Chapter 9

Machine Learning of Computer Vision Algorithms

STEVEN R. SCHWARTZ
Motorola Inc.
Arlington Heights, Illinois

BENJAMIN W. WAH
Center for Reliable and High-Performance Computing
Coordinated Science Laboratory
University of Illinois, Urbana-Champaign
Urbana, Illinois

HANDBOOK OF PATTERN RECOGNITION
AND IMAGE PROCESSING: COMPUTER VISION

1. INTRODUCTION

Both human vision and human learning are far from fully understood, yet machine imitation of these abilities has proved very useful. Research up to now has made much progress in tracing the desired result backward to the algorithms that implement it. For example, we know how or what we want the machine to perform, but lacking a complete understanding of the biological processes, we proceed to find an algorithmic solution. In both the fields of machine learning and computer vision this has been the case. This chapter presents an approach to integrating machine learning techniques to improve the performance of computer vision algorithms. The topics considered here are the techniques for machine learning, examples of how they can be applied, and a specific case study detailing the application of the methods. Topics not considered here include reasoning about detected objects and planning interaction with the environment.

Before discussing the integration of learning and vision it is best to start with a brief description of the sought-after result: namely, what vision tasks do we now expect the machine to perform. This will lead to the difficulties that must be overcome and how machine learning can play a role in surmounting them. The full utility of human vision is far beyond our current reach with computational techniques, yet our present demands are often less ambitious than full human vision and are within our grasp. Three general and important goals are effective performance of (1) two-dimensional object recognition, (2) three-dimensional object recognition, and (3) integration with robotics.

These goals are general, but applications toward achieving limited forms of these goals are of key interest here. Two-dimensional object recognition has found much application in manufacturing as an assembly automation technique. Other uses lie in spotting equipment and hostile movement for defense purposes from aerial surveillance imagery. Three-dimensional object recognition is also of use in manufacturing and is vital to systems that interact in the real world. Effective 3D vision is one of the key requirements for automatic navigation in mobile robots. The last goal, integrating vision with robotics, is primarily an application of the first two but also underlines a motive for computer vision in the first place: vision in environments where it is more practical to send a machine. An example of the tasks in demand is in parsing scenes viewed by an exploratory mobile robot. Such robots are often outside the reach of remote operation by humans, thus necessitating a degree of autonomy.

2. THE NEED FOR MACHINE LEARNING IN VISION

The goals outlined above are achievable in a limited sense, but to go beyond them it is necessary to overcome a variety of problems. The first problem is that simple cognitive tasks are difficult for machines because the algorithm designer must specify every step in the task. This is exemplified in that gaining Gestalt information from a scene is often very difficult. Often we ourselves don't know the correct way of combining components to make the whole. Also, much of the difficulty in computer vision lies in getting the machine to ignore unimportant details. Another problem is that the design of practical vision systems is still very much an art. What is needed is to take advantage of the machine to develop and refine the algorithm itself.

In real-world applications we need to be pragmatic. To incorporate machine learning we must first approach a reasonable portion of the vision task. A logical partitioning of the algorithm construction effort relys on human expertise for the design of the general algorithmic framework, while relying on computer exploration within this framework to search for the appropriate complete algorithm. This necessitates using machine learning techniques to achieve effective results, and can work well with many vision algorithms because of the large number of heuristic components. With this approach, given the parametric form of the algorithm, it becomes the responsibility of the machine to find suitable "values." Here the term *parameter* is used loosely to refer to any adjustable component and can be numeric as well as symbolic; the value in the latter case is the particular expression used for the component.

These parameters in the algorithm may have varied roles. Some examples include detection thresholds, set sizes, and the configuration of image preprocessing stages. Frequently, the parameter values that are used in the final algorithm are set heuristically. There is often little or no theory behind the values that are used, and a significant amount of hand-tuning is used to find acceptable values. (Developers of real-world systems can attest to this.) Since parameter tuning is a tedious process, machine automation allows more extensive testing than humans may find personally tolerable. As recognized in a panel discussion chaired by Sklansky in 1988 (14), one of the bottlenecks to effective vision applications is the amount of effort necessary for tuning the algorithm. Tuning is necessary not only for the initial implementation in the laboratory but also for the installation of the system in the field. Additional factors that necessitate modification include varying image domains, different lighting conditions, drifting camera parameters, and the implementation of the algorithm on different computer systems. In the latter case, the performance bottleneck can change, thus shifting the time critical area of the algorithm elsewhere.

As an example, different sets of parameters may be needed for a system designed for both day- and night-vision applications. Additionally, parameters may need adjusting after the vision system is ported to another computer; for example, one that might lack special image processing hardware. These adjustments can be costly because the novice user is typically unqualified to make them. Worse yet, even an expert can be inadequate because many systems require the extensive knowledge of the original code to make appropriate adjustments. A benefit of machine assistance is the ability to train for different domains and automatically switch context as necessary. By putting the responsibility for tuning in the machine, the need for human intervention can be greatly reduced. This results in a cost savings of both time and effort.

3. THE ROLE OF MACHINE LEARNING

Computer vision encompasses everything from low-level image preprocessing to high-level construction and interpretation of visual information. Machine learning can be applied at any stage in this cognitive process. The procedures for integrating learning techniques with existing vision algorithms is best done incrementally. The initial step is to recognize areas in the vision task where the strengths of machine learning can be exploited. Massive bookkeeping and computation can prove very fruitful when guided by relatively simple learning techniques. Before addressing these points further, however, it is necessary to outline what machine learning is and what can be expected.

Most people assume they know what learning is for humans, but anthropomorphizing the term for machines does not clarify what is meant by machine learning. A useful working definition is supplied by Simon (13): "[Machine] Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time."

With this interpretation the purpose of using machine learning techniques is to improve computer vision performance by *automating algorithm improvement*. Machine learning can address problems in both low- and high-level computer vision. In low-level vision the machine can learn what should be seen, that is to say, what should be detected from the image for use in the higher levels of the vision algorithm. Parameters can have a strong influence on what the machine "sees." In high level vision, the machine can help learning by "seeing." This includes generalizing from examples. Each of the problems mentioned above can be addressed through learning techniques.

Learning can be performed over the target domain by allowing the vision algorithm to adapt and improve its performance. Feedback from the measured algorithm performance can also be used by the learning mechanisms at the initial image processing stage. This can have the effect of training the system to eliminate any unnecessary detail. This is a very powerful technique that is evident in biological systems (5, pp. 12–15). Gestalt knowledge of a scene can be obtained by a vision system that is able to generalize. An example of this is generalization by training the system to associate various sets of image primitives with the same label. For instance, differentiating rivers from roads in reconnaissance images can be done by learning techniques that select the discriminating features rather than through hand analysis by experts. Finally, because machine learning systems must be able to judge algorithm fitness, it forces the use of an objective algorithm-quality measure. Of course, quality measures will vary for individual vision tasks, but consistent use of objective metrics can give needed insight into system performance.

4. MACHINE LEARNING TECHNIQUES

This section presents an overview of available learning techniques. These techniques are described relative to the domain of computer vision, and with the goal of improving the performance of the algorithm. First, common components in learning systems are explored, then in each subsection that follows the technique is presented along with example application domains. It is hoped that from this information the designer will be able to select the most appropriate learning mechanism for the intended application. This information is presented in a general form to stress how the methods are applicable to a variety of domains. To clarify the information, a specific case of applying the techniques and strategies is presented in the second half of this chapter (Section 5).

Before presenting the individual learning mechanisms, it is useful to discuss some characteristics common to all the learning methods that will be discussed. One common attribute is the use of stored knowledge. "Knowledge" is simply the term used for the collection of tools or building blocks that the learning system has available to it. Indeed, the strength of a learning system stems from its ability to manipulate knowledge into useful forms. Therefore, both the mechanism to manipulate knowledge and the structure of the knowledge itself can have an impact on system performance. For example, in a particular system, the knowledge might take the form of a rule-base used to represent the archetype of an object's 3D framework. Initial knowledge could be "hard-coded" into the system, and new knowledge could be acquired through manipulation or extrapola-

tion of existing rules. Another example of knowledge would be the structure of preprocessing stages used to extract edges in the scene for use in a recognition task. The knowledge base would be the set of stage organizations that have been explored or are currently under consideration. The rules for manipulating the stages are also part of the system knowledge. It is important to note, however, that the goal of learning is not simply to acquire new knowledge but to work with existing knowledge to develop a better algorithm.

An effective system may also choose to forget, or to "prune" knowledge. In the example given above, this would include removing stage combinations that have proved less effective. In this case it may also be necessary to construct a class of knowledge that was purged to avoid generating it in the future.

The method of classifying and storing knowledge is closely related to another common attribute of the learning mechanisms: a learning bias (11). This determines what can and cannot be learned and, therefore, should be adjusted carefully. Bias is exhibited in both the knowledge representation scheme and the knowledge manipulation tools. In terms of computer vision, this will limit the form of algorithms that can be discovered and that need to be avoided.

Nevertheless, bias is a necessary restriction. Without bias, the learning system can have an infinitely wider domain of potential solutions to discover. Such a system, however, would have little focus, and as a result, little hope of discovering anything promising. Balancing the tradeoffs of bias can be performed dynamically and to some degree automatically. By properly scheduling the operations in a learning system, the system can gradually increase its bias throughout the learning process. Figure 9.1 shows this graphically over the concept space. As time progresses, the bias

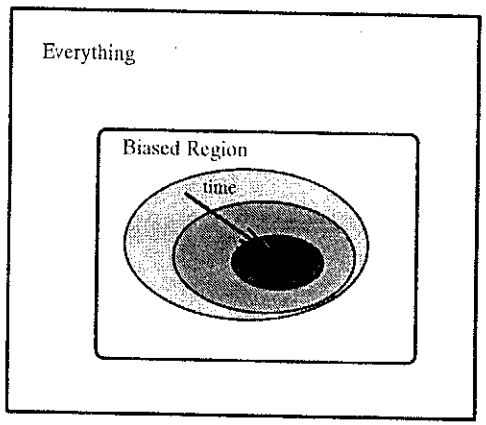

Figure 9.1. Bias (size of concept space) increasing over time.

should become stronger and stronger, limiting the search space to smaller regions. The tradeoff is that with a smaller space, the relative performance can be better. This form of scheduling helps determine what areas should be given preference. In practical applications, it is important to balance bias with scheduling in order to narrow the search, since only a finite amount of time is available.

Machine learning can be divided into several different strategies. Common divisions include (9):

- Rote learning
- Learning by instruction
- Deductive learning
- Learning by analogy
- Inductive learning

Rote learning and learning by instruction are the most primitive. *Rote learning* simply refers to knowledge being directly implanted. *Learning by instruction* differs from rote learning only in that the machine is responsible for compiling the knowledge into a readily usable format. These methods are inappropriate as a primary mechanism because none of the burden of discovery is taken on by the machine. *Deductive learning* is the process of using a set of axiomatic knowledge and truth-maintaining transformation rules to construct more useful forms of knowledge. This method is also of little utility in computer vision because the foundation of many practical algorithms is heuristic in nature. *Learning by analogy* is the process of mapping knowledge from one domain to another. This method is useful primarily when the learning domains are very similar. Because the knowledge must already be in a machine-usable form, it is often coupled with another learning technique used in the original domain. An example of learning by analogy would be transforming the algorithmic knowledge learned for a simulated vision domain to that of the real world. The domain change could also be less drastic, perhaps consisting of the extension of day-vision to a night-vision system. This transformation could be performed under user supplied rules formed independently of the knowledge in the source domain.

4.1. Inductive Learning

Learning by induction is by far the most powerful of the learning strategies for the domain of computer vision algorithms. This strategy encompasses many subcategories, some of which can be combined: learning by example, learning by exploration, data-driven learning, and model-driven learning.

The first category, learning by example, is both simple and powerful. In this case, training samples are presented from an external entity, and it is the responsibility of the system to structure the knowledge constituting the presented class. For example, a high-level vision system may be trying to recognize flat parts on an assembly line based on their 2D outline. Part of the recognition algorithm is the model used by the machine to perform identification. A system to tackle this problem could be composed of an edge detector and an outline constructor. Sample images of the object to recognize, along with its correct identity, would be presented to the system. In each training instance, the system would construct a model for the object. As more examples are presented, the object model used by the machine would adjust to accommodate variances. In this case the machine is learning the knowledge that will be used by the system to perform the recognition task.

Training samples presented in this way can be either positive or negative. In the case mentioned above, positive samples were used. The object to be recognized—say, a bracket for a car—is presented in image form, and the system is told to associate the label "bracket" with the model that it built. The training sample, however, could also have been a negative one. This would mean presenting something that the system should not recognize as being a bracket. The model modification algorithm would then make sure that the stored outline for a bracket did not fit the outline extracted for the negative example. Training with both positive and negative examples can be important for a system that must differentiate multiple objects. This kind of training ensures that key *differentiating* traits are emphasized in the models rather than just *identifying* traits.

The presentation of training samples to the learning system can have an impact on the type of algorithm that is learned and also how much training is necessary to achieve adequate results. Because formation of the acquired or learned portions of the algorithm is performed incrementally, the initial training samples can play a more significant role than latter ones. Additionally, it is important that the training domain represent the intended target domain and not simply a single case of each type of the scenes that will be encountered. This point is best illustrated by an example.

Suppose a vision system is constructed to recognize navigable terrain for a mobile robot. There are several factors that must be considered in training the system. If the system operates on solar power, it may be necessary to explore primarily during sunlight. Therefore, the training domain would emphasize brightly-lit daytime scenes with perhaps a small number of dawn and dusk images. The algorithm learned for one domain may not be best suited for all. If the risk of making a mistake is higher during the period of lower light, however, it may be best to over-represent training images during these periods as a counter measure.

Ideally the system would perform its learning by actually working in the domain for which it was intended. This is called learning by exploration. However, in this situation it becomes more difficult to verify the performance of the algorithm, so the result is that the system receives less effective guidance in modifying existing and creating new forms of the algorithm.

Learning by exploration can still be effective if an oracle is available to provide feedback to the system. One method of implementing the concept of an oracle is to make the entire training database available to the vision system, but have the system responsible for selecting the training cases. This can be especially beneficial if training is expensive in terms of image acquisition costs or available time. Typically training time greatly exceeds the time to apply the algorithm that has been learned. If the training database is available, the system can apply the current algorithm to training cases only in the cases where the algorithm produces undesirable results. An example of this would be in an artificial neural network designed for character recognition (8). The inputs would be the grid of pixels containing the character. In this case it would save time to train only over the cases where the existing network weights led to incorrect classification. Nevertheless, a predetermined database does not have to be created for the disposal of the system. A human could act as the oracle, or another vision system could be used as the oracle. (This would be useful if the oracle system has either impractical cost or size to be used in the target system.) This approach still allows the system to explore on its own and train based on its own scene selections.

Another division in the structure of an inductive learning algorithm is data-driven learning versus model-driven learning. The data-driven approach starts from a specific form and generalizes (relaxes the model) as more examples are presented. The model-driven approach starts with a generalization (a user-supplied model) and refines (specializes the model) as more examples are presented. In the example of learning the archetype for the car-bracket recognition problem, either the data-driven or the model-driven approach could be used. In the data-driven approach the system may start initially without a model. The first training instance would supply the initial concept of the archetype. Additional training instances would broaden the model to encompass the new instance as well. This could be accomplished by increasing matching tolerances in the model. Depending on the format of the model, the tolerances might be for lengths of edge segments or angles of connection of the segments.

The same target problem could be model-driven. In this case the system would be primed with a very broad description of the part and would have wide tolerances. Then the model could be refined by training with either positive or negative training cases. With negative training cases the model tolerances would be reduced if an accidental classification of the negative

example occurred. For positive training cases a stepwise procedure could be used to reduce the tolerances to more aptly fit the training case.

In summary, inductive learning can be applied under a variety of conditions. It is primarily effective when training data is available either as positive or negative examples. The training data can be a prepared set, or can be selected by the machine under its own guidance. (In either case, however, the correct answer needs to be available for the purposes of feedback to the learning system.) An indication of algorithm learning performance can always be obtained by reserving some instances of the training set for testing purposes.

One drawback of learning systems is that it may be forced to make a tradeoff with its bias. If the system is strongly biased, it will have a narrower set of solutions that can be learned. This can lead to a higher learning rate by bypassing ranges of knowledge, but if an effective concept lies in the excluded range then it will never be discovered.

5. CASE STUDY

This section presents a case study to demonstrate how the methods presented in the last section can be applied. The design of the system is developed step by step, and the rationale for the necessary tradeoffs is presented as appropriate. Finally, results are presented to show what type of gain a system can experience with the addition of learning techniques.

The target system is a general stereo vision system. It is designed to be used as an integral stage in a 3D object recognition system or an image understanding system. It is general in the sense that it performs standard feature matching to achieve correspondences used to gauge depth. The following section briefly describes the stereo matching system. This is necessary to understand the rationale for the tradeoffs to be made in the system with the addition of a learning component.

5.1. System Background

In our study, we use a correspondence-based stereopsis algorithm whose local correspondences are constrained by a number of parameters such as epipolar constraints and orientation. Our algorithm belongs to Marr and Poggio's paradigm that uses monocular feature extraction, local stereopsis, and global stereopsis (6). Other models of stereopsis (16, 18) are not considered in this study.

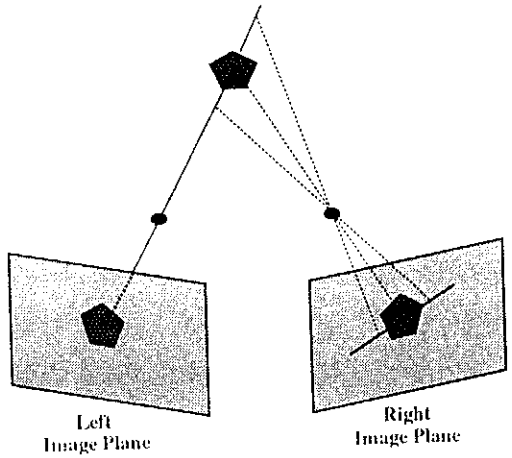

Figure 9.2. Stereo camera setup.

The standard two-camera (binocular) stereo vision setup is shown in Fig. 9.2. Here the two cameras are pictured facing into the page. The object being imaged is the pentagon. One image alone (left) narrows the position in 3D space to a line passing through the focus of the camera and the position of the object in the image. The matching position of the object in the second image (right) conveys where along that line the object lies. The general procedure for discrete binocular stereo vision can be summarized in five steps:

1. Obtain two images from different viewpoints.
2. Extract the tokens, or scene features, from each image to use for matching. (This information is known as the *primal sketch*.) (5, pp. 188–193).
3. Determine the correspondence between tokens found in each image. (This results in the disparity map.)
4. Translate disparity values into depth values. (This results in the depth map.)
5. Interpolate over the depth map as necessary for the desired resolution of depth information.

The first step can be accomplished using two cameras separated by a small distance, say, 25 cm. A typical image might be 512×512 pixels with 8 bits for recording pixel intensity, i.e., 256 gray levels.

In the second step, the matchable features are extracted from the intensity information. Edge detection preprocessing is used to extract these higher-level tokens. Tokens, such as edges, tend to emphasize areas

of interest for object identification, and these are points at which depth information can be useful.

The third step, known as the *correspondence problem*, has traditionally been the most difficult. If the matching criteria are too loose, then depth information is lost because of ambiguous potential matches. However, if the matching criteria are too restrictive, then depth information is lost because no match can be found. There is no known analytic function to optimize for the selection of the matching criteria. The problem is best approached by heuristic methods generated by either human experts or machines.

The fourth step in the stereo matching procedure is to transform disparities into depths. This is straightforward when given the geometry of the camera setup. Horn (3) gives a concise outline of this procedure.

The fifth and final step of interpolating depth values can be solved with a variety of interpolation methods as well as with knowledge-driven interpolation schemes. For the sake of simplicity, the nearest-neighbor method is used here.

The procedure in steps 2 and 3 is repeated iteratively to refine the depth measurements made by the system. The idea is to start by extracting few, but prominent, tokens using a large search region. This provides a good initial depth estimate. Further stages extract more tokens but can use previous depth information to reduce the size of the search region. Each stage of this algorithm is called a *channel*, and each channel has a granularity defined by the "coarseness" of the tokens extracted at that stage. One attractive aspect of this approach is that it is thought to be similar to the process used in biological visual systems (2). A pictorial view of the algorithm is shown in Fig. 9.3.

The matching of corresponding features in the images occurs on a medium level. The tokens that are matched consist of edge pixels or *edgels*. The edgel extraction is done using an enhanced version of the Canny edge detector (1) that employs hysteresis for greater edge continuity and retains midprocessing information to be used for matching.

5.2. System Analysis

After choosing the vision algorithm to be used, the designer needs to analyze the algorithm and determines the goals of the learning system. At this point, the exact learning mechanisms are of secondary importance as compared to the goals of learning, which drives the strategy-selection process. (Strategy selection is presented in Section 5.3.)

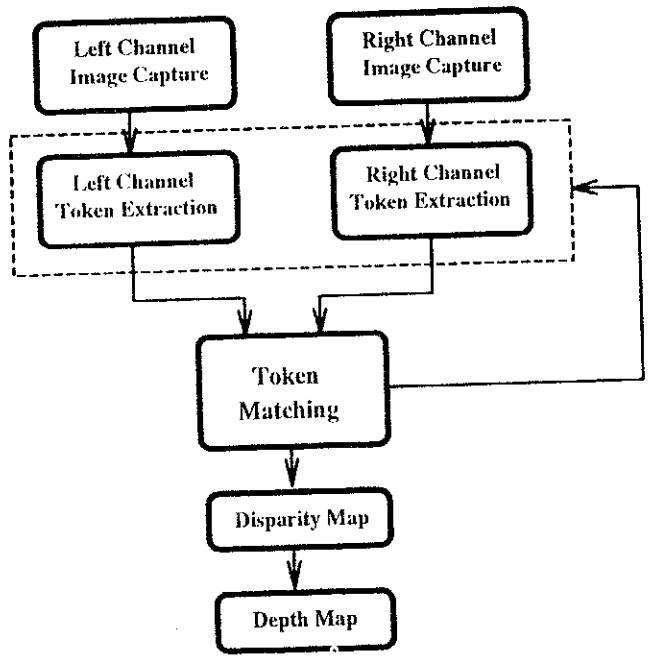

Figure 9.3. Stereo algorithm flowchart.

5.2.1. Specification of the Target Domain

This first step is to select the target domain based on the application requirements. Since the system will be learning on its own, it is necessary to provide it with training cases. The main effort of this step is, therefore, to decide the types of images that are representative of the domain. In our study, we have chosen day scenes of an outdoor environment. A sample stereo pair is presented in Fig. 9.4. The image training domain consists of 30 image pairs. Each stereo pair is a 128×128 gray-scale image with 256 gray levels.

To allow for automated experimentation, it is also necessary to supply the system with correct results for guidance. Here we want to ensure the accuracy of token matching; hence, we enhance the image database by adding the correct matches for sample components within the image. We manually selected and matched 10 sample test points from each image and recorded the correct disparity in the database.

At this point, it is also necessary to divide the training set in order to reserve some of the samples for testing the final vision algorithm to be learned. Another reason for reserving samples is to help detect common problems among learning systems: sometimes systems become very proficient at processing over their training set, yet they become so optimized that their performance actually drops on other test cases. Having a

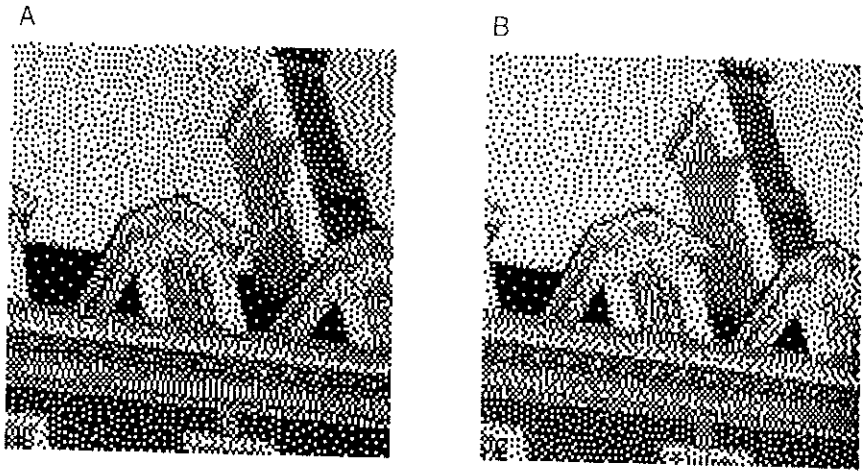

Figure 9.4. Sample stereo pair: (A) right image; (B) left image.

fraction of the training set reserved for testing can identify when this problem occurs.

5.2.2. *Component Selection*

The next step in the analysis phase is to isolate the components, or parameters, that are appropriate for machine learning. Using the general algorithm as outlined above, the methods suited for machine tuning lie in steps 2 and 3. These areas have the greatest impact on system performance and also the greatest flexibility. From these steps, the designer may first enumerate what exactly is open to machine adjustment. Table 9.1 lists the adjustable components that may be selected in this general token-based stereo matching algorithm. (Note that the unit of "pixel" as used in the table is a distance measure equal to the width of one pixel.) The list of parameters considered in Table 9.1 assumes a vision algorithm similar to Marr and Poggio's (6).

There are two problems facing the selection of the appropriate parameters. First, as is evident from Table 9.1, the parameters are continuous and can sometimes be unbounded. This means that the potential parameter space is infinite and, without a strong bias in the learning system, it becomes impossible to explore many of the possible combinations. The second problem is that there is little information available to guide the search for the appropriate parameters. Our approach is to use parameters determined intuitively by designers as a good starting point, and apply a machine-coordinated search for better values.

TABLE 9.1

Stereo Algorithm Parameter List

Parameter	Function and Purpose	Range	Units
Number of channels	Aids matching density by refining the search region and obtaining increasingly dense depth estimates	≤ 1	
Blurring kernel size	Width (σ) of the Gaussian filter used to select the strength of detected edges; along with the edge detection thresholds, this controls the density of edges detected in a particular channel	0.1–5.0	pixels
High threshold	Upper limit of gradient strength for hysteresis of edge thresholding	1–255	$\frac{\partial \text{ intensity}}{\partial \text{ pixel}}$
Low threshold	Lower limit on gradient strength allowed for edge classification	0–255	$\frac{\partial \text{ intensity}}{\partial \text{ pixel}}$
Initial search window	Size of region to initially consider for a match	1–200	pixels
Similarity thresholds	Used to determine if tokens are a match (similar)		
Gradient	Difference in gradient magnitude	1–50	$\Delta \frac{\partial \text{ intensity}}{\partial \text{ pixel}}$
Orientation	Difference in edge orientation	$\lvert\cdot\rvert \leq 30$	degrees
Vertical position	Difference in vertical position	$\pm$ 1–3	pixels

The first step is to narrow the possible component space to the most important factors. The choice for the initial subset is made regarding the effect and function that the parameters had on performance. Of the parameters listed, the number of channels, the blurring factor for each channel, and the edge detection gradient thresholds have been selected as the domain for the machine to search through. The other parameters were assigned values selected by experimentation and were held constant throughout the experiments. This set provides an acceptable balance between the size of the search space and the importance of the parameters; it is best to start with a small learning domain and increase the range searched incrementally. For a typical candidate, the number of channels varied from one to five, and the σ of the blurring kernel varied from 0.1 to 5.0.

The final form for a candidate parameter can be viewed as a set of triplets. Each triplet gives the parameters for a particular channel. For

example, the expression

$$\{(\sigma_1, lt_1, ht_1), (\sigma_2, lt_2, ht_2)\}$$

would represent the parameters comprising a two-channel candidate.

5.2.3. *Expressing the Learning Objective*

For the learning system to operate autonomously, it must be able to gauge its performance. The designer must specify the performance metric precisely so that it can be expressed in a machine-usable form. This metric is known as the *objective function*, whose inputs can be thought of as the algorithm components to be learned. In general terms, the goal of the system is to maximize the objective function over the learnable component space given the training database as input. To formulate the objective function, the designer must assess the goals of the vision system and represent them concisely for the learning system to use. Note that one possible way that the objective can be expressed is an equation with both symbolic and numerical components. Other possible forms include constraint equations that define where desirable parameter sets would lie.

In most applications, the designer is faced with overall performance requirements that must be met. It is from this that the quality of an algorithm is measured. For example, requirements could consist of a time limit or a minimum accuracy. In this respect, however, the objective is ill-defined. In other words, the precise performance tradeoff of accuracy or speed is not known and can be left to the system to decide. With respect to the stereo-matching problem, there are three factors used to judge its general performance:

1. *Speed*—the rate at which a unit area of the depth map is calculated
2. *Density*—the fraction of the image for which depth information is determined
3. *Accuracy*—the average error in depth measurements

The relative importance of each factor, of course, depends on the parent application employing the stereo system. The bias of application-specific modifications to the algorithm can be set by the user by formulating a particular equation of these three measurements. Without specifying minimum allowable values in the performance criterion, a simple performance equation might be

$$\text{Performance} = \text{speed} \times \text{density} \times \text{accuracy}. \tag{1}$$

This example objective function is specified as a mathematical function of three parameters, but because the actual objective is ill-defined, we should use a more flexible objective that can adapt to various application requirements. One possibility is to parametrize Eq. (1) as a family of functions and set bounds on allowable time, acceptable density, and acceptable accuracy of depth information. For example, the accuracy component can be expressed parametrically and could take a functional form itself. Similarly, other application-specific measures may be substituted as components in the objective function. An example might be the end goal of the vision system, such as the 3D model to be matched. This would allow the stereo vision system to be tuned with respect to its specific impact on the vision task at hand. An example of a formulation used in our case study is given in Eq. (4) later in this chapter. This function consists of a time penalty, an accuracy weight, and a density weight to measure performance.

In the rest of this chapter, we assume that it is the task of the learning system to hypothesize a set of more specific alternative objectives by performing tests. These objectives will be implicitly arrived at by natural tradeoffs made by the system. In turn, the particular objective function defined indirectly determines the types of candidates that the system finds. The objective function would be a function of only the stereo vision algorithm parameters and would return a measure of quality. It is expressed as a function of intermediate values gathered by performing tests on the candidate component set. Of course, it is the user's responsibility to review the machine-generated results and determine the most appropriate *explicit* tradeoff needed.

The objective defined in Eq. (1) involves tradeoffs among the cost of running the vision algorithm and the quality of the matched tokens returned by the algorithm. We assume that a fixed amount of time is allowed for completing the algorithm. Exceeding (and even approaching) this limit penalizes the fitness of a candidate. This is consistent with many real-world situations in which the vision system plays a role in a chain of cognitive acts and reacts to inputs in real time. Performing faster than this deadline would not be considered particularly useful. *Cost* is then defined as a penalty in exceeding the specified time limit, where the penalty function is defined in terms of t, the time for the test

$$p(t) = \frac{1 + e^{-qt_0}}{1 + e^{q(t - t_0)}}. \tag{2}$$

The parameter q allows tailoring the penalty function with respect to the time limit t_0. The graph of the penalty function for $q = 0.05$ and $t_0 = 120$ seconds is shown in Fig. 9.5.

Other than the speed of the stereo vision algorithm, it is also important to maximize the density as well as the accuracy. *Density* is measured in

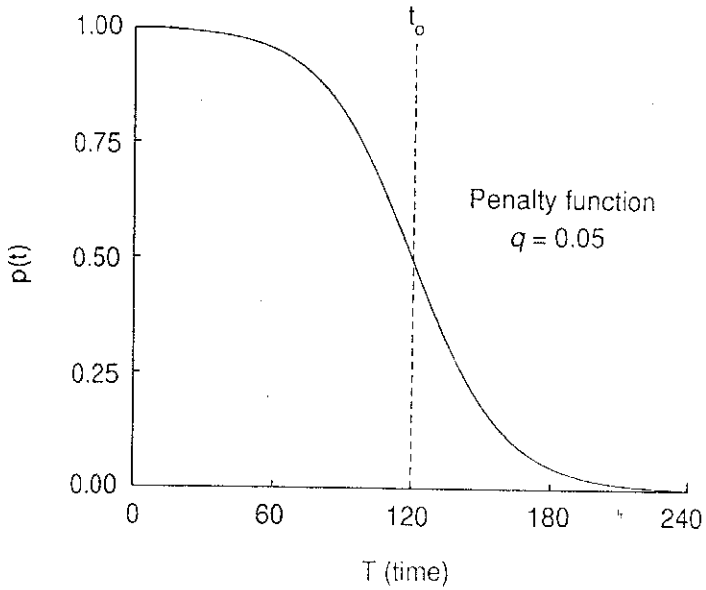

Figure 9.5. Time penalty function.

terms of the fraction of the image for which corresponding edgels were found. *Accuracy* is measured by probing the resulting disparity map and comparing the values with known disparities. For the test images in our database, 10 points were measured per image. The accuracy used by the objective function is the normalized average disparity error with 0 error mapping to 1, and an error of $e_{\max}$ pixels or more mapping to 0. (The value of $e_{\max}$ used for our system was varied to simulate the effect of different application requirements. See Section 4.)

The final form of the objective function [Eq. (4)] is then a product of density (in matches per square pixel), normalized accuracy, and the time penalty function. It is computed as the average quality of the candidate as measured over each test image in the database. The quality measure over one image i with parameter set P is $q(i, P)$. This is a function of three intermediate test results that are each functions of the image and the algorithm parameters. The three components are $d(i, P)$, density of disparity measurements; $a(i, P)$, average accuracy of disparity measurements; and $t(i, P)$, time to process the image

$$q(i, P) = d(i, P) \times a(i, P) \times p[t(i, P)]. \tag{3}$$

The objective function, $O(P)$, now takes the form shown in Eq. (4), where P is the set of candidate parameters and B is the set of the N images in the database

$$O(P, B) = q_{\text{av}} = \frac{1}{N} \sum_{i \in B} q(i, P). \tag{4}$$

The objective function stated here is really one in a large family of possible objectives. By varying components such as t_0, q, and $e_{\max}$, the user can make the system search for particular parameter sets for different applications (cf. Tables 9.4 and 9.5). This kind of information is of interest because it shows how the system focuses on different classes of candidates under different system requirements.

5.3. Machine Learning Enhancements

After analyzing the vision problem from an algorithmic perspective, we need to select the appropriate learning mechanisms. These mechanisms include the structure of system knowledge and the methods for manipulating knowledge. Together these components will form the bias of the system and determine what can and cannot be learned.

5.3.1. Knowledge Selection

To specify the format for system knowledge, it is helpful to analyze how a human would behave in the machine's place. A person would track candidate performance and would develop an understanding of what makes one candidate better than another. A person would also keep track of the deadline for finding the solution. This would have an influence on how many experiments would be performed as time passed. In the beginning of the process, it might be best to try a variety of different strategies to gain a feeling for the relationship that particular parameters have on system performance. As time runs out, it would be best to focus on fine-tuning the parameters.

With this view, the system will need to store and manipulate three types of knowledge: (1) past candidate performance, (2) candidate performance relationships, and (3) system constraints and goals.

Past candidate performance can be stored in a performance database indexed by candidate identifiers. These identifiers are simply tags unique to the algorithm components. The values of past performance are the intermediate performance measurements obtained by applying the algorithm to sample images. There will be a performance entry in this database for every test image to which the candidate has been applied.

The relationship between candidates is important because it can be used to guide the creation of new candidates. Each relationship is stored as a "delta" vector in the candidate components with an associated value for the measured change in performance. This type of knowledge must be carefully maintained for two reasons. First, it can grow exponentially with the number of candidates. Therefore, it is best to limit remembered relationships to neighbors that are close within the component space. The second reason for maintaining the knowledge is that ineffective relationships can cause an unnecessary waste of time and storage space. The system must be able to track relationships and learn when a potentially promising one has created only false leads. The specific details of relationship knowledge and how they are to be used are discussed in more detail in Section 5.3.2.

The final piece of knowledge that the system must maintain is the set of constraints and the goals of the system. Any practical learning system will be faced with resource constraints, and, therefore, must tailor its actions to best reach the assigned goals in view of the constraints. The goals are represented in either the objective function or in the cost and quality constraints, and can take the form of a mathematical function or a symbolic expression. The knowledge of the time constraints are stored explicitly in the form of a deadline, but the knowledge that guides how to coordinate achievement of the goals under the deadline is stored implicitly in the candidate-generation mechanism. This mechanism is described next.

5.3.2. *Candidate Creation*

Initially, some sample parameter sets (or candidates) are specified by the user. As testing progresses, however, the system needs to be able to create new candidates. Since each candidate the system creates represents a potential increase or decrease of algorithm performance, it is important to create new candidates that are guided by structures of promising candidates (those with good sample performance values). If they were created haphazardly, we could almost be guaranteed to spend our computational resources on many poorly performing ones.

The generation of new candidates is handled by using rules to represent knowledge in candidate generation. Because of the limited amount of time available for testing, the candidate generator should refer to the performance of previous candidates when generating new ones. Two methods are used here: pseudorandom and greedy. In the pseudorandom approach, a new candidate is generated by a perturbation from one of the existing candidates. The candidate used as a basis for this perturbation must lie in the top third of the existing candidates. In the greedy approach, the generator tries to follow the "direction" of the greatest improvement in

performance, based on the performance of candidates already generated. This direction is expressed as a vector of the delta values between the parameters of the two candidates.

The knowledge for candidate creation is stored in the form of rules. This rule-based implementation is flexible enough to allow the addition of further domain knowledge as it becomes available. New operators and rules can easily be added to give incremental improvement. The rule base consists of a set of assertions called *working memory* and a set of rules for operating on it. A rule consists of two parts: a list of conditions and a list of actions. When all conditions within a rule are met, the rule is said to fire, and all the associated actions are carried out. In general, a rule-based system allows asynchronous execution of rules by relying on a conflict-resolution strategy to select one rule to fire at a time. In this case, however, the rule base executes the rules sequentially. A condition can be either a pattern or a function. A pattern is considered met by finding a unification between the pattern and an assertion within the working memory, whereas a function is considered met when it returns a nonempty value. Variable assignments and wildcards can be used in both patterns and functions. There are three basic types of actions: (1) delete an existing assertion from working memory, (2) add a new assertion to working memory, and (3) execute a function.

Actions can use values of variables assigned in the conditional parts of rules. This approach allows a flexible system for analyzing existing candidates and for generating new ones.

The creation of a candidate follows a regular procedure. During this process, only well-tested candidates are considered worthy of reference. (Relatively untested candidates are avoided because their "goodness" may not yet be particularly reliable or stable.) A candidate is considered well tested if the confidence of its performance is above a given level for its sample mean to lie within a certain range of its population mean.

The first step in candidate creation is to select a reference candidate from which to create the new one. This candidate is randomly selected from the top third of the well-tested candidates in the pool. Next, the candidate's neighbors are checked to see if they have better performance. If so, then the reference candidate is updated to be the best neighbor. This candidate will now serve as the basis for the new candidate. Following this, the neighbors of the reference candidate are again polled for their performance values. The "direction" of the greatest change in performance is then recorded. This direction is represented as a numerical vector (called a *transform*) that specifies the change in the parameters between the two candidates. If no existing candidates lie in the path of this transform, then the transform is applied to the reference candidate to generate the new one. However, if this transform leads into a region of the parameter space already occupied by a candidate, then a random transfor-

mation is created to move into unexplored space. The random transform is created by first recording the transforms from the reference candidate to the neighbors. One of the channels of the transform is then selected for modification. An individual replacement channel transformation is then randomly created. If it is found to be similar to any of the existing transforms for that channel, then the channel transform is recreated, but with a greater magnitude. This process continues until the new transform leads to an unexplored region of the parameter space.

This method of candidate generation combines two important concepts. The first is that existing good transformations are followed. This gives an effect similar to hill-climbing in the parameter space. The second benefit of this method is that randomness is employed as a hedge against stagnation in a local maximum on the objective-function "surface." This randomness is exploited in two ways. First, it is used as the reference candidate and is set by randomly selecting from the top third of all candidates, rather than just using the top, or *incumbent*, candidate. Second, as the local regions of the parameter space become more well searched, the transforms try to break away from the current region by issuing larger perturbations.

Table 9.2 lists the candidate-generation rules more explicitly. In the table, C_{ref} is the chosen reference candidate, C_{max} is the best neighbor of the reference candidate, and C_{inc} is the incumbent (or best) candidate.

TABLE 9.2

Rules Used for Candidate Generation

Rule no.	Description
1	Randomly select C_{ref} from the top third of the well-tested candidates
2	If there is a candidate in the same neighborhood as C_{ref} that is better than C_{ref} and has a greater quality disparity from C_{ref} than all others in the neighborhood, then use that candidate instead of C_{ref}
3	If no candidate in the neighborhood of C_{ref} has a performance in the top $\frac{2}{3}$ of all well-tested candidates, then use C_{inc} instead of C_{ref}
4	Find a candidate C_{max} with the greatest disparity from C_{ref} among all candidates in the neighborhood; record T_{max} as the transformation that creates this disparity and Δ as the change in quality
5	Use the transform that caused the highest increase in performance in the neighborhood of C_{ref} as the basis transform for generating a new candidate
6	If there are no existing candidates in the direction of the basis transformation, then use the basis transformation to generate the new candidate
7	Otherwise, generate progressively greater transforms until one is found that leads to an unexplored region of the parameter space; then apply this transform to C_{ref} to generate the new candidate

5.3.3. Scheduling

Unfortunately, the domain of stereo vision parameter tuning is knowledge-lean; that is, the knowledge that leads to the generation of better candidates is either very difficult to acquire or too expensive to be useful. Therefore, it is more important to have an efficient procedure for testing the generated candidates than to develop methods for capturing new knowledge for candidate generation. Generating parameter sets and testing them in a controlled manner is commonly used by knowledgeable experts who hand-tune the parameters themselves. This tedious task is best handled by automated machine learning; the difficulty, of course, is to transfer some of the expertise from the designer to the system so that the added speed and patience are put to proper use.

Each unique set of parameter values is considered a separate entity, or candidate, that could possibly lead to an improvement. The creation mechanism decides how many new candidates to create, how to create them, and how to schedule the available testing time. On the basis of a statistical model, the system trades between the quality and the cost of the vision algorithm learned, and incorporates methods such as gradient ascent to overcome the knowledge-lean aspect of the learning problem.

Even with the measure of the algorithm quality clearly defined, it is still not straightforward to automate the process of finding better parameter values. To fully evaluate the performance of a parameter set, it is necessary to test it over the entire training domain database. Clearly, a gradient-ascent-type method is impractical here. For such a method to work, it would be necessary to perform full evaluation frequently. The primary tradeoff is between the number of candidates to be evaluated and the amount of evaluation performed on each.

The generate-and-test framework used here is geared toward searching an ill-defined space under a given time constraint (see Fig. 9.6). Before examining its structure, it is necessary to define some terms. The set of all candidates currently under consideration for testing is called the *candidate pool*, and the set of images used for testing the candidates is called the *test database*.

The three main parts that constitute the core of the framework are the *candidate generator*, the *candidate evaluator*, and the *resource scheduler*. The candidate generator creates new candidates for consideration; the candidate evaluator tests the candidates on the test images and records their performance; and the resource scheduler determines which candidate to test next.

If there is much domain knowledge available, then the candidate generator can be very sophisticated. In this case, the candidate evaluator plays a subordinate role. However, in knowledge-lean domains, the candidate generator is relatively primitive. Therefore, the burden of selecting good

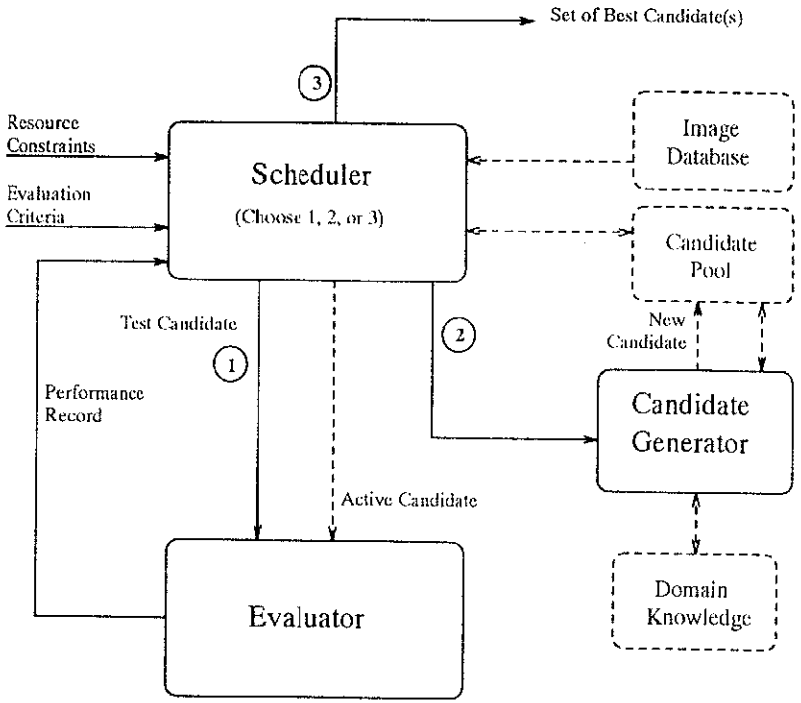

Figure 9.6. Generate-and-test framework.

heuristics shifts to the resource scheduler and the candidate evaluator. Such is the case in the stereo vision domain. Consequently, the focus is placed on the resource scheduler and the candidate evaluator. (It is worth noting, however, that the strategies presented here are general enough to be applicable to knowledge-rich domains as well.)

To avoid spending a large amount of time on poor candidates, the evaluation process is divided into small subtests called *quanta*. This allows the system to perform additional tests on candidates *only* if they demonstrate some merit during prior quanta. During one quantum of time, tests are performed on the selected candidate using test images from a data-base supplied by the user. There are two possibilities for the test images used here. They can be either relatively small images (64×64) that give only a mild indication of fitness of the candidate, or they can be fairly large images (512×512) that give a good indication of candidate fitness. The advantage of the latter approach is that the candidate is tested under more realistic conditions; however, its drawback is that the duration of one quantum must be large enough to accommodate the test. This can then have an adverse effect on the system's performance. In the stereo vision domain, parameter-set performance scales relatively well from small to

large test images. Therefore, candidates are tested using small images, with the goal being to find the candidate with the greatest average performance. Of course, the optimal objective is to find a candidate that performs the best in all cases. This, however, is impractical or impossible to verify unless all candidates are evaluated over the exact same (and possibly infinite) set of test images representing the application domain. This would imply that all candidates (including the worst) must be evaluated to an equal degree.

At the end of each quantum the scheduler selects one of the following actions to perform:

1. Select the next candidate to test from the candidate pool.
2. Generate a new candidate to be placed in the candidate pool (and possibly remove an existing one from the pool).
3. If the deadline has been reached, select a set of the best candidates and terminate testing.

The decision between choices 1 and 2 is based on the current performance of the candidates in the pool and the amount of evaluation that has been performed on each. One simple method for determining when to generate new candidates for the pool is to generate new ones whenever existing ones have been evaluated to have a performance value known to within a statistical confidence level.

If the decision is made to pursue choice 1 or 3, then the candidate(s) is (are) selected on the basis of an *evaluation criterion*. This consists of two parts: the *goodness function* and the *guidance strategy*.

The goodness function is an estimator of the value of the objective function [Eq. (4)]. It is used to select from the pool the candidate that most likely performs the best. It is needed because candidates may not be fully evaluated to within a statistical confidence when testing is terminated.

The guidance strategy is used (if testing is continued) to select the candidate to be evaluated during the next quantum. The goal of the guidance strategy is to choose a candidate that maximizes the probability that the candidates with the highest objective values also have the highest goodness values. It is not always best to select the most promising candidate to test because a candidate may show less promise when limited tests have been performed but might appear better with more tests. Moreover, with limited resources, it might be necessary to explore more candidates early in the generate-and-test process and focus on a limited set of promising ones as time runs out. This tradeoff is addressed by using a statistical model for sequential selection.

If the decision is made to pursue choice 2, then a new candidate must be generated. The candidate generator used here is driven by a fixed set of

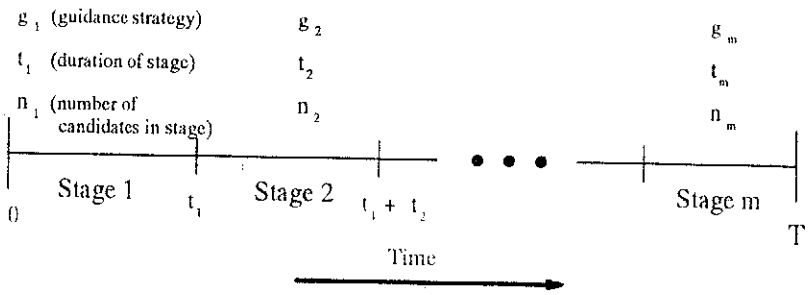

Figure 9.7. The multistage testing procedure.

rules. An outline of the actual code comprising this framework can be found in Schwartz (12).

Guidance strategies must be developed to take into account limited resources such as testing time, and that can deal with situations in which there are so many candidates that it would be impossible to test them all even once. Because of these conditions, the guidance strategies must trade between the number of candidates that can be tested and the accuracy of the sample mean for selecting the best candidate in the end. If more candidates are tested, then the accuracy of the estimated performance would decrease, and the final selection of the best candidate would likely be in error. If a candidate is tested more thoroughly, then fewer total candidates can be tested and, therefore, a greater chance of missing a good one entirely.

To overcome these problems, it is necessary to formulate a general guidance strategy $G(T)$. This is accomplished as a series of stages in which each stage is represented by $G_i(g_i, t_i, n_i)$, where i ranges from 1 to m, the number of stages. Each stage is then characterized by the triplet g_i, t_i, n_i. The particular guidance strategy to be used for stage i is g_i; the duration of the stage is t_i; and the number of candidates to be considered for testing during that stage is n_i (see Fig. 9.7). In this methodology, the early stages correspond to coarse initial testing used to weed out unworthy candidates, and the latter stages correspond to more careful evaluation of the better candidates. Of course, only the candidates that have the top n_{i+1} sample-mean performance values at the end of stage i are carried over into stage $i + 1$ for further testing. Finally, only the top candidate is selected at the end of the last stage. This entire procedure is a way of incrementally increasing the bias of the learning system by allowing many candidates to be tested in the early stages, and by obtaining accurate sample-mean values for the better candidates by more thorough tests in later stages. For a full discussion of a variety of guidance strategies the reader is referred to Ieumwananonthachai *et al.* (4).

5.4. Results

This section presents some of the actual results of implementing the system described in the previous sections. The results were generated from actual runs of different durations and with different objective-function formulations. (In addition, simulated runs are presented to provide a more complete picture.) Before performing the experiments in the actual runs, the candidate pool was seeded with six candidates that were tuned by hand (see Table 9.3). All experiments started with these candidates to show the different types of candidates that can be discovered as the available time and the objective function are modified. From these initial candidates, the rules shown in Table 9.2 were applied to generate new candidates as necessary. At the end of each run, all candidates were then fully evaluated over all images in the database. Here the database consisted of 30 images. Testing of the candidates was performed on a Sparc IPC workstation using the general binocular stereo algorithm implemented in C language. Typical execution times were roughly 20 seconds per channel on the 128×128 gray-scale images. Of course, this time varied widely depending on the number of tokens found in each channel. This full evaluation allows judging the performance of the system by giving insight into the quality/time tradeoff. The quality loss is the difference between the best found under the guided system and the best found by

TABLE 9.3

Hand-Tuned Seed Candidates

Candidate number	Channel width	Low threshold	High threshold
1	1.3	2.0	5.0
2	0.6	2.0	5.0
3	0.9	2.0	5.0
	0.6	2.0	5.0
4	1.8	2.0	5.0
	1.5	2.0	5.0
5	2.4	2.0	5.0
	1.7	2.0	5.0
	1.0	2.0	5.0
6	3.0	2.0	5.0
	1.0	2.0	5.0
	0.4	2.0	5.0
7	1.8	2.0	5.0
	1.3	2.0	5.0
	0.8	2.0	5.0
	0.6	2.0	5.0

TABLE 9.4

Performance from Actual Tests with Varying Objectives

Objective parameters	Total duration (in no. of tests)	
	200	400
$t_0 = 60$	2.4–2.0–5.0/1.7–2.0–5.0/1.3–0.5–2.2(22)	2.4–2.0–6.6/1.5–1.7–5.0/1.0–2.0–5.0(18)
$e_{max} = 5$	0.0369	0.0416
$t_0 = 60$	1.3–0.5–4.3/1.5–2.0–4.8(29)	2.4–2.5–5.0/1.7–2.0–5.0/1.0–1.6–5.0(140)
$e_{max} = 2.5$	0.0143	0.0251
$t_0 = 45$	1.6–1.4–2.5/0.7–4.0–8.3(30)	3.0–2.0–5.0/1.4–2.1–2.1/1.1–1.7–2.2(109)
$e_{max} = 5$	0.0284	0.0430
$t_0 = 30$	2.4–2.0–5.0/1.4–2.0–6.2/1.0–2.0–5.0(48)	1.8–0.6–5.0/1.7–4.0–4.9/1.3–2.0–5.0/0.9–0.7–5.6(134)
$e_{max} = 2.5$	0.0199	0.0200

exhaustive tests. The time gain is the amount of time saved by using a guidance strategy rather than blind exhaustive tests. The full evaluation corresponds to finding the values of the objective function for the given candidates.

During actual tests, the testing strategy that was used was a two-stage round-robin–minimum-risk strategy (4, 12). The division of time between the two stages was equal.

Results from a combination of actual and simulated runs are combined to present a full picture, because it is very time-consuming to perform extensive tests on candidates whose individual test times can range from 1 to 2 minutes. (Simulated runs were performed by using preevaluated candidates rather than generating them on the fly.) For 200 candidates tested on 30 images, the total testing time can last up to 7 days. For this reason, it can be beneficial to perform additional simulations using results recorded from actual tests, if the simulations are close to observed actual results. Since a simulation can take less than an hour to complete, it clearly results in a great time savings. The problem with simulation, however, is that candidate generation is not modeled. Therefore, the candidates that were generated by the one recorded run will be the only ones used during the simulation. Nevertheless, empirical evidence has shown that simulation performance is very close to that of actual tests. Some justification for this conclusion can be found by comparing the results shown in Table 9.4 for real tests (those with candidate generation) and the results shown in Table 9.5 (the simulations).

For the actual (nonsimulated) tests of the system, three objective functions were tested for three different durations. The parameters of the objective functions that were used as well as the test durations are

TABLE 9.5

Performance from Simulated Tests with Varying Objectives

Time limit (t_0 in seconds)	Error Cutoff (e_{max})			
	2.5	5	7.5	10
5	2.4–2.0–5.0/1.7–2.0–5.0/1.0–2.0–5.0 0.0163	1.5–2.3–2.7(24) 0.0225	3.0–2.0–5.0/1.4–2.1–2.1/1.1–1.7–2.2(109) 0.0319	1.5–2.3–2.7(24) 0.0342
10	2.3–2.0–5.0/2.3–1.0–4.3/1.0–2.0–3.7(156) 0.0146	3.1–1.2–5.8/1.1–4.6–4.6/0.8–1.1–3.7(122) 0.0225	2.8–2.0–5.0/1.9–3.5–4.4/1.2–2.5–6.1(48) 0.0312	1.5–2.3–2.7(24) 0.0350
20	2.2–0.7–3.6/1.5–2.4–3.1/1.0–2.2–5.0(34) 0.0167	2.7–2.7–7.2/1.7–2.0–5.0/1.0–2.0–3.7(85) 0.0264	3.0–2.0–5.0/1.4–2.1–2.1/1.1–1.7–2.2(109) 0.0384	3.0–2.0–5.0/1.4–2.1–2.1/1.1–1.7–2.2(109) 0.0408
30	2.2–0.7–3.6/1.5–2.4–3.1/1.0–1.5–4.1(58) 0.0173	2.8–2.0–5.0/1.9–3.5–4.4/1.2–2.5–6.1(48) 0.0319	3.0–2.0–5.0/1.4–2.1–2.1/1.1–1.7–2.2(109) 0.0431	2.4–1.0–6.4/1.7–2.0–5.0/1.0–2.0–3.7(104) 0.0390
60	2.4–2.0–5.0/2.1–2.0–5.4/1.0–0.5–5.2(69) 0.0266	2.4–1.0–6.4/1.7–2.0–5.0/1.0–2.0–3.7(104) 0.0392	2.4–1.3–6.3/1.7–2.0–5.0/1.0–2.0–3.7(45) 0.0460	2.8–2.0–5.0/1.3–3.5–4.6/0.6–2.5–6.1(39) 0.0456

presented in Table 9.4. Each entry of this table lists the identity of the best parameter set that was found, as well as its performance, that is, the value of the objective function after 30 tests. The parameter set for each channel is encoded in the form (channel width–low-threshold–high-threshold) with channels separated by a "/". The number in parentheses at the end of the candidate name indicates that this was the nth candidate to be generated.

Simulated results for the case of a fixed duration and different objectives are shown in Table 9.5. Here two of the parameters constituting the objective function are shown on different axes (horizontal and vertical) of the table. Each entry in the table shows the identity of the best candidate as well as its performance value. The duration for each learning experiment was 400 tests.

As can be seen from Table 9.5, the objective function plays a major role in determining the sought-after solution. The difference in the results for varying t_0 and e_{max} demonstrates the two purposes of the objective function: the ability to express the goals of the designer, and the guidance of the search in an automated fashion. It is also beneficial to see the effect of extending the available testing time. As would be expected, spending more time should produce better solutions.

The best candidates found by the various objectives shown above are difficult to appreciate when the results are listed in a strictly numerical format. Presented here are some graphical results in the form of input images and matched results for the best performing seed candidate and the candidate found after 200 tests. These were generated with $t_0 = 30$, and $e_{max} = 5$. Each set of images consists of the stereo pair, the left and right edge maps found on the original channels, and those found on the discovered channels. The 3D relief of the matched edges and the disparity map are also presented for both parameter sets. The disparity map is generated by interpolating the entire image area by nearest-neighbor disparity values. It is meant to give only an indication of depth throughout the scene, since this type of interpolation does *not* preserve object outlines and appearances. The left and right images are displayed with the right image on the left so that it is possible for the viewer to merge them by crossing the eyes if desired. One thing that is not appreciable from the pictures, however, is the speed-up in execution. In all cases, the machine-tuned parameters performed faster and resulted in a 15–30% improvement as measured by the objective function. In the cases shown here, the best original parameter set was candidate number 5 from Table 9.3. The best parameter set found after a simulation with 600 time units with the above stated objective was used for the results shown below. This candidate was 2.2–2.0–7.0/1.6–3.8–3.8/1.3–3.0–3.6(41).

This first example, shown in Fig. 9.8, is of a sample building scene. Figures 9.9–9.14 show the old and new edge maps extracted for each channel. This image was processed in 115.26 seconds with the original

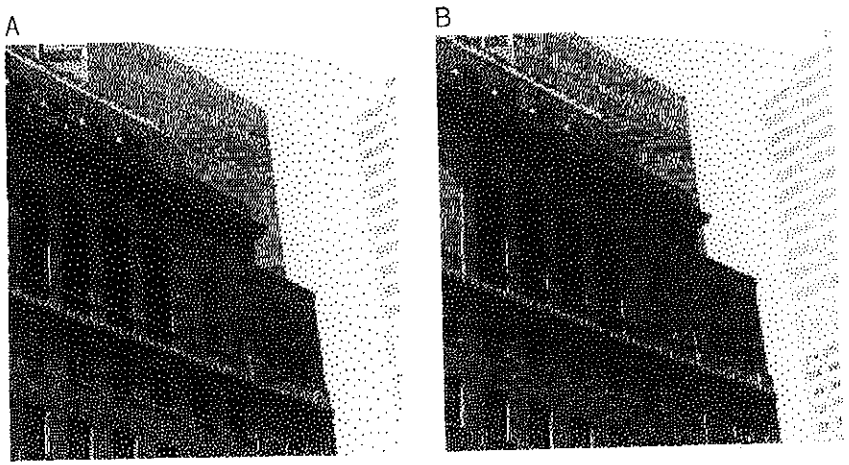

Figure 9.8. Building stereo pair: (A) right image; (B) left image.

parameters, and 78.83 seconds with the discovered (new) ones. The accuracy, however, dropped from 2.83 pixels to 3.12, but as the objective function emphasized speed, the new parameter set had a performance of 0.021 as opposed to the original 0.017. Figures 9.15(A) and (B) compare the 3D outline for the original and the new parameters, respectively. Figure 9.16 displays the same scene except in the form of an interpolated depth map with intensity representing depth.

Figure 9.17 shows a sample street scene. This image was processed in 130.71 seconds with the original parameters, and 100.10 seconds with the

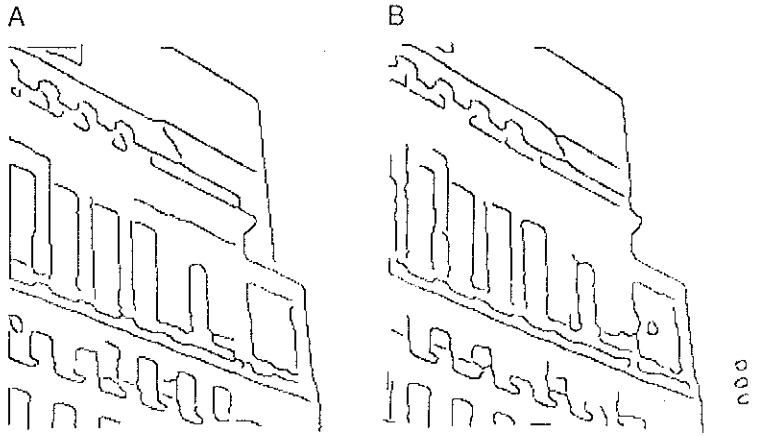

Figure 9.9. Original channel 1: (A) right; (B) left.

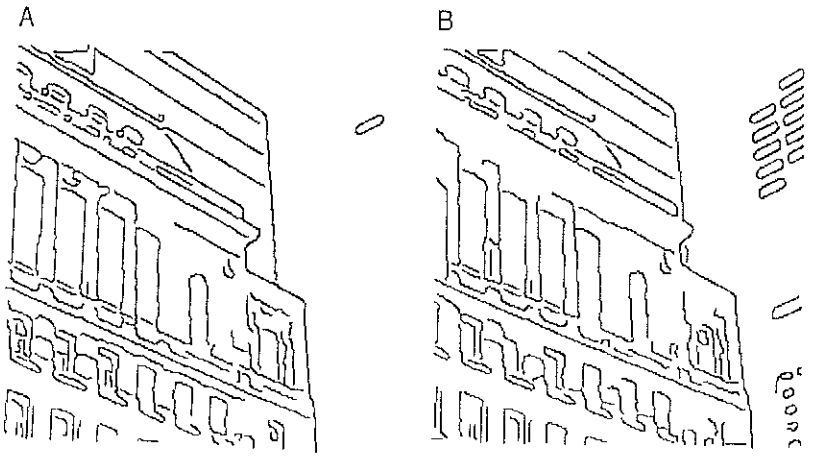

Figure 9.10. Original channel 2: (A) right; (B) left.

discovered (new) ones. In this case, the accuracy dropped from 1.4 pixels to 2.4. Nevertheless, the great increase in speed allowed the objective function to increase from 0.024 to 0.034, a 42% improvement. Figures 9.18–9.23 show the old and new edge maps extracted for each channel. Figures 9.24(A) and (B) compare the 3D outline for the original and the new parameters, respectively. Figure 9.25 displays the same scene except in the form of an interpolated depth map with intensity representing depth.

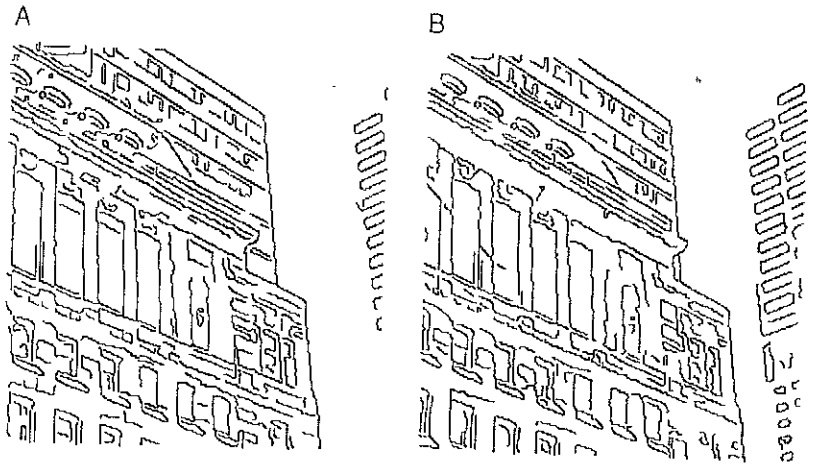

Figure 9.11. Original channel 3: (A) right; (B) left.

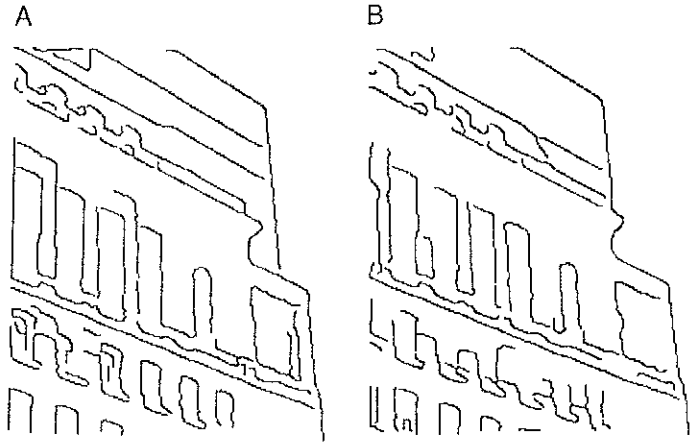

Figure 9.12. New channel 1: (A) right; (B) left.

Figure 9.13. New channel 2: (A) right; (B) left.

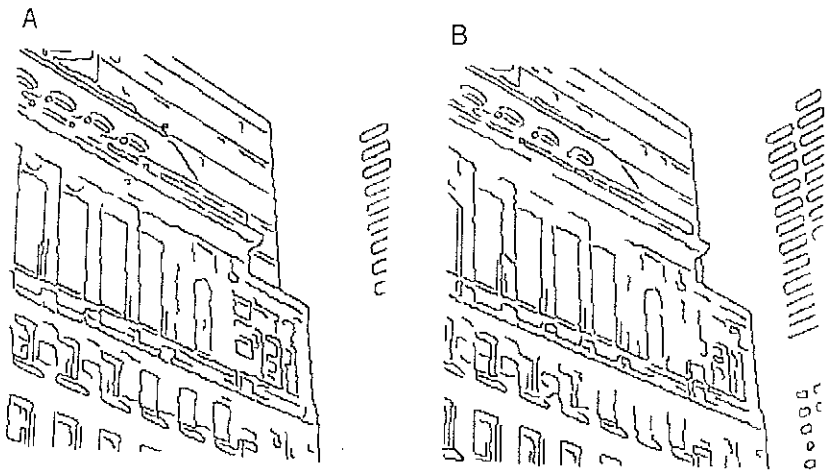

Figure 9.14. New channel 3: (A) right; (B) left.

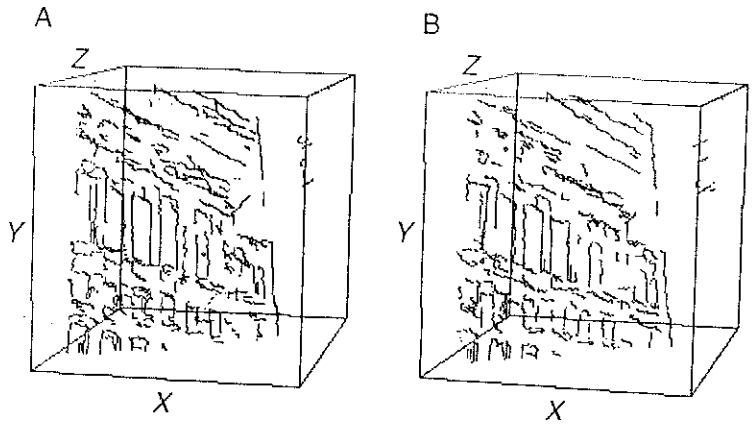

Figure 9.15. Scatter plot of points, 3D comparison: (A) original; (B) new.

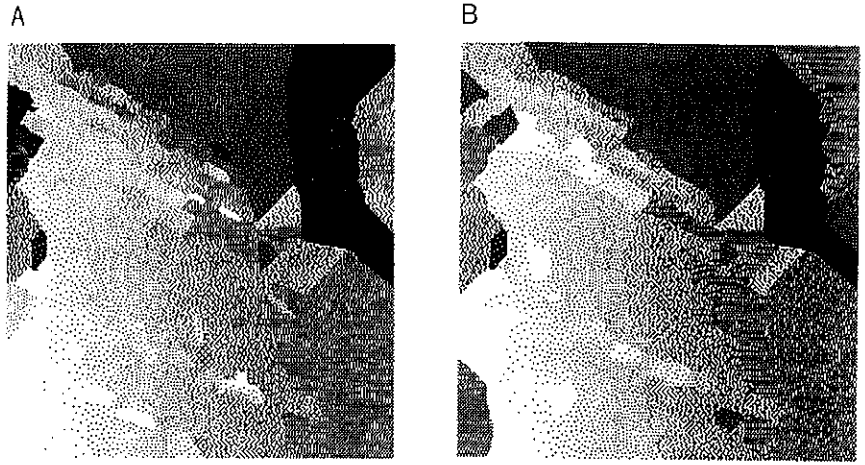

Figure 9.16. Three-dimensional interpolation comparison: (A) original; (B) new.

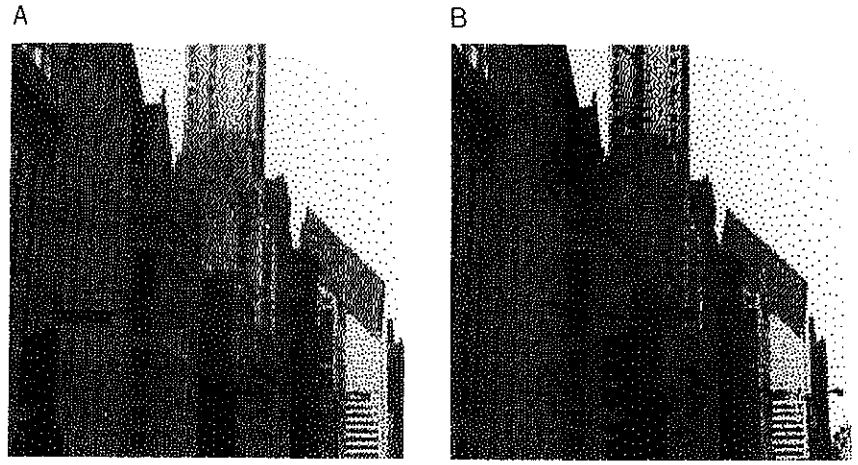

Figure 9.17. Building stereo pair: (A) right image; (B) left image.

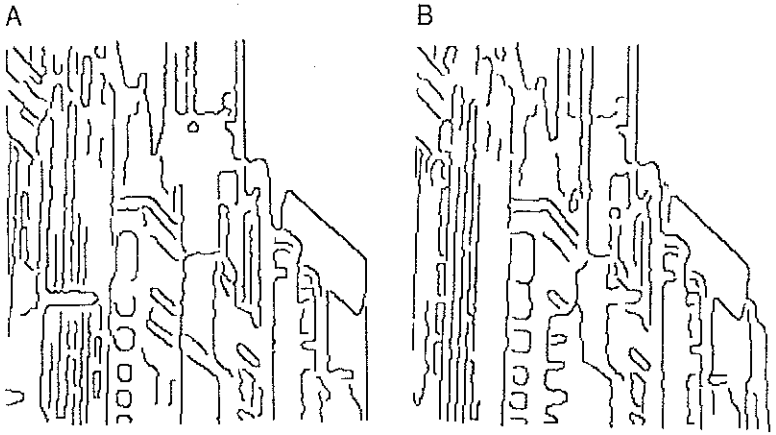

Figure 9.18. Original channel 1: (A) right; (B) left.

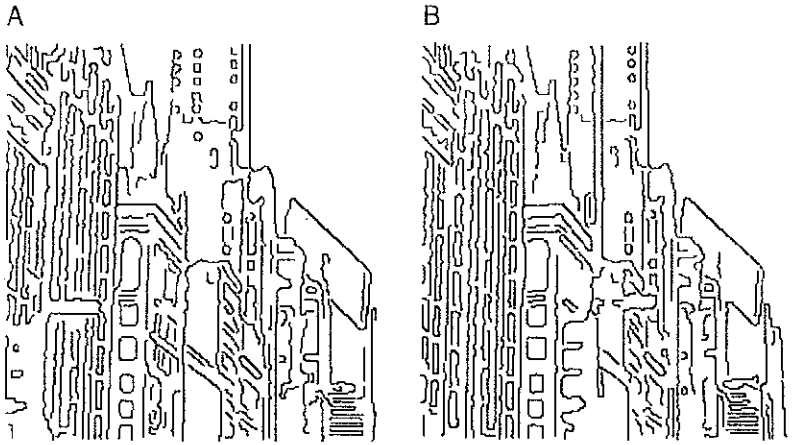

Figure 9.19. Original channel 2: (A) right; (B) left.

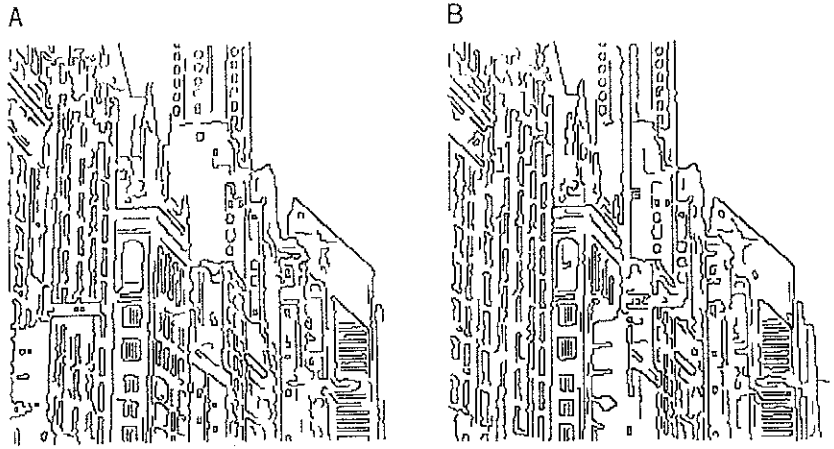

Figure 9.20. Original channel 3: (A) right; (B) left.

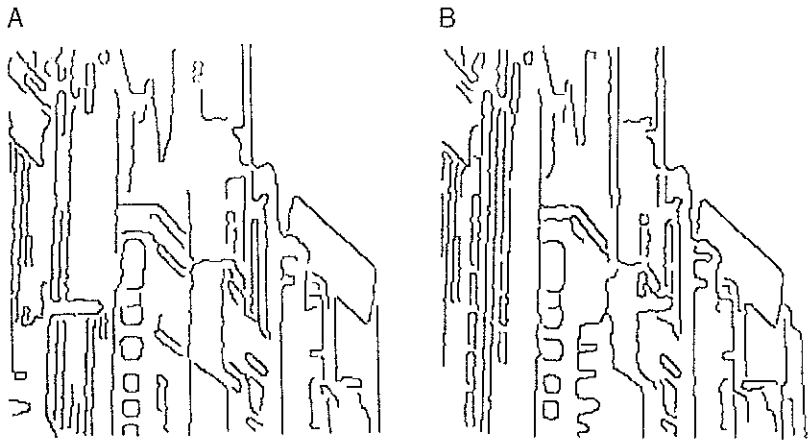

Figure 9.21. New channel 1: (A) right; (B) left.

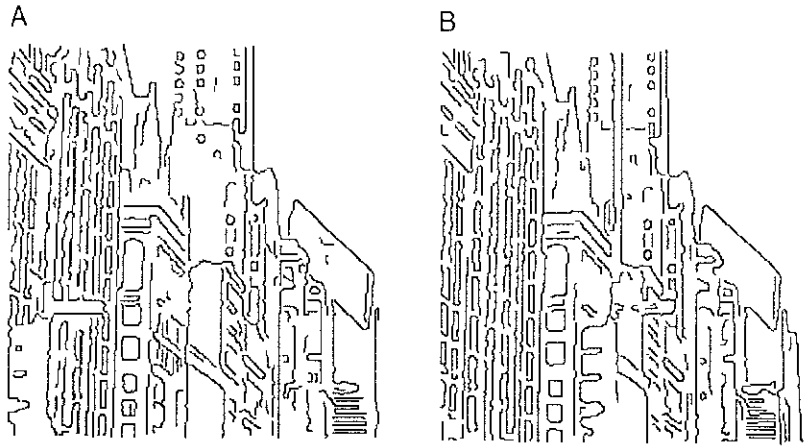

Figure 9.22. New channel 2: (A) right; (B) left.

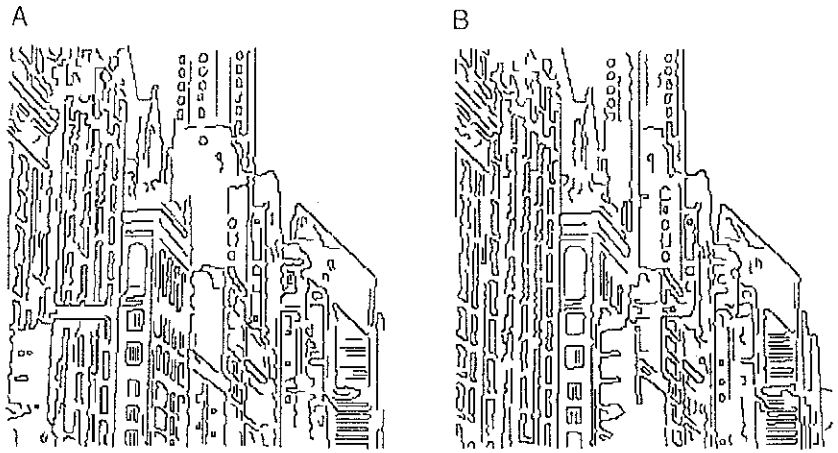

Figure 9.23. New channel 3: (A) right; (B) left.

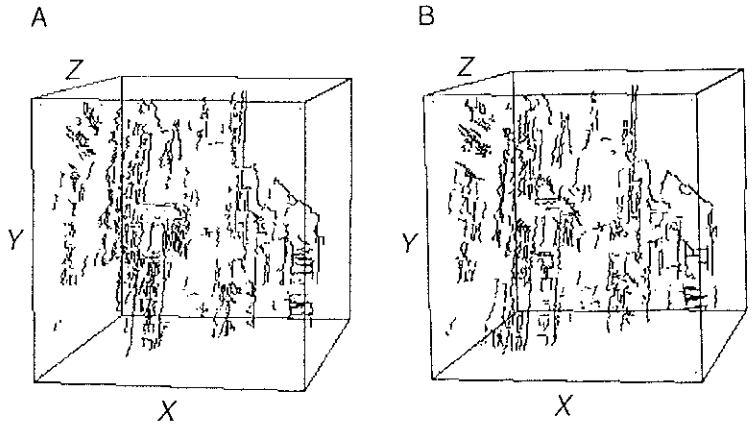

Figure 9.24. Scatter plot of points, 3D comparison: (A) original; (B) new.

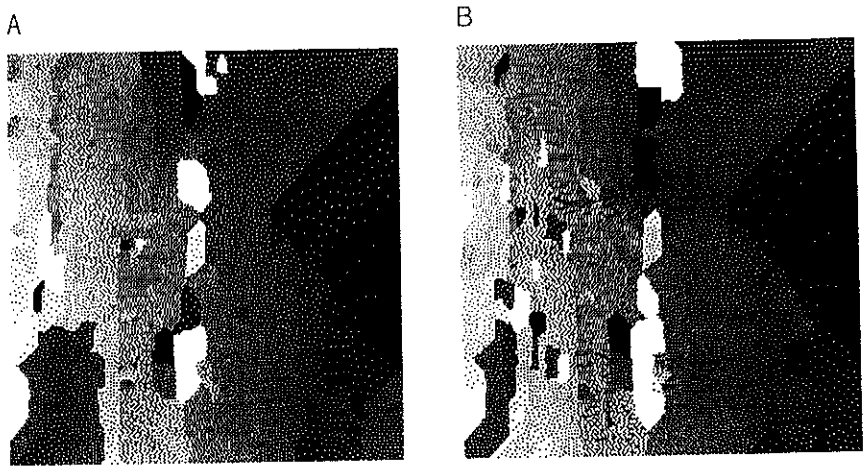

Figure 9.25. Three-dimensional interpolation comparison: (A) original; (B) new.

6. CONCLUSION AND FUTURE WORK

As can be seen from the case study, the addition of a learning component to an existing vision system can have a significant impact on performance. The approach to take is one of pragmatism with an eye toward incremental improvement. The key decisions that the designer must make

in this task are:

1. Analyze the current structure of the vision algorithm.
2. Select the areas where a learning component will perform best.
3. Express the learning objective and constraints formally.
4. Choose the learning mechanisms that will act on the algorithm.

For the first step, the designer should "dissect" the system and view it as a set of logical modules. To accomplish the second step, it is best to focus on tasks that are tedious or areas that have little theory to guide decisions. The third step can be straightforward, but any oversight in specifying the objective will certainly cause the machine to find an algorithm that satisfies the objective in a trivial way. Here it is best to express everything of importance in the objective that must be optimized and constraints that must be satisfied. If there is a tradeoff that the machine should not be allowed to make on its own, then penalty components might be included in the objective or constraints to guide it. The last step is the most difficult. For many systems, an inductive learning approach will serve best. Also, it can be helpful at this stage to review how a human expert might address the problem. This can help significantly in designing ways to perturb the vision algorithm. In any case there will be a variety of options available, and it is best to focus on what is feasible before attempting a task more complex than redesigning the vision algorithm itself.

One limitation of our case study is that we did not consider objectives other than that defined in Eq. (3). We have found that anomalies in cost and quality of the algorithm learned may happen when learning is carried out under a single objective function defined as a mathematical function of subobjectives (15). An alternative way to learn is to define constraints (or requirements) on the cost and quality of the algorithm to be learned, and direct the learning system to find good feasible algorithms satisfying the constraints. We plan to carry out this approach in our future study.

Another restriction of our case study is that we did not consider domain knowledge already known about the performance of the vision algorithm with respect to changes in these parameter values. In the future, we plan to incorporate existing error analysis methods on various aspects of binocular stereo vision (7, 10, 17) in our objective function.

Finally, our stereo vision algorithm is based on a paradigm that recovers dense-range maps via interpolation of sparse feature maps. There exist other stereo vision algorithms in the literature. We plan to study in the future the applicability of our learning system for these other algorithms.

ACKNOWLEDGMENT

Research was supported by National Aeronautics and Space Administration Contract NCC 2-481 and National Science Foundation Grant MIP 92-18715.

REFERENCES

1. J. Canny, A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-8**, 679–698 (1986).
2. W. E. L. Grimson, "From Images to Surfaces: A Computational Study of the Human Early Visual System." MIT Press, Cambridge, MA, 1981.
3. B. K. P. Horn, "Robot Vision." MIT Press, Cambridge, MA, 1986.
4. A. Ieumwananonthachai, A. Aizawa, S. R. Schwartz, B. W. Wah, and J. C. Yan, Intelligent process mapping through systematic improvement of heuristics. *J. Parallel Distributed Comput.* **15**, 118–142 (1992).
5. D. Marr, "Vision." Freeman, New York, 1982.
6. D. Marr and T. Poggio, "A Theory of Human Stereo Vision," AI Memo 451. Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, 1977.
7. D. Marr and T. Poggio, A computational theory of human stereo vision. *Proc. R. Soc. London, Ser.*, **204**, 301–328 (1979).
8. J. L. McClelland, D. D. Rumelhart, and G. Hinton, The appeal of parallel distributed processing, *In* "Parallel Distributed Processing: Explorations in the Microstructure of Cognition" (D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, eds.), No. 1. MIT Press, Cambridge, MA, 1986, pp. 3–44.
9. R. S. Michalski, Understanding the nature of learning: Issues and research directions. *In* "Machine Learning: An Artificial Intelligence Approach" (R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, eds.), No. 2. Morgan Kaufmann, Los Altos, CA, 1986, pp. 3–25.
10. H. K. Nishihara, Practical real-time imaging stereo matcher. *Opt. Eng.* 23, 536–545 (1984).
11. L. Rendell, R. Seshu, and D. Tcheng, More robust concept learning using dynamically-variable bias. "Machine Learning." Boston, 198, pp. 66–78..
12. S. R. Schwartz, "Resource Constrained Parameter Tuning Applied to Stereo Vision." M.Sc. Thesis, University of Illinois, Department of Electrical and Computer Engineering, Urbana, IL, 1991.
13. H. A. Simon, Why should machines learn? *In* "Machine Learning: An Artificial Intelligence Approach" (R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, eds.), pp. 25–37. Morgan Kaufmann, Los Altos, CA, 1983.
14. J. Sklansky, Bottlenecks to effective application of machine vision. *In* "Machine Vision: Algorithms, Architectures, and Systems" (H. Freeman, ed.), pp. 187–192. Academic Press, San Diego, CA, 1988.
15. B. W. Wah, Population-based learning: A new method for learning from examples under resource constraints. *Trans. Knowl. Data Eng.* **4**, pp. 454–474 (1992).
16. J. Weng, A theory of image matching. *Proc. Int. Conf. Comput. Vision, 3rd*, Osaka, Japan, 1990, pp. 200–209, IEEE.
17. R. P. Wildes, Direct recovery of three-dimensional scene geometry from binocular stereo disparity. *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-13**, 761–774 (1991).
18. A. Witkin, D. Terzopoulis, and M. Kass, Signal matching through scale space. *Proc. Natl. Conf. Artif. Intell., 5th*, Philadelphia, 1986, pp. 714–719.