected) for each parameter set to move up the gradient. Thus, the amount of computation is three times of that required by the reinforcement learning system at each iteration. The decision of where to look next critically influences the computational efficiency of the optimization process. Like the reinforcement learning method, however, a priori gradient information is not available. It has to be estimated by sampling the search space. The average results (labeled as "hill-climber") are shown in Fig. 10. A comparison of these results (Fig. 10) clearly demonstrates that the reinforcement learning system performed significantly better than the stochastic hillclimber, despite the fact that it took more computation time at every iteration.

5 Conclusions

Our choice of the team architecture is motivated by its simplicity and its generality as a representation scheme. What kind of problems an architecture can represent is to be determined experimentally and theoretically. In particular, one explanation for the successful results from the architecture presented in this paper is that the images used exhibit well-behaved smoothness, that is, similar images in feature space require similar parameters for segmentation. We believe that this is true in general. In addition, determining how close an architecture can approximate an unknown mapping is a classical problem, and many criteria have been proposed, such as the Probably Approximately Correct (PAC) learning theory and the Minimum Description Length (MDL) measure. It is in general an ill-posed problem. In practice, however, it is often determined experimentally through cross-validation. The architecture presented in this paper seemed to approximate the unknown mappings sufficiently well, for nearly optimal performance has been achieved. These results shed light on the large potential of the proposed architecture, and present a basis for additional experiments to determine the scope of applicability of the architecture beyond the current problem reported in this paper.

The key contribution of the paper is the general framework for the usage of reinforcement learning in a model-based object recognition system. Our investigation into reinforcement learning-based object recognition shows convincingly that a robust and adaptive system can be developed that autonomously determines the criteria for segmentation of the input images and selects useful features that result in a system with high recognition accuracy when applied to unseen images. Note that the performance of any learning-based computer vision system depends on the vision algorithms that are used, e.g., the *Phoenix* algorithm used in this paper for the segmentation of color images. Future research will address extensions for enlarging the scope of the approach to encompass closed-loop 3D object recognition and problems in active vision where reinforcement learning could be extremely useful. Furthermore, incorporation of "delayed" reinforcement learning could adequately address the inherent multilevel nature of vision systems [20].

Appendix A: The *Phoenix* Segmentation Algorithm

The *Phoenix* image segmentation algorithm is based on a recursive region splitting technique [15]. It uses information from the histograms of the red, green, and blue image components to split regions in the image into smaller subregions on the basis of a peak/valley analysis of each histogram. An input image typically consists of red, green, and blue image planes, although monochrome images, texture planes, and other pixel-oriented data may also be used. Each plane is called a feature or feature plane.

Fig. 16 shows a conceptual description of the *Phoenix* segmentation process. It begins with the entire image as a single region. It then fetches this region and attempts to segment it using histogram and spatial analyses. If it succeeds, the program fetches each of the new regions in turn and attempts to segment them. The process terminates when no region can be further segmented.

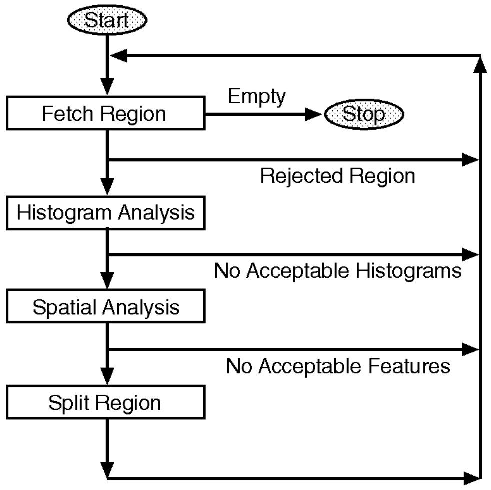


Fig. 16. Conceptual diagram of the *Phoenix* segmentation algorithm.

The histogram analysis phase computes a histogram for each feature plane, analyzes it and selects thresholds or histogram cutpoints that are likely to identify significant homogeneous regions in the image. A set of thresholds for one feature is called an interval set. During the analysis, a histogram is first smoothed with an unweighted window average, where the window width is *hsmooth*. It is then broken into intervals such that each contains a peak and two "shoulders." A series of heuristics is applied to eliminate noise peaks. When an interval is removed, it is merged with the neighbor sharing the higher of its two shoulders. *Splitmin* is the minimum area for a region to be automatically considered for splitting.

Two tests determine if an interval should be retained. First, the ratio of peak height to the height of its higher shoulder must be greater than or equal to the *maxmin* threshold. Second, the interval area must be larger than an absolute threshold and the relative area, percent of the total histogram area. The second highest peak can now be found, and peaks lower than the *height* percent of this peak are merged. The lowest valley is then determined, and any interval whose right shoulder is higher than *absmin* (*Phoenix's* parameter) times this valley is merged with its right neighbor. Finally, only *intsmax* (*Phoenix's* parameter) intervals are retained by repeatedly merging intervals with low peak-to-shoulder ratio.

The spatial analysis selects the most promising interval sets, thresholds the corresponding feature planes, and extracts connected components for spatial evaluation. The feature and the interval set providing the best segmentation (the least noise area) are accepted as the segmentation feature and the thresholds.

The histogram cutpoints are now applied to the feature plane as intensity thresholds and connected components are extracted. After each feature has been evaluated, the one producing the least total noise area is accepted as the segmentation feature. If no suitable feature is found, the original region is declared terminal. Otherwise, the valid patches, merged with the noise patches, are converted to new regions and added to the segmentation record. In either case, a new segmentation pass is scheduled. For additional details, see [15].

APPENDIX B: THE CLUSTER-STRUCTURE ALGORITHM

The cluster-structure algorithm can be divided into the following main steps:

1) Determine Disparity Matrix,
2) Initial Clustering,
3) Sequencing,
4) Final Clustering,
5) Transform Computation.

The algorithm first computes the disparity matrix. It determines the segment length of each line and the angles between successive lines from the set of vertices for the model and the image input to the program. At this point, every segment in the model will be compared against every segment in the image. If segment lengths and successor angles are compatible, the algorithm computes the rotational and translational disparity between pairs of segments. These values are stored in the disparity matrix and are indexed by the segment numbers in the model and the image. The algorithm continues until all segments have been compared. It then computes the range of rotational and translational values present in the matrix, and normalizes them over their appropriate range.

The initial clustering determines clusters from the normalized values in the disparity matrix. At each step, the program clusters all of the samples, recomputes the new cluster centers, and continues until none of the cluster centers change their positions. The program then selects the cluster having the largest number of samples. Also selected are the clusters that are within 20 percent of the largest one. Each cluster is considered separately and the final transform comes from the cluster that yields the highest confidence level.

The sequencing step uses the samples in the current cluster to find all sequences in the samples. This provides the critical structural information. Samples that are not placed in any sequence are discarded. The program also removes sequences that have a segment count of less than three (three segments comprise the basic local shape structure). It then computes the rotational and translation averages of each sequence that has been located.

Using the sequences and the sequence averages, the final clustering step clusters these values to find those sequences that lead to the same rotational and translational results. This is achieved by using the iterative technique of clustering, evaluating, clustering, etc. The program then selects the cluster that contains the largest number of sequences and passes this cluster to the final step.

The final step of the algorithm computes the confidence level of the transformation determined by each cluster. The cluster having the highest confidence level is selected as the final transformation cluster. It assembles the set of matched segments in the sequences in this cluster. The final output of the program is the rotation and the vertical and horizontal translation necessary to locate the model within the image. The program also produces a confidence level indicating the likelihood that the final matching is correct. For further details, see [4].

ACKNOWLEDGMENTS

This work was supported by DARPA/AFOSR grants F49620-97-1-0184 and F49620-95-1-0424. The contents of the information do not necessarily reflect the position or the policy of the U.S. Government.

REFERENCES

[1] A.G. Barto, R.S. Sutton, and C.J.C.H. Watkins, "Learning and Sequential Decision Making," COINS Technical Report 89-95, Dept. of Computer and Information Science, Univ. of Mass., Amherst, Mass., 1989.
[2] B. Bhanu and T. Jones, "Image Understanding Research for Automatic Target Recognition," *Proc. DARPA Image Understanding Workshop*, pp. 249-259, 1992.
[3] B. Bhanu and S. Lee, *Genetic Learning for Adaptive Image Segmentation*. Boston, Mass.: Kluwer Academic Publishers, 1994.
[4] B. Bhanu and J. Ming, "Recognition of Occluded Objects: A Cluster-Structure Algorithm," *Pattern Recognition,* vol. 20, no. 2, pp. 199-211, 1987.
[5] B. Bhanu, S. Lee, and S. Das, "Adaptive Image Segmentation Using Genetic and Hybrid Search Methods," *IEEE Trans. Aerospace and Electronic Systems*, vol. 31, no. 4, pp. 1,268-1,291, Oct. 1995.
[6] B. Bhanu, S. Lee, and J. Ming, "Adaptive Image Segmentation Using a Genetic Algorithm," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 25, no. 12, pp. 1,543-1,567, Dec. 1995.
[7] D. Chapman, "Intermediate Vision: Architecture, Implementation, and Use," *Cognitive Science,* vol. 16, pp. 491-537, 1992.
[8] R.T. Chin and C.R. Dyer, "Model-Based Recognition in Robot Vision," ACM Computing Surveys, pp. 67-108, Mar. 1994.
[9] M.A. Fischler, "On the Representation of Natural Scenes," *Computer Vision Systems*, A.R. Hanson and E.M. Riseman, eds. New York: Academic Press, 1978.
[10] K. Fukushima, S. Miyake, and T. Ito, "Neocognition: A Neural Network Model for a Mechanism of Visual Pattern Recognition," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 13, no. 5, pp. 826-834, Sept. 1983.
[11] D.E. Goldberg and J.H. Holland, Special Issue on Genetic Algorithms, *Machine Learning*, 2/3, 1988.
[12] R.C. Gonzalez and P. Wintz, *Digital Image Processing*. Addison-Wesley Publishing Co., 1977.
[13] R.M. Haralick and L.G. Shapiro, "Image Segmentation Techniques," *Computer Vision, Graphics, and Image Processing,* vol. 29, pp. 100-132, 1985.
[14] H.G. John, R. Kohavi, and K. Pfleger, "Irrelevant Features and the Subet Selection Problem," *Proc. 11th Int'l Conf. Machine Learning*, pp. 121-129, 1994.
[15] K. Laws, "The *Phoenix* Image Segmentation System: Description and Evaluation," SRI Int'l Tech. Rep. TR289, Dec. 1982.
[16] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation,* vol. 1, pp. 541-551, 1989.
[17] J.L. Marroquin and F. Girosi, "Some Extensions of the K-Means Algorithm for Image Segmentation and Pattern Classification," A.I. Memo No. 1390, MIT AI Lab, 1993.
[18] K.S. Narendra and M.A.L. Thathatchar, *Learning Automata: An Introduction*. Englewood Cliffs, N.J.: Prentice Hall, 1989.
[19] R. Ohlander, K. Price, and D. R. Reddy, "Picture Segmentation Using a Recursive Region Splitting Method," *Computer Graphics and Image Processing*, vol. 8, pp. 313-333, 1978.
[20] J. Peng and B. Bhanu, "Delayed Reinforcement Learning for Closed-Loop Object Recognition," *Proc. DARPA Image Understanding Workshop*, pp. 1,429-1,435, Feb. 1996.

[21] V. Ramesh, "Performance Characterization of Image Understanding Algorithms," PhD Thesis, Dept. of Electrical Eng., Univ. of Washington, Seattle, Washington, 1995.

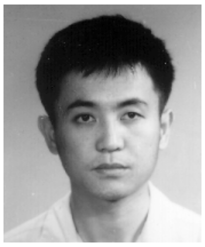
[22] S. Shafer and T. Kanade, "Recursive Region Segmentation by Analysis of Histograms," *Proc. IEEE Int'l Conf. Acoustics, Speech, and Signal Processing*, pp. 1,166-1,171, 1982.

[23] P. Suetens, P. Fua, and A.J. Hanson, "Computational Strategies for Object Recognition," *ACM Computing Surveys*, vol. 24, no. 1, pp. 5-59, 1992.

[24] S. Wang and T. Binford, "Local Step Edge Estimation—A New Algorithm, Statistical Model, and Performance Evaluation," *Proc. ARPA Image Understanding Workshop*, pp. 1,063-1,070, Apr. 1993.

[25] R.J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," *Machine Learning*, vol. 8, pp. 229-256, 1992.

[26] R.J. Williams and J. Peng, "Function Optimization Using Connectionist Reinforcement Learning Algorithms," *Connection Science*, vol. 3, no. 3, 1991.



Jing Peng received the BS degree in computer science from the Beijing Institute of Aeronautics and Astronautics, Beijing, China. He also received the MA degree in computer science from Brandeis University and the PhD degree in computer science from Northeastern University, Boston, Mass. Recently, he has been a research scientist with Visualization and Intelligent Systems Laboratory at the University of California at Riverside. Dr. Peng's research interests include machine learning, computer vision, image databases, and learning and vision applications.

Bir Bhanu received the SM and EE degrees in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, Massachusetts, the PhD degree in electrical engineering from the Image Processing Institute, University of Southern California, Los Angeles, and the MBA degree from the University of California, Irvine. He also received the BS degree (with Honors) in electronics engineering from the Institute of Technology, BHU, Varanasi, India, and the ME degree (with Distinction) in electronics engineering from Birla Institute of Technology and Science, Pilani, India.

Since 1991, he has been a professor of electrical engineering and computer science and director of Visualization and Intelligent Systems Laboratory at the University of California, Riverside. Previously, he was a Senior Honeywell Fellow at Honeywell Systems and Research Center, Minneapolis, Minnesota. He has been on the faculty of the Department of Computer Science at the University of Utah, Salt Lake City, and has also worked with Ford Aerospace and Communications Corporation, INRIA-France, and IBM San Jose Research Laboratory, California. He has been the principal investigator of various programs for DARPA, NASA, the U.S. National Science Foundation, AFOSR, ARO, and other agencies and industries in the areas of learning and vision, image understanding, pattern recognition, target recognition, navigation, image databases, and machine vision applications. He is the coauthor of the books *Computational Learning for Adaptive Computer Vision* (Plenum, forthcoming), *Genetic Learning for Adaptive Image Segmentation* (Kluwer, 1994), and *Qualitative Motion Understanding* (Kluwer 1992). He received an outstanding paper award from the Pattern Recognition Society. He has also received industrial awards for technical excellence, outstanding contributions, and team efforts. He has been the guest editor of several IEEE transactions and journals and is on the editorial board of various journals. He holds 10 U.S. and international patents and has published more than 150 reviewed technical publications in the areas of his interest. He was the General Chair for the first IEEE Workshop on Applications of Computer Vision, Chair for the DARPA Image Understanding Workshop, and General Chair for the IEEE Conference on Computer Vision and Pattern Recognition.

Dr. Bhanu is a Fellow of the IEEE and AAAS (American Association for the Advancement of Science). He is a member of ACM, AAAI, Sigma Xi, Pattern Recognition Society, and SPIE.