AI Magazine Volume 9 Number 2 (1988) (© AAAI)

Evidence Accumulation & Flow of Control in a Hierarchical Spatial Reasoning System

K. M. Andress and A. C. Kak

A fundamental goal of computer vision is the development of systems capable of carrying out scene interpretation while taking into account all the available knowledge In this article, we focus on how the interpretation task can be aided by the expected scene information (such as map knowledge), which, in most cases, would not be in registration with the perceived scene The proposed approach is applicable to the interpretation of scenes with three-dimensional structures as long as it is possible to generate the equivalent two-dimensional orthogonal or perspective projections of the structures in the expected scene The system is implemented as a two-panel, six-level blackboard and uses the Dempster-Shafer formalism to accomplish inexact reasoning in a hierarchical space Inexact reasoning involves exploiting, at different levels of abstraction, any internal geometric consistencies in the data and between the data and the expected scene As they are discovered, these consistencies are used to update the system's belief in associating a data element with a particular entity from the expected scene

A fundamental goal of computer vision is the development of systems that are capable of carrying out scene interpretations with the aid of all available knowledge. To elaborate, suppose a helicopter-based computer vision system is looking at a snow-covered terrain; this terrain knowledge must then be explicitly taken into account in a target recognition procedure. Clearly, the processing required for a snow-covered background is different from that for, say, a wooded area in spring.

As a simpler example of knowledge-based processing, consider the problem of self-location for a vehicle-mounted vision system (Kak et al. 1987). Let's say the vehicle's whereabouts are approximately known from the position encoders mounted on the wheels, the precision of this information limited by the extent of slippage in the wheels, and so on. Given this approximate information, is it possible to make a more precise fix on the location of the vehicle by integrating the vision data with the map knowledge while the two are out of registration? This problem of robot self-location was the original goal of the Production System Environment for Integrating Knowledge with Images (PSEIKI), a system currently under development in the Robot Vision Lab at Purdue University. We felt this simple exercise in knowledge-based processing would give us the expertise to set up more complex reasoning structures for incorporating other kinds of knowledge sources in an image interpretation task. (To digress briefly, the reader interested in robot self-location might want to know that in contrast with the quantitative approach typified by PSEIKI, it is also possible to design qualitative reasoning systems for navigation and self-location using visual landmarks to represent environmental location [Kuipers 1977; Levitt et al. 1987].)

As it now stands, PSEIKI, a production system in OPS83, requires that the world knowledge be presented to it as a line drawing of the expected scene. For applications such as automatic target recognition, the line drawing can include environmental effects, such as the attenuation of infrared energy through the atmosphere. The PSEIKI system is implemented as a two-panel, six-level blackboard and uses the Dempster-Shafer (D-S) formalism to accomplish inexact reasoning in a hierarchical space. Our use of the D-S theory should not be construed to imply the inappropriateness of the other available formalisms. Using the Bayes-net methodology developed by Pearl (1986), one could also employ a Bayesian formalism, as Binford, Levitt, and Mann (1987) do in the SUCCESSOR system for model-based machine vision

Although PSEIKI was originally developed for carrying out knowledge-based experiments in robot self-location, the current implementation is general enough to be used in any application where a good estimate of the expected scene is available to the vision system. The system can be used as a general vision verification module either in a robotics context or for automatic target recognition. PSEIKI contains two features that keep it domain independent.[1] First, the knowledge used by PSEIKI consists of a line drawing of the expected scene (which in most applications would not be in registration with the observed image). For example, for robot navigation applications, line

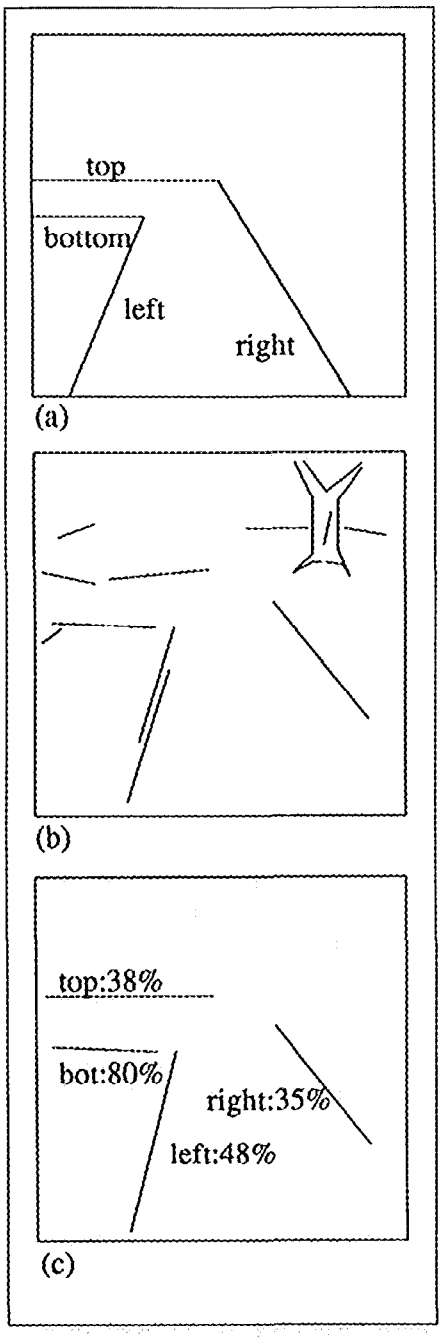


Figure 1. Typical Images Used by PSEIKI

(a) An example of a line drawing of an expected scene with edges labeled; (b) A simple example of the output of an edge-based preprocessor that PSEIKI could use as input data (c) The final output of PSEIKI, with labeled edges and associated belief values.

drawings can easily be generated from road maps. For verification vision, a line drawing of the object whose identity, location, and orientation need to be verified can be generated from a three-dimensional model of the object. In industrial two-dimensional vision applications, computer graphics or computer-aided design systems can be used directly to generate the line drawings. The other feature that provides the system domain independence concerns how the system presents its results. The output of PSEIKI consists of a mapping from elements detected in the input image to elements in the expected scene.[2]

The mapping generated by PSEIKI is expressed by labeling the detected edges with the names of the corresponding lines in the expected scene; a belief value is also attached to each label, indicating the confidence of the mapping found. Furthermore, a belief value is estimated for the entire mapping process. If this overall belief value does not exceed a threshold, the entire mapping is rejected. To illustrate what PSEIKI does, refer to figure 1. If figure 1a is a line drawing of an expected scene and figure 1b a depiction of the edges that might be found in the vision data collected for the scene, then PSEIKI produces an output similar to that in figure 1c, where labels attached to some of the edges found in figure 1b and the corresponding belief values are shown For example, the label right:35% means that PSEIKI has found the expected scene edge labeled right in figure 1a to be compatible with the lower right edge in figure 1b with a belief of 35 percent. In this case, the rest of the belief, 65 percent, would be apportioned to either this particular label being incorrect or the system professing ignorance on the subject of assigning the label right to this edge in the vision data. The reader might note that the edge labeled top:38% actually corresponds to two edge segments in figure 1b. This merger of nearly compatible edges in the vision data is one consequence of the various tests PSEIKI makes for internal geometric consistencies in the vision data. Because PSEIKI only generates a mapping from the edges in the input image to the expected scene, it is left to a higher-level system to make global interpretations based on the mapping found

PSEIKI is also a test bed for carrying out experiments in how inexact reasoning can be achieved on hierarchical representations of scenes. Gordon and Shortliffe (1985) discuss the problem of diagnostic reasoning in medicine and present a technique that allows the D-S formalism (Shafer 1976) to be used in a system which groups hypotheses into strict hierarchies. Shafer and Logan (1987) also discuss the problem of combining hierarchical evidence; they look at the problem in a more formal manner and are able to do without some of the approximations used in Gordon and Shortliffe (1985) We can not use the methods presented in Gordon and Shortliffe (1985) and Shafer and Logan (1987) directly because PSEIKI does not employ strict hierarchies (an edge can be a member of two faces if it is a part of the border between them). In the current implementation of PSEIKI, the blackboard architecture is exploited to permit exact and inexact reasoning in a tangled hierarchy The D-S formalism is used for pooling uncertain evidence in the hierarchy.

PSEIKI is able to handle significant perspective effects Many previous systems, again most notably aerial interpretation systems, were able to assume that the images were obtained by an orthographic imaging system. Although perspective distortions make image interpretation difficult because metric properties, such as length and orientation, depend on the object's position in the image, they also provide clues to the structure of objects in the image.

A brief report on PSEIKI was presented in Andress and Kak (1987). This article is more of a tutorial and discusses in greater detail the evidence accumulation mechanisms and the control structures in PSEIKI.

Related Image-Understanding Systems

ACRONYM by Brooks (1981) is a model-based image-understanding system. The system's task consists of finding instances of known objects in the image. To perform object identification, the system first builds a pic-

ture graph of the image and an observability graph that specifies information about objects which could be in the image. The system identifies instances of objects in the image by matching nodes of the observability graph with sets of nodes in the picture graph. The objects in the observability graph are represented in slot, filler structures where any slot that can accept numeric values can also accept algebraic constraints expressed as inequalities. The system can then manipulate these constraints and determine if properties of objects detected in the image meet these constraints. The objects used to generate the observability graph are represented as generalized cones. Inexact reasoning is not used, and the system utilizes backward chaining to arrive at an interpretation.

Davis and Hwang (1985) describe the SIGMA image-understanding system for aerial image interpretation. The system uses both forward and backward chaining to arrive at an interpretation and represents its object classes hierarchically using frames Furthermore, the system is able to integrate hypotheses about specific objects in the scene. The system does not use uncertain reasoning but instead is able to control its focus of attention based on the strength of a situation.

Another aerial interpretation system is described by Nagao and Matsuyama (1980); the system is based on the blackboard architecture and uses multispectral images in the interpretation process. To accomplish the interpretation task, the system first performs a global survey of the entire image and labels regions without using any domain-specific knowledge. The characteristic regions that it finds, such as water, vegetation, and roads, are then used to generate context information for further processing. This processing consists of a detailed analysis of local areas in the scene using context information provided by the characteristic regions and applying context-specific object-detection subsystems.

SPAM, a system designed by McKeown, Harvey, and McDermott (1985), is also an aerial image interpretation system. The system was originally

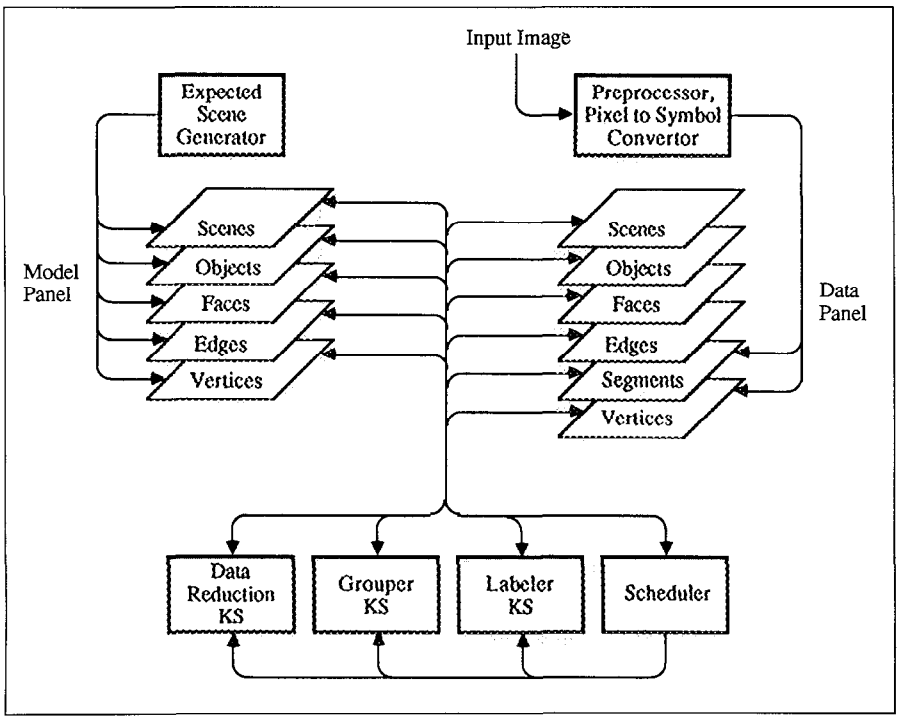


Figure 2 PSEIKI's Architecture

constructed to interpret airport scenes but has been expanded with a rule generator; so, it can now interpret scenes from other domains. SPAM uses confidence values to aid labeling and can manipulate these values based on the consistency of the various labelings.

VISIONS, first reported on by Hanson and Riseman (1978), is a blackboard expert system designed to analyze color images. The system uses a flexible control scheme, hierarchical scene representation, and a number of knowledge sources to accomplish the scene interpretation. VISIONS is domain independent but uses schemas to tune the system for a particular application.

The image segmentation expert system developed by Nazif and Levine (1984) contains two global memories. The global long-term memory contains rules that are applied to the data stored in its short-term memory. The system is rule based and uses modules to update lines, regions, and areas in the image. The expert system also contains a set of metarules and can control its focus of attention.

Barnard (1983) describes a system that deals with perspective images. The system is able to use the Gaussian sphere to determine the vanishing points of the scene being analyzed. The back projection of angles and curvatures is also used to aid the interpretation task.

Barrow and Tenenbaum (1981) discuss the problem of interpreting line drawings. They are able to use junction libraries and knowledge of differential geometry to discriminate between different kinds of boundaries. This knowledge is then used to determine how the surfaces should be constrained.

PSEIKI differs from these systems in three main areas. First, PSEIKI's task differs from those of previous systems. Most of the other systems were designed to find object instances in the image and, through such discoveries, to arrive at a global interpretation of the image. PSEIKI's task is limited to integrating expected scene information with the observed image, the result being a set of consistent labels, with associated belief values, for the edge elements in the image.

PSEIKI differs from SPAM and SIGMA and, to a certain extent, VISIONS in that it does not rely on domain-dependent information. For example, SPAM uses airport design knowledge when interpreting airport

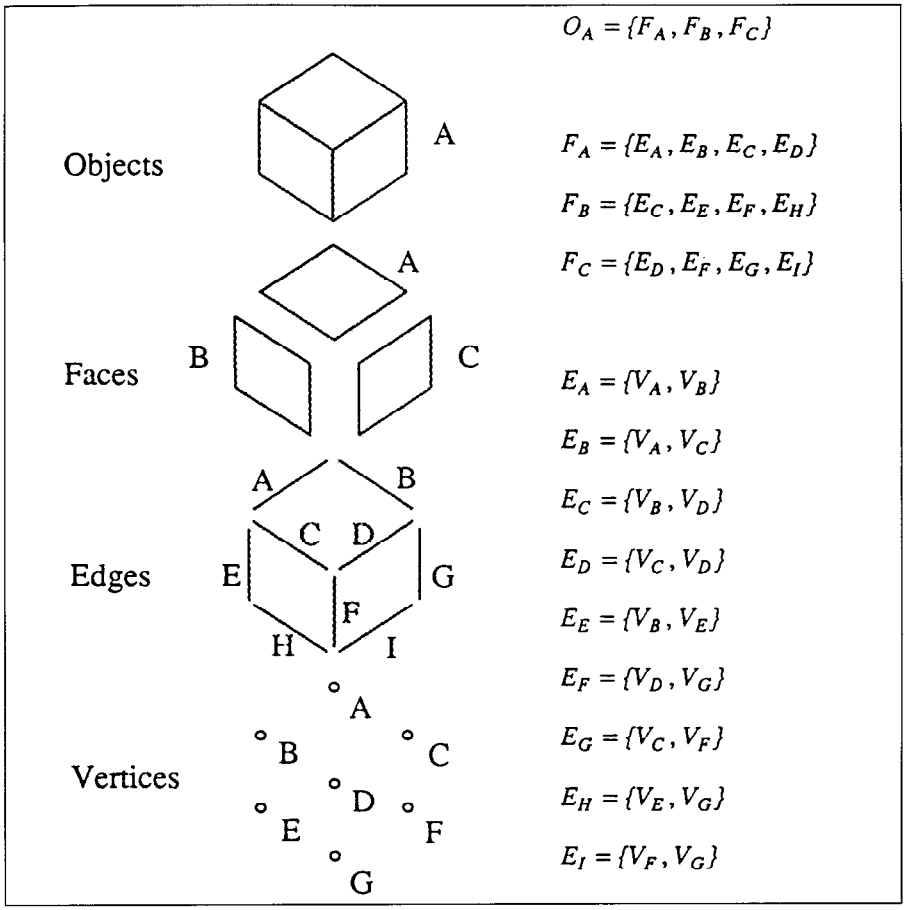


Figure 3 Example of Information in Different Levels of the Model Panel for a Simple Object

scenes. Context cues have also been used extensively in past computer vision systems. For example, if SIGMA detects a driveway in an image, it searches for a house and for roads connected to the driveway. Because PSEIKI is provided with a good estimate of the expected scene, it does not have to perform inferences of this type. Although it might be said that context cues are indispensable for scene interpretation because they make deductions more powerful, their use necessarily introduces some domain dependence. Therefore, it is our philosophy to separate the generation of the mapping from the formation of an overall scene interpretation. If context cues are desired by a system using PSEIKI, then it is up to the higher-level system to provide PSEIKI with a line drawing incorporating the information contained in the cues.

PSEIKI also differs from previous systems in its method of performing inexact reasoning. Many systems, including ACRONYM, SIGMA, and the system by Nazif and Levine, use no uncertain reasoning in the image interpretation process. Because of the overwhelming amount of data in an image, most of the inexact reasoning schemes used in the past have been fairly simple to avoid becoming bogged down in certainty value computations. However, inexact reasoning in PSEIKI is based on the D-S formalism in a tangled hierarchical space. The use of a hierarchy curtails the number of uncertainty calculations and is made possible by the blackboard architecture.

Overview of PSEIKI

PSEIKI's architecture is shown in figure 2. The system has been implemented in OPS83 as a two-panel, six-level blackboard. One panel is reserved for the expected scene; we frequently refer to it as the model panel. The other panel initially contains data derived from the image and eventually the abstraction hierarchies derived from these data; this panel is the data panel.[3]

The input to PSEIKI is provided by an image preprocessor, also shown in figure 2. The preprocessor converts the pixel data in the input image to a symbolic form that is deposited on the first two levels of the blackboard data panel. In our current implementation, we use an edge-based preprocessor, the output of which is a collection of piecewise linear segments detected in the image. Currently, the information that is deposited on the model panel does not change during the processing of a given image frame. However, our hope is that at some future time this model information will be made dynamic, allowing the system to automatically invoke a different expected scene if the current scene cannot be matched with the data at a sufficiently high degree of overall belief.

Each blackboard panel contains the following levels to represent the images: scenes, objects, faces, edges, segments, and vertices. Each element in a level is defined by a finite collection of elements on lower levels. For example, a scene is made of a union of objects, and a face is defined by the group of edges that form its borders. Figure 3 shows the data on the model panel for a trivial expected scene, a simple block. It shows each element's label and the subelements from which it is composed (note that the segment level does not exist for the model panel).

On level 6 are stored the *scenes*. The entire scene (expected or observed) is represented on this level. The scene is defined as the union of all objects in level 5 of the hierarchy. It provides a way of labeling multiple objects that otherwise would not be possible. On level 5 are the *objects*. Each element on this level corresponds to a distinct physical object. The objects are defined as the union of all boundary faces from level 4. Level 4 contains the *faces*. The elements on this level represent the polygonal faces that form a boundary representation of the observable portion of the objects. A face is defined by the edges from level 3 that form its border.

On level 3 are the *edges*. These ele-

ments form the boundaries of the faces in level 4 of the hierarchy. This level is included to provide a way to compensate for segmentation deficiencies. Highly collinear segments from level 2 are grouped to produce an edge in this level. Level 2 stores the *segments*. The piecewise linear segments produced by the image preprocessor are represented on this level. Note that this level is not necessary on the model panel and is not provided. Finally, on level 1 are the *vertices*. The vertices are the end points of the segments and edges from the next two higher levels. On the data panel, the vertices are provided by the low-level vision system, the preprocessor

PSEIKI has three main knowledge sources (KSs) that it uses to establish a mapping from the observed to the expected scene: Data-Reduction, Labeler, and Grouper (see figure 2). The data-reduction KS cleans up the segments deposited by the image preprocessor on the lowest level of the blackboard data panel. The grouper KS determines which data elements in the lower levels of the hierarchy should be grouped to form a data element on a higher level The third activity, the labeling of data elements, is done by the labeler KS.

In the next section, we focus on the evidence accumulation mechanisms in the labeler and the grouper KSs. This information is followed by a brief discussion of some of the data structures used; we believe the reader needs to appreciate the structure of the knowledge source activation record (KSAR) to understand the flow of control achieved by the scheduler and the monitor, both discussed in Scheduler and Monitor for Blackboard Control.

We do not discuss the image preprocessor that converts pixels into binary edges, represented as piecewise linear segments; for preprocessing steps, see Andress and Kak (1988). We also do not discuss the methods used to generate of the expected scene that resides on the model panel. The key to handling perspective effects in PSEIKI lies in generating the perspective projections of world scenes; methods used for this process are also discussed in Andress and Kak (1988).

Evidence Accumulation Mechanisms in KSs

As mentioned previously, there are three knowledge sources in PSEIKI: Data-Reduction, Grouper, and Labeler The data-reduction KS is not discussed here in any detail except to mention that its purpose is to carry out chores, such as the elimination of small dangling edges and small edges generated mostly by noise. In short, the data-reduction KS helps us reduce the complexity of the problem by retaining only those edges which are either strong or long.

We now detail the other KSs, which accumulate evidences for various possible labels for the elements on the data side of the blackboard. We show how the labeler KS updates the belief value of a particular label for a data element if it finds the data element to be geometrically compatible with the neighboring data elements. Of course, before such belief revision can take place, the labeler KS must also compute its initial belief in associating a particular label with a data element. Techniques for transforming feature measurements in vision data into belief values are still in their infancy; the reader is referred to Reynolds et al. (1986) for some earlier work on this subject.

Labeler KS

This KS performs element labeling and confidence estimation for different possible hypotheses using the D-S formalism. The combinatorial explosion of uncertainty calculations usually associated with the D-S scheme is avoided with the use of a hierarchical reasoning space

The hierarchical structure of the blackboard data provides a natural basis for a hierarchical reasoning space. The levels of this space correspond naturally with the levels of the data elements on the blackboard. For example, assume that in figure 3 F_A on the model panel is composed of edges $\{E_A, E_B, E_C, E_D\}$.[4] Also, assume that edge E_1 on the data panel is part of the group which constitutes face F_1. If F_1 is labeled F_A, then E_1 can be labeled only as one of $\{E_A,..., E_D\}$. If the data were not arranged hierarchically, it

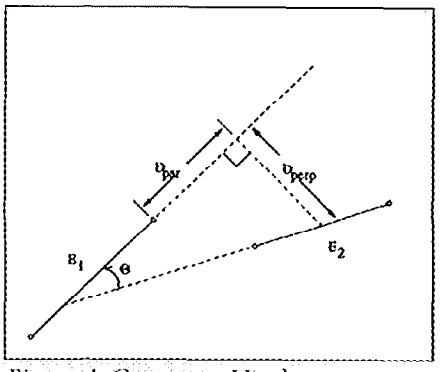


Figure 4 Geometry Used in the Definition of Collinearity.

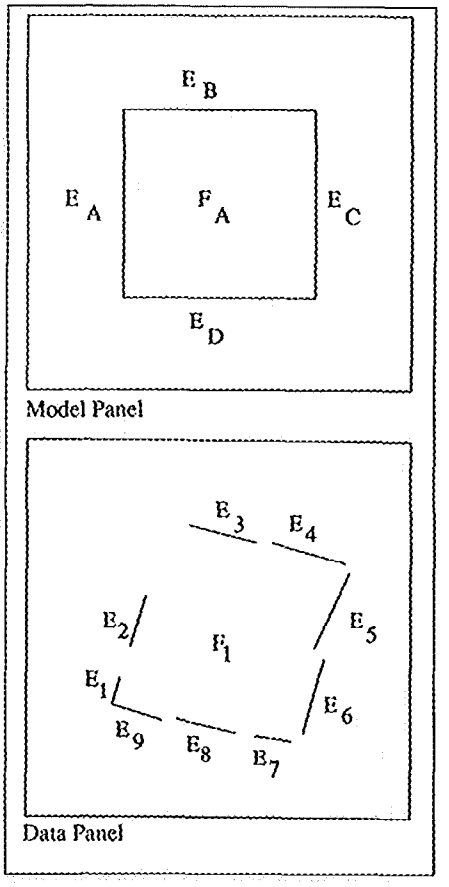


Figure 5. Information on the Model and Data Panels at the Edge Level of the Blackboard

would be necessary to consider every element on the model panel when assigning labels and performing consistency checks. To curtail the number of uncertainty calculations, consistency checks are not made directly between two elements at the same level of the hierarchy if they do not have a common parent. In the previous example, edge E_1 would only be checked for consistency with edges that are children of face F_1 Consistency checks between two nonsiblings can be made indirectly by propagating an element's confidence value up through the hierarchy until a common ancestor is reached and then back down to the second element.

The frame of discernment (FOD) for any element is defined by the labels that can be given to the element.[5] In our example, because F_A is under consideration as a label for the face F_1 the FOD for edge E_1 is

$$\Theta = \{E_A, E_B, E_C, E_D\}.$$

A data element's *label* is defined as the element from its FOD that has the greatest belief value attached to it. If the belief value of an element on an upper level of the hierarchy is changed, then all of its descendants must change their FOD. Thus, it is advantageous to first perform compatibility checks between elements on upper levels of the hierarchy to avoid performing unnecessary calculations on lower levels when FODs are changed. This necessity for checking global consistency before local consistency seems reasonable and is a part of the control strategy discussed in Scheduler and Monitor for Blackboard Control.

Two metrics are required when updating the label belief values at any level of the hierarchy, one for measuring compatibility and the other for measuring incompatibility between two given elements on the data panel. When initially assigning belief values to data elements, these two metrics must also measure compatibility between a data element and a model element. To facilitate the correspondence between them and certainty values, both metrics should range between 0.0 and 1.0. Obviously, the metrics need not be the same for all levels of the hierarchy.

To show what these metrics look like, consider the following: At the segment and edge levels of the blackboard when two data segments (or edges), E_1 and E_2, are thought of as belonging to the same model element, the compatibility between the two is measured by the following collinearity measure:

$$\text{collinearity}(E_1, E_2) = (D_{max} - D_{perp}) / D_{max} \times \cos(\theta)$$

where θ is the acute angle between the two edges, and D_{perp} is the distance from the middle of E_2 to the line defining E_1 (see figure 4). D_{max}, the maximum allowable value for D_{perp}, is a user-specified heuristic parameter or function. In our current implementation, D_{max} is set equal to the length of E_1; the justification is that for two data elements to belong together in some sense, the maximum allowable distance between them should not be measured in an absolute manner but relative to the sizes of the data elements

Likewise, the incompatibility between the same two edges, E_1 and E_2, can be measured by calculating the noncollinearity (E_1, E_2).

$$\text{noncollinearity}(E_1, E_2) = D_{perp} / D_{max} \times \text{scale}(E_1) \times \sin(\theta)$$

where scale(E_1) depends on the length of E_1 [6]

Because the (in)compatibility measures are defined heuristically, it is usually advantageous to limit the amount of evidence that they can provide. This limit can be achieved by scaling the measures by a level-specific scale factor SF, $0 \leq SF \leq 1$. Thus, the (in)compatibility measures for the segment and the edge levels can be defined as

$$\text{compatibility}(E_i, E_j) = \text{collinearity}(E_i, E_j) \times SF_{edge} \quad (1)$$

and

$$\text{incompatibility}(E_i, E_j) = \text{noncollinearity}(E_i, E_j) \times SF_{edge} \quad (2)$$

Note that in computing both these measures, the elements E_i and E_j on the data panel must correspond to the same element on the model panel, unless, of course, the system is computing the initial belief values by comparing the data and the model information; in this case, the argument E_i can refer to a model element and E_j to a data element The computation of initial belief values is discussed in greater detail in the following subsection.

The compatibility and incompatibility metrics must, of necessity, be different at the various levels of the blackboard. At the face level, for example, a metric that is used to compute the incompatibility between two faces on the data panel measures the overlap between them normalized by the average area of the two faces. To understand how this metric is used, consider the following example· Suppose for two faces F_1 and F_2 on the data panel, the normalized overlap, as measured by the function overlap (F_i, F_j), returns 10 percent. If the maximum-belief model labels for F_1 and F_2 are F_A and F_B, respectively, and if it is known that there is no overlap between F_A and F_B, then the incompatibility between F_1 and F_2 can be measured by incompatibility $(F_1, F_2) = \text{overlap}(F_1, F_2) = 0.1$

Going back to the segment and edge levels of the blackboard, if two segments (or edges) on the data panel correspond to different model elements, a rigid motion transform is applied to one of them before the computation of the (in)compatibility metrics, which has the effect of enforcing relational constraints between the two data elements. For example, if edges E_1 and E_3 are thought to correspond to model edges E_A and E_B respectively, then the measure of compatibility between E_1 and E_3 is defined as

$$\text{compatibility}(E_1, E_3) = \text{collinearity}(E_1, T_{E_B \rightarrow E_A}(E_3)),$$

where $T_{E_B \rightarrow E_A}$ is the rigid motion transformation that makes model edge E_B collinear with model edge E_A in the following sense: First, for a given pair of nonparallel edges, we distinguish between the vertices on the convergent and the divergent sides; by *convergent side*, we mean the side on which the edges would meet if extended. For the transformation $T_{E_B \rightarrow E_A}$, the edge E_B is rotated about its convergent vertex through an angle which makes the edges parallel; subsequently, E_B is translated so that the two convergent vertices are coincident. For further details on this transformation, see Andress and Kak (1988). Performing this transformation forces model elements to be com-

patible; in other words,

collinearity$(E_A, T_{E_B \to E_A}(E_B)) = 1.0$

Computing Initial Belief Values for Elements in a Group. To fully describe how we compute the initial belief values for, say, the edge elements at the edge level on the data panel in figure 2, consider the following example. Let's say that at this level, the information in the model panel and the data panel is as shown in figure 5.

The grouper KS hypothesizes that the edges $\{E_1, \ldots, E_9\}$ be grouped together; let this grouping be designated by the face F_1. In order to label the edges in the data, the labeler KS constructs the following FOD for each edge in the group: $\Theta = \{E_A, E_B, E_C, E_D\}$.

Let's now focus on the labeling process for edge E_1. The labeler KS computes a basic probability assignment (bpa) (see appendix) over Θ by applying a collinearitylike measure to the pairs (E_A, E_1), (E_B, E_1), (E_C, E_1), and (E_D, E_1). The collinearity measure used here differs from the collinearity metric of the preceding subsection in that it also constrains the maximum permissible misregistration between a data edge and a model edge. The following formula shows the form of the measure when applied to the data edge E_1 and the model edge E_A

$$\text{ES\_compatibility}(E_A, E_1) = \frac{D_{max} - D_{perp}}{D_{max}} \times \frac{D_{max} - D_{par}}{D_{max}} \times \cos\Theta$$

with the same geometric visualization as in figure 4, D_{perp} is the distance from the middle of E_1 to E_A, D_{par} the misregistration along the direction E_A, D_{max} the maximum allowable value for both misregistrations, and θ the acute angle between the segments. The prefix ES in the name of the measure stands for expected scene.

For the computation of initial belief values, D_{max} is set differently from the manner described earlier; its value is set by the user and reflects the maximum expected mis-registration between the data and the expected scene. Note that although E_A and E_1 exist in different frames, it is possible to speak of distances and angles between them because for the purposes of blackboard processing, they are both projected into the same world coordinate system. For a mobile robot navigating on a flat ground plane, all the edges are transformed onto the ground plane.

Let's assume, for the sake of discussion, the compatibility calculation produces the following results for E_1:

ES_compatibility$(E_A, E_1) = 0.7$
ES_compatibility$(E_B, E_1) = 0.1$
ES_compatibility$(E_C, E_1) = 0.4$
ES_compatibility$(E_D, E_1) = 0.05$

Note that although E_1 is approximately parallel to both E_A and E_C, the ES_compatibility calculation yields a larger result for E_A because of the distance dependence of the calculation. If the sum of the ES_compatibility measures exceeds unity, as is the case in our example, the measures are normalized by the summed value. Therefore, in our example,

ES_compatibility$(E_A, E_1) = 0.56$
ES_compatibility$(E_B, E_1) = 0.08$
ES_compatibility$(E_C, E_1) = 0.32$
ES_compatibility$(E_D, E_1) = 0.04$

These ES_compatibility measures define a bpa over Θ for our example, as given by

$m_{E_1}(\{E_A\}) = 0.56$
$m_{E_1}(\{E_B\}) = 0.08$
$m_{E_1}(\{E_C\}) = 0.32$
$m_{E_1}(\{E_D\}) = 0.04$
$m_{E_1}(\ ) = 0.0$ for all other subsets of Θ

A slightly different approach is taken when the sum of ES_compatibility measures is less than 1. Assume for a moment that because of the distance cutoff, D_{max}, the ES_compatibility measure of (E_C, E_1) is zero; that is,

ES_compatibility$(E_A, E_1) = 0.7$
ES_compatibility$(E_B, E_1) = 0.1$
ES_compatibility$(E_C, E_1) = 0.0$
ES_compatibility$(E_D, E_1) = 0.05$

Because the measures now sum to less than unity, there is no reason to normalize. Instead, we now convert them directly into bpa's in the following manner:

$m_{E_1}(\{E_A\}) = 0.7$
$m_{E_1}(\{E_B\}) = 0.1$
$m_{E_1}(\{E_C\}) = 0.0$
$m_{E_1}(\{E_D\}) = 0.05$
$m_{E_1}(\Theta) = 0.15$
$m_{E_1}(\bullet) = 0.0$ for all other subsets of Θ

Note that we have now assigned 0.15 belief to Θ; 0.15 is what remains after we subtract the sum of ES_compatibility measures from 1. Assigning a part of our belief to Θ seems intuitively plausible for the simple reason that ES_compatibility is a good measure of "E_1 is a part of E_A." Clearly, if E_1 does not match with E_A, E_B, E_C, or E_D to a sufficiently high degree, then we can leave some belief uncommitted. In this assignment, $m_{E_1}(\Theta) = 0.15$ represents the uncommitted portion of our belief.

Let's say that the procedure just described yields the following bpa's for all the data edges.

$m_{E_1}(\{E_A\}) = 0.7$	$m_{E_2}(\{E_A\}) = 0.8$
$m_{E_1}(\{E_B\}) = 0.1$	$m_{E_2}(\{E_B\}) = 0.1$
$m_{E_1}(\{E_c\}) = 0.0$	$m_{E_2}(\{E_C\}) = 0.0$
$m_{E_1}(\{E_D\}) = 0.05$	$m_{E_2}(\{E_D\}) = 0.00$
$m_{E_1}(\Theta) = 0.15$	$m_{E_2}(\Theta) = 0.1$

(3)

$m_{E_9}(\{E_A\}) = 0.2$
$m_{E_9}(\{E_B\}) = 0.1$
$m_{E_9}(\{E_c\}) = 0.1$
$m_{E_9}(\{E_D\}) = 0.60$
$m_{E_9}(\Theta) = 0.0$

These bpa's then constitute the initial bpa's for the data edges.

Revising Belief by Enforcing Mutual Consistency Constraints. From the set of initial bpa's, the labeler KS then seeks those observed edges which have maximum bpa for the same model edge. To illustrate, in the example here, E_1 and E_2 exhibit maximal beliefs for the same model edge, E_A.

The edge labeler now applies a local consistency enforcer to the belief values. Let's say that because E_1 and E_2 have maximal similarities with E_A, we wish to use E_2 to revise our beliefs regarding E_1. To do so, the labeler measures the collinearity of E_1 and E_2 and then the noncollinearity of the same two edges. Let's say that by using equations 1 and 2 we get

compatibility$(E_2, E_1) = 0.8$, and
incompatibility$(E_2, E_1) = 0.1$.

We now construct an "updating" bpa for E_1, as follows:

$$m_{update2\text{-}1}(\{E_A\}) = m_{E_2}(\{E_A\} \times \text{compatibility}(E_2, E_1) = 0.64$$

$$m_{update2\text{-}1}(\{\neg E_A\}) = m_{E_2}(\{E_A\} \times \text{incompatibility}(E_2, E_1) = 0.08$$

$$m_{update:\,2\text{-}1}(\Theta) = 0.28$$

where, again, the bpa for the FOD was set to the uncommitted portion of belief. The ¬ symbol as used in this context, denotes set negation, that is, $\neg E_A$ To find the new bpa over Θ for E_1, we use Dempster's rule, (see appendix), to combine m_{E_1} with $m_{\text{update: } 2\rightarrow 1}$ to obtain a new m_{E_1}, given by

$m_{E_1}(\{E_A\}) = 0\ 87$
$m_{E_1}(\{E_B\}) = 0\ 04$
$m_{E_1}(\{E_c\}) = 0\ 0$
$m_{E_1}(\{E_D\}) = 0\ 02$
$m_{E_1}(\{\neg E_A\}) = 0.02$
$m_{E_1}(\Theta) = 0\ 05$ (4)

Revising Belief Values by Enforcing Relational Constraints. The labeler KS also contains procedures for enforcing relational constraints within each grouping which is done in the following manner. Consider the bpa's over Θ for the edges E_1 and E_9, as illustrated in figure 5 Although the bpa for E_1 takes its maximum value for the label E_A, the bpa for E_9 is a maximum at E_D. Let's say we wish to compute, using relational constraints, the bpa $m_{\text{update: } 9\rightarrow 1}$, which is the additional belief generated by E_9 for E_1's label. To find $m_{\text{update: } 9\rightarrow 1}$, we note that the geometric relationship between E_A and E_D is already known because their positions in the world coordinate system are known Let $T_{E_A\rightarrow E_D}$ be the transform that makes the model edges E_A and E_D coincident To get a measure of the extent to which the geometric relationship between E_1 and E_9 is the same as the one that exists between E_A and E_D, the labeler carries out the following (in)compatibility computations:

compatibility(E_9, E_1)
= collinearity$(E_9, T_{E_A\rightarrow E_D}(E_1))$
and
incompatibility(E_9, E_1)
= noncollinearity$(E_9, T_{E_A\rightarrow E_D}(E_1))$

Clearly, if compatibility (E_9, E_1) = 1 0 (when such happens, we also have incompatibility $(E_9, E_1) = 0.0$), the geometric relationship between E_1 and E_9 in the data is exactly the same as between E_A and E_D in the model To explain how the relational information is used to update the bpa distribution for, say, E_1, we go back to our figure 5 example Let's say that the compatibility calculations yielded the following results:

compatibility$(E_9, E_1) = 0.7$
and
incompatibility$(E_9, E_1) = 0\ 4$

Because these measures sum to more than 1, they are normalized by their sum to yield the final (in)compatibility measures:

compatibility$(E_9, E_1) = 0\ 64$
and
incompatibility$(E_9, E_1) = 0.36$

We now construct another bpa for E_1, as follows

$m_{\text{update } 9\rightarrow 1}(\{E_A\})$
$= m_{E_9}(\{E_D\}) \times \text{compatibility}(E_9, E_1)$
$= 0.6 \times 0.64$
$= 0.38$

$m_{\text{update: } 9\rightarrow 1}(\{\neg E_A\})$
$= m_{E_9}(\{E_D\}) \times \text{incompatibility}(E_9, E_1)$
$= 0.6 \times 0.36$
$= 0\ 22$

$m_{\text{update: } 9\rightarrow 1}(\theta) = 0\ 4$

where the value of $m_{E_9}(\{E_D\})$was obtained from the table of values in equation 3. By using Dempster's rule to combine this update bpa with that in equation 4, we obtain a further revision of the belief values over the FOD for E_1.

Combining Belief Revisions from Relational Constraints and Mutual Collinearity Constraints. Actually, a single procedure is used for enforcing both the mutual and the relational constraints within a group. Note that the (in)compatibility calculations for relational constraints in the previous subsection reduce to the computations required for (in)compatibility calculations for mutual consistency in the subsection Revising Belief by Enforcing Mutual Consistency Constraints if we use an identity transformation for $T_{E_X\rightarrow E_Y}$.

Therefore, we can use the following integrated algorithm for enforcing the relational constraints and the local collinearity constraints. Suppose n edges—E_1, E_2, ..., E_n—have been grouped together in the data panel and are hypothesized to belong to a single face. Also, suppose that the edges in one of the possible corresponding faces in the model panel are E_A, E_B, . , E_Z. Clearly, the FOD, Θ, for each data edge is

$\Theta = \{E_A, \ldots, E_z\}$

Let's further say that as described in Computing Initial Belief Values for Elements in a Group, measuring local collinearities of the data edges against those of the model edges yields the following initial bpa distribution for each data edge:

$$m_{E_i}(\{E_\alpha\}) \text{ for } \begin{matrix} i = 1, \ldots, n \\ \alpha = A, \ , Z \end{matrix}$$

with the assumption that initially

$$m_{E_i}(\theta) = 1 - \sum_{\alpha=A}^{z} m_{E_i}(\{E_\alpha\})$$

and $m_{E_i}(\bullet) = 0.0$ for all other nonsingleton subsets of Θ

For each data edge E_i, let $E\alpha_{max}$ be the model edge for which the bpa takes a maximum value; that is,

$m_{E_i}(\{E\alpha_{max}\}) \geq m_{E_i}(\{E\alpha\})$
for $\alpha = A \ldots, Z$

For each edge E_1, a single model edge, $E\alpha_{max}$, exists for which the bpa value is maximum; we denote this maximum belief model edge by $E\alpha_{max(i)}$ In case of a tie because of two or more model edges yielding the same value for bpa, we arbitrarily select one of them

When incorporating new evidence for an element's label, we can ease the computational load by using the property that when using Dempster's rule of combination, the order in which evidences are combined does not matter. To elaborate, if we update an element's bpa incrementally with every piece of new evidence from the element's siblings, then the new bpa is computed as

$m_{new} =$
$(((m_{old} \oplus m_{\text{update: } 1\rightarrow j}) \oplus m_{\text{update: } 2\rightarrow j})$
$\oplus \ldots \oplus m_{\text{update: } n\rightarrow j})$

where $\oplus$ denotes that Dempster's rule of combination is being used for combining evidence, and $m_{\text{update:}j\rightarrow 1}$ is the updating bpa for the new evidence that element j is providing for element i. Because Dempster's rule of combination is invariant with respect to the order of combination, the new bpa can be expressed as

$$m_{new} = m_{old} \oplus m_{update}$$

where

$m_{update} = ((m_{\text{update: } 1\rightarrow j} \oplus m_{\text{update: } 2\rightarrow j})$
$\ldots \oplus m_{\text{update: } n\rightarrow j})$

Now we can take advantage of the fact that the updating bpa's use binary FODs (if element E_t is labeled E_A, then in each updating bpa, the belief is

nonzero for only the proposition E_A, $\neg E_A$ and Θ itself). Barnett (1981) describes how Dempster's rule can be implemented in linear time if all focal elements of the belief functions are either singletons or their complements. Currently, we do not use Barnett's formulas because FODs for the elements on the blackboard are fairly small. However, we do take advantage of the fact that with the formulation presented here, we only have to deal with three elements in the power set of the updating FOD instead of all the $2|\Theta|$ elements.

A function, **update-belief**, that provides an updating bpa for an arbitrary element, E_i, using compatibility and incompatibility measures is shown in figure 6. It takes the element being updated as a parameter, and it returns the updating bpa for this element. Thus, the bpa for E_i could be updated with the following function call:

$m_{E_i} = m_{E_i} \oplus$ **update-belief**(E_i)

Evidence Propagation between Levels in the Hierarchy. Mutual consistency constraints and geometric constraint relations are not the only sources of knowledge used to update an element's belief function. A mechanism is also provided for passing belief values between different levels of the hierarchy. This passing is done to satisfy the intuitive argument which says any evidence confirming an element's label should also provide evidence that its parent's label is correct. We also require that disconfirming evidence be passed to the lower levels of the hierarchy (for example, if we think that a face is mislabeled, then all its constituent edges are also most likely mislabeled).

We use the updating bpa, m_{update}, as computed by the update-belief function, when passing evidence up the hierarchy. We combine m_{update} not only with the bpa for the element in question but also with this element's parent Combining the updating bpa with an element's parent makes intuitive sense because all new evidence generated on a level comes from the (in)compatibility between elements on this level. If the children of an element have consistent (compatible) labels, then these child elements should provide evidence that the label given to the parent element is correct. Likewise, children with inconsistent labels provide evidence that an element's label is incorrect. Thus, by passing the updating bpa's to each parent element on a higher level of the hierarchy, we are providing new evidence for these elements based on the consistency or the inconsistency of its descendants.

Evidence from an element cannot be directly applied to its parent because FODs of an element and its parent are composed of different types of data elements. However, we show that it is possible to build a FOD that can be used to update the belief functions of elements on a higher level of the hierarchy Assume that the model panel is as shown in figure 3 and that edge E_1 on the data panel is a child of face F_1. Furthermore, assume that E_1 is labeled E_A, and F_1 has label F_A. Because the bpa for edge E_1, as given by $m_{update:\ E_1}(\{E_A\})$ arises from the consistency of E_1's label with its sibling's labels, it can be considered a weighted vote of confidence that face F_1's label is correct. Likewise, because $m_{update:\ E_1}(\{\neg E_A\})$ arises from the inconsistency of E_1's label with its sibling's labels, it can be considered a (weighted) vote of no confidence in the correctness of face F_1's label. Thus, $m_{update:\ E_1}(\Theta)$ can be considered to be the amount of ignorance about F_1's label. Using this rationale, we can define an updating bpa for face F_1 with the following nonzero basic probability numbers:

```
function update-belief (Ei)
    ;; initialize updating bpa as a vacuous belief function
    mupdate(Θ) = 1.0
    mupdate(·) = 0.0        ; all other subsets
    ;; use all n elements in the data panel with the same parent group
    ;; when updating Ei
    For each j = 1, 2, ..., n
        begin
            cj = compatibility(Ei, Ej)
            dj = incompatibility(Ei, Ej)
            ;; normalize (in)compatibility measures if needed
            if (cj + dj > 1.0)
                begin
                    cj = cj / (cj + dj)
                    dj = dj / (cj + dj)
                end
            ;; create updating bpa corresponding to element Ej
            mupdate:j→i({Eαmax(i)}) = cj × mEj({Eαmax(i)})
            mupdate:j→i({¬Eαmax(i)}) = dj × mEj({¬Eαmax(i)})
            mupdate:j→i(Θ) = 1.0 − mupdate:j→i({Eαmax(i)}) − mupdate:j→i({¬Eαmax(i)})
            ;; accumulate into updating bpa
            mupdate = mupdate ⊕ mupdate:j→i
        end     ; (for loop)
    ;; return the accumulated sum of new evidence
    return (mupdate)
End.    ; update-belief
```

Figure 6 Algorithm Used to Generate an Updating bpa

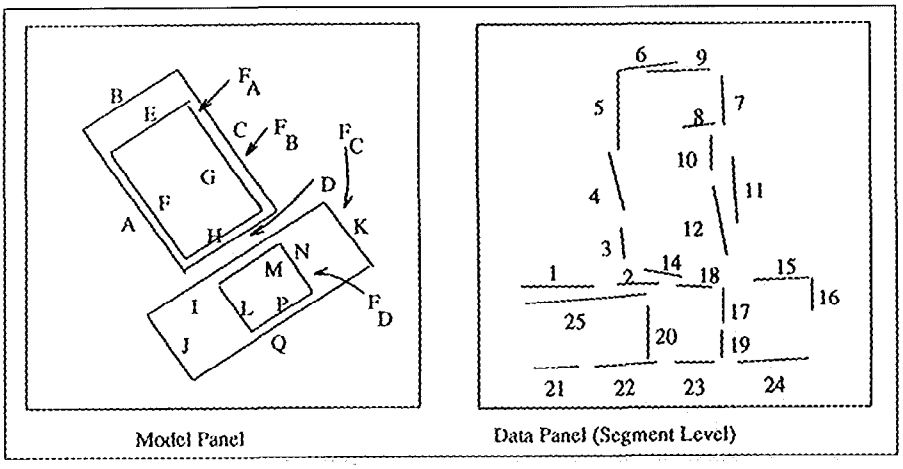


Figure 7
Left The Expected Scene Information on the Blackboard Model Panel; Right. The Data at the Segment Level of the Data Panel

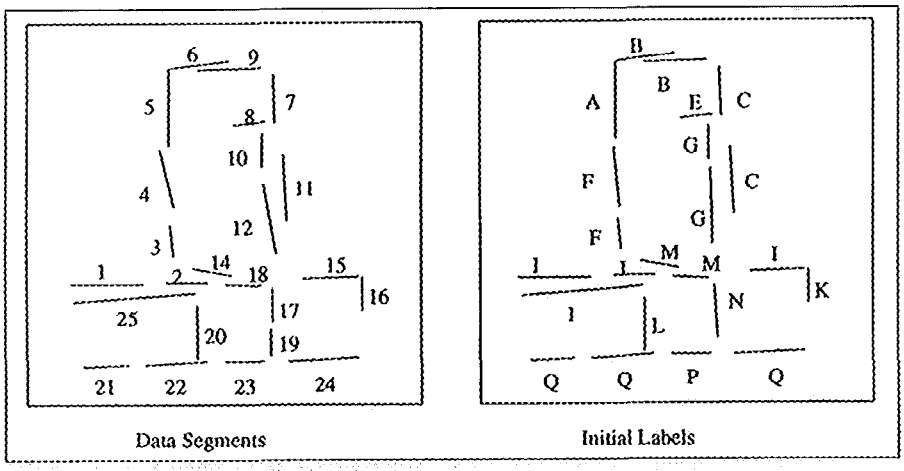


Figure 8 Initial Labels for Data Segments

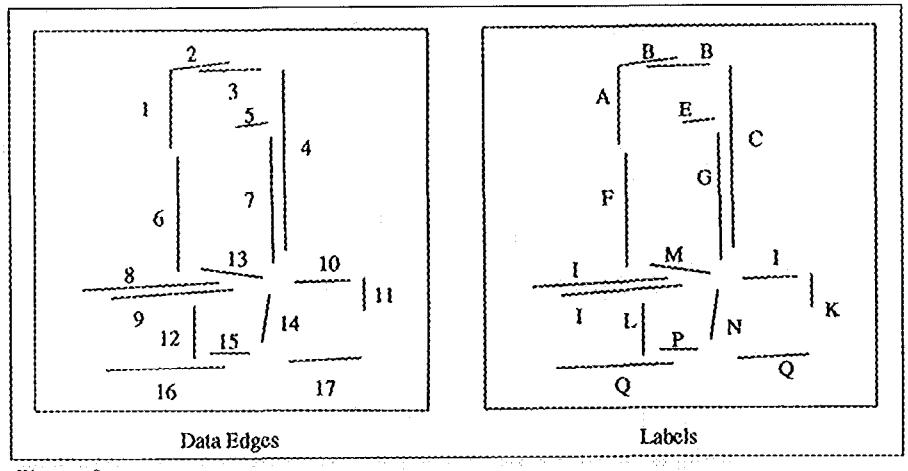


Figure 9.
Edges with Initial Labels Derived from Data Segments. (Note the numeric labels for the data edges on the left are independent of the numeric labels used for the data segments in figure 8)

bpa for F_1. As described in the last subsection, after initial belief values are assigned, the FOD for an element's updating bpa is really binary in nature. Thus, the only nonzero elements of the updating bpa for E_1 are $m_{update:\ E1}(\{E_A\})$, $m_{update:\ E1}(\{\neg E_A\})$, and $m_{update:E1}(\Theta)$. Because these elements are the only nonzero basic probability numbers from E_1s updating bpa, they sum to 1 Given this fact, it is trivial to show that $m_{update:\ E1 \rightarrow F1}$ is also a bpa. We can now express the total accumulated new belief for face F_1 from its children E_1, E_2, ..., E_n as

$$m_{update:\ F1} = (m_{update:\ E1 \rightarrow F1} \oplus m_{update:\ E2 \rightarrow F1}) \oplus \ . \ \oplus m_{update:\ En \rightarrow F1})$$

Information is passed down the hierarchy only if it is disconfirmatory. Currently, this downward propagation of information takes the form of the reassignment of FODs caused by the ancestor of an element having its label changed. In the previous example, this form of downward propagation could happen if the label for face F_1 were changed to F_B. Using information from the model panel (as shown in figure 3), we note that the FOD for E_1 would then be changed to

$$\Theta = \{E_C, E_E, E_F, E_H\}$$

Grouper KS

The grouper KS builds data elements on the upper levels of the hierarchy from the segments and vertices deposited by the low-level vision system. It does this building in a data-driven manner by first grouping segments into edges and then grouping the edges into faces, and so on. The KS is activated (by mechanisms described in Scheduler and Monitor for Blackboard Control) by a request to find the parent of an ungrouped data element, called the *seed element.* In response to such a request, the KS first creates a dummy parent element one level up on the blackboard from the seed element and then tries to find all the data elements compatible with the seed element. These compatible data elements are found by first determining the most probable model label for the data element and then finding the siblings of this model label and the data elements that correspond to these model siblings. The set of compatible

data elements thus found form the seed element's siblings on the data side. It is the job of the labeler KS to eventually give a label to the dummy parent element of the data group so formed.

In general, two data elements on the same level must satisfy two requirements if they are to be grouped together. First, the elements must satisfy a level-specific adjacency constraint. These constraints usually force the KS to consider only elements that are physically close when it forms groups. The other requirement is obtained from the compatibility metric used by the labeler KS; that is, the elements must be highly consistent in order to be grouped. For example, at the segment level of the blackboard, two segments must be highly collinear, and, similarly at the edge level, two edges must lie on a common plane to be grouped. Although the KS should be able to group elements based solely on their geometry, it must also be able to take advantage of any label information that the labeler KS has provided. For example, suppose the labels for a set of data edges are known to belong to a single face on the model panel, the grouper KS should group the data edges together.

Although eventually the grouper and labeler KSs work in concert on an opportunistic basis, some initial groupings must be formed for the labeler KS to act at all. In other words, the grouper KS must be able to generate some initial groupings. To illustrate how this function is performed, consider the example in which the image preprocessor has deposited the data, shown on the right in figure 7, on the segment level of the data panel. Information about the expected scene on the model panel is shown on the left.

Again, remember that the information in both the model and the data panels is in the same world coordinate frame. (For a mobile robot with cameras slanted downward, this coordinate frame corresponds to the flat ground plane.) For generating initial groupings, every segment in the data panel is compared with all the model edges that are in the vicinity of the segment, the basis of comparison being collinearity. For each data segment, we retain that model edge label which yields the highest value for collinearity In figure 8, the left frame shows the data segments and the right frame a possible label for each segment. All the adjoining segments that have the same initial labels are now joined together into edges, as shown in the left plate in figure 9.[7] Note that the numeric labels assigned to the data edges in figure 9 are independent of the numeric labels of the data segments in figure 8.

Using the labeling shown on the right in figure 9, the grouper constructs the following initial groupings from the data edges shown in the figure:

$$\begin{aligned} F_{1\_initial} &= \{E_5, E_6, E_7\} \\ F_{2\_initial} &= \{E_1, E_2, E_3, E_4\} \\ F_{3\_initial} &= \{E_8, E_9, E_{10}, E_{11}, E_{16}, E_{17}\} \\ F_{4\_initial} &= \{E_{12}, E_{13}, E_{14}, E_{15}\} \end{aligned} \quad (5)$$

It is not uncommon for an initial grouping to be contaminated by multiple renditions of the same edge in a scene. Suppose the gray-level variations corresponding to a scene edge do not exhibit a monotonic variation in directions perpendicular to the edge; depending on the size of the edge detection operator, the result can be multiple parallel edges in close proximity to one another For example, segments 1 and 25 in figure 7 could be examples of such an artifact. An important job assigned to the grouper is the detection of such parallel edges. The grouper does this detecting by measuring the angle between the elements that have the same label in an initial group and the extent of the overlap between the two edges, the overlap being measured by projecting the smaller edge on the longer one. When such competing parallel edges are found, multiple groupings are formed from an initial group by retaining only one competing parallel edge at a time.

To illustrate, in the initial groupings shown in equation 3, edges 8 and 9 in $F_{3\_initial}$ are found to be competing;

```
type Data=element
  (
    id:              integer;        -- Unique id number
    panel:           integer;        -- Panel in the blackboard
    type:            symbol;         -- Type of data (two_d, three_d, model)
    level:           symbol;         -- Level in the panel
    children:        child_list;     -- Element ids of children of element

    -- General Parameters
    value:           integer;        -- Edge strength, etc.
    size:            integer;        -- # of pixels in edge, etc.

    -- Parameters valid only for vertex elements
    image_coord:     coord;          -- (vertex) coordinate of vertex
    world_coord:     vector;         -- (vertex) coordinate of vertex

    -- Parameters used for uncertainty management
    frame:           fod;            -- Frame of discernment
    bpa:             bpas;           -- Basic probability assignments
    positive:        real;           -- Update bpa - belief
    negative:        real;           -- Update bpa - disbelief
    label:           integer;        -- Label of element
    belief:          real;           -- Belief in label
  );
```

Figure 10 WME Class Definition for Data Elements

```
type KSAR=element
  (
     KS:              symbol;        -- Knowledge source being triggered
     action:          symbol;        -- action KS is to perform

     data_element:    integer;       -- Data Element being focused on
     level:           symbol;        -- Level being focused on
     panel:           integer;       -- Panel Being focused on

     id:              integer;       -- KSAR id #
     status:          symbol;        -- KSAR status

     trigger_cycle:   integer;       -- cycle KSAR was formed
     trigger_KSAR:    integer;       -- KSAR which triggered this one
     active_cycle:    integer;       -- cycle during which this KSAR was active
     priority:        integer;       -- KSAR priority.
  );
```

Figure 11 WME Class Definition for KSAR

```
--
-- RULE: select_max_benefit_KS
--     IF: PSEIKI is "running" and the current strategy is to maximize benefit
--     THEN: select the KS such that no other KS has greater benefit.
--         Mark the KS as "active."
--
rule select_max_benefit_KS {
              (Context current=run_pseiki);
              (Strategy current=maximize_benefit);
       &ks    (KS   status<>active);
              (KS   status<>active; benefit > &ks.benefit);
-->
       modify &ks (status=active);
};
```

Figure 12 Rule Used to Select Active (KS, action) Pair.

the same is the case with edges 2 and 3 in $F_{2\_initial}$. Thus, the initial groupings lead to the following groups:

$F_1 = \{E_5, E_6, E_7\}$
$F'_2 = \{E_1, E_2, E_4\}$
$F''_2 = \{E_1, E_3, E_4\}$
$F'_3 = \{E_8, E_{10}, E_{11}, E_{16}, E_{17}\}$
$F''_3 = \{E_9, E_{10}, E_{11}, E_{16}, E_{17}\}$
$F_4 = \{E_{21}, E_{13}, E_{14}, E_{15}\}$

Some Data Structures Used by PSEIKI

Fundamentally, a blackboard is a data structure that is operated on by a number of KSs in an opportunistic manner (Hayes-Roth 1985; Nii 1986a, 1986b). The PSEIKI system uses the working memory of OPS83 for this data structure; each working memory element (WME) corresponding to the blackboard data structure describes a data element at some level of the blackboard.[8] In addition to being a host for the blackboard data structure, the working memory is also used for storing the knowledge source activation records (KSARs); a KSAR is created by the monitor when the trigger conditions for a KS are satisfied by some data element. (It is the job of the monitor to keep track of the data on the blackboard and constantly check whether a newly created data element satisfies the triggering conditions for a KS.) KSARs can also be created by KSs, allowing KSs to trigger other KSs explicitly. Each KSAR holds the identity of the data element that meets the triggering conditions of a KS; the relevant KS; and other pertinent information, such as the cycle during which the KSAR was created. This information indicates to the KS which object should be worked on and is used by the scheduler when it chooses a KSAR to activate.

In the rest of this article, we describe the data structures used for representing the data elements and KSARs. Subsequently, we discuss some of the productions used by the scheduler, the monitor, and the grouper KS.

WMEs for Representing Data

As mentioned earlier, a single WME class is used for all the data elements regardless of the blackboard level at which the data element might reside. In other words, the same WME class is used for edges, faces, objects, and so on. The distinctions between different types of data elements are introduced by using appropriate values for the attribute "level." Using the same WME class allows generic functions to be freely applied to all the data levels.

Figure 10 shows the definition of the WME class for representing data. Most of the WME fields are self-explanatory. The element's **id** number is a unique identifier used to keep track of individual data elements; data elements are always referenced by their id numbers. The **panel** and **level** fields specify the element's location on the blackboard. The **type** field is used to specify the type of data from which the element is derived; the values that it can assume are two_d, three_d, and model. The next few fields are parameters of the data element. The **value** field is a generic attribute used to specify a level-specific value. For example, it is used to specify the strength of an edge or the average gray level associated with a face. The **size** parameter is also generic; this parameter is used to specify the degree, length, area, or volume if an element is a vertex, edge, face, or object, respectively. The next two parameters specify the data element's location if it is a vertex. The **image_coord** attribute indicates a vertex's coordinate on the image plane if

it was obtained from 2-D data. Likewise, the **world_coord** attribute specifies a vertex's location in the world coordinate frame.

The remaining fields shown in figure 10 hold the uncertainty information about a data element and are used by the labeler KS. The frame attribute holds the element's FOD, and the bpa attribute holds the element's basic probability assignment. An element's updating bpa is stored in the **positive** and **negative** attributes; these values indicate the new belief and disbelief in the element's label. This new evidence is obtained from the compatibility of the element with its siblings and its children's compatibility, as discussed in Evidence Accumulation Mechanisms in Knowledge Sources. Finally, the element's label and belief in this label are indicated by the next two attributes.

The WME Class for Representing KSARs

Figure 11 shows the WME class definition for representing a KSAR. The **data_element, level,** and **panel** fields specify the focal element of the KSAR. Using these three fields, we can focus on the following kinds of entities: the entire blackboard, a blackboard panel, a level of the blackboard, a level of a specific panel of the blackboard, or a specific data element on the blackboard. The KS and action fields of the KSAR specify what action is to be performed on its focal element. PSEIKI's scheduler uses the **trigger_cycle, trigger_KSAR,** and the **priority** fields when ranking KSARs for firing. The **id** field is used to keep track of the KSARs, and the state of any KSAR is determined by its status field.

A KSAR is originally created with its status marked pending, which means that the KS has been triggered but has not been run yet. When the scheduler decides to fire on a KSAR, it changes the KSAR's status to active. The KS's precondition and poisoning productions are now allowed to fire; it is their job to mark the KSAR's status as running if the preconditions are met or as poisoned if they aren't If the KSAR is determined to be poisoned, the KS's body productions are not allowed to fire, and control passes back to the scheduler. Otherwise, if the status has been set to running, the KS's body productions will be allowed to fire. After the KS has accomplished its goal, it marks the KSAR's status field as finished and returns control to the scheduler.

Scheduler and Monitor for Blackboard Control

In this section, we first describe the operation of the schedule and monitor. Subsequently, we comment on how the KSs interact with the schedule and monitor. Finally, we detail the operation of the grouper KS as an example of KS processing.

Scheduler Operation

It can reasonably be stated that the scheduler is the heart of any blackboard. It is the scheduler's job to choose what action to perform at any cycle of the blackboard operation; it carries out this job by selecting one of the pending KSARs and activating the corresponding KS. PSEIKI's scheduler, which consists of a set of metarules, runs by default; that is, it runs automatically when no KSs are active. Initially, when the data, in the form of edge segments, are deposited on the blackboard, the scheduler is invoked to get the entire process started.

The strategy used by PSEIKI's scheduler for selecting one of the pending KSARs is patterned after the Hearsay system (Erman et al. 1980) The scheduler first forms a set of (KS, action) pairs for all the KSs. Note that each KS can be used to accomplish different actions; for example, the labeler KS can both initialize and update labels. Thus, the priority accorded a KS should depend on which action is desired. The pairs are then ranked, and one is selected and marked as active on the basis of the costs and benefits associated with the different KSs, the cost being a measure of the computational difficulty of running a KS and the benefit a measure of a heuristic estimate of the power of a KS to accomplish the overall goal. After selecting a (KS, action) pair, the scheduler ranks all the pending KSARs that seek to invoke the pair. If the selected (KS, action) pair has no pending KSARs, the scheduler selects the next most highly ranked pair. Ranking of the pending KSARs is based on a combination of their priority, focal element, focal element's blackboard level, and focal element's recency It then selects the top-ranked KSAR and marks it as active; the corresponding KS is then fired. Figure 12 is a sample rule that performs the KS ranking and selection on the basis of the KS's relative benefit.

In this rule, the first two condition elements (CEs) specify that PSEIKI must be running and that the current KS selection strategy is based on maximizing KS benefit The next two CEs perform the work of the rule. The third CE matches any nonactive (KS, action) pair; the WME corresponding to this (KS, action) pair is marked active by the scheduler. The last CE prevents the rule from firing if there is an available inactive (KS, action) pair with greater benefit. Thus, the overall action of this rule is to find the inactive (KS, action) pair with the greatest benefit and to mark its status as active

After a (KS, action) pair is chosen, a particular KSAR must be selected from the group of pending KSARs that use it. Associated with every KS is a different method of choosing the best KSAR For example, if the labeler KS is active, the scheduler tries to choose a KSAR that focuses on the highest-level data element possible. As we mentioned previously, the strategy of focusing on the highest-level data element is equivalent to checking global consistency before local consistency. Conversely, the scheduler forces the grouper to group data elements on the bottom levels of the hierarchy first and then work its way up Figure 13 shows a rule that selects a labeler KSAR.

Once again, the first CE specifies that PSEIKI must be running in order for the rule to fire. The next two CEs find a pending KSAR that focuses on the active (KS, action) pair The third CE matches the WME that is the focus of the KSAR The structure on the next line uses a feature of OPS83 to choose a data element at the highest possible level on the blackboard. OPS83 uses the value in the square brackets to rank rule instantiations in

```
--
-- RULE: select_label_KSAR
--     IF: PSEIKI is running and a KS and action have been selected.
--     THEN: select the KSAR that focuses on the highest level
--         data element possible and mark the KSAR as "active."
--
rule select_label_KSAR {
                    (Context current=run_pseiki);
        &ks         (KS name=label; status=active);
        &ksar       (KSAR KS=label; action=&ks.action; status=pending);
        &el         (Element id=&ksar.data_element);
        {level_no(&el.level)};    -- choose the highest level possible
-->
        modify &ksar (status=active; active_cycle=&current_cycle);
        make (Context current=reset_KS_status);
};
```

Figure 13 Rule Used to Select a KSAR that Seeks to Invoke the Labeler KS

```
--
-- RULE: group_trigger_1
--     IF: An orphaned and labeled data element is found
--         AND there is no KSAR to group that element.
--     THEN: Create a new KSAR to group that element and find its parent.
--
rule group_trigger_1 {
        &el   (Element type<>model; level<>vertex; label<>0);
        ~     (Element child(&el.id, @.children));
        ~     (KSAR KS=group; action=initialize;
                    (@.status=pending ∨ @.status=poisoned);
                    data_element=&el.id);
-->
        make (KSAR KS=group; action=initialize;
                trigger_cycle=&current_cycle;
                id=&next_KSAR_id; status=pending;
                data_element=&el.id; priority=0.5);
        &next_KSAR_id = &next_KSAR_id + 1;
};
```

Figure 14 Monitor Demon Used to Create a KSAR for the Grouper KS.

the conflict set; everything else remaining equal, OPS83 selects the rule instantiation for firing that yields the greatest value for the expression inside the brackets. Thus, if the function level_no() returns numbers that increase in value for higher levels on the blackboard, this rule chooses a data element on the highest level possible.

The strategy used by the scheduler to rank the (KS, action) pairs is controlled by the fields of a special WME; these fields determine which (KS, action) selection rules are allowed to fire. At this time, this WME is specified before the processing begins and remains fixed thereafter. However, our plan is to institute a set of metarules that will dynamically alter the fields of this WME in response to diminishing returns for a given strategy.

Monitor Operation

The monitor is the watchdog for the blackboard; it is the monitor's job to keep track of the data on the blackboard and trigger KSs when specific conditions are met. After the monitor triggers a KS by building a pending KSAR, it can also execute some immediate code. It is also up to the blackboard monitor to watch the blackboard and determine if the status of any poisoned KSARs should be reset to pending This resetting of the status happens if the KS action on the specified data element has once again become valid. It is also up to the monitor to determine if any poisoned KSARs should be deleted; deletion occurs if there is no chance that the KSAR could once again become valid.

The blackboard monitor makes extensive use of OPS83 demons. A *demon* in OPS83 is a rule whose first CE is not a context, goal, or KSAR. Because of the OPS83 rule selection strategy, these rules take precedence over ordinary rules (for example, rules inside of KSs or scheduler rules) and fire as soon as they become completely instantiated. Thus, a demon in OPS83 can be thought to operate outside of any context, KS, or goal search.

For example, a monitor rule used to trigger the grouper KS is shown in figure 14. This rule fires when it finds any data elements without any par-

ents (orphan elements). The rule then creates a KSAR that directs the grouper KS to find the element's parents. This rule works as follows: The first CE matches any new data element if it is not a model element and has a label; this data element is the focus element of the rule. The second CE allows the rule to fire only if its focus element is an orphan. This CE uses the function child() to match any WME that has the first CE's id number in its list of children. The tilde in front of the CE acts as a negation symbol; that is, it allows the rule to fire only if no WME matches the CE. Thus, the tilde in front of the second CE of this rule keeps the rule from firing if the focus element has a parent The last CE keeps the rule from firing if the grouper KS has already been triggered on this data element; the rule is only fired if a pending grouper KSAR focused on the same element can not be found. Notice that this rule simply builds a pending KSAR and that no immediate code is executed.

```
--
-- RULE: poison_group_1
--     IF: a KSAR indicates that en element should be put into a group
--         and that element already has a parent (and thus is in a group)
--     THEN: mark the KSAR as poisoned.
--
rule poison_group_1 {
        &ksar       (KSAR KS=group; action=initialize; status=active);
                    (Element child(&ksar.data_element, @.children));
-->
        modify &ksar (status=poisoned);
};
```

Figure 15. An Example of a Poisoning Rule

```
--
-- RULE: group_into_element_driver
--     IF: grouper KS is running and the focal element does not yet
--         have a parent.
--     THEN: create a parent element and a KSAR to label the parent.
--
rule group_into_element_driver {
        &ksar       (KSAR KS=group; action=initialize; status=running);
        &el         (Element id=&ksar.data_element; level<>segment);
        ~           (Element children[1]=&ksar.data_element);
        &model      (Element child(&el.label, @.children));
-->
        -- Make dummy parent element
        &max_id = &max_id + 1;
        make (Element type=&el.type; panel=&el.panel; level=&model.level;
                id=&max_id; size=&el.size; value=&el.value;
                label=&model.id; positive=&el.belief;
                children[1]=&el.id);
        write () 'initializing group with ', &el.level, &el.id, '\n';

        -- Make KSAR to requesting that the parent's belief be initialized
        make (KSAR KS=label; action=initialize;
                trigger_cycle=&current_cycle;
                id=&next_KSAR_id; status=pending;
                data_element=&max_id; priority=0.5);
        &next_KSAR_id = &next_KSAR_id + 1;
};
```

Figure 16 An Example of a Driver Rule

Operation of the KSs

Even though the various KSs perform greatly different tasks, many of the same actions are performed by all of them during the task-solving process. These actions start when the scheduler marks a KSAR's status as active. After a KS becomes active, its poisoning rules are allowed to fire; these rules make sure that the KS's preconditions have not become invalid since the KS was triggered. If a poisoning rule does fire, it sets the KSAR's status to poisoned and returns control to the scheduler If none of the poisoning rules fire, a rule that marks the KSAR's status to running fires by default.

After the KS starts running, the control flow becomes more KS specific, but it still follows the same pattern. Usually, the first few rules that fire after the KS starts running are *driver rules* These rules don't contribute directly to the solution of the KS's task; instead, they initialize elements in the working memory that the KS needs to solve the task. These driver rules can generate contexts that are needed by the KS in its problem-solving activity. They can also put dummy data elements on the black-

```
--
-- RULE: group_into_faces
--     IF: Edges are being grouped into a face
--       AND: There is an edge whose label element is a child of
--          the parent's label element
--       AND: That edge is not in a group that has the same label as the parent
--     THEN: compute the colinearity of that edge and a transformed version of
--          the focus element.  If this exceeds a threshold, add the edge
--          to the face's child list
--
rule group_into_faces {
        &ksar           (KSAR KS=group; action=initialize; status=running);
        &face           (Element children{1}=&ksar.data_element);
        &model          (Element id=&face.label);
        &edge1          (Element id=&ksar.data_element);
        &edge2          (Element type<>model; level=edge; id<>&edge1.id;
                                child(@.label, &model.children);
                                (~child(@.id, &face.children)));
        &xfrm           (Model_xfrm from=&edge1.label; to=&edge2.label);
        -- get parameters needed in rhs computations
        &s1             (Element id=&edge1.children{1});
        &e1             (Element id=&edge1.children{2});
        &s2             (Element id=&edge2.children{1});
        &e2             (Element id=&edge2.children{2});
        &dist           (Constant type=group_threshold);
-->
        local &collin: real;
        local &s1_xfrm, &e1_xfrm: vector;
        local &i, &next_child: integer;

        call apply_xfrm(&s1_xfrm, &xfrm.xfrm, &s1.world_coord);
        call apply_xfrm(&e1_xfrm, &xfrm.xfrm, &e1.world_coord);
        &collin = find_collinearity(&s1_xfrm, &e1_xfrm,
                                &s2.world_coord, &e2.world_coord,
                                &group_slop);
        if (&dist.real_value < &collin) {
                &next_child = 0;
                for &i = (1 to &max_children)
                        if ((&next_child = 0) ∧ (&face.children[&i] = 0))
                                &next_child = &i;
                modify &face  (children[&next_child] = &edge2.id;
                                        update_belief(&face, &collin * &edge2.belief, 0.0));
                write () |   grouping edge |, &edge2.id, '\n';
        };
};
```

Figure 17 Rule Used to Group Edges into Faces

board that are "fleshed out" during the course of the KS's processing. After the KS's driver rules are fired, the control flow becomes KS specific In the next subsection, we show an example of KS processing to demonstrate the control flow inside a KS.

Grouper KS Operation

To illustrate the flow of control inside a KS, we examine the operation of the grouper KS as it initializes a collection of edges into a face, using the example from figures 7, 8, and 9 Assume that a KSAR focused on the element E_9 of figure 9 has just been activated with (KS, action) pair set equal to (grouper, initialize). As we described previously, when the KS is first activated, the poisoning rules are allowed to fire. Figure 15 is an example of a poisoning rule used by the grouper KS This rule is meant to poison a KSAR if it tells the grouper that a nonorphaned element should be put into a group.[9] The rule works in the following manner: The first CE matches the active KSAR if its action is to initialize a group. The second element uses the same child() function as the monitor rule discussed previously. If this CE matches a WME, then the focus element already has a parent; the rule then fires, and the KSAR is marked poisoned. If no poisoning rules fire, another rule fires by default and marks the KSAR's status running Thus, if we assume that element E_9 is an orphan at this point, then the active KSAR's status is set to running.

The grouper KS uses driver rules to initialize internal processing; these rules fire immediately after the KS starts running. When the KS is initializing a group, the driver rule builds a dummy data element, the parent element, on the blackboard. This element is the parent of the focus element of this KS activation. It is the job of the grouper KS to group the focus element and its siblings into this element. Figure 16 shows the driver rule for group initialization.

The rule in figure 16 works as follows: The first two CEs match the running KSAR and the focus element. The purpose of the third CE is to prevent the rule from firing more than once during any KS activation by allowing the rule to fire only if no parent of the seed element is already in the working memory. It would be able to find the parent element because a parent element always has the focus element of the KS as its first child. The last CE is designed to find an element that could possibly be the model panel counterpart of the parent element by finding the parent of the focus element's label element.

The rule performs two actions when it fires. First, it builds the parent element. As we mentioned previously, the KS's focus element and its siblings are grouped into this element The parent element is initialized with appropriate parameters: panel, data type, level, id number, size, and so on. Furthermore, the parent element's first child is set as the focus element to prevent the driver rule from firing twice and allow the remaining KS body rules to easily find both the focus and parent elements. The rule also builds a KSAR that requests the parent element be labeled.

Because edge E_9 is an orphan in our example, this driver rule will fire. When the firing happens, a new element, say, element F_{20}, is created and deposited on the blackboard. This new element is on the face level of the data panel with label F_C and initially has element E_9 as its only child. Now it is up to the rest of the KS body rules to find element E_9's siblings and group them into the face F_{20}

After the driver rule initializes the parent element, the remaining KS body rules can fire. Just one KS body rule needs to fire to group edge elements into a face element. This rule is nontrivial and fires at least one time for every edge that can be grouped into the face This rule is shown in figure 17; as one can see, it is fairly complex.

The first four CEs of the rule in figure 17 find the active KSAR, the parent element, the model label of the parent element, and the focus element, respectively. The fifth CE finds a candidate to group into the parent This CE makes sure that the candidate is on the same level and panel as the focus element and that it has not been grouped into the parent already. The rest of the CEs are present just to get data that is needed in the right side of the rule. This CE group includes the sixth CE that matches a WME which holds a homogeneous *transformation matrix*. The transformation matrix is defined to transform the focus element's label element so that it is compatible with the candidate's label element.

When the rule fires, the collinearity between the candidate and a transformed version of the focus element is computed. If this value is greater than a threshold, the candidate is grouped into the parent element. This grouping is done by finding the next empty slot of the parent's child list and inserting the candidate's id number. The new edge also contributes evidence that the parent's label is correct. Notice that if the candidate element doesn't meet the criteria to be grouped, then nothing in the working memory is changed, and refraction prevents the rule from firing again with the same instantiation.

In our example, any edge that has one of the labels I, J, K, or Q is a can-

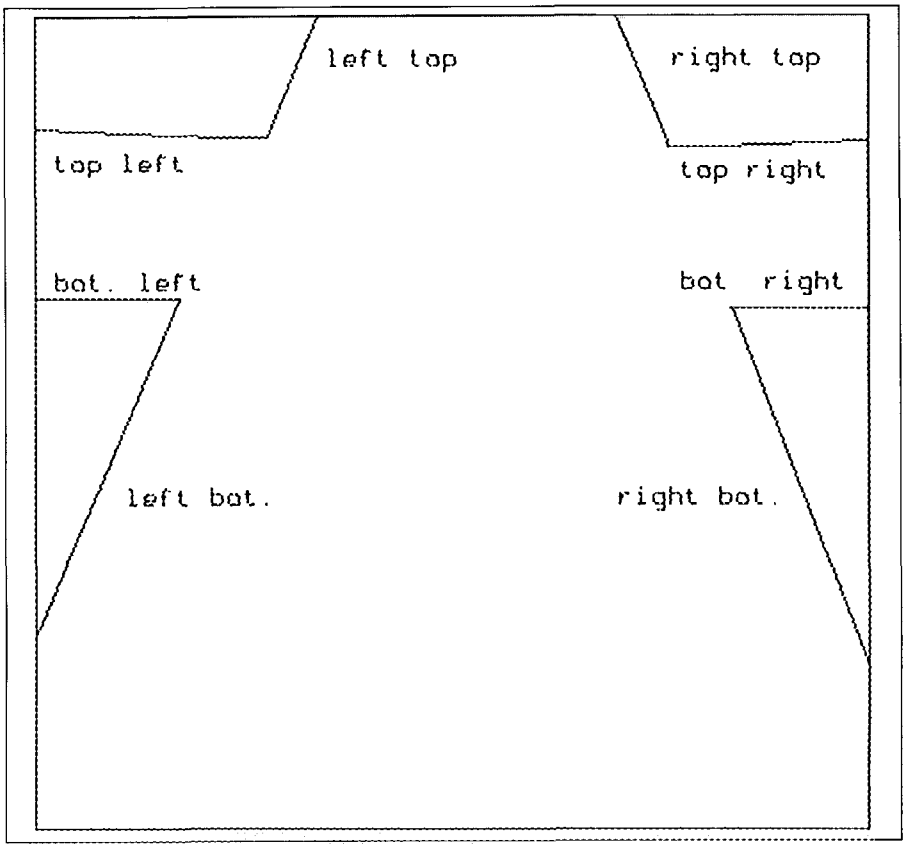
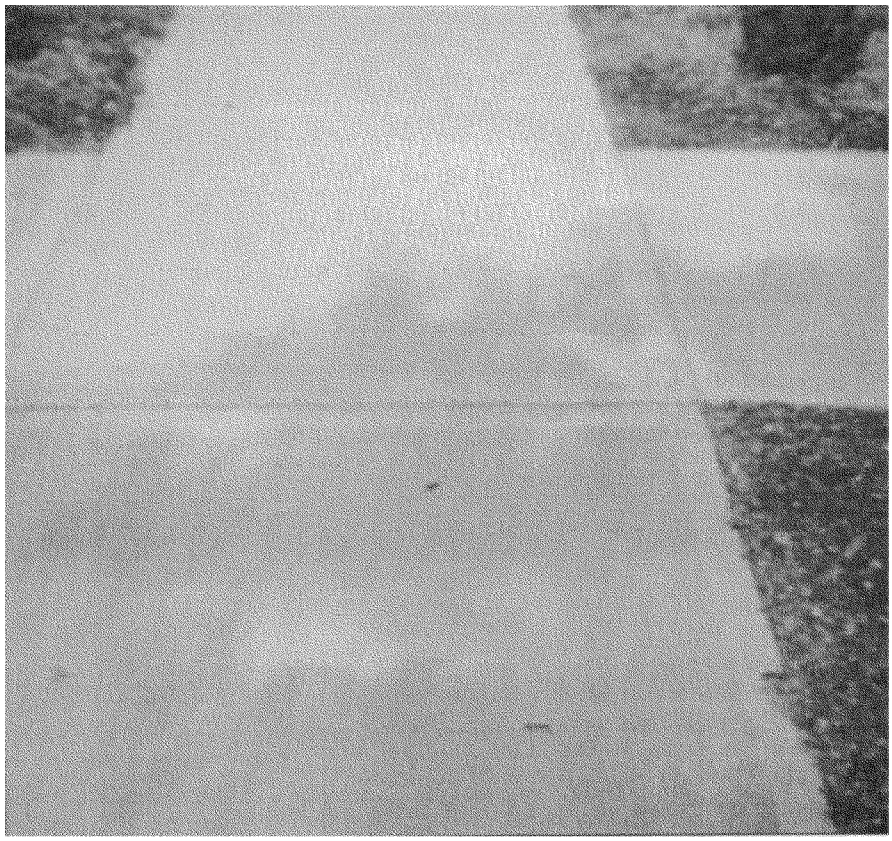


Figure 18 (Above) Line Drawing of the Expected Scene with Edges Labeled
Figure 19 (Below). A Sidewalk Image Used for Illustrating PSEIKI Processing.



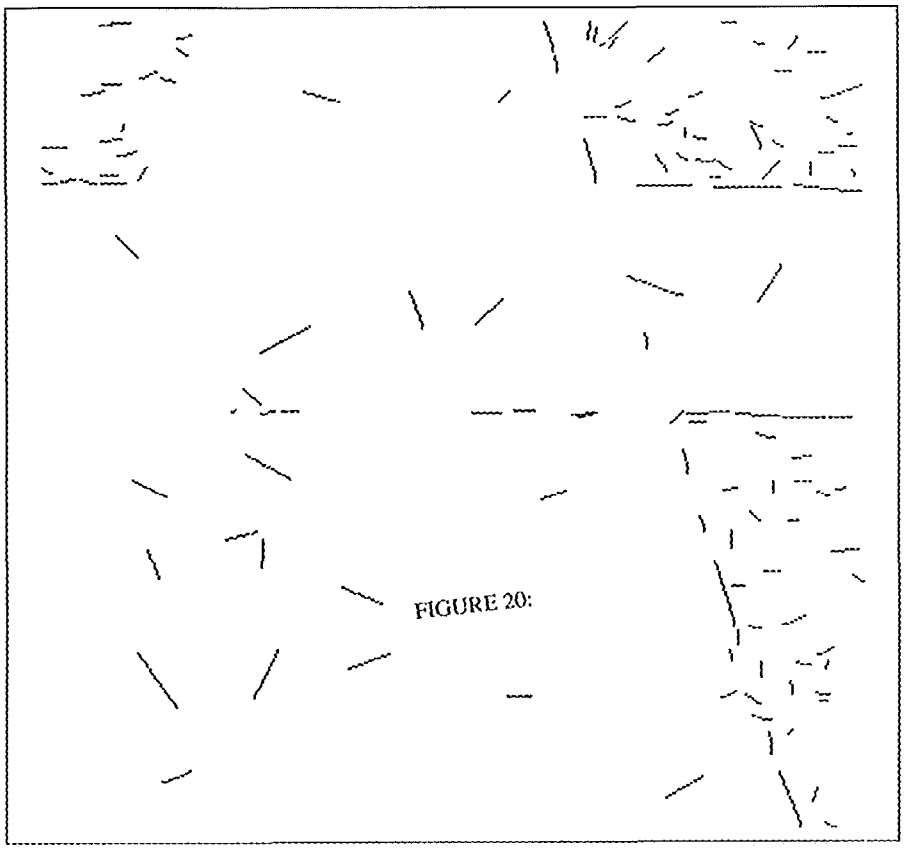
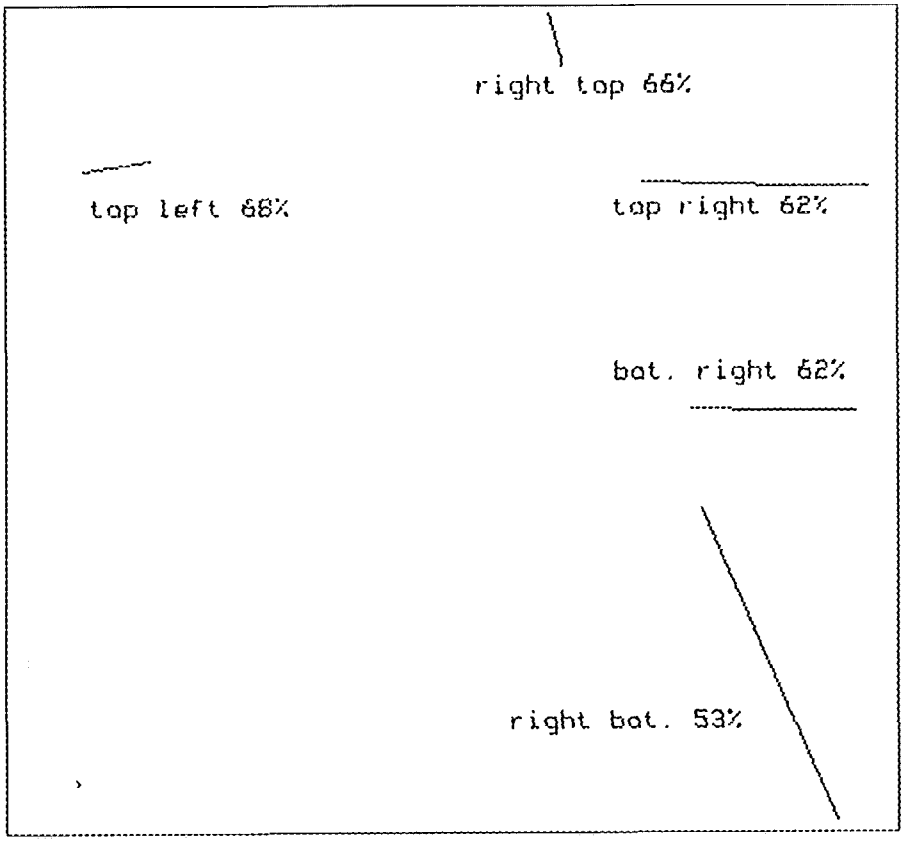


Figure 20 (Above) Input to PSEIKI from the Preprocessor.
Figure 21 (Below) Output of PSEIKI Displaying Detected Edges, Their Labels, and Associated Belief Values.



didate to be grouped with edge E_9 into face F_{20} Edges E_{10}, E_{11}, E_{16}, and E_{17} meet this criterion. Thus, any edge that is collinear with the transformed version of the focus element is grouped into the parent. If all but E_{11} were collinear with the transformed E_9, then the children of F_{20} would be the edges E_9, E_{10}, E_{16}, and E_{17}.

Experimental Results

PSEIKI was run on a number of images typical of what would be seen by a sidewalk-navigating mobile robot with downward-slanted cameras. For one such run, figure 18 shows the edges representing the expected scene and figure 19 the actual observed image. Note that there is significant misregistration between the expected scene and the observed image; two of the major edges in the expected scene, in the lower left, are missing entirely in the observed image. The reader should also note the presence of shadow edges in figure 19. The output of the preprocessor is shown in figure 20.

The final result produced by PSEIKI consists of labels, with associated belief values, attached to entities at the edge and higher levels on the data panel of the blackboard. For the example in figures 18-20, if we select from the scene level (the highest blackboard level) the entity with the maximum belief and then retain from the lower blackboard levels only those groups which correspond to this scene entity, at the edge level we obtain the result shown in figure 21. This figure shows the edges and their associated labels in the scene interpretation that PSEIKI found most plausible. In line with the earlier discussion, the percentage values associated with a label indicate the belief that PSEIKI has in the correctness of the label. For example, PSEIKI has a belief of .53 that the lower right edge is the right-bottom edge from the expected scene, implying that at a belief level of .47, PSEIKI either believes that the label is not right-bottom or that PSEIKI is simply ignorant about the matter of labeling the edge in question.

Concluding Remarks

If our aim was limited to labeling the edges in simple-looking images, such

as the one in figure 18, the reader might accuse us of creating the AI equivalent of the Monty Python skit where the characters hunt sparrows with rocket launchers However, note that what we have accomplished so far is only a stepping stone and is intended to give us the expertise for incorporating more complex spatiotemporal context in solving image-understanding problems. Of course, the following question remains: Does our solution scale up for more complex images? We believe that by reasoning in a hierarchical space, we have gained substantial leverage over the computational complexity involved. Admittedly, we are not yet in a position to present any formal expressions for the time complexity functions associated with PSEIKI, but then we are not aware of any having been derived previously for blackboard processing in general.

We hope we have provided the reader a sense of how evidence is accumulated in PSEIKI from both the expected scene information and any internal geometric consistencies that might exist in the image data. Notwithstanding the fact that no single large-scale implementation of any uncertainty formalism exists where researchers did not have to take massive liberties with the underlying assumptions, our use of Dempster's rule is not entirely beyond reproach because we did not address the requirement that evidence sources be independent before being combined. Loosely said, the independence requirement states that the evidence from one source not depend on the evidences from other sources; however, it is often difficult to formalize this notion and develop computationally feasible tests for its validity Yet, given the importance of the issue, this area is active with research; the reader is referred to Dubois and Prade (1985, 1986), Huntsberger and Jayaramamurthy (1987), Kyburg (1987), Smets (1986), and Yen (1986) for further discussions on the subject.

Acknowledgments

We would like to thank Lynn Garn, team chief of the Image Understanding and Artificial Intelligence Group and member of the Army Center for Night Vision and Electro-Optics, and Tim Williams, also of the Army Center, for many fruitful discussions that educated us on the limitations of the existing methodologies for automatic target recognition. We would also like to thank Richard Coe for his work on the design and development of PSEIKI's debugging software

Appendix

In this appendix, we give a short review of some relevant terms from the D-S theory of evidence accumulation. For a detailed presentation, see Shafer (1976)

In a random experiment, the frame of discernment (FOD), Θ, is the set of all possible outcomes For example, if we roll a die, Θ is equal to the set of possibilities "the number showing is i," where $1 \le i \le 6$; therefore, Θ can be set equal to the set {1,2,3,4,5,6} The $2^{|\Theta|}$ subsets of Θ are called propositions, and the set of all the propositions is denoted by 2^{Θ} In the die example, the proposition "the number showing is even" is represented by the set {2,4,6}

In the D-S theory, probability masses are assigned to propositions, meaning to some of the sets in 2^{Θ}; therein lies a major departure of this theory from the Bayesian formalism, in which probability masses must be assigned to the individual elements of Θ These probability masses must add up to one, and the probability mass assigned to Θ represents ignorance The interpretation to be given to the probability mass assigned to a subset of Θ is that the mass is free to move to any element of the subset; this interpretation agrees with the probability mass assigned to Θ representing ignorance because this mass can move to any element of the entire FOD When a source of evidence assigns probability masses to the propositions discerned by Θ, the resulting function is called a basic probability assignment (bpa) Formally, *bpa* is function $m: 2^{\Theta} \rightarrow [0,1]$, where

$$0\,0 \le m(\bullet) \le 1\,0, \quad m(\varnothing) = 0 \text{ and } \sum_{x \subseteq \theta} m(X) = 1\,0$$

A belief function, Bel(X), over Θ is defined by

$$Bel(X) = \sum_{Y \subseteq X} m(Y)$$

In other words, our belief in a proposition X is the sum of probability masses assigned to all the propositions implied by X

Dempster's rule of combination states that given two bpa's, $m_1(\bullet)$ and $m_2(\bullet)$ corresponding to two independent sources of evidence, we can combine them to yield a new bpa $m(\bullet)$ using

$$m(X) = m_1 \oplus m_2 = K \sum_{X_1 X_2} m_1(X_1) m_2(X_2)$$

$$x_1 \cap x_2 = X$$

where

$$K^{-1} = 1 - \sum_{X_1 X_2} m_1(X_1)\, m_2(X_2)$$

$$x_1 \cap x_2 = \varnothing$$

This formula is commonly called Dempster's rule or Dempster's orthogonal sum

References

Andress, K M., and Kak, A C 1987 A Production System Environment for Integrating Knowledge with Vision Data In Proceedings of the 1987 AAAI Workshop on Spatial Reasoning and Multi-Sensor Integration, 1-12. Los Altos, Calif.: Morgan Kaufmann.

Andress, K M , and Kak, A C 1988 The PSEIKI Report—Version 2, Technical Report, TR-EE-88-9, School of Electrical Engineering, Purdue Univ

Barnard, S T 1983 Interpreting Perspective Images *Artificial Intelligence* 21: 435-462

Barnett, J A 1981. Computational Methods for a Mathematical Theory of Evidence. In Proceedings of the Seventh International Joint Conference on Artificial Intelligence, 868-875. Los Altos, Calif.: Morgan Kaufmann

Barrow, H G , and Tenenbaum, J M 1981 Interpreting Line Drawings as Three-Dimensional Surfaces *Artificial Intelligence* 17: 75-116

Binford, T O ; Levitt, T S.; and Mann, W. B. 1987 Bayesian Inference in Model-Based Machine Vision. In Proceedings of the AAAI Workshop on Uncertainty in Artificial Intelligence, 86-92 Los Altos, Calif. Morgan Kaufmann

Brooks, R A. 1981 Symbolic Reasoning among 3-D Models and 2-D Images *Artificial Intelligence* 17: 285-348

Brownston, L.; Farrell, R ; Kant, E ; and Martin, N 1985 *Programming Expert Systems in OPS5* Reading, Mass : Addison-Wesley

Davis, L. S., and Hwang, S S V 1985. The SIGMA Image Understanding System. In Proceedings of the IEEE Workshop on Computer Vision, Representation, and Control, 19-26 Washington D C : IEEE Computer Society

Dubois, D , and Prade, H 1985 Combination and Propagation of Uncertainty with Belief Functions In Proceedings of the Ninth International Joint Conference on Artificial Intelligence, 111-113 Los Altos, Calif.: Morgan Kaufmann

Dubois, D , and Prade, H. 1986 Set-Theoretic Operations on Bodies of Disjunctive or Conjunctive Evidence In Proceedings of the 1986 Conference of the North Ameri-

can Fuzzy Information Processing Society, 107-124 Columbia, S.C.: Univ of South Carolina

Erman, L D ; Hayes-Roth, F ; Lesser, V. R ; and Reddy, D R 1980 The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty. *Computing Surveys* 12: 213-253

Gordon, J., and Shortliffe, E H 1985 A Method for Managing Evidential Reasoning in a Hierarchical Hypothesis Space *Artificial Intelligence* 26: 323-357

Hanson, A R , and Riseman, E M 1978 VISIONS: A Computer System for Interpreting Scenes In *Computer Vision Systems*, eds A R Hanson and E M Riseman, 303-333 New York: Academic.

Hayes-Roth, B 1985 A Blackboard Architecture for Control. *Artificial Intelligence* 26: 251-321

Huntsberger, T. L , and Jayaramamurthy, S N. 1987 A Framework for Multi-Sensor Fusion in the Presence of Uncertainty. In Proceedings of the 1987 AAAI Workshop on Spatial Reasoning and Multi-Sensor Fusion, 345-350 Los Altos, Calif : Morgan Kaufmann.

Kak, A. C ; Roberts, B A ; Andress, K M ; and Cromwell, R. L. 1987 Experiments in the Integration of World Knowledge with Sensory Information for Mobile Robots In Proceedings of the IEEE International Conference on Robotics Automation, vol 2 , 734-741 Washington D C : IEEE Computer Society

Kuipers, B 1977 Representing Knowledge of Large-Scale Space, Technical Report, AI-TR-418, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Kyburg, H E , Jr 1987 Bayesian and Non-Bayesian Evidential Updating *Artificial Intelligence* 31 : 271-293

Levitt, T S ; Lawton, D T ; Chelberg, D M.; and Nelson, P C 1987 Qualitative Navigation In Proceedings of the DARPA Image Understanding Workshop, 447-465 Los Altos, Calif : Morgan Kaufmann

McKeown, D. M., Jr ; Harvey, W A., Jr ; and McDermott, J 1985 Rule-Based Interpretation of Aerial Imagery In *IEEE Transactions on Pattern Analysis and Machine Intelligence* 7(5): 570-585

Nagao, M., and Matsuyama, T 1980 *A Structural Analysis of Complex Aerial Photographs* New York: Plenum

Nazif, A. M , and Levine, M D. 1984 Low-Level Segmentation: An Expert System. In *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6(5): 555-577

Nii, H. P 1986a Blackboard Systems, Part One: The Blackboard Model of Problem Solving and the Evolution of the Blackboard Architectures. *AI Magazine* 7(3): 38-53.

Nii, H P 1986b. Blackboard Systems, Part Two: Blackboard Application Systems and a Knowledge Engineering Perspective *AI Magazine* 7(4): 82-106

Pearl, J 1986 Fusion, Propagation, and Structuring in Bayesian Networks *Artificial Intelligence* 29: 241-288

Reynolds, G ; Strahman, D ; Lehrer, N ; and Kitchen, L 1986 Plausible Reasoning and the Theory of Evidence, Technical Report, 86-11, Dept of Computer and Information Science, Univ of Massachusetts

Shafer, G. 1976 *A Mathematical Theory of Evidence* Princeton, N J.: Princeton University Press

Shafer, G , and Logan, R 1987 Implementing Dempster's Rule for Hierarchical Evidence *Artificial Intelligence* 33: 271-298

Smets, P 1986 Combining Non-Distinct Evidences In Proceedings of the 1986 Conference of the North American Fuzzy Information Processing Society, 544-548 Columbia, S C.: Univ of South Carolina

Yen, J. 1986 A Reasoning Model Based on an Extended Dempster-Shafer Theory In Proceedings of the Fifth National Conference on Artificial Intelligence, 125-131 Los Altos, Calif.: Morgan Kaufmann

Notes

1 We say a system is domain dependent when domain-specific knowledge is embedded in various components of the inference engine, such as the rules or the knowledge sources PSEIKI is domain independent in this sense; the context information that PSEIKI uses is encoded entirely in the form of a line drawing of the expected scene

2. We refer to this mapping throughout the article

3 Note that these two panels correspond to the observability and picture graphs in ACRONYM

4. Note that in the following discussion, the elements on the model panel have capital letters as subscripts, and the elements on the data panel have numeric subscripts.

5 In the appendix, we define some of the relevant terms of the Dempster-Shafer formalism for evidence accumulation

6 The scale factor is provided to limit the amount of disconfirming evidence generated by small edges that might be the result of noise

7 The labels shown in figure 9 are intended only for the purpose of explanation In actual practice, even for simple imagery, the initial label map can be much more chaotic, depending on the extent to which an image is degraded by noise and other artifacts.

8 If not already familiar with terms such as working memory and production memory, the reader is referred to Brownston et al (1985) for an exposition on the architecture of a production system. The OPS83 used for PSEIKI is a direct descendant of the OPS5 system described in Brownston et al (1985). Much more so than OPS5, OPS83 allows functions and procedures to coexist with rules and working memory elements

9 This does not imply that a data element can only participate in a single group An edge element is, for example, allowed to be in two or more groups if it is on the common boundary between them However, any edge element can serve as a seed for only one group. Therefore, an edge element that belongs to two or more groups can trigger the formation of only one of them; other edges would have to act as seeds for the other groups.