PERCEPTUAL STRATEGIES FOR PURPOSIVE VISION

Technical Note 117

September 1976

By: Thomas D. Garvey

Artificial Intelligence Center

The research reported in this paper was sponsored primarily by the Advanced Research Projects Agency of the Department of Defense under Contract No. DAH04-72-C-0005 with the U.S. Army Research Office.

ABSTRACT

This report describes a computer program that approaches perception as a problem-solving task. The system uses information about the appearances of objects, about their interrelationships, and about available sensors to produce a plan for locating specified objects in images of room scenes. The strategies produced allow for cost-effective processing by utilizing "cheap" features in the early stages, reserving more complex operations for later in the process, when the content has been sufficiently restricted.

The general strategy paradigm used by the system is to acquire image samples expected to belong to the target object; validate the hypothesis that the acquired samples do belong to the target; and finally to bound the image of the object by outlining it in a display picture.

Sensors used by the system include a vidicon with three color filters and a master-scanned, laser-rangefinder capable of producing a "range image." In addition, the range-finder measures the reflectivity at the laser wavelength, at each post, producing a gray-scale image in perfect registration with the range image. The primitive attributes of brightness, line, saturation, height, and local surface orientation at specified range locations can be computed from these images. These attributes represent the new data available to the system for recognizing and identifying objects. Object descriptions are provided initially by indicating the object to the system, and allowing the system to measure attributes. Other data, such as typical object relationships, are provided interactively by the user.

When required to locate an object, the system computes a distinguishing features representation that will serve to separate parts of the target from those of other objects. These distinguishing features are combined into a strategy represented as a planning graph. This graph contains optimal subgoals for achieving the goal of locating the object. An execution routine selects the "best" subgoal, executes it, rates its effect, and selects the next best goal, continuing with the process until either the object is located, or there are no options remaining.

This approach offers several contributions to perception research. By capitalizing on the goal-directed aspects of the problem, the system is able to select relevant information from a mass of irrelevant data. The system is able to organize its processing in such a way as to optimize the use of sensor data. By generating strategies when needed, the program allows for the easy introduction of new objects and new sensors. The system allows for the logical introduction of new information deriving algorithms.

CONTENTS

FIGURES

TABLES

I INTRODUCTION

A. Overview

Imagine a not uncommon situation; you are driving home after a pleasant day at the beach. The air turns a little chilly, and you suddenly wonder whether you remembered to bring your sweater back from the beach. Happily, however, you stop the car, turn around, spy the beach bag, and recognize the small piece of green cloth sticking out of the top as your sweater, and put it on.

Let us look at this simple scenario in more detail. First, your initial goal was to locate your sweater. Second, you knew that if your sweater was in the car, it was most likely in the beach bag. So, rather than waste time searching throughout the car, you first looked for the beach bag; this became a subgoal for the solution of the first goal. Finally, when you found the beach bag, you recognized the sweater, merely on the basis of a bit of green cloth. The fact that the little piece of cloth was sufficient was due to the limited context of the possible items in the beach bag. In that context, the green cloth distinguished the sweater from all other objects, and provided a satisfactory basis for recognition. And, since there was no great risk in being mistaken, that was sufficient.

In this thesis we explore an approach for generating perceptual strategies which capitalize on the purposive aspects of perception. In particular, we report experimental results with a computer based system that relies heavily on a priori knowledge to find specified objects in relatively complex real-world environments. Objects are shown to the system by using a cursor to indicate examples in a displayed image. The indicated object is then automatically characterized by histograms of local surface attributes such as hue and orientation and by spatial relationships among the surfaces. The system can use this knowledge about the makeup of the current pictorial domain and knowledge of its available picture processing techniques to automatically formulate a strategy for finding the object. The resulting program is run on representative scenes and should errors materialize, debugged interactively.

The system is oriented toward a verification approach to scene understanding. It seeks particular objects, using a description of its target. Most of its effort is spent verifying or rejecting hypotheses that the area of the image currently under consideration belongs to that target object. This approach differs from the usual one where entities in the scene are first described and identifications are then made from the descriptions.

Since it is often possible to reject or reinforce hypotheses about object identifications on the basis of local attributes, the system relies more on these, and less on global descriptions such as shape or structured descriptions. Instead, the system is based on the premise that in real scenes there exist easy ways of "seeing" things. The many contextual relations among real-world objects, together with multisensory (i.e., range and color) data, suggest that a combination of perceptual cues can be found that would enable the system to easily locate and identify a given object. For example, to locate a tabletop, it is usually sufficient to find a horizontal surface 2-1/2 feet above the floor, since these features distinguish tables from other objects. And, to find a telephone, it is usually easier to first find the table, and then to find the telephone as a small compact mass of black points on the table, instead of attempting the much harder task of finding a telephone solely on the basis of its shape (the principal basis in past scene-analysis systems) using no contextual aids. However, even when such complicated features as shape are required, if the context is suitably limited, a simple distinguishing measure related to that feature may suffice to resolve remaining recognition ambiguities. Thus, the ratio of maximum height to base surface area is sufficient to distinguish the shape of a telephone from that of other black objects normally found on a desk top (e.g., loose-leaf binders). This approach to perception is termed, "vision by distinguishing features."

These example strategies are ad hoc and highly context dependent. We believe that the effective use of such domain specific knowledge is essential for machine perception. Of course, the most successful existing scene analysis programs also rely heavily on ad hoc heuristics (see, for example, Yakimovsky and Feldman[1*] or Bajcsy and Lieberman[2] The distinction between these programs and the approach we advocate lies in generality. Programs based on a large amount of ad hoc knowledge were thought to be "cheats" because the knowledge was inflexibly embedded. Thus, minor modifications of the object or environment often necessitated major reprogramming.

A system must instead be supplied with the higher level knowledge and problem-solving capabilities needed to choose for itself the most effective way of finding any object in any context, utilizing all currently available knowledge about the world. Specifically, the computer must use its knowledge to select features of the desired object that are both distinguishing and easy to see. The utility of such

---

* References are listed at the end of the thesis.

"distinguishing features" is critically dependent on what is known at the current stage of analysis. Hence, it is unreasonable to preprogram these recognition strategies, except in the simplest and most static environments.

This approach clearly requires a fairly complete description of most items likely to be found in the scene. Obviously, if the attributes being used to distinguish the desired object from others are shared by an unknown object, the system is liable to err. An attempt is made to circumvent this problem by checking other attributes of the object in order to validate the hypothesis that it is the desired object, but the basic problem is that any two items that share a given set of attribute values must have attributes outside that set in order to be distinguished. By increasing the set of measured attributes, or by increasing the resolution of the measurements, we can increase the total number of objects that can be distinguished. Alternatively, by restricting the context to a small set of objects, we can improve the effectiveness of a limited set of coarse attribute measurements.

B. Background

Until recently, research on machine perception has concentrated on the exhaustive description of simple geometric environments appearing in a black-and-white television image[3]. The "blocks" world was originally chosen to simplify the problems of feature extraction and thus to accelerate the development of high-level perception programs. Unfortunately, it may also have impeded progress by diverting investigators to difficult but less important problems attributable to its very sterility. First, the lack of variety and of meaningful contextual constraints in this artificial world limited the extent to which perceptual strategies could capitalize on task goals and expectations; exhaustive descriptions were invariably sought. Moreover, the unavailability of attributes like color and texture led to preoccupation with shape descriptions as the principal basis for recognition, which, in turn, diverted a disproportionate amount of effort to the problem of extracting perfect line drawings of homogeneous objects viewed under homogeneous, shadowless illumination.

The ad hoc, primarily bottom-up, techniques developed under these constraints were unsatisfying because they would be overwhelmed by the irrelevant detail found in virtually any other type of scene. Highly tuned edge followers, for example, suffocate in a world with texture. Moreover, the elaborate topological descriptions that conveniently characterized simple polyhedra are inappropriate for many real-world objects (e.g., trees, lakes) even if they could be obtained with reasonable effort.

The thesis confronts several important perceptual issues, such as information overload and generality of strategies, that arise in real world scenes. Instead of simplifying the environment, our system copes with the complexity of real-world scenes by capitalizing upon their natural redundancy of descriptive features and contextual constraints.

We have chosen room scenes as a research domain, because they are naturally complex, semantically rich, and experimentally convenient. Many of the lessons initially learned in this domain should prove applicable generally to complex scenes with adequate semantic richness; for example, the analysis of outdoor scenes and earth resource satellite imagery.

Our objective is to find designated objects in uncluttered rooms like that shown in Figure 1 using brightness, color, and range as sensory inputs. The system will show its comprehension by outlining the indicated object in a displayed image of the scene.

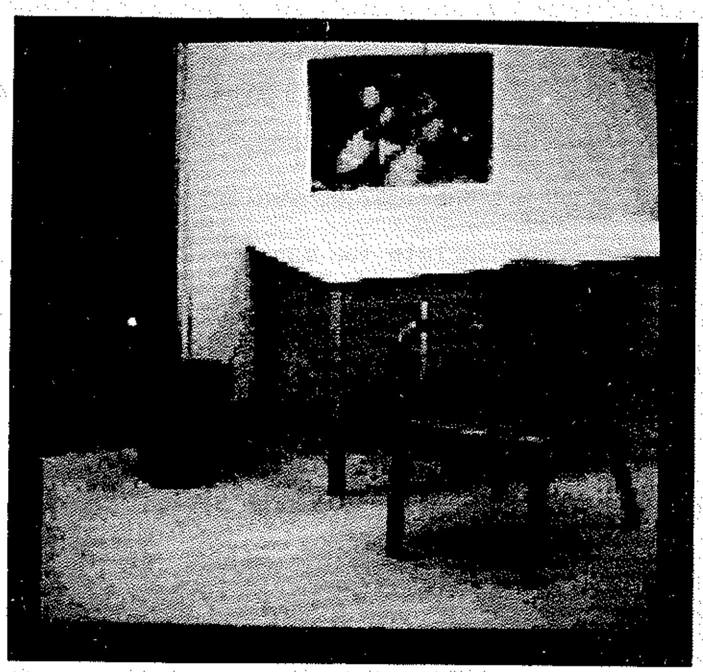


Figure 1. A Simple Room Scene

Our system is roughly partitioned into an image-processing section consisting of a set of primitive perceptual operators (collectively known as ISIS) and a perceptual executive. The executive knows about many kinds of things; among them are perceptual properties of objects, their contextual relations, the operating characteristics of the available primitive operators, and partial results generated by an ongoing analysis. The task of the executive is to combine these various kinds of information in order to define the next step of analysis; that is, to decide which operator should be next applied to which portion of the scene.

C. Design Considerations

The above point of view suggested the following design considerations and implementations for our perceptual system.

1. Perception as Problem Solving

Perception should be construed as a problem-solving process; the system must utilize its knowledge of the current real-world environment and of its own perceptual capabilities to plan where and how to look for a specified object. Specifically, the computer must use its knowledge to select features of the desired object that are both distinguishing and easy to see. The utility of such "distinguishing features" is critically dependent on what is known at the current stage of analysis. Hence, it is unreasonable to preprogram these recognition strategies, except in the simplest and most static environments. Moreover, a system that can plan its own strategy has inherent generality; it should be able to function in any environment for which its knowledge base and perceptual primitives are adequate.

Our system contains a strategy generating program which plans to a sufficient level of detail to begin execution. Fine details are planned when required by the execution of a coarse planning step.

2. Sequential Decision Paradigm

It is usually unnecessary to examine all features of an object to arrive at a confident recognition hypothesis. Simple descriptive attributes (e.g., color, size) rapidly limit consideration to a small subset of the objects possible in a given environmental context. Remaining ambiguities can then be resolved using distinguishing components of more complex attributes (e.g., shape).

The system generates predicates which check sufficient local features to either identify the surface in question, or to reduce the set of possible ambiguous objects to the point that a small number of global distinguishing tests can disambiguate.

3. Multiple Sensors

The likelihood of finding suitable surface attributes for distinguishing a given object increases with the number of independent sensory modalities. Furthermore, simple discrimination in each of several sensory modalities should be a cheaper, more reliable alternative to using more detailed descriptions in a single modality. It is substantially easier to interpret a set of brightness values observed through several filters as a color, or a set of range values as a surface orientation, than to interpret a set of grey-scale intensities as a shape or texture.

Our programs make use of color information (represented as hue and saturation components), range information (from which is extracted height and local surface orientation), and intensity data (grey-level data taken though a neutral density filter).

4. Goal Directed Feature Extraction

A key problem in perceiving by distinguishing features is to extract the features reliably. One lesson that has been learned in a decade of vision research is that good feature extraction is quite difficult to perform adequately bottom-up for use by a knowledge-based interpretative process. However, if the system is integrated so that recognition strategies are based on knowledge of which features are easy to extract in a given context, then low-level routines could concentrate on extracting those specific features.

Global surface features like shape are the hardest to extract because the surface must first be segmented from the rest of the scene. Fortunately, with multisensory data, in a context that is sufficiently restricted, we can rely heavily on local surface attributes known to be both homogeneous over the goal object and distinguished from those of other nearby surfaces previously found or anticipated in the environment.

The planner uses its knowledge of the costs of the various available perceptual procedures to organize its search to examine cheap attributes first, later resorting to more complex and expensive tests.

One problem we, as well as anyone else who is interested in cost-effective planning, must face is the question of how to estimate the pertinent parameters well enough to allow us to create effective strategies. While it is usually the case that any estimates are likely to improve the system's performance over the situation where there is no information at all, clearly, the better the the system's knowledge, the better it will work. We currently model those processes that are either simple enough, or bounded well enough to make the models meaningful. The models do predict the performance of many of the system modules well enough that the system gets real benefit from them.

5. Incremental Acquisition of Knowledge

The substantial amount of ad hoc world knowledge required to plan perceptual strategies is most reasonably acquired in an incremental fashion. Insufficient information for a task is usually indicated by a strategy failure. In this case, it is generally easy to determine the problem and add enough new facts to allow the system to replan a correct distinguishing strategy.

The perceptual system is embedded in ISIS (Appendix 2), which gives the user full interactive capabilities for examining the attributes of a point or a region and to add or modify information in appropriate data structures.

D. Perceptual Strategy

The writing of a perceptual system to locate objects in scenes required the implementation of certain high-level capabilities. First, the system required the ability to locate single, isolated sufaces of objects. Then, from known surfaces and known relationships among object parts, the system needed to be able to locate other parts. Finally, we required the ability to use information about one object to find other objects.

The search for an object proceeds in three phases: acquisition, validation, and outlining (or bounding). During acquisition, the multisensory image is sampled for characteristic surface attributes of the desired object. If a sample satisfying all criteria is found, a sequence of top-down validation tests determines whether the acquired sample does, in fact, belong to the desired object or to another object with similar surface characteristics. The validated object samples are used as starting points for an outlining routine.

Chapter I INTRODUCTION

1. Acquisition

The objective in acquisition is to rapidly disqualify obviously irrelevant areas of the scene so that subsequent processing can concentrate on the most promising locations for the goal object. This objective is realized by sampling the scene at a density determined by the goal object's size, maximum range, and least favorable viewing orientation. Attributes of the goal object are tested at each sample in an order likely to cause the earliest disqualification of irrelevant points. The selection of attributes suitable for acquisition is based on such considerations as:

(1) Criteriality--the attribute should invariably be associated with the object (e.g., the floor is always horizontal).

(2) Distinguishability--the attributes should not also be characteristic of other objects expected in that context.

(3) Measurability--the attribute should be reliably obtained from simple, localized processing of sensory data.

Acquisition can also be done indirectly, as in the example of locating the telephone by first locating the table, or in the case of finding objects with multiple parts. That is, clues for locating the target object, or for narrowing the search space, can be provided by some other previously located object with a known relationship to the target. In other words, the acquisition of some part or object may itself involve finding some other object or part.

2. Validation

The second phase of the strategy scheme, validation, provides increased confidence that the acquired samples belong to the target. This happens in two ways. First, the target object may be ambiguous with respect to other known objects. In this case, the system attempts to distinguish the target from other members of this set. Second, the system then tries to raise its confidence that the acquired points belong to the correct object by making other tests. This phase of validation is necessary for two reasons: First, acquisition attempts to distinguish samples of the target from samples of other known objects, but, some attempt must be made to guard against unknown objects. Second, since objects are characterized rather coarsely, there may be points on some other, known object that are not reflected in the model, but that satisfy the acquisition tests. Unexpected spurious points of this type are filtered out by this second validation stage.

Validation tests can also include more global features too costly to test at each sample: e.g., global attributes (such as size and shape) of the surface surrounding the acquisition sample. We rely on the consensus of several crude, individually tolerant tests related to the surface attributes, rather than actually extracting a detailed description of any particular one. For example, the ratio of perimeter squared to area, or the ratio of length to width, might suffice as representations of shape. Many global attributes of a surface can be grossly tested even without finding a detailed boundary. For instance, a surface can be disqualified as too small merely by failing to find a sufficient number of appropriate samples within a gross window centered on the acquisition sample.

3. Outlining

To complete its objective, the system must finally display an outline of the object it has found. Several techniques can be used to determine this outline from previously validated kernel points. The simplest is a region growth process which applies a predicate to neighboring points, keeping those that pass and recursively applying the predicate to their neighbors.

The system can also utilize high-level knowledge about a scene to rapidly obtain approximate boundaries of located objects without exhaustive low-level operations (such as edge following or region growing followed by line fitting). An entity in the scene is completely mapped onto a rigid object model by supplying the model's identity and a sufficient set of correspondences between model points and image points to constrain position and orientation. Detailed boundaries are then obtained by projecting the model onto the scene and validating the predicted edges against image data.

E. Example

We will use the example of finding the telephone in the image shown in Figure 2 to illustrate the process outlined above.

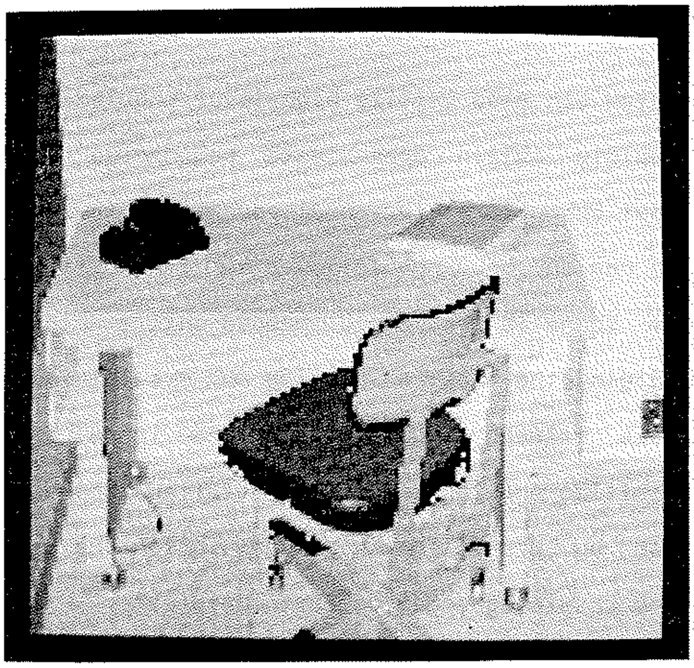


Figure 2. Image of Telephone on Table

In formulating its strategy, the system realizes that, due to its small size, it is undesirable to search for the telephone directly. Instead, it decides on an indirect approach, and chooses to find the table first. To locate the table, it searches the scene with a predicate that rejects all points except those belonging to a horizontal surface within a given height range. In Figure 3, we see the initially acquired table points, which, after validation, are used as the starting points for an outlining routine. This program scans outward to find the table's edges, which are then used to constrain the location of a projected horizontal rectangle, as shown in Figure 4. The table now provides a "window" Figure 5. in which the telephone can be distinguished as a small black region. The acquired samples are shown in Figure 6. Finally, the region corresponding to the telephone is extracted by a region growing technique for the final results shown in Figure 7.

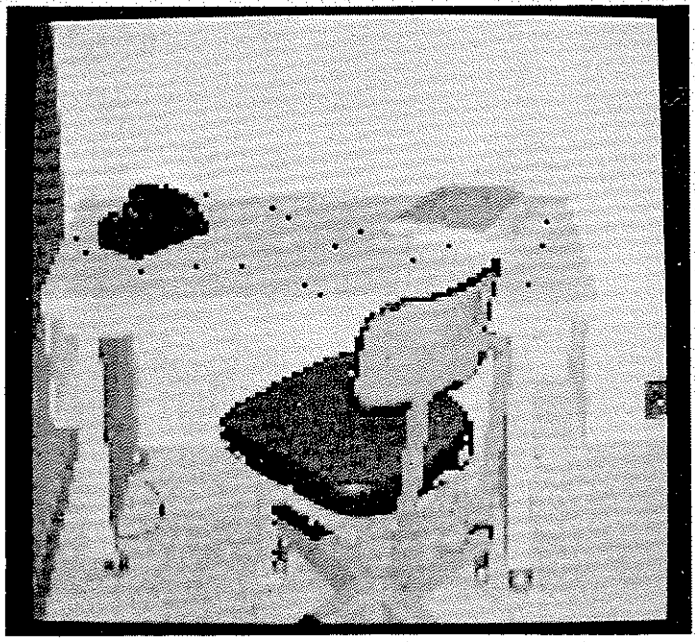


Figure 3. Initially Acquired Table Points

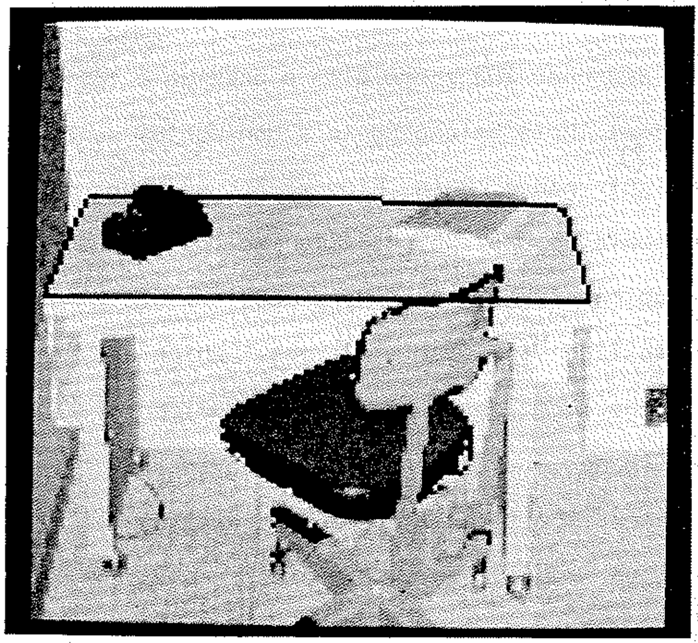


Figure 4. Table Boundary Located with Projected Rectangle

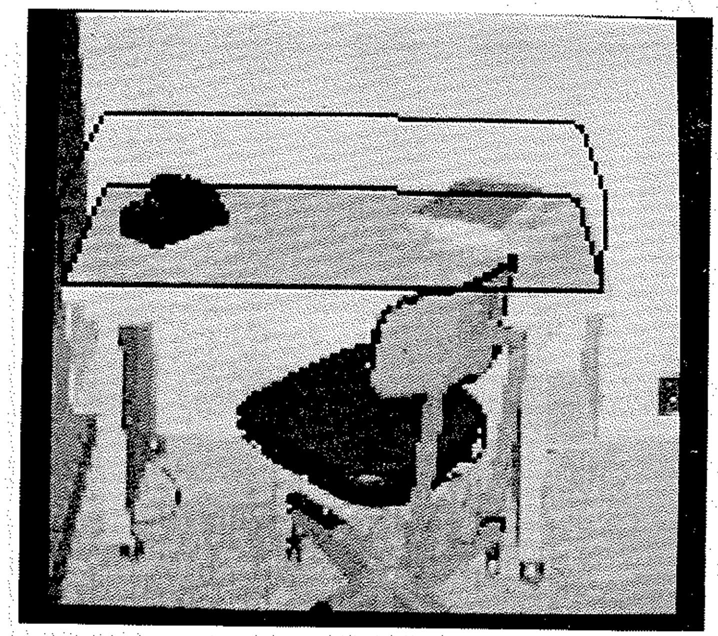


Figure 5. Window Determined by Table

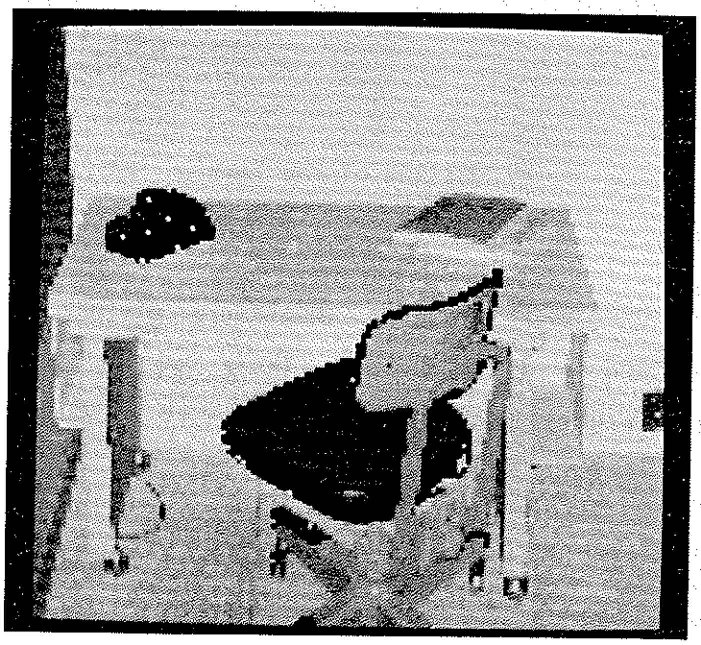


Figure 6. Telephone Acquisition Samples

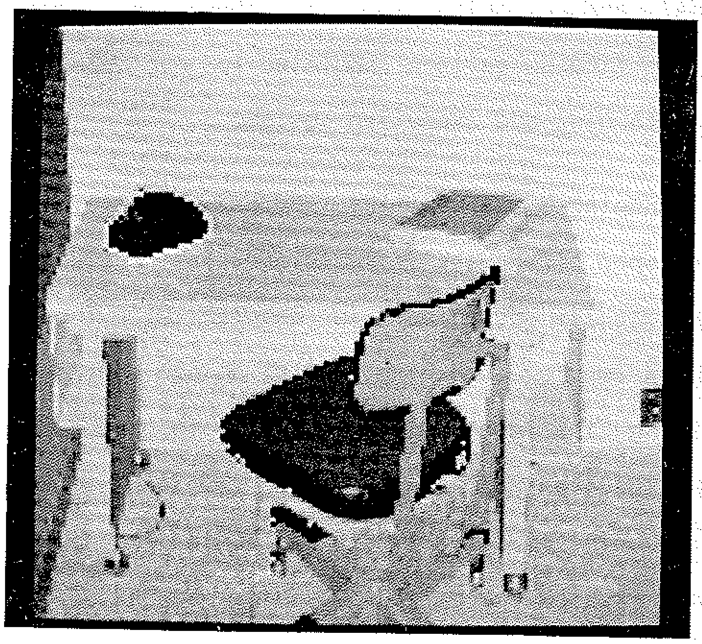


Figure 7. Final Telephone Outline

F. Outline of the Thesis

The remainder of this thesis is divided into five chapters, and three appendices. These are briefly outlined below as an aid to the reader.

The interactive system ISIS is described in some detail in Chapter 2. This system was developed as part of this research to facilitate experimentation with visual strategies. We discuss how ISIS aided in the development of interactive strategies, and describe some of these in detail. These interactive strategies were emulated in our creation of the automatic planning and execution system.

In Chapter 3 we describe several program modules that were developed after their usefulness was indicated by the manual strategies described in Chapter 2. These modules include programs for acquiring initial object points, programs for verifying the identity of these points, and programs for then using these points to locate the boundary of the object. We describe both the actual operation of the programs, and how we estimate various parameters such as the expected cost of execution, and the reliability of operation of the module. These parameters are used by the planning system.

The planning and execution system is described in Chapter 4. The planner creates a strategy whose operators consist of instances of the modules described in Chapter 3. This plan is passed to the plan executor which makes a cost effective decision about which goal should be executed, and executes it. It then determines whether the main goal was satisfied in the course of executing the goal, and if not chooses the next goal to be attacked. This sequence is continued until the main goal is satisfied, or no other options remain. Although the planner was developed principally for use by this system, we feel that its applicability extends to other areas of Artificial Intelligence.

Chapter 5 describes several experiments with the system. Objects are indicated to the system in several images, and it then locates instances of these items in other images. We discuss some errors and omissions, and their implications.

In Chapter 6, we discuss the results we were striving for, and the results we actually achieved. We point out the strengths and weaknesses of the approach, and indicate where we think it fits into an integrated visual system. We discuss our contributions, and indicate possible future directions for research.

The appendices contain information and computations that, while germane to the work, seemed out of place in the main body of the text. Appendix 1 contains the derivation of cost and confidence used in the evaluation of sequential detector strategies. Appendix 2 provides a rather complete description of the ISIS system functions; it is actually intended as a manual for the system. Appendix 3 explores the use of information gained from a failure to aid in future planning and execution.

II ISIS AND INTERACTIVE STRATEGY GENERATION

A. Introduction

We have chosen as our objective the task of finding specified objects in relatively complex real-world scenes. Ideally, one would like to program a computer to find an unfamiliar object as one would instruct a person, by providing a crude description of the desired object. A tabletop, for example, might be described as a "horizontal, buff-colored, planar region, 2-1/2 feet above the floor." The computer could demonstrate comprehension by outlining instances of the described object in a displayed image. The programmer could then refine the description empirically to correct errors in the computer's interpretation.

Communications on this level with a machine is hampered because the machine does not share an "understanding" with the human of basic descriptive concepts like "horizontal," "buff," "above," and "planar." Such concepts are often difficult to express in natural language, much less in conventional programming languages; primitive pictorial concepts are better conveyed in terms of pictorial examples.

The difficulties incurred in the symbolic description of pictorial concepts suggest a pragmatic approach to representation. Although we may not know, in general, how to describe shapes and textures, such concepts can often be adequately discriminated by simple feature extraction operators when the context is suitably limited.

Representations of this sort are founded solely on empiricism; we have no theoretical understanding of perception from which they can be deduced. Our aim has therefore been to create an interactive system to facilitate the empirical design of scene analysis programs. The system allows an experimenter to describe basic perceptual concepts to the computer using pictorial examples. The examples are designated graphically by circling areas of a displayed scene with a cursor. Given a pictorial example, a representation can be empirically constructed by trying pictorial operators in order of increasing complexity until the example is sufficiently distinguished from previously determined representations. Pictorially defined concepts constitute a shared vocabulary that can be used to describe objects or to define more complex concepts. This paradigm is intended to elevate the user above the details of handcoded algorithms, allowing him instead to concentrate on the construction of descriptions and strategies.

B. Defining Pictorial Representations Interactively

1. Overview of System

The system consists basically of an image file, a library of primitive feature extraction functions (see Appendix 2), a means for applying selected primitive operators to graphically designated areas of a scene, a semantic data structure for accumulating concept definitions, and an iconic data structure for retaining pictorial examples of those concepts. Images are stored in sample arrays, with each sample characterized by brightness measured through red, green, blue, and neutral density filters and by a range measurement. (Range is measured by a time-of-flight laser range finder.) The feature extraction operators transform the raw data into more meaningful dimensions such as hue, saturation, and local surface orientation.

Iconic regions are represented as an explicit list of picture samples and/or as a list of vertices describing a closed polygonal boundary. A bounding rectangle enclosing the region is also stored for each region. Efficient routines exist for determining whether a given picture element is contained within a bounded region, for obtaining a set of samples over a bounded region, and for fitting a boundary around a set of samples (see Appendix 2).

The primitive functions and symbolic data structure reside within the interactive environment of INTERLISP on a PDP-10. The raw data arrays, and support routines for graphics, file handling, coordinate transformations and such reside in a FORTRAN environment accessible as a lower fork through the TENEX operating system. The graphics hardware consists of a monochromatic ADAGE vector display with a "mouse" and a RAMTEK color display with trackball*.

Briefly, the system's role can be explained as follows. To describe concepts in terms of the computer's primitive functions, a human must know (or be able to determine easily) what those primitive functions can distinguish. Our system provides the trainer with tools to choose primitive functions empirically. The trainer can circle regions of the displayed image and obtain from the system average or extreme numerical values for local operators such as height, hue, saturation, and surface normal. Thus, by applying operators to examples and counterexamples of pictorial concepts in the image, the trainer can discover directly which operators provide sufficient

---

* The mouse and trackball both allow us to manipulate a cursor on the screen. We will use the term, mouse, to refer this cursor.

discrimination. The trainer can try out a proposed description by requesting the system to indicate all parts of the displayed scene that correspond to the description.

Let us now illustrate, with portions of actual protocols, how an investigator might use the system to define pictorial concepts in a simple room scene such as Figure 1. (The examples presume a slight familiarity with LISP notation, but should be largely self explanatory.)

2. Description of Objects by Pictorial Example

The trainer's immediate objective in a new domain is to describe the principal surfaces of interest (e.g., wall, floor, tabletop in a room scene). The object will be to allow the system to develop characterizations for the objects that can be used to describe the surface characteristics. The user can also interact with the system to provide relational data (such as the TELEPHONE is on the TABLE). This process of interacting with the system is used to develop descriptions for use by the automatic portions of the system.

Using the cursor, the trainer begins by outlining the tabletop surface in Figure 1. Tabletop is now a symbolically accessible concept, defined by pictorial example. From the defined region, the system selects random samples (as shown in Figure 8), and uses these for computing primitive features.

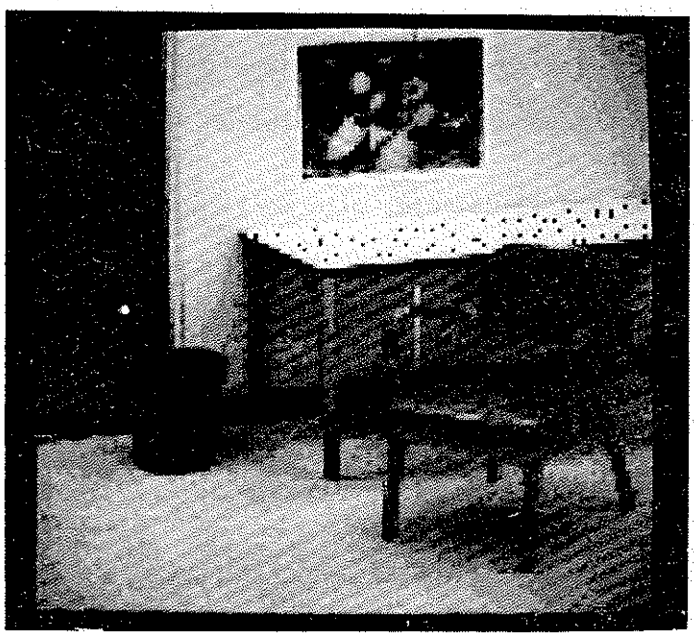


Figure 8. Sample Points Used to Compute Table Properties

The trainer can obtain the values of these primitive features by using the functions in Appendix 2. For instance:

```
        (HEIGHT TABLETOP)    -- user input shown capitalized
        2.4                  -- machine responds with average
                                height of the encircled surface,
                                shown here in feet.

        (HEIGHTEXT TABLETOP)
        (2.43 2.48)          -- (minimum height, maximum height)

        (HUE TABLETOP)
        31.5                 -- average hue over the tabletop,
                                expressed as the angle (in
                                degrees) measured from pure red
```

around the periphery of a color triangle (see Appendix 2).

The values of these surface attributes are computed by applying primitive operators to the randomly sampled image points. The attribute values are converted into histograms of probabilities. These histograms allow us to determine the probability that an object will have an attribute value within a given range. By examining these probabilities, it is possible to determine exactly which tests are characteristic of the object. Used in conjunction with histograms for other objects, it is possible to make up distinguishing tests.

3. Testing Descriptions

The trainer uses the information supplied by the system to select primitive functions that best distinguish the tabletop from other surfaces in the scene. The adequacy of a set of primitive functions can be tested empirically by requesting the system to intensify all points in the displayed image that satisfy a proposed description. More specifically, the description is formulated as a LISP predicate using the primitive operators and the predicate functions in Appendix 2. The routine FILTER applies this predicate to randomly sampled points in the image. All points satisfying the predicate will be displayed and retained in order to test subsequent refinements to the description. To illustrate, if the trainer wished to determine whether tabletops were adequately distinguished by their color and height, he could request the system to filter random image points according to the predicate:

```
(AND (COLORP (HUEEXT TABLETOP)
             (HUE X))
     (LIMITP (HEIGHTEXT TABLETOP)
             (HEIGHT X)))
```

This predicate translates roughly as: "Find all points, X, such that the hue and height of X are within the range of colors and heights observed on the tabletop." When the system applied this predicate to the random image samples displayed in Figure 9 the display in Figure 10 was generated. The left half of this display shows the location in the scene of those points from the original set having appropriate height and hue. The corresponding color coordinates of these selected points are displayed in the color triangle to the right. Note that all of the selected points fall within the wedge delimiting the range of hues observed in the tabletop example. Figure 10 shows that not only did the predicate select all of the points on the tabletop, it also accepted some points at table height on the buff colored wall and brown door. Thus, the definition proposed for the tabletop is not sufficiently discriminating to select only the tabletop, and must be augmented with a

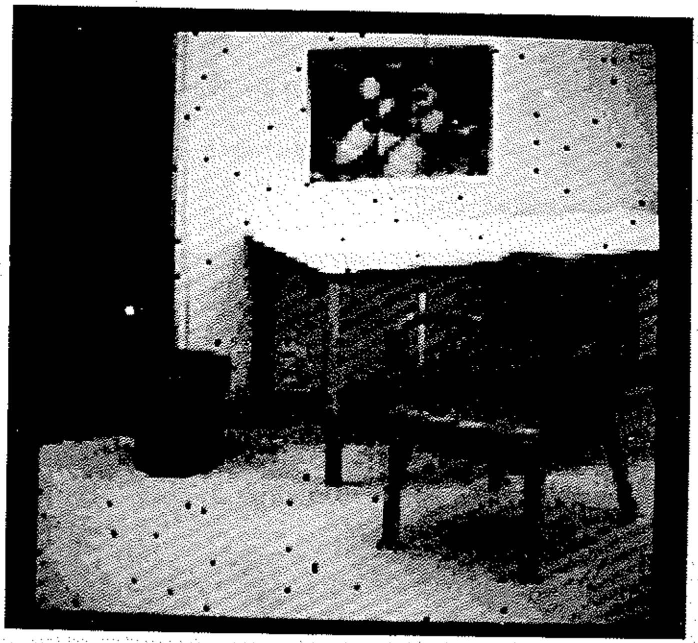


Figure 9. Random Image Samples

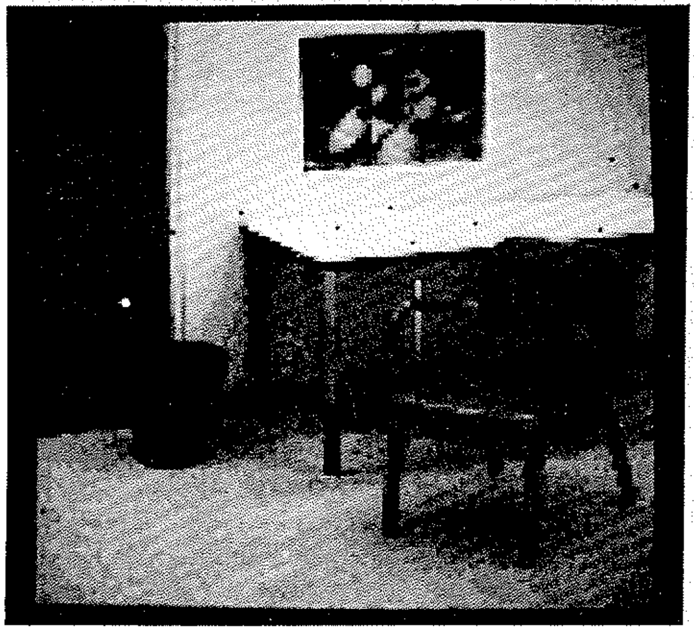


Figure 10. Random Samples with Hue and Height of Tabletop

constraint on color saturation or on surface orientation. Figure 11 shows the result of further filtering the points of correct height and color with a predicate that eliminates those with improper orientation,

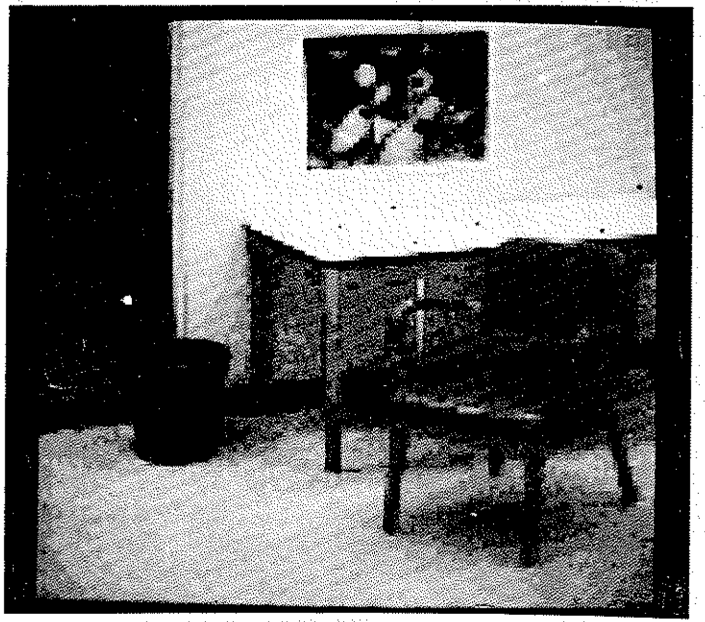


Figure 11. Tabletop Samples Further Distinguished by Orientation

```
(LIMITP (ORIENTEXT TABLETOP) (ORIENT X)).
```

Figure 11 confirms that color, height, and orientation comprise a description that distinguishes the tabletop, at least in this particular scene. The corresponding three-clause predicate

```
                    (AND (COLORP (HUEEXT TABLETOP) (HUE X))
                         (LIMITP (HEIGHTEXT  TABLETOP) (HEIGHT
X))
```

```
                    (LIMITP (ORIENTEXT TABLETOP) (ORIENT
X)))
```

expresses the description in procedural form. This expression can now be used as a predicate to locate tabletop points in other images. It can also appear as a clause in more complex predicates, such as procedures for finding tables or objects on tables.

Since all of the primitive operators appearing in the tabletop predicate accept a variety of arguments, the predicate can be used not only to filter image samples, but also to test whether objects or regions qualify as tabletops. In particular, the trainer can have the system apply the predicate to every surface description in the data base, in order to determine whether tabletop is adequately distinguished from objects that appeared in other scenes.

In addition to using extreme values of attributes for descriptions, it is often useful, when more than a few objects must be distinguished from each other, to use descriptions that are more restrictive. Many objects tend to have overlapping ranges of attribute values. For example, the saturation over a region of tabletop may vary from .2 to .4 while the saturation of wall may vary from .3 to .5. Obviously, saturation alone may not always be adequate to distinguish tabletop samples from wall samples. This becomes a problem when we are trying to locate samples of one object or the other in an image where both may be found. A more promising approach is to refine the description in such a way that only a part of the full range of attribute values for the object is used to locate initial samples, and then use these initial samples as starting points for a region grower. Obviously, this would not describe all the points on the surface, but would hopefully match at least a few (although, without correlated measurements, it is possible to inadvertantly create descriptions made up of partial attribute value ranges that failed to match any point on the surface). The region grower would examine points adjacent to these points collecting those that had attribute values within the full range of allowable values for the object.

This partial attribute range is then carefully chosen so as to both distinguish the object from other objects, and also be characteristic of the object we are describing. We may then need to augment the description by the addition of other attributes in order that the description be truly distinguishing.

Since the attribute value functions mentioned previously and in Appendix 2 are not adequate for determining these partial attributes ranges, we have included the capability to display the probability histograms mentioned previously. One-dimensional histograms allow us to examine the complete spectrum of attribute values for an object. In addition, we can display two-dimensional histograms (scatter plots) of selected attribute pairs (e.g., hue vs. saturation). Points in the histograms can be selected and image points with corresponding attribute values can be displayed. The histogram display functions also allow us to examine windows in the picture (in addition to single points), and check them for appropriate statistical distributions of attribute values. This would be useful for experimenting with certain classes of texture operators. By comparing histograms from the object we are describing with those from other objects, we have been able to interactively determine good distinguishing descriptions for all the objects in our scenes.

4. Description of Attributes

Instead of describing surfaces pictorially, the trainer can first define common surface attributes, such as the color "buff" and the orientation "horizontal," with which to describe surfaces symbolically. These attribute labels can usually be defined by a characteristic range of values for a single primitive operator. The range of values may be determined empirically by applying the operator to pictorial examples of the attribute. Horizontal, for instance, might be defined as any surface whose normal is within 5 degrees of the Z axis (based on values of the ORIENT operator obtained when pointing at image samples on the floor and tabletop). Attribute labels may also be extensionally defined in terms of the observable properties of pictorially described objects. For example, buff can be defined for the system as the color of the tabletop:

NEWATTRIBUTE (BUFF)

defined as: (HUEEXT TABLETOP)

Attribute descriptions are tested and refined empirically in the same manner as object descriptions. For instance, the definition of horizontal would be tested by requesting the system to intensify points randomly sampled in the image that satisfied the predicate:

(HORIZONTAL X).

The symbol HORIZONTAL evaluates to the range of orientation (-5 5). Hence the predicate will accept all points, X, with orientations in this allowed range. Figure 12 validates this definition of horizontal, since all intensified points are on surfaces such as the floor, tabletop, and chairseat normally thought of as horizontal. This description is again preserved in procedural form as the function definition of HORIZONTAL.

5. Symbolic Description of Objects

Defined attributes can be used symbolically to describe new objects. For example, floor can now be described to the system in terms of buff and horizontal as follows:

NEWOBJECT (FLOOR)

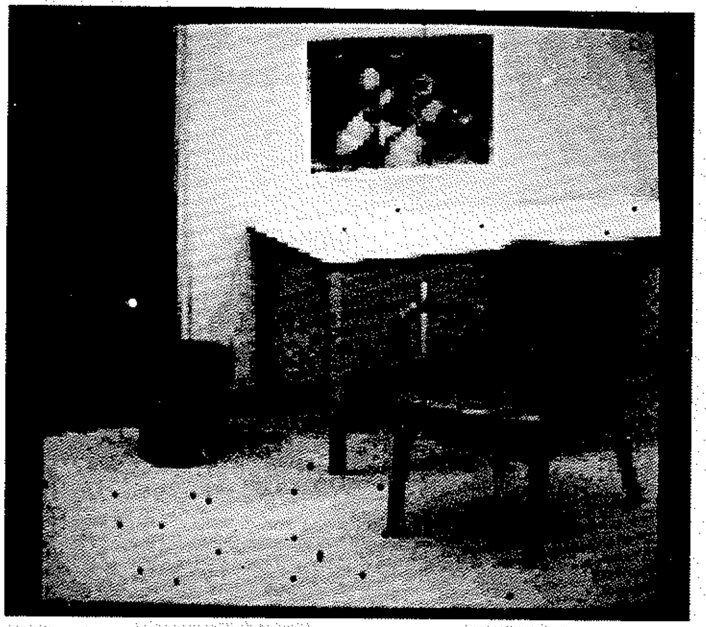


Figure 12. Horizontal Image Samples

```
(ORIENTEXT FLOOR HORIZONTAL)
(BUFF FLOOR)
(HEIGHTEXT FLOOR (-0.1 0.2))
```

NEWOBJECT first establishes floor as an object in the data base. The primitive functions then deposit their second argument as corresponding property-list values of floor. The height range (-0.1 0.2) was determined empirically by applying the HEIGHT operator to a few manually designated floor samples. Since no pictorial example of floor is provided, any attributes not explicitly supplied are indeterminate. The function call, (BRIGHT FLOOR), for example, would return as its value the atom UNDEF (standing for undefined).

The description of a floor can be tested in the usual way by applying the following predicate to randomly sampled image points:

```
                (AND
                     (HORIZONAL X)
                     (BUFF X)
                     (LIMITP  (HEIGHTEXT  FLOOR)  (HEIGHT
X))).
```

The satisfactory result is displayed in Figure 13.

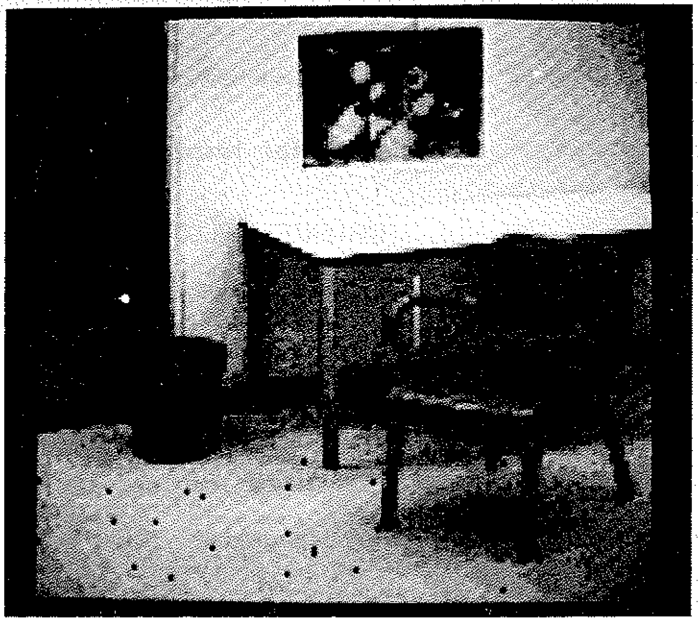


Figure 13. Floor Image Samples

6. Description of Complex Objects

We use the representations developed for simple objects and surfaces to form articulated descriptions of complex objects. A chair, for example, might be described at the top level as an object with two principal parts, seat and back, in the spatial relationship (ABOVE BACK SEAT). The attributes of seat and back in correct spatial juxtaposition adequately distinguish chairs from other office furniture (e.g., tables), and may do so more effectively than attributes of parts such as legs. At the next level of description the sub-objects, seat and back, might be adequately described in terms of their distinguishing surface attributes. A seat, for instance, might be characterized simply as a horizontal surface in a suitable height range, perhaps with specified colors and linear dimensions.

Articulated descriptions can be developed empirically in the same way as surface descriptions; the proposed chair description, for example, might be tested by writing a program to find chairs, based on predicates which find the parts: e.g.,

```
(PROG (X Y)
   A (COND
       ((SETQ X (SEAT))
        (COND ((SETQ Y
                  (SCAN-ABOVE X (QUOTE BACK)))
               (RETURN (LIST X Y)))
              (T (GO A))))
```

```
          (T (RETURN NIL))))
```

This example LISP function attempts to find the chair by first finding the seat, and then looking in a window above the seat region for samples that look like the back. If it finds both, it returns the two regions. If it finds samples that look like the seat, but cannot find the back above the seat, it continues looking for the seat. Finally, if it cannot find both regions, it returns NIL. For efficiency in searching the image, parts like the seat that have distinguishing attributes should be sought first, in order to narrow the search rapidly to likely areas of the scene.

C. Interactive Strategies

It is not always possible to specify a simple predicate that will select all image samples belonging to a desired object and to no others; in such cases, specialized object-finding programs may be needed to determine the actual boundaries of an object in the image.

The interactive features of our system provide unique experimental capabilities for rapidly developing specialized scene analysis procedures. The system offers researchers the following advantages:

(1) Complete compatibility with INTERLISP, facilitating the development and integration of specialized scene analysis algorithms.

(2) Symbolic access to a large library of primitive feature extraction operators as well as to a wide variety of utility programs (e.g., coordinate transformations, display routines) underlying the implementation of the top level system.

(3) An extensible vocabulary of semantic concepts that can be used to express perceptual strategies.

(4) An interactive graphics package specifically designed for scene analysis experimentation.

(5) An efficient and versatile iconic data structure for regions and edges, complete with editing and display routines.

The system has been used extensively for constructing specialized programs that find and outline the various objects in a domain of uncluttered office scenes. In this work, the system's basic repertoire of primitive operators was augmented to include several higher-level functions. The following examples demonstrate the use of ISIS for interactively generating strategies and techniques for finding several objects in room scenes. Later chapters will describe the implementation of programs to emulate these techniques automatically.

1. Development of a Door-Finding Program

One might assume from previous examples that, to find the door, the system need only apply a suitable predicate to extract door samples in the image, and then fit a boundary if desired. Door samples, however are not always easily distinguished from those of other objects on the basis of local attributes. This fact was established empirically by circling a door in a test image and requesting the system to select random image points that fell within the ranges of hue, saturation, height, and surface orientation observed in that region. (Brightness was considered too variable to use as a criterion for acquisition.) Figure 14 shows that many irrelevant points met the criteria, including points on the chair back, points in the picture hanging on the wall, and even points on the portion of wall visible under the table. (The saturation of wall points under the table is unusually high because incident light is reflected from the brown floors. The saturation of some door points is reduced because of specular reflection.)

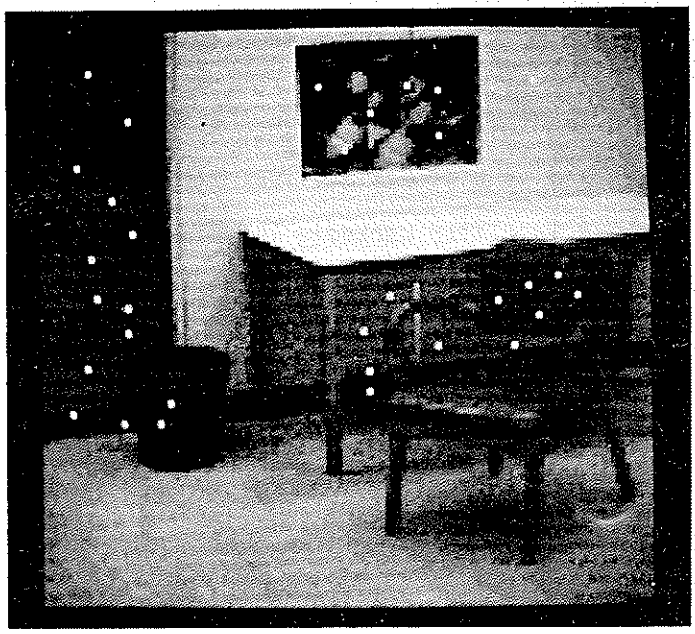


Figure 14. Random Image Samples with Local Attributes of Door

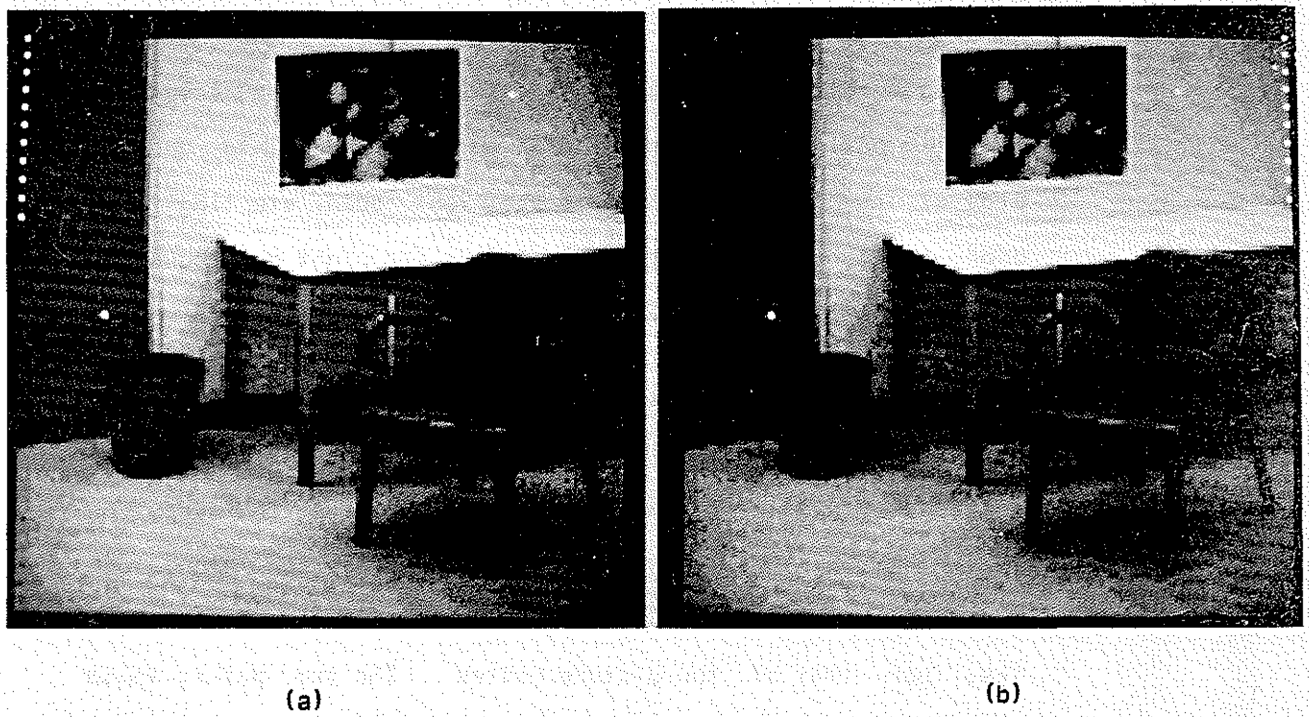
The vertical rectangle bounder (VRB) is a highly specialized program module that can utilize a priori knowledge to obtain an approximate door boundary rapidly, without recourse to edge-following or region-growing. All SRI office doors are known to be vertical, rectangular, plane surfaces, 3.5 feet in width, 7 feet high, and bounded below by the floor (height 0). The boundary of a door in an image is thus completely constrained given the world coordinates of a single point on either vertical edge and the surface orientation measured on the door's side of the edge*. The task of finding the door thus reduces to that of obtaining the two pieces of information, an edge location and a surface orientation required by the vertical rectangle bounder.

In the current image (Figure 15), observe that the height range above the tabletop and below the picture is occupied only by the door and the directly illuminated wall. These objects were observed to differ markedly in saturation, suggesting that a door edge could be located by scanning horizontally at this distinguished height, seeking a saturation discontinuity.

The above conjecture was confirmed with some brief experimentation: First we obtained numerical values of saturation using the primitive SAT. The saturation was observed to be 0.6 when pointing at a door sample and 0.25 when pointing at a wall sample. We then scanned a predicate

---

* Assuming the detected edge has world coodinates (X, Y, Z), the door boundary is a vertical rectangle with two corners at (X, Y, 0) and (X, Y, 7) and two other corners at the same heights but 3.5 feet horizontally displaced along the known plane of the door.



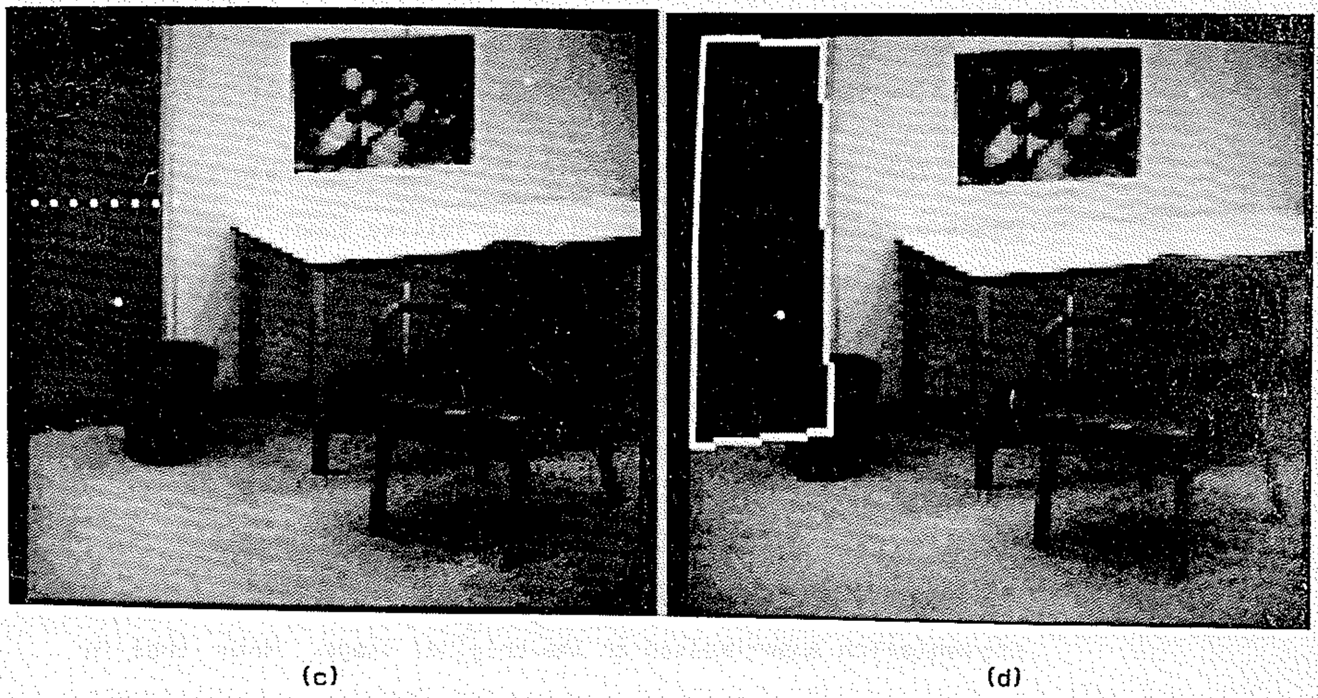


Figure 15. Operation of Door-Finding Program

horizontally across the image in the specified height range seeking a saturation jump of at least 0.2 (the scan locus was manually designated by pointing at suitable end points on each side of the image). The left to right scan, starting on a door, sample successfully terminated at the right door boundary where the required saturation discontinuity was first detected.

To determine a suitable scan line automatically, we first determined the actual height of the bottom edge of the picture frame (2.8 feet) by pointing in the image and executing (HEIGHT MOUSE). We then scanned down from the top along each side edge of the image, seeking the first sample with height less than that of the picture bottom. The two points found determine a suitable horizontal locus on which to seek a saturation discontinuity. The operation of this phase of the door-finding program is summarized in Figure 15(a-c).

The world coordinates (X, Y, Z) of the resulting edge point were then obtained using the primitive function XYZ. The surface orientation of the door was found by applying the primitive function SORQ to a door sample on the high saturation side of the discontinuity. Finally, the door boundary was computed from this information by the vertical rectangle bounding module.

The computed boundary was passed to a display module for presentation superimposed on the original image. Note that in the present scene the door extends above and to the left of the image borders. Hence the display module must clip the actual computed boundary to coincide with the visible portion of the door. Figure 15d shows the resulting boundary display. The total development time for this strategy including several false starts was about four hours. Strategies outlined below for finding the wall, picture, table and chair involved comparable effort.

2. Some Illustrative Object Finding Strategies

In this section, we briefly outline several additional object finding strategies developed empirically with our system.

a. Walls

The wall-finding strategy was developed by slightly modifying the door-finding strategy. A step decrease in saturation while scanning horizontally left to right marks the left edge of the wall. Unlike doors, however, the widths of wall sections are not known a priori. Hence the scan must continue seeking a step increase in saturation that marks the right edge of the wall. Given these two lateral extremes, and the known vertical extent, the VRB can infer the complete boundary of a section of wall.

b. Pictures

Since the height and width of pictures are both variable parameters, the VRB requires at least one edge point on all four borders to compute the picture boundary. The search for the border samples is constrained by the fact that pictures hang on walls. Hence the first step in picture finding is to call the wall finder. The program next scans in horizontally at nominal picture height from both the left and right wall edges, seeking discontinuities in hue or saturation. These discontinuities mark the left and right vertical picture borders respectively (an extent check is made to distinguish these edge points from those on the wall-door boundary). The horizontal top and bottom picture borders are found by scanning vertically down from the top of the image and up from table height in between the previously detected lateral extremes of the picture, seeking color discontinuities. The picture boundary is then a rectangle consisting of two vertical lines through the (X,Y) world coordinates of the left and right edge points, extending from the height (Z coordinate) of the bottom edge point to the height of the top edge point and the two horizontal lines connecting the tops and bottoms of these verticals.

c. Tabletop

The tabletop boundary is found using GROW to extend a region outward from points located on the tabletop by the predicate (TABLETOP X). The predicate used by the region grower is one composed of tests for points having attribute values within the extreme values for tabletop.

D. Discussion

The interactive style of programming encouraged by this system is in marked contrast to the awkwardness usually experienced in writing and debugging picture-processing routines; strategies that are trivial to describe to a human programmer invariably take weeks to translate into code using conventional programming systems. We feel that high-level graphical interaction is an invaluable tool for developing scene analysis procedures.

1. Generality

The programs and representations discussed in this thesis admittedly are ad hoc and highly context dependent; it is for this reason that they are so effective. These programs are based on a type of domain-specific knowledge that is difficult to utilize in a general-purpose scene analysis system, but very easy for a person to supply interactively.

Such highly specialized strategies can be criticized for their lack of generality. The door-finding program can be made to fail, for instance, by placing a tall object on the table and slightly shifting the point of view: the basic philosophy followed, though, is not to complicate a strategy or representation unnecessarily, by anticipating all possible contingencies. Should a program actually fail, we would then try to patch it by interactively experimenting with additional constraints. For example, if the door-finding program became confused by the edge of a tabletop object we could easily install an additional test that would check the vertical extent of a discontinuity before calling the bounding module. Alternatively, the door side of the discontinuity could be checked for proper hue, saturation, and surface orientation. As a final check, a mask can be used to find discontinuities in the image that confirm the boundaries predicted by the bounding module. It is as hard to guess which refinements will prove effective in overcoming a bug as it is to anticipate which bugs will materialize. An advantage of our interactive system is that it allows refinements to be added empirically when contingencies arise.

2. Primitive Functions

The ability to describe concepts and strategies interactively in a particular domain depends on the availability of appropriate primitive functions for that domain. In office scenes, there was heavy reliance on multisensory features (hue, surface orientation, height) obtained from color and range data to increase the likelihood of finding simple, distinguishing attributes for each object. These features would obviously be ineffectual for distinguishing objects in a black and white outdoor scene, where objects are distinguished principally by attributes such as shape and texture as well as by spatial relationships. We are currently implementing some additional primitive functions which will extract relevant features for representing shapes and textures.

The scene analysis literature abounds with descriptions of low-level operators (Rosenfeld[4] provides a fairly complete list). Unfortunately it is very difficult to anticipate which set of operators will provide adequate discrimination in a given domain. Given the interactive nature of the approach, it is more reasonable to start with a few likely operators and let the set grow with necessity; i.e., when the available operators are unable to distinguish examples of a new concept from examples of previously defined concepts, then additional operators are introduced. The ability to experiment interactively is an enormous help in selecting new operators and ultimately may help to establish a taxonomy of operator capabilities.

3. Distinguishing Features

We wish to emphasize that our use of distinguishing features to avoid detailed description should not be construed as a denigration of other work on complex representations. Assuredly, occasions will arise when, like a person, a machine will have to resort to detailed descriptions in order to discriminate among very similar objects. Our main contention is that in all but very complicated scenes, it is usually possible to find relatively simple representations to distinguish objects from one another. The advantages of a simple description include economy, flexibility, and most important, the ability to find complex objects (like trees) for which descriptively adequate representations are still unknown. Since all representations are incomplete at some level of detail, there should not be undue concern over using the simplest representation that provides adequate discrimination for the current task.

4. User Interface

During the design of the present system, considerable attention was paid to the man-machine interface. We wanted a system that was easy to learn and that would encourage people to experiment with a variety of descriptions and perceptual strategies. The following design features contribute to the operating simplicity of the system:

(1) The principal power of the system resides in a handful of versatile top-level functions listed in Appendix 2. These functions respond appropriately to any meaningful argument. For instance, an image sample, the cursor, a region, an object, and a defined color are all valid arguments to HUE.

(2) All functions default interactively; if the experimenter uses an unknown concept to describe a new object, the system will request a definition (in terms of the attributes of some known object or by graphical example) before allowing him to continue.

(3) Requests for input are stated concisely, often with a single prompt character, to spare the patience of experienced users. An inexperienced user can always request an explicit list of options by responding with a "?".

(4) The system maintains a continuity in the man-machine dialogue by interpreting underspecified commands in the context of recent processing. For instance, after requesting the system to find all "green" regions, the experimenter need only enter the function (HEIGHT) to obtain a distribution of heights for all green areas of the scene.

Amenities such as these make it possible for many visitors to use the system after only a brief demonstration.

5. Incremental Growth

We view our system as a prototype facility for use in developing spcialized scene-analysis programs in many application areas. The power of this facility can be expected to grow incrementally with use. The more primitive operators that have been programmed, the more likely one is to find distinguishing representations for new semantic attributes. Similarly, as more semantics are defined, it becomes easier to describe new objects. Finally, a large repertoire of object-finding strategies will frequently contain useful components of new strategies. In the office domain, a previously developed wall-finding program was utilized in a strategy for finding pictures that are constrained to hang on walls. Judging from experience in office scenes, when appropriate primitive operators, semantics, and procedures have been accumulated for a domain, programming in that domain becomes easy.

III MODULES FOR OBJECT FINDING STRATEGIES

A. Overview

In the preceding chapter we described a schemata for locating objects in images. Now we are interested in generating a system that can use this schemata to find objects automatically. This system should be successful in finding the target object* most of the time, and should be cost effective in its use of detectors. It is important also, that the system retain the flexibility required to allow for both the easy addition of objects to its data base and the extension of its capabilities through the addition of new sensors. This will not only provide robustness (by allowing the system a redundent set of sensors to choose from), but also allow (relatively) easy extension to other pictorial domains.

Creating a plan to find an object is analogous to writing

---

* The object being sought will often be referred to as the "target object", or simply the "target."

a program to accomplish the same task. A good programmer segments his code into modules, each of which performs a specific job, thus simplifying interfaces and control structures and allowing a straightforward description of the expected results of the module. For these same reasons, we have created a set of execution modules for use specifically by an object finding program. Simplified interfaces allow for easy specification of arguments and initial conditions required for a module to be executed. The simplified control structure makes it possible for a machine to generate programs based on schemata such as the one we have described. The simplified descriptions allow the system to understand the effects of executing a module. These modules are made up from primitive feature extraction functions, basic predicates, and a few higher level programs (such as have already been discussed). The modules are usually more complex than our primitive functions, yet simple enough not to be overly restricted in their applicability.

Just as a programmer has to be able to decide when it is appropriate to use a certain module, we also require the ability to decide which of several modules is most appropriate for a particular task. The first thing that must be specified is the structural embedding required to establish the preconditions for the module. Then, when several modules are applicable to a task, a way of choosing among the candidates is required.

The structural embedding required for the module is provided by planning modules. These are rules specifying sequences of actions for the solution of a goal, and are executed during planning to produce combinations of other planning and execution modules. They are similar to macros in a programming language that are expanded to produce source code which is passed back to the input scanner. They usually result in constructs which either force the execution of modules in a particular order, or require the system to eventually select the best order for execution. Most execution modules have built-in checks to ensure any preconditions are satisfied. For example, a program to operate on a particular set of points will check for these points before proceeding, and fail if they are not available.

When the system has the responsibility to select the best module for execution, it attempts to make a cost effective decision. We use a probabilistic model with three parameters in order to facilitate these decisions. The first two of these planning parameters model the efficacy of the module, the third estimates its cost, expressed as the expected processing time required to run the module.

When a module is executed, it reports either success or failure upon completion. This report is usually based upon some internal decision made by the module, and can be in error. The first two planning parameters, $P(T)$ and $P(\alpha|T)$, provide our model of the outcome of the execution of a module. The first, P(T), estimates the a priori likelihood that the module will report success in its task. The second, $P(\alpha|T)$, estimates the probability that the module is correct in reporting success. That is, the second parameter is the likelihood that the module had a good outcome* given that it reported success.

Obviously, the notion of successful termination and a good outcome are related to the goal. For example, a module which uses a predicate to filter the image for samples belonging to a particular object, will report success if it finds a sufficient number of samples satisfying the predicate. An estimate of the probability of this is provided by the first parameter. However, some of the samples selected by the predicate may well belong to other objects. Therefore, we need the second parameter to estimate the likelihood that a sufficient number of the accepted samples will belong to the desired object.

When a module reports failure, we assume that it did, indeed, fail, and it is not considered further. This is a useful simplification for our purposes, but it does overlook two points. The first was raised above; sometimes the module was successful even when it reported failure. We are making the implicit (and simplistic) assumption that, normally,

* We will define a "good outcome" of a module as the desired outcome.

$P(\alpha|F)$, the probability of a good outcome given failure of the module, is zero*. The second, and more serious, point is that even when a module fails, some useful information about the world could be gained from that fact. If the module is supposed to locate a certain feature in the image and fails, that does not necessarily mean that the feature is not present in the image, but it should, perhaps, lower the a priori likelihood of its being there. Appendix 3 examines a potential extension that would allow the exploitation of failure information.

In some cases, when the computation is sufficiently well bounded, it is possible to make very accurate and reliable estimates of the parameters. In other cases (for example, region growth), it is difficult to bound the computation, and so estimates may be based on expected computational bounds (such as the anticipated size of the final area in the case of region growth), or on estimates compiled from many runs of the module.

We would now like to describe specific techniques for accomplishing each of the the three main phases of our strategies, acquisition, validation, and bounding.

----
* Although, this is probably a reasonable assumption when applied to the case of a detector rejecting a sample, for higher level modules, it is probably a much better idea to allow the execution of a module to change the cumulative probability of a good outcome, $P(\alpha)$, and reserve judgement about success or failure to a higher level of the program.

Chapter III MODULES FOR OBJECT FINDING STRATEGIES

1. Acquisition Methods

The most straightforward way of acquiring samples of the object is to filter random picture samples with a predicate which rejects any point not matching the desired object. For example, the sampled scene may be filtered to retain only those samples within a certain height range characteristic of the target. Alternatively, the scene can be sampled along a locus to detect distinguishing boundary discontinuities. These predicates are generated automatically from pictorial examples. In addition, it is frequently possible to restrict the search area by the use of information about other objects or parts. As mentioned, knowing the location of the table can greatly aid in finding objects known to be supported by it.

2. Validation Methods

Acquired samples are validated by checking additional local characteristics that distinguish the desired object from others with common acquisition attributes. These additional attributes are examined sequentially, reducing the number of samples that must be examined by successive tests. Total cost is minimized by deferring computationally expensive procedures. When the cost of planning the optimal order exceeds the known cost of using all remaining attributes, samples are classified with respect to possible objects. Those samples which have a high enough classification probability of being the desired object are retained. Samples along a boundary can be further validated by testing the boundary length, strength, and orientation using suitable masks.

3. Bounding Methods

The final bounding phase is accomplished in various ways. Given a suitable geometric model of the object and a projective camera transform (or range data), a procedure can compute analytically the best global boundary consistent with detected edge points. The computed boundary can be projected onto the image and validated using extent masks as above. Procedural models are available for common surface types such as horizontal and vertical rectangles. In the absence of such models, a region may be grown around validated points; neighbors of validated samples which pass a chosen predicate (often the acquisition predicate) are collected, and their neighbors are then examined recursively. The set of accepted points form a region, for which a crude boundary can be obtained by computing a convex hull.

B. Execution Modules for Locating Surfaces

Most of the execution modules described here apply a previously created detector to a window of the image. In this section, we describe just the modules themselves; a subsequent section will be devoted to automatic detector generation. However, before we begin the discussion, we will require some definitions and notation.

A "window" is a subregion of the image (the initial window is usually the entire image). A detector (or a "simple detector") is a procedure that checks whether a sample's attribute value lies within a given contiguous range of attribute values. A "composite detector" is a conjunction of simple detectors. A "predicate" is the corresponding LISP program for a detector. A predicate returns T (true) if the value is within the range, and NIL (false) otherwise*.

A detector will be represented by the symbol, D. A sequence of detectors (i.e., a composite detector), $<D_1, \ldots, D_n>$, will be indicated as D^n. The probability of a

* The terms detector and predicate will often be used synonymously.

randomly chosen sample being accepted by D^n is $P(D^n)$. We will also represent the probability of a sample being part of object, O, as P(O). As a simple extension of this notation, $P(O,D^n)$ is the probability that a sample both belongs to O, and is accepted by D^n.

If we have the probability, p, of an event occurring and the probability, (1 - p), of it not occurring, then the expected number of times the event occurs in N trials has a binomial distribution. The expression for the probability of an event occurring precisely i times in N trials is just

$$P_i = C(N,i)p^i [1 - p]^{N-i},$$

where C(N,i) is the binomial coefficient,

$$\frac{N!}{i!(N-i)!}.$$

The probability of at least k occurrences of the event can be expressed as the sum of the individual probabilities, P_k through P_N, or,

$$P_{\geq k} = \mathrm{SUM}_{i=k}^{N}[C(N,i)p^i [1 - p]^{N-i}].$$

This can also be expressed as the complement of the probability of less than k occurrences. Or,

$$P_{\geq k} = 1 - \mathrm{SUM}[C(N,i)p^{i}\,[1 - p]^{N-i}]_{i=0}^{k-1}.$$

This latter form is more easily computed for the small values of k and large values of N that we will typically deal with. We can now describe the operation of the modules and the computation of their planning parameters.

Chapter III MODULES FOR OBJECT FINDING STRATEGIES

1. Acquisition Modules

Acquisition modules have the task of selecting object points within a window possibly containing several objects. We have two basic techniques for accomplishing this. We can filter the set using a predicate created especially for acquiring the target, or, when we have a starting point, we can scan across the image searching for points matching the target. FILTER-WINDOW is the main filtering routine; it requires the program, SAMPLE-WINDOW, to generate the set of samples to be filtered. In addition, there is a set of scanners for searching along specific line segments.

a. SAMPLE-WINDOW

For a given object, predicted image, and detector for that object, it is possible to compute the density of samples required to ensure (with reasonable probability) having a certain minimum number of samples which are both on the object and accepted by the detector. We will refer to this minimum as N_D (see Appendix 1). SAMPLE-WINDOW computes the appropriate sampling density, and samples the window accordingly, collecting N_W points altogether. The probability of success is 1, since the program always samples the window. A good outcome for SAMPLE-WINDOW is the event that at least N_D samples on the object, which are acceptable to the detector, are chosen. The actual number of samples satisfying this requirement forms a binomial probability distribution where the probability of a single sample being simultaneously on the object, O, and acceptable to the detector, D^n, is $P(O,D^n)$. Therefore, the probability that at least N_D "good" samples are selected is

$$P_{\geq N_D} = 1 - \sum_{i=0}^{N_D - 1} [C(N_W, i)[P(O,D^n)]^i [1 - P(O,D^n)]^{N_W - i}].$$

In most cases, the value of this expression is approximately 1.

The cost of executing SAMPLE-WINDOW is essentially the cost of sampling any region. Although the INTERLISP random number generator must be called twice (to compute X and Y) for each point, most of the time is spent in setting up a grid over the region from which the points will be chosen. This makes the average execution time (about 1500 milliseconds) relatively insensitive to the number of points selected.

b. FILTER-WINDOW

FILTER-WINDOW requires a predicate (i.e., a detector), an object name, and a window as input. The window is assumed to have been previously sampled at a density appropriate for the detector and the object. It applies the detector to each sample in the window, adding any that pass to a specially created acquisition region for the object. This set of samples satisfying the predicate is accessible to other modules for further processing.

FILTER-WINDOW is successful if it accepts any samples from the window. It has a good outcome if at least one of those samples belongs to the target object. If N_W is the total number of samples in the window, and $P(D^n)$ is the probability that any randomly selected sample will be accepted by the detector sequence, D^n, then the probability of selecting at least k samples is

$$P_{\geq k} = 1 - \mathrm{SUM}_{i=0}^{k-1}[C(N_W, i)[P(D^n)]^i [1 - P(D^n)]^{N_W - i}]$$

Therefore, the probability of FILTER-WINDOW selecting at least one sample is just

$$P_{\geq 1} = 1 - [1 - P(D^n)]^{N_W} .$$

If we let $P(O|D^n)$ be the probability that a sample accepted by the detector sequence D^n also belongs to object O (see Appendix 1), and let $m = N_W P(D^n)$ be the expected number of samples selected by FILTER-WINDOW, then, the probability (weighted by the distribution of m) that at least k of the samples selected by FILTER-WINDOW will also belong to O is

$$P_{O,\geq k} = 1 - \mathrm{SUM}_{i=0}^{k-1}[C(m,i)[P(O|D^n)]^i [1 - P(O|D^n)]^{m-i}].$$

Again, the likelihood of at least one sample belonging to O is

$$P_{O,\geq 1} = 1 - [1 - P(O|D^n)]^m .$$

We use these expressions to compute the first two planning parameters for FILTER-WINDOW. The final parameter, the expected cost of applying the detector to the samples, is derived in Appendix 1.

c. Scanners

The scanning programs we use search sequential points along a line segment with a predicate, until either a point satisfies the predicate (i.e., successful completion), or the end of the segment is reached (i.e., failure). If the condition, which is defined by a predicate, can be satisfied, the program terminates successfully, else, unsuccessfully.

The primary use of the scanners is to locate an object after first locating some other object. It is often the case that given the location of some object you can infer the probable location of others. The scanner, then, will search from the initial object in the expected direction of the target object, testing points with an appropriate predicate, until some point is accepted. The predicates are usually designed so as to specifically accept points from the target, and reject points from the initial object.

Scanning programs in our repertoire include: SCAN-UP, SCAN-DOWN, SCAN-RIGHT, SCAN-LEFT, and SCAN-LINE. They require the names of the initial object and the target object, from which an appropriate predicate is manufactured (as described in a subsequent section). This predicate is scanned sequentially along the line, starting with a point (usually the center of mass) in the acquisition region of the starting object, and continuing until one of the termination conditions is met.

The probability of a successful completion is the probability that one of the points on the scan line will pass the detector, D. In the worst case, with no knowledge of spatial relationships, this probability is just P(D) for each sample. On the other extreme, when the acquired object, O_1, is known to be adjacent to the target object, O_2 (and therefore, the sample must be taken from oe of the two), this probability is precisely the sum,

$$P(D,O_1) + P(D,O_2).$$

In between these two extremes is the situation encountered for objects such as the chair, where the back is constrained by the seat, but samples from the space separating the two surfaces may belong to any of several other objects. If the set of possible objects can be limited, then we have an extension of the latter situation, where more than two objects are involved. If the set of objects cannot be limited, then we are left with the former situation, where all possible objects must be considered.

One other factor that influences the probability calculations is that the local likelihoods of a point belonging to one object or another are usually not constant along the line. As the scan proceeds outward from the starting point towards a point expected to belong to the target region, the probability of these points belonging to the first object decreases, and the probability of them belonging to the second (target) object increases. This suggests using a weighting of the relative likelihoods as a function of distance from the end points (which is not presently done).

We will derive the probability formulation from this description. Call the probability that one of k samples examined by the scanner will succeed, P_k. The probability of the i^{th} sample being accepted by the predicate, had it been randomly selected, will be p_i*. But, since the process is sequential, the fact that the i^{th} sample is being tested means that the preceding samples failed.

Therefore, the probability of one of the samples passing the detector is just one minus the probability that all k samples fail to pass the detector, or,

$$P_k = 1 - \prod_{i=1}^{k} (1 - p_i)$$

The probability that the successful outcome was also a good one is just the probability that the accepted point (which

* p_i is the representation of the probability function discussed above.

passed the detector) is a sample from the target object, O_2, or $P(O_2|D)$.

The expected cost is the per sample cost of applying the detector, C_s, times the expected number of samples processed. The expected number of samples is computed by summing the expected number of times we will examine exactly i samples, for i from 1 to k. This sum is

$$1 + 2p_2(1 - p_1) + 3p_3(1 - p_2)(1 - p_1) + \ldots$$
$$+ k(1 - p_{k-1})\ldots(1 - p_1).$$

Or,

$$C_k = C_s[[\sum_{i=1}^{k-1} ip_i[\prod_{j=1}^{i-1}(1 - p_j)]] + k[\prod_{j=1}^{k-1}(1 - p_j)]] .$$

These scanners, as described, are relatively unsophisticated. It would be preferable for a scanning program to continue checking a certain number of samples past the first successful one, in order to increase the likelihood of a correct identification.

2. Validation Modules

Validation modules are used to increase the confidence that the implicit hypothesis (that the acquired samples belong to the target object) made by the acquisition module is correct. They either attempt to distinguish the target from other objects sharing the acquisition characteristics, or use the remaining characteristics of the surface to increase the initial confidence.

a. DISTINGUISH

DISTINGUISH is the module that attempts to select the samples belonging to the target from those that may belong to other similar objects. For each point in the acquisition region, DISTINGUISH uses the measured set of properties from the point to classify it as belonging to one of the possible objects (i.e., either the target, or one of the set of ambiguous objects). If the classification likelihood for the target is greater than some cutoff probability value, p_c (e.g., .8), the sample is retained. Otherwise, it is rejected. DISTINGUISH succeeds if at least one point from the acquisition set is retained. It has a good outcome if at least one sample from the retained set belongs to the target. The points retained by DISTINGUISH comprise a new validated region associated with the target, which is available for subsequent processing.

To compute the probability of success of the module, we compute the likelihood that it will succeed given that the samples it examines were already acquired. Let $P(T|A)$ be the probability that a single sample that was acquired will be accepted by the distinguishing test. We write the formula to take into account the possible contributions to the probability value of all the objects in the ambiguous set. That is,

$$P(T|A) = \mathrm{SUM}_{O_j}[P(T|A,O_j)P(O_j|A)],$$

where $P(O_j|A)$ is just the probability that an acquired sample belongs to object O_j. However, since the probability of success of the distinguishing test actually depends only on the objects present, and not on the fact that the samples were acquired, we can rewrite the equation as

$$P(T|A) = \mathrm{SUM}_{O_j}[P(T|O_j)P(O_j|A)].$$

Now, since the classification test is structured so that it is very unlikely that a sample from object O_2 will be classified as object, O_1, we make the simplifying assumption that $P(T|O_1)$ (where O_1 is the target) is one, and $P(T|O_i)$ for $i \neq 1$ is zero. And therefore,

$$P(T|A) = P(O_1|A)$$

for each sample. Now, if m samples were acquired, the probability of the distinguishing test passing at least k of them will be

$$P_{\geq k} = 1 - \mathrm{SUM}_{i=0}^{k-1}[C(m,i)[P(O_1|A)]^i[1 - P(O_1|A)]^{m-i}].$$

And in particular, the probability of at least one sample being selected will be

$$P_{\geq 1} = 1 - [1 - P(O_1|A)]^m.$$

Since only samples whose classification likelihood is greater than p_c are accepted, this cutoff probability is a lower bound on the a priori likelihood that a sample accepted by the test will also belong to the target. The a posteriori probability will be the actual classification likelihood.

If $n = mP(O_1|A)$ is the expected number of samples retained by the module, the a priori likelihood (averaged over the distribution of n) that at least k of them will belong to the target is

$$P_{\geq k} = 1 - \mathrm{SUM}_{i=0}^{k-1}[C(n,i)p_c^i(1 - p_c)^{n-i}].$$

And, the probability that at least one will belong to the target is

$$P_{\geq 1} = 1 - [1 - p_c]^n .$$

The expected cost of application is the cost, C_s, of classifying a single sample times the expected number of samples tested. This number is obtained from the model of the appropriate acquisition process.

b. VALIDATE

In order to increase confidence in the acquisition hypothesis, VALIDATE measures any untested attributes of the acquired (and possibly distinguished) samples. The tests that are performed generally check whether the values of the sample's attributes fall within the allowable range of values for the object, based on information in the model. A successful completion for VALIDATE occurs if at least one of the samples it processes passes the tests applied to it. A good outcome occurs when at least one of the samples passed is also a point on the target.

For VALIDATE to succeed, a sample which was previously acquired (usually by passing some particular set of tests) must pass a further set of tests. The validation tests are devised so that they will pass any point from the target, but very few from other objects. We now must compute the likelihood, $P_m = P(D^m_{n+1} | D^n)$, that a sample which has already passed the set of tests, D^n, will pass another set, D^m_{n+1} (i.e., the set $\{D_{n+1}, D_{n+2}, \ldots, D_m\}$). This probability is expanded over the set of objects as

$$P(D^m_{n+1} | D^n) = \mathrm{SUM}_{O_j}[P(D^m_{n+1} | O_j, D^n) P(O_j | D^n)].$$

Since the outcomes of the detectors depend only on the objects, and not on the outcomes of previous tests, we can eliminate the dependency on D^n in the first term to get

$$P(D^m_{n+1} | D^n) = \mathrm{SUM}_{O_j}[P(D^m_{n+1} | O_j) P(O_j | D^n)].$$

$P(O_j | D^n)$ is available from the acquisition model. Therefore, only $P(D^m_{n+1} | O_j)$ need be computed*. Since the system does not retain the correlated data required to compute $P(D^m_{n+1} | O_j)$, we estimate the probablity by using the minimum of the set,

$$\{P(D_m | O_j), \ldots, P(D_{n+1} | O_j)\}.$$

Therefore,

$$P_m = \mathrm{SUM}_{O_j}[[\min\{P(D_m | O_j), \ldots, P(D_{n+1} | O_j)\}] P(O_j | D^n)].$$

As we have seen previously, if n samples are passed to the validation module for checking, the probability of at least k being accepted is

---

* Of course, since the validation tests are designed to pass all points on the target, $P(D^m_{n+1} | O_{target}) = 1$.

$$P_{\geq k} = 1 - \mathrm{SUM}_{i=0}^{k-1}[C(n,i)P_m^{\,i}\,[1 - P_m]^{n-i}].$$

And, of course, the probability of at least one being accepted is

$$P_{\geq 1} = 1 - [1 - P_m]^n.$$

The expected number of samples accepted is $q = nP_m$. The probability that any particular sample, accepted by the validation test, also belongs to the target is $P(O|D^m)$ (see Appendix 1). Therefore, the probability of a good outcome is

$$P_{\geq 1} = 1 - [1 - P(O|D^m)]^q.$$

The expected cost is the per sample cost of applying the distinguishing tests times the expected number of samples to be processed. As above, the anticipated number of samples is available from the models of the acquisition modules.

3. Bounding Modules

Once initial samples have been acquired and validated, another module is called to outline the object in the image. The region resulting from the process of bounding is not only the principal output of the location process, but also is used for any final distinguishing tests (such as shape and size evaluations) that still remain.

The modules described below are of two general types, those that operate in an undirected mode, and those that are directed. GROW-REGION, an undirected module, relies primarily on point attributes to extend the boundaries of an acquisition region outward to the boundaries of the object. It terminates when no new samples with the appropriate attributes can be found. On the other hand, the rectangle bounding modules, VRB and HRB, both use size and shape information about the object in question to produce tests for locating and verifying the boundary.

The specification of a good outcome for the bounding modules is rather tricky. Obviously, we want a good outcome to be the case where the module obtains an accurate outline of the target object. The likelihood of this is very hard to predict, since the procedure is relatively global in scope, and involves many poorly specified facts about the world. The object could be occluded by other objects. It could be oriented in such a way as to make certain measurements unreliable, or it could be adjacent to a "similar" object. What this means, is that it is not productive to try and model the likelihood of producing an accurate boundary, but rather base estimates on the results of past trials. This is the approach which will be used.

Similarly, we assume that the bounding modules are called after samples have been acquired and validated. Therefore, the modules are provided with samples that have a certain probability of belonging to the target object. For directed modules, the predicted boundary depends on the target. The assumption is made that for the right object, the predicted boundary will match the actual boundary (allowing for occlusions), and will not match the boundary of other objects. Therefore, the probability of success for these modules is the probability that the samples provided belong to the target. For the undirected modules*, they (practically) always succeed, since they are only required to collect samples until they can find no more. For this reason, the probability of success for these modules will be one.

a. GROW-REGION

This module obtains an object's outline by applying a predicate to samples adjacent to the validated samples, keeping

* in this case, only GROW-REGION is undirected.

any that pass the predicate. The neighbors of retained samples are examined recursively. When no more samples are accepted, the module calls the ISIS system function, BLIST, to generate a boundary for the region. The default boundary routine, which generates the convex hull of the region, can be overridden to allow for experimentation with other bounding programs. A successful outcome for GROW-REGION is the addition of any new samples.

The expected cost of the module is computed by multiplying the cost for checking each sample times the number of samples likely to be tested. We estimate this number by comparing the size of the starting (acquisition) region to the expected size of the bounded region. The program terminates only after it has failed to add any more new samples to the region. This means that all samples immediately adjacent to the boundary of the region, but not part of it must be checked (and rejected). Therefore, we use an estimate of the number of external points checked in the cost calculation. The actual computation performed is to subtract the starting region's area from the expected area of the final region (to estimate the number of internal points examined), and add the expected perimeter of the region (as an estimate of the number of external points checked). This number is then divided by the frequency with which points are checked (i.e., the reciprocal of the spacing) to get the expected number of points checked. This number multiplied by the expected cost per point gives the total expected cost.

b. VRB and HRB

The VRB module is used to locate boundaries for objects whose outlines are vertical rectangles (such as a door). The process is equivalent to projecting the rectangle onto the image, and using the resulting outline to direct the search for the actual outline of the object. The module uses a predicate which distinguishes the target object from others likely to be adjacent to it to scan horizontally for an edge. The three dimensional coordinates of this point fix the edge in space. If the size of the rectangle is known, the location of the parallel edge can be computed. When the size is not known, another scan for the parallel edge is made. At this point, if the height of the lower edge and the vertical extent of the rectangle is known, the complete boundary is fixed. Otherwise, the program must scan upwards and downwards to locate the remaining edges. After fixing the rectangle, any edges that extend beyond the borders of the image are clipped. The final stage is to test the points on the hypothesized edges to ensure that they are not only located appropriately, but also to detect any occlusions.

The cost of this module is the sum of the costs of scanning to the edges of the region, the cost of the projection, and the costs of checking the boundary. The scanning costs are estimated as $.5WK_T$, where W is the expected dimension parallel to the scan, and K_T is the expected cost of the edge test. The number of scans required is a function of the amount of a priori information available. Projection costs are essentially fixed. Finally, the cost of the boundary test is K_T times the number of points in the perimeter, assuming the same edge test is used.

The HRB module, used to bound horizontal rectangles, is similar to the VRB module, except that the rectangle now has an extra degree of freedom, namely the angular orientation around a normal to the surface. Therefore, before the program can project the rectangle into the image, it must eliminate this degree of freedom. It can either locate a point on two edges, or establish the orientation of the first edge (by, for instance, using an edge operator which gave orientation information in the original acquisition of the edge).

C. Automatic Predicate Generation

In the course of our work we attempted to automate three key decisions involved in planning a distinguishing features strategy:

(1) What attributes should be used for acquiring initial samples?

(2) What attributes should be used for validating those samples?

(3) What attributes should be used for determining boundaries?

All three decisions involve distinguishing a desired surface in a context of other surfaces that may be present. Acquisition involves distinguishing a surface from other known surfaces in a domain, validation involves distinguishing a surface from other surfaces known to satisfy the acquisition attributes, and bounding involves distinguishing a surface from all surfaces that could possibly be adjacent to it in the image.

We recall that objects are characterized by local surface attributes, symbolic properties, and relations with other objects. The local surface attributes that are currently used as descriptors are: brightness, color (separated into hue and saturation components), and when range data are available, height and local surface orientation (referred to as "orientation"). Local properties are stored as cumulative histograms representing the probability density functions for the object's attribute values. These histograms allow the probability of an object having an attribute in a given range to be computed by a simple subtraction. These histograms may be augmented by symbolic properties such as boundary descriptions, size and extent values, surface characteristics (e.g., horizontal), and spatial relations (e.g., adjacent to, above, or left of). The creation and modification of these descriptions are discussed below.

1. Acquisition Predicates

Acquisition predicates are generated by a heuristic program called DACQ (for Default ACQuisition) that attempts to satisfy two simple requirements. First, the predicates should pass a few points on the desired object while excluding all points from other objects. Second, they should be as cheap and reliable as possible. Naturally, these requirements are somewhat contradictory; usually a detector that allows most points on the target object to pass will also pass some points from other objects. If the detector is made more restrictive, it usually becomes prohibitively expensive. To resolve the difficulty, the program attempts to generate compound detectors that exclude all samples from undesired objects, allowing at least a few points on the desired object to pass. Since this is not always possible, the program returns a list of any undesired objects that cannot be completely excluded, so that these may be distinguished subsequent to filtering.

The expected cost and confidence of applying a detector can be computed for each object. The confidence of an object given a detector is loosely defined as the probability that a sample accepted by the detector belongs to the object in question. The complete derivation is contained in Appendix 1. Cost is defined as the expected cost (in milliseconds of processing time) of application of the detector to the image. Given a detector, a set of points to be filtered, and the joint probability that a random point from the set will both be part of the object and also accepted by the detector, it is possible to compute the appropriate sampling frequency and the anticipated cost. This derivation is also contained in Appendix 1.

The quality of a detector is evaluated using a heuristic function, Q, of cost (K) and confidence (C) that reflects the design priorities listed above*. The generation of good

* The quality function, Q, which we currently use, is just C/K, the quotient of confidence and cost.

compound detectors now becomes (for combinatorial reasons) a problem in heuristic search. The search proceeds as follows. First a good initial set of simple detectors is selected. The candidate detector with the best Q is then combined with others to see if the combination improves the Q function. The best combination, in turn, is combined with remaining candidates, and so on, until one of the following terminating conditions is reached, either the set of candidates is exhausted, some combination exceeds a preset Q threshold, or an effort bound is exceeded.

Two heuristic programs are used to select candidate simple detectors. The first, PEAKS, locates peak attribute values for the object. It examines the characterization of the object (which are probability density curves and will be discussed later) for a peak providing a certain minimum area underneath. The function will then narrow the peak if it can do so without significantly decreasing the area. Values of minimum acceptable peak size and other variables were empirically determined through experimentation with ISIS. PEAKS, which finds a single peak for each attribute curve, outputs a set of simple detectors corresponding to the attributes considered.

The second selection program, MINRANGE, generates distinguishing detectors by looking for contiguous ranges of attribute values which minimize the set of ambiguous objects.

The program scans a range of an attribute for the target object, checking to see what other objects share the range. MINRANGE also returns a set of single detectors.

The present implementation of the heuristic predicate generator creates only composite detectors from the initial set supplied by PEAKS and MINRANGE. The program cannot yet generate detectors that are disjunctions (such as might be required for locating a red and white checkered tablecloth) or decision trees. Despite these limitations (which we expect to eliminate in the future), the detectors generated are quite effective.

To generate these composite detectors, DACQ selects the current best (in terms of Q) detector on the list and tries to combine it with all the simple detectors on the initial list. Detectors that would violate the description given above (e.g., which would combine two detectors with the same attribute), or those in which the composite does not have a smaller ambiguity set than the parents are not considered. When a candidate is selected, a composite detector is formed and retained only if its value of Q is higher than that of both parents. Before adding the new candidate to the appropriate spot in the list of detectors, a quick check is made to see if it could ever be better than the current best detector. This is done by computing Q for a hypothetical detector with the same cost, but with a confidence of 1. If the Q of this combination is higher than the best Q so far, the new candidate is kept. Otherwise, it is discarded.

A candidate that is retained is added to the proper location in the ordered list of candidates. After the first detector has been tried with all the possible single detectors, it is marked as having been expanded, and DACQ proceeds with the best (unexpanded) detector on the new list. If no candidates are retained, DACQ continues with the next best detector on the list.

When there are no more candidates, or when a detector is generated with a Q above a certain threshold, the program terminates, returning the complete ordered list of detectors generated.

DACQ spends considerable time attempting minor improvements to its best candidates. Therefore, a CPU time limit was provided to allow the user to decide how much effort DACQ should expend. If the time quantum provided by the user is exceeded, the program interrupts, prints the best candidate, and asks the user whether it should continue. If the reply is NO, or there is no reply within ten seconds (equivalent to a NO reply), DACQ terminates with the best so far. If the user wishes to proceed, DACQ will continue for another quantum of time. The predicates produced by DACQ within the time constraints (currently about 10 seconds) are typically the same as those produced by letting it run to completion. Also, the detectors produced are usually as good as, or better than, predicates defined manually by experienced users (often they are the same).

When DACQ terminates, the best compound detectors are converted to equivalent LISP programs and compiled for use by acquisition modules.

2. Design of Validation Predicates

Validation typically proceeds in one of two ways. Either the sample is classified as belonging to one of the members of the ambiguity set, or further tests are made to increase the confidence that the point is from the target. In the first case, no predicate is required; the actual measured attribute values of the points are used. In the second case, the test usually involves measuring the remaining characteristics (i.e., the ones not used for acquisition), and ensuring that they fall into the range of allowable values for the target.

3. Design of Bounding Predicates

The next step in object finding after validating the acquisition samples is to extract the boundary of the surrounding surface. Bounding a single, specific object is a significantly different task than global scene partitioning as described (say) by Yakimovsky[1].

In object bounding, a region is grown from acquisition samples by including all contiguous samples that satisfy an appropriate bounding predicate. Specifically, the program, GROW, applies the predicate to samples adjacent to the acquisition samples. If a sample is accepted, its neighbors (either 4-adjacent or 8-adjacent) are added to a list to be recursively examined. When no further samples can be added, GROW returns the resulting new region.

The bounding predicates differ from acquisition predicates in two important ways. First, the particular region being bounded need only be distinguished from pictorially adjacent objects, as opposed to all objects in the scene. Second, region growth requires predicates capable of collecting all points on the object.

Since the predicate must accept all points on the object, the task of generating an initial predicate is greatly simplified--we need only create detectors based on the complete range for all attributes (a "full-range detector"). There is relatively little risk in using these wide range detectors, since they need only discriminate objects adjacent to the acquired object. However, if the system has knowledge about which objects could be adjacent, the predicate can be refined by specifically disallowing points from these objects.

A major problem with the above approach is that the region growing predicates are applied uniformly to all points. Often, it is known that the color (say) of a region is darker at the bottom than at the top. Such information should be used to produce more selective predicates that respond differently at different points in the region. Adaptive predicates which would continuously change their expectation of what should come next, based on what has recently been seen, could also be designed.

D. Higher Level Planning Considerations

In planning a strategy, we wish to be able to use one object to fix the location of others. To accomplish this requires an extended set of planning modules. In this section, we wish to describe the concepts we have developed to facilitate finding objects indirectly.

1. Windowing

The whole philosophy of indirect searches (and, in fact, our whole approach) is to limit the context in which we look for the object. We provide the information required for this restriction by the use of "windows." In this section, we will use window in two different ways: first, it can designate some area of the image under current consideration; more often, it will mean a symbolic data structure associated with an object. The distinction will usually be obvious. The two concepts are related in that a window into an image will usually be provided from a symbolic window.

A window associated with an object consists of a set of related objects and its relationships with them. These objects may consist of ones that are considered to be "functionally" related, and those that might be considered as "fortuitously" related. The first category includes objects that have a bona fide relationship to the first object. For example, functionally related objects for a table might include books, plates, chairs, and telephones. While, on the other hand, fortuitously related objects might include walls, doors, pictures, and wastebaskets. These are all objects that may be associated with tables, but usually are not thought of in the same context. Their importance here is that they may be pictorially adjacent, and therefore may be important to a strategy involving tables.

It is obvious that many objects, while associated with a particular object, may have a fairly tenuous relationship. A telephone may have a fairly close relationship to a table (e.g., we may have a rule: Telephones are always on tables), but a chair, while normally associated with a table is much less constrained in relation to it, and therefore has a looser relationship than a telephone.

Because of this, we allow for many different windows for an object, all specified symbolically. We might wish to differentiate along any of the lines indicated. For example, some windows for table might be the set of objects likely to be on the table, or the set of objects normally found in the same room with a table, or perhaps the set of objects which might be pictorially adjacent to the table. Each of these might have a place in our planning. In particular, each of these might result in different mappings onto windows in the image. The pictorially adjacent (symbolic) window would typically result in an image window consisting of all samples some distance out from the object, while a symbolic window consisting of objects supported by an object might imply an image window which only extended upward from the surface. Relations such as ABOVE or LEFT-OF further constrain the associated image window.

In order to select useful windows, we need some likelihoods on inter-object relationships. These likelihoods are used to compute probabilities for indirect searches. In particular, the process of locating an object indirectly has two principal phases. First, the related object must be located, then by some other process (e.g., scanning or filtering), the window in the image implied by the objects and their relationship must be searched for the second object. If we assume that we will find the related object, O_R, with probability $P(O_R)$*, and that given that object, the relationship, R, with the target object, O_T, holds with probability, $P(R|O_R)$, then we are in a position to compute the

* Of course this is really $P(O_R|S_R)$, where S_R is the strategy for locating O_R.

likelihood of finding O_T indirectly from O_R. We assume the indirect strategy used to locate O_T from O_R works with probability, $P(S_I|R,O_R)$. This may be some expectation based on the operation of a scanning program, for instance, or we will often choose to hold this probability fixed, and allow the likelihood of the relationship, $P(R|O_R)$, to dominate the computation. However, for now we see that the likelihood of locating the target indirectly from O_R is just

$$P(S_I|R,O_R)P(R|O_R)P(O_R).$$

This is the probability of successful completion of the sequence of events, that is, the likelihood of reporting success. The probability of a good outcome, given success is, just $P(O_T|S_I)$. But we can expand this to give

$$P(O_T|S_I) = \frac{P(S_I|O_T)P(O_T)}{P(S_I)}.$$

We will not go into the complete details of the computation here, but will note that, in this context,

$$P(O_T) = P(R,O_R) * \frac{S(O_T)}{S(W_R)}.$$

That is, the probability of a randomly selected area in the window specified by O_R, belonging to the target, is the probability of the target being in the window (i.e., $P(R,O_R)$), times the ratio of expected sizes of the target and the window.

2. The Choice of Direct vs. Indirect Search

We have just seen one way of deciding whether to try to locate an object directly or indirectly. That is to compute the relative costs and likelihoods, and decide on that basis. This is often the basis for the decision, but there are other factors. These factors usually relate to a cost-effective approach, but have become more heuristic.

In general, if we are interested in locating a small (or otherwise difficult to see) object, and if we know a relationship exists between a larger object and the target, we will try to locate the larger one in hope of being able to use it to locate the smaller. We will often do this without following the plan completely to the end. The rationale is that the large object will likely be of aid in the task, and it is more worthwhile to just go find it, than to spend (what we expect to be) a great deal of time planning for finding the target. If we find the larger object, then we can concentrate on locating the target, but if we fail to find the larger object, then we have likely spent less time looking for it than we would have spent thinking about what to do afterward. An extension of this heuristic is the idea of choosing to find a related object because it is especially distinguished in the scene, where the target is not. Typically, the most easily distinguished objects are also the largest ones, but this is not always true. For example, it might be wise to locate the table by finding the desk lamp, the brightest object in he room.

Another situation in which we would usually like to utilize an indirect search is that where we will need to find a set of related objects. This would be the case, for example, when we were trying to outline the chair. In this case, we need to locate the seat and back (say). Since one will constrain the location of the other, it makes sense to attempt to find one part directly, and use that to constrain the location of the other part.

The final situation where we might choose an indirect search is one where we had already located some objects. These would then provide the context for locating the target. If we had already identified the door, for example, we would probably choose to look for the wall as an adjacent object, even though it was easy enough to find the wall directly.

IV PLAN GENERATION AND EXECUTION

A. Introduction

The difficulty facing artificial intelligence problem solvers has traditionally been to generate any plan for performing a desired task; since the choices for solving the problem were generally limited, the choice of which subgoal to try next has not been a difficult decision. The planner to be described here generates a strategy for locating an object in an image, but then, more importantly, organizes the plan so as to know in which order to attack the subgoals. The problem at any point in execution is to know what to do next. An example of a simple plan for finding a chair is shown in Figure 16. We will describe the generation of this plan in detail below.

The planner first elaborates a goal into the subgoals required for its solution. These subgoals are elaborated in turn until no more remain. The result is an AND-OR planning graph with tip nodes that are executable subgoals. Next, the

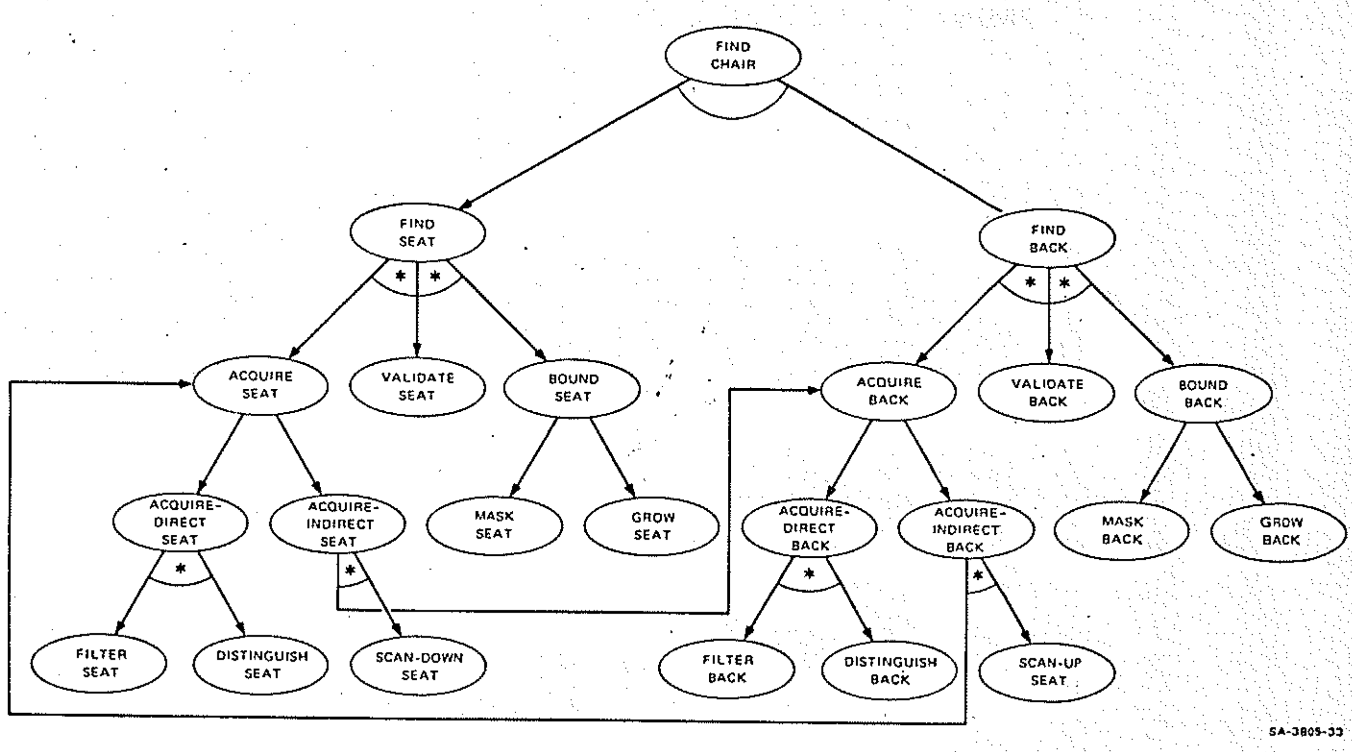


Figure 16. Diagram of a Plan for Finding a Chair

scorer organizes the graph so that the executor can find the best terminal node to begin evaluating. The plan executor executes the best subgoal and then propagates the success or failure of the goal through the original plan. The scorer reorganizes the plan based on the outcome of the subgoal, and execution continues until the top goal either succeeds or fails.

B. Planner

1. Definitions

We begin with a few definitions and descriptions of strategies, plans, and elements of plans. Although many of these definitions have appeared previously, we include them here for completeness.

Goal - a representation of some state to be achieved. A goal is either directly realizable (i.e., executable), or is an intermediate goal, which is satisfied by satisfying its subgoals. An executable goal is a terminal goal.

The format of a goal is a list, the first element of which is usually the activity represented by the goal (e.g., FIND, VALIDATE). The second element is usually the object for which the goal is being executed. Succeeding elements (if any) usually contain information pertinent to the goal. For example,

```
(FILTER-WINDOW  DOOR
      DOOR:AF
      ((HEIGHT (6 . 8))
       (SAT (7 . 12))))
```

is a typical goal. The activity is FILTER-WINDOW, the object is DOOR; the next element is the LISP predicate to be used in filtering, and the last element is the internal representation of that predicate. The last two elements are particular to FILTER-WINDOW.

Subgoal List - a list of subgoals preceded by an operator from the set: AND, *AND, OR, or ↑. The operator indicates how solution of the subgoals relates to solution of the goal. AND and *AND imply that all the subgoals must be achieved to satisfy the goal; OR implies that achieving any one of the subgoals will suffice. *AND requires that the subgoals be achieved in the given sequence. ↑ is a dummy operator which is used when there is only one subgoal and means that satisfying the subgoal is equivalent to satisfying the goal. This operator is used primarily to minimize the complexity of the subgoal lists. Some example subgoal lists are:

```
(AND (FIND SEAT)(FIND BACK)),

(*AND (ACQUIRE DOOR)(GROW DOOR)), and

(OR (ACQUIRE-DIRECT PICTURE)
    (ACQUIRE-INDIRECT PICTURE)).
```

The first example is the subgoal list for (FIND CHAIR); the second for (BOUND DOOR); the third for (ACQUIRE PICTURE).

Node - (or goal node) is a list consisting of the goal, the subgoal list for the goal, a set of parameter lists for scoring purposes, and the parents of the goal. For example,

```
((ACQUIRE DOOR)
 (OR (ACQUIRE-DIRECT DOOR)
     (ACQUIRE-INDIRECT DOOR))
 ((.9) (.95) (.855) (50000.) (58500.))
 ((BOUND DOOR)))
```

could be the goal node for the goal (ACQUIRE DOOR), which, in turn, is a subgoal for (BOUND DOOR).

Plan - the set of goal nodes generated for satisfying the initial goal. In its most general form, the plan is an AND-OR graph since the subgoals may point to one another through their own subgoal lists. The graph structure appears when a goal has some other goal for a subgoal, which in turn has the first as one of its subgoals (although not necessarily as a direct descendant). For example, (as shown in Figure 16) to find the seat of a chair, it might be possible to first find the back and use that to localize the seat. However, to find the back, it might also be possible to first find the seat and then localize the back. This type of reasoning results in plans with loops.

Module - a module is a program or a statement used in satisfying a goal. We have two types of modules: planning modules and execution modules. Planning modules are the programs executed in the course of planning a strategy. They usually result in combinations of other modules. Execution modules are run during the course of executing the strategy, and are usually image processing programs of some sort. A module may have various functions associated with it, such as COSTFN (which computes the cost of execution of the module), and PTFN (which computes the probability of success). In addition, the module may also have associated functions used for deciding whether a previously generated goal is equivalent or related to the current one.

Cost - in the context of planning, the cost is the estimated time that would be spent trying to satisfy a goal. In the case of bounded processes (such as filtering, the cost can be estimated rather closely. For less bounded processes (such as region growth), the cost is approximated from past experience and whatever relevant information may be available (for example, the expected size of the region to be grown).

Confidence - in the context of planning, confidence is the probability that the execution of the goal will have a good outcome. This is composed of two parts, P(X|T), the probability of a good outcome given that the node succeeds, and P(T), the a priori probability of success of the goal. In cases where the process is well modeled, the estimates are relatively good; where there is no good model, the estimates are based largely on past experience. Confidence, of course, is always between 1.0 and 0.0.

Score - the score of a node is the value of a function which computes a cost figure normalized by confidence. Our current score function is cost divided by confidence.

2. Plan Elaborator

To create a plan, the plan elaborator (PE) expands the initial goal (the "top goal", which is typically to find some object, e.g., FIND(TABLE)) into the subgoals required for its solution*. The subgoals are recursively elaborated in the same way. If the PE cannot generate subgoals, it checks whether the goal is executable, and if so computes parameters (cost and confidence) for later use in scoring. If it is not executable and has no subgoals, it is marked for later deletion.

As goals involving new objects are examined by the PE, their names are added to an "instance list". This list is used as a global "note-pad" to maintain up-to-date information discovered about objects during execution of the plan. The initial entries into the instance list are taken from a globally maintained list of prototypes. For example, when the chair seat is noticed for the first time in the course of planning, the name is looked up on the list of prototypes. The generic information contained in the prototype description is then added to the instance list. Data contained in the prototype consists of such general information as relations between the object and other objects, or even previous plans created for locating that object. This information is available to the planner during elaboration, and during

---

* That is, it "elaborates" on the solution of a goal.

execution is updated by executed modules. For example, the chair seat prototype could contain the information that the chair seat is below the chair back. During execution of a plan to find the chair seat, the acquistion module (discussed below) would save any acquired chair seat samples on the instance list, which would then be available to the validation module for further processing (e.g., to a program for outlining the chair).

Before generating the subgoals of a goal, the elaborator checks to see if any special modules exist for satisfying the goal for that particular object. This allows the user to provide advice to the planner about certain goals. For instance, the planner could be instructed to always look for the telephone on the tabletop. If no special routines are available, the planner looks for routines associated with the activity and that are generally useful for all objects.

For example, to elaborate the goal (ACQUIRE TABLE), the planner might be instructed to use a horizontal plane finder, since the tabletop is a horizontal plane. Alternatively, if there was no information associated with TABLE, the elaborator would look into general purpose routines pertinent to ACQUIRE. The graph in Figure 16 provides another example. There is a module for FIND which says to find an object, find its parts. Therefore, to find the chair a plan is generated for finding its major parts, the seat and the back.

In general, several planning modules may be associated with a given activity. In order to select the expansion modules appropriate to a particular goal, the planner applies a predicate function associated with each to the goal in question. If it returns a true value, the planner adds the module to its list of applicable modules. Special cases of the predicate function can cause the module always to be selected (to provide defaults) or can cause the module to be selected by goals specified explicitly. This predicate allows us to have several modules, each an expert in a relatively narrow domain, and each selectable on the basis of that domain. For example, associated with the activity, ACQUIRE, could be a program for locating horizontal planes. This program would have a predicate that would return true when applied to a horizontal object.

Before a newly generated goal is added to the list of goals to be elaborated, a check is made to see if it has already been generated, or if any equivalent subgoals have been generated (the subgoal must have a special function to allow the planner to decide this). In either event, it is not processed further, and the subgoal list of the goal being expanded is modified to point to the previous goal. The planner will also check for related subgoals (that can succeed or fail based on other goals that are not directly related, i.e., not parents or children). For example, the planner may generate a strategy which includes multiple filter subgoals where one filter predicate subsumes another. If the less restrictive predicate fails to turn up the desired object, it is guaranteed that the more restrictive one also will not (when executed on the same set of samples). Therefore, if the less restrictive filter is selected and executed, its outcome could imply the outcome of the related, restrictive filter. The planner maintains an explicit list of related subgoals, for this type of situation.

3. Scorer

After a plan has been elaborated, the next step is to decide which subgoal to attempt, or, in general, to decide what to do next. The choice depends on the local expense of executing the goal, the more global expense due to the fact that some goals may be easier after other goals have been executed, and the likelihood of correct results after the goal has been executed.

We compute a score for a node which allows us to make a selection from proposed alternatives for satisfying the goal. This score function will be cost normalized by confidence, with the result that high confidence, low cost goals will have a smaller score than low confidence, high cost goals, and medium confidence, medium cost goals will fall somewhere in between.

Before discussing the techniques of scoring a plan, we digress briefly for a preview of plan execution. The plan executor receives a completely scored plan. It starts at the top node of the plan, and proceeds downward through the plan, always selecting the best branch (on the basis of score) until it reaches a terminal node. This executable goal is then evaluated. If the subgoal fails, the result is propagated backwards through the plan. However, if the node succeeds, then the confidence of the node is propagated back. That is, all nodes dependent upon the successful node have their a priori confidence altered to reflect the success of the node. When the propagation ceases, if the top goal has not been satisfied, the updated plan is rescored and execution proceeds as above.

The output of the plan elaborator is a planning graph. This representation of the strategy, although usually non-reentrant, may, in general, have loops. While the problem of backing up costs and confidence through a tree structure is quite simple, the same problem in a graph with loops is not quite so straightforward. We will first describe the technique for a tree, and then show how we generalize it to a graph.

The scoring process uses a relaxation technique. All goals in the plan are put on a list to be examined. For each one in turn, the score is computed (if possible). If the score of a node changes from the last value, its parents are put on the list to have a new score calculated. The process continues until the scores stabilize. If a node has a score, it means that some way (presumably) exists to satisfy that goal.

Since the scores for terminal (executable) nodes do not change during the scoring process, the only things that need be considerd are the intermediate nodes. We begin by describing how scores are passed from a subnode to its immediate predecessor.

As mentioned previously, there are four possible types of intermediate nodes, *AND, AND, OR, and ↑. The ↑ node passes all of its scoring parameters to its parents, and will not be discussed further. Since the *AND and AND nodes require all the subgoals to be satisfied, a score for this type of node can only be computed if scores for all the immediate subgoals have been computed. However, an OR node merely requires that one of the subgoals be satisfied. This means that even if only one of the subgoals can be scored, the OR node can be scored. Effectively, any unscored node is considered to have an infinite score. The scores of AND or *AND nodes with any unscored subgoals is also infinite. Unscored subgoals of OR nodes can be ignored, and the score computed on the basis of the ones already scored. In practice, what occurs is that an initial score is determined for some subgoals, which allows other subgoals to receive scores. These subgoals in turn propagate their scores back, and the process continues to completion. At the end of the process, any nodes which haven't received scores are deemed impossible to satisfy, and are deleted from the plan. In operation, the actual parameters that are backed-up are cost and confidence, and the score of the node is computed from them directly.

We will use the symbol S to stand for score, C for confidence, and K for cost. We will also use T for the successful execution of a node, T^k for the sequence of successful executions of nodes 1 through k, $\bar{T}$ for unsuccessful execution of a node (i.e., failure), and $\bar{T}^k$ for the sequence of unsuccessful executions of nodes 1 through k. Thus, S_i is the score of the i^{th} node, K_i the anticipated cost, and C_i the confidence. T_i is the representation of successful completion of node i.

The confidence is the a priori joint probability of success and a "good outcome" from the node. We expand this to represent confidence as the product of two factors: the probablity of a good outcome from the node, given the success of the node, and the probability of success of the node. If we let α_i represent a good outcome from the i^{th} node, then we have $C_i = P(\alpha_i | T_i)P(T_i)$. Thus, we see that confidence is just the product of the two planning parameters described in Chapter 3. We will also note that the a posteriori probability of a good outcome from the i th node is zero if it failed, and $P(\alpha_i | T_i)$ if it succeeded.

As mentioned above, the score of a node will be cost normalized by confidence, that is $S = K/C$. In the course of searching for the best path through the graph, we will attempt to minimize this function at each node. In order to compute the score of intermediate nodes, costs and confidences must be backed-up from subnodes to their superiors.

a.. Scoring an OR node

We begin the discussion with the OR node in Figure 17. We assume that the subnodes are ordered by increasing score. The backed-up parameters will be dependent on the order, and it can be shown that this particular ordering will result in a minimum score for the node.

For an OR node to have a good outcome, one of the subnodes must have a good outcome. Therefore, we can write the confidence as the sum of the probabilities of a good outcome from each node. That is,

$$C_{OR} = P(\alpha_1, T_1) + P(\alpha_2, T_2, \overline{T}_1) + \ldots + P(\alpha_i, T_i, \overline{T}^{i-1})$$

$$+ \ldots + P(\alpha_n, T_n, \overline{T}^{n-1}),$$

or,

$$C_{OR} = \mathrm{SUM}_{i=1}^{n}[P(\alpha_i, T_i, \overline{T}^{i-1})].$$

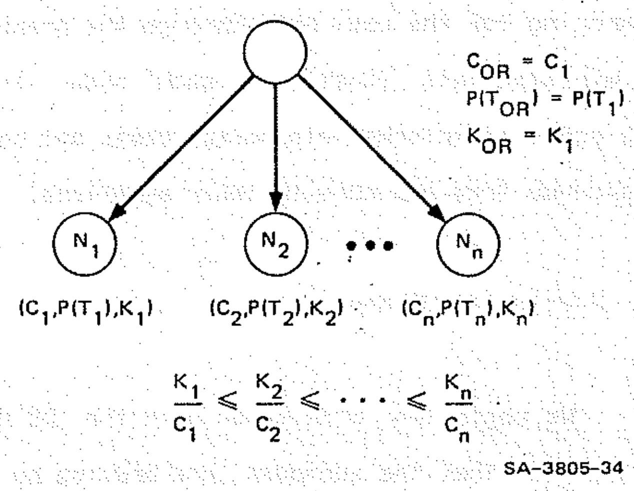


Figure 17. Backed-up Planning Parameters for an OR Node

Expanding this formula gives

$$C_{OR} = \mathrm{SUM}_{i=1}^{n}[P(\alpha_i | T_i, \overline{T}^{i-1})P(T_i | \overline{T}^{i-1})P(\overline{T}^{i-1})].$$

We have in $P(\alpha_i, T_i, \overline{T}_{i-1})$ a context sensitive measure of confidence. That is, it is a confidence measure that takes the past history of the parent node into account. Since we do not make use of failure information to adjust likelihoods in our current system, we must make the weakening assumption of independence between T_i and $\bar{T}^{i-1}$, and also assume that α_i is dependent only on T_i, not on $\bar{T}^{i-1}$. We can now rewrite the formula as

$$C_{OR} = \mathrm{SUM}_{i=1}^{n}[P(\alpha_i \mid T_i)P(T_i)P(\bar{T}^{i-1})]$$

or

$$C_{OR} = \mathrm{SUM}_{i=1}^{n}[C_i P(\bar{T}^{i-1})] .$$

And, therefore

$$C_{OR} = \mathrm{SUM}_{i=1}^{n}[C_i \prod_{j=1}^{i-1} (1 - P(T_j))] .$$

The expected cost of the node is the sum of the expected costs of each of the individual subnodes, which is just the cost of the node times the probability of executing it. The probability of executing a node is the probability that all preceding nodes fail. Therefore, we can write the expected cost of the node as:

$$K_{OR} = \sum_{i=1}^{n}[K_i P(\bar{T}^{i-1})]$$

With the same assumptions of independence as above, we can write the cost formula as

$$K_{OR} = \sum_{i=1}^{n}[K_i \prod_{j=1}^{i-1} (1 - P(T_j))]$$

The probability of success of the OR node is just $1 - P(\bar{T}^n)$, that is, the probability that they do not all fail. This is rewritten

$$P(T_{OR}) = 1 - \prod_{i=1}^{n} (1 - P(T_i)).$$

The score is computed from the backed-up cost and confidence. It is apparent, however, that this cost function allows a very expensive subnode to pull the score of the whole node down, even though it may be likely that it will never be executed. Since the score of a node should reflect the best that can be expected from it, we define the score to be the minimum score of a set of subsequences of the sequence of subgoals. That is, we can compute the score for the whole sequence, then for that subsequence without the last goal, then without the last two subgoals, and so on. We then take the minimum of these scores to be the score of the node. But this definition implies that the score of an OR node will always be the score of the first subgoal in the sequence. Therefore, the first subgoal in the sequence will represent the node. As a result, the planning parameters for the node become:

$$C_{OR} = C_1$$

$$K_{OR} = K_1$$

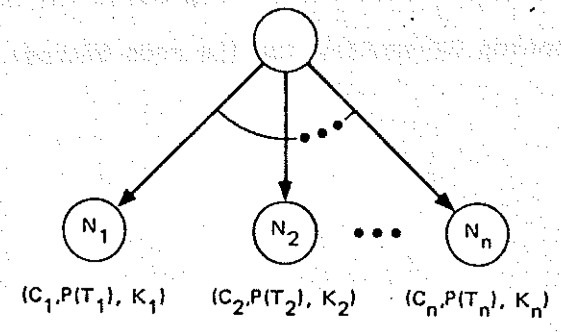
$$P(T_{OR}) = P(T_1)$$

$$S_{OR} = S_1$$

Therefore, with this particular score function, the best strategy for choosing an option is to select the one with the lowest score. For planning purposes, the parameters of only that node are used. It is important to note that this result is strictly due to the choice of score function. The more general results derived would be required if a different score function was chosen.

b. Scoring an AND node

We assume that the subnodes of an AND node have been ordered by $K_i/(1 - C_i)$, as this can be shown to result in a minimum score for the node. A *AND node is already ordered, so the formulas derived for an AND node will apply equally to a *AND node. The AND node is shown in Figure 18.



$$C_{AND} = \prod_{i=1}^{n} C_i$$

$$P(T_{AND}) = \prod_{i=1}^{n} P(T_i)$$

$$K_{AND} = \sum_{i=1}^{n} K_i \prod_{j=1}^{i-1} P(T_j)$$

$$\frac{K_1}{(1-C_1)} \leqslant \frac{K_2}{(1-C_2)} \leqslant \cdots \leqslant \frac{K_n}{(1-C_n)}$$

SA-3805-35

Figure 18. Backed-up Planning Parameters for an AND or *AND Node

The confidence in this case is the probability that all the subgoals will have a good outcome, or

$$C_{AND} = P(\alpha^n, T^n) = P(\alpha^n | T^n) P(T^n).$$

With our assumptions of independence, we have

$$P(\alpha^n | T^n) = \prod_{i=1}^{n} P(\alpha_i | T_i),$$

and

$$P(T^n) = \prod_{i=1}^{n} P(T_i).$$

Therefore,

$$C_{AND} = \prod_{i=1}^{n} P(\alpha_i | T_i)] [\prod_{i=1}^{n} P(T_i)$$

or

$$C_{AND} = \prod_{i=1}^{n} [P(\alpha_i | T_i) P(T_i)].$$

And, therefore,

$$C_{AND} = \prod_{i=1}^{n} C_i .$$

If a subnode of an AND node fails, we terminate execution. Therefore, the expected cost of a single subnode in the sequence is the cost of the node times the probability of executing it. This probability is the probability of all preceding nodes succeeding. The cost, therefore, is

$$K_{AND} = \mathrm{SUM}_{i=1}^{n} [K_i P(T^{i-1})],$$

or,

$$K_{AND} = \mathrm{SUM}_{i=1}^{n} [K_i \prod_{j=1}^{i-1} P(T_j)].$$

The probability of success of the node is

$$P(T_{AND}) = \prod_{i=1}^{n} P(T_i) .$$

The new score is computed from these backed-up values.

c. Scoring a Plan

With these techniques for computing the score of individual nodes based on their subnodes, we can discuss an algorithm for computing the score of a complete plan. The algorithm is:

```
1 Set the scores of all intermediate nodes to
  infinity (all tip nodes have already received
  a score);

2 Place all goals on the list, CHECK;

3 If CHECK is empty, exit with scored plan;

4 Set CURRENT to the first goal on CHECK, and
  remove it from the list;

5 If CURRENT is a tip node, go to step 3;

6 If CURRENT cannot be scored *, then go to step
  3;

7 Compute the score for CURRENT, if it is
  different from the previous value, then add
```

* An AND type node can be scored iff all of its subgoals have non-infinite scores. An OR node can be scored if any of its subgoals has a non-infinite score.

the parents of CURRENT to CHECK (therefore, if the score of a node changes, we insure that its parents will be reconsidered, as their scores might change also);

8 Go to step 3.

We can see that the effect of this algorithm on a plan with a tree structure (i.e., no loops) is straightforward. The algorithm has the effect of starting at the tip nodes, and propagating the scores upward in the tree. Any tip node is removed from CHECK as soon as it becomes CURRENT, and will never be put back. An intermediate node with only tip nodes for subgoals, will likewise be scored, its parents added to check if the score changed, and it will be deleted from CHECK, never to reappear (since none of its subgoals' scores can ever change). Eventually all the nodes one level up from the tip nodes will be scored and removed from CHECK. Conceptually, this process continues to the next level and so on up the tree. Although the order of scoring may be arbitrary, the number of times a goal is looked at (in a tree) is bounded by the total number of intermediate subgoals it has.

The process of scoring a plan with loops, however, (such as the one in Figure 16) is more complicated. The existence of a loop in the plan means that some node depends upon the solution of another node for its solution, but the second node also depends upon the first for its solution. This is a result of planning to gain information, as opposed to creating plans where execution of a step causes some change in the world.

An extremely simple plan with a loop is shown in Figure 19. In this case, there is no way for the top goal to succeed, since it requires itself as a subgoal. However, in Figure 20, the plan has an OR node with a subgoal that can be executed as an alternative to the loop. This executable subnode in the loop is necessary in order for the plan to be executable. It is clear that the structure forming the subnode of the OR node shown in Figure 21 is immaterial, as long as it is an executable subnode. This loop with an executable subnode as part of it will be termed an executable loop.

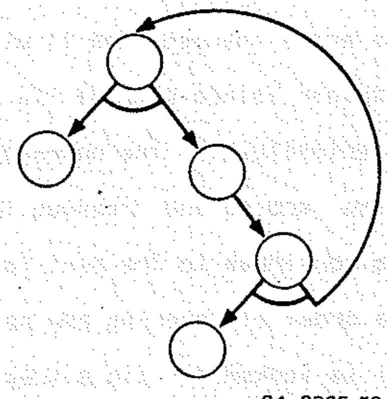


Figure 19. Simple Plan With Loop

Now, let us define "executable subnode" more precisely as either: a tip node; an OR node with at least one executable subnode; an AND (*AND,↑) node with only executable subnodes; or, an executable loop. The process of scoring a plan converges if.

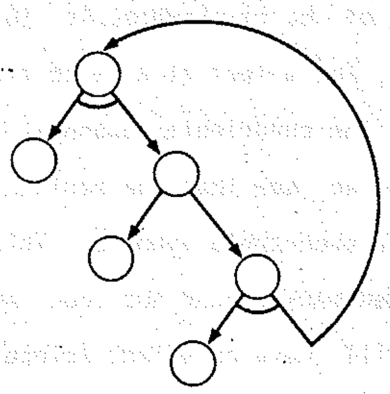


Figure 20. Executable Plan With Loop

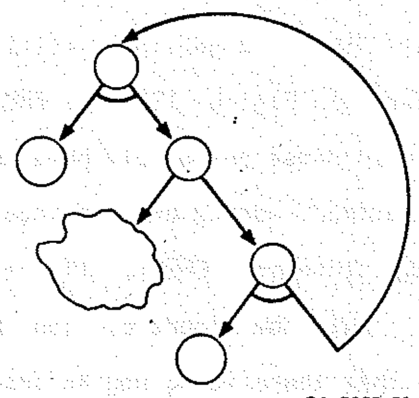


Figure 21. Plan With Arbitrary Executable Subnode

a) the score of OR nodes is non-increasing; and,

b) for each loop in the plan, there is OR node in the loop, such that the OR node is an executable subnode.

The effect of b, of course, is to ensure that the loop is executable. The effect of a is to clamp the score at a maximum provided by the executable subnode. If we look at Figure 20, for example, we see that the maximum score of the OR node is fixed by its executable subnode. This means that if this score is propagated back around the loop, and feeds into the OR node again, it will have no effect (since the score, which is the effective cost, must go up as the loop is traversed). This clamping effect prevents infinite looping in the algorithm.

In the case where the plan has a loop which is not executable, the algorithm still terminates (without converging). In Figure 19, the AND nodes cannot be scored since their subnodes do not all have scores. They are removed from CHECK without adding any new nodes to the list. After a single pass through CHECK, it is empty, the algorithm terminates, but the score of the plan is still infinite, indicating that there is no way to ever satisfy the top goal.

4. Execution

After plan scoring is complete, the execution program selects the best executable subgoal, and evaluates it. The success or failure is propagated upwards to all parents, and any related subgoals are dealt with also. If the strategy has not succeeded or failed at this point, the planner calls the scorer to rescore the net and then continues with the next goal.

To do this, the executor starts at the top goal of the net, and proceeds downwards through the subgoals. Wherever there is an option, it chooses the subgoal with the lowest score. When it finds a node where the best subgoal is executable, it passes that subgoal to an evaluation function. This function executes the subgoal and propagates the success or failure backwards. It keeps track of subgoals that were executed, along with their outcomes on a special list.

The propagation of failure is straightforward. If any subgoal of an AND or *AND goal fails, then the goal itself also fails and continues the propagation upward. A subgoal of an OR node that fails is merely removed from the list of subgoals. When any subgoal list is reduced to a single subgoal, the operator is replaced with the ↑ operator which always propagates the outcome of the subgoal directly upward. When a subgoal succeeds, its a priori confidence value, C, is replaced by the a posteriori confidence, either $C/P(T)$ (i.e., $P(X|T)$), or a value computed by the module. The cost is set to zero, $P(T)$ to one, and the parameters of its predecessors are adjusted to reflect the outcome. If the first subgoal of an AND node succeeds, then the confidence of the node becomes

$$[\prod_{i=1}^{n} C_i]/P(T_1) .$$

After the k^{th} node succeeds, the confidence is

$$[\prod_{i=1}^{n} C_i]/[\prod_{i=1}^{k} P(T_i)].$$

The remaining cost after success of the k^{th} node is

$$\sum_{i=k+1}^{n} [K_i \prod_{j=k+1}^{i-1} P(T_j)].$$

Since the confidence is increasing, and the cost decreasing, the score of the node decreases. This means that the subgoals of the node will be executed in order as long as they are successful. The final confidence of a successful AND node is

$$[\prod_{i=1}^{n} C_i]/[\prod_{i=1}^{n} P(T_i)]$$

If a subgoal of an OR node succeeds, then the OR node itself succeeds with a posteriori confidence, C/P(T) (or a value computed by the module). This is propagated back along with the cost (which is zero) and P(T) (which is one) to the parents of the OR node.

After propagating the outcome of a goal, the executor checks the list of related goals to see if any other action needs to be taken. Since in many cases the outcome of one goal can determine the outcome of another, this is the appropriate point at which to check.

If the outcome of a subgoal does not propagate to the top goal, the net has a new score computed for it in the same way as was done initially, starting with the set of subgoals affected by the last step; execution continues with the next goal.

C. Conclusions

We have described an artificial intelligence planning and execution program that is able to make cost effective decisions about what to do next in executing a plan. We have described in some detail the effects of a particular score function (i.e., K/C). Another possibility which could be explored is to treat an OR node as if there were a discrete cost-confidence function associated with it. That is, considering each subgoal, there is a certain reliability to be expected for a certain cost. We can consider these cost vs. reliability values as points on a curve, and the complete set provides a cost-confidence function. The function can then be propagated upward instead of making a decision about what the score should be at the local node level. This way, the parameters can be passed up to the executor which could then decide (using a variety of criteria) just what should be done. With this kind of information available, the executor could use budgetary considerations (how much resource is available to expend for the job) or considerations about required reliability.

V EXPERIMENTS

A. Introduction

In this chapter we will discuss some experiments with the system. These experiments were designed to illustrate a number of important points: the overall operation of the system on a variety of scenes, the options the system has at various stages of execution, the results obtained from the execution of plan steps, and the effects of unexpected objects on the plan.

Before discussing the examples, we would like to establish the context in which the experiments were performed. We begin with a discussion of the equipment used, and the data available from it. After describing the primitive operators and object represention, we will be in a position to discuss the experiments.

Chapter V EXPERIMENTS

1. Equipment

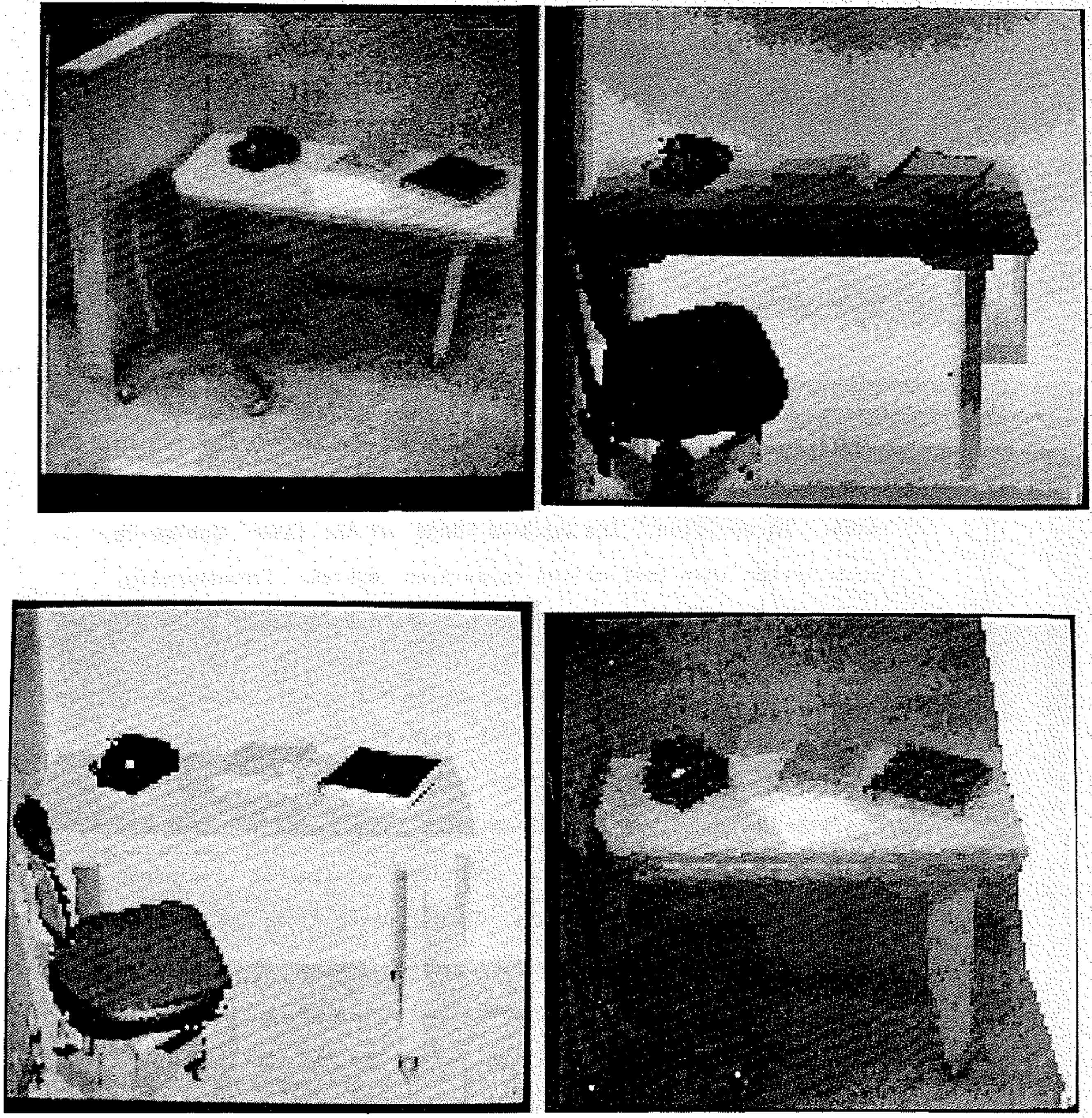
The set of images making up a complete picture are obtained from two different devices. A set of four television images[*], taken through red, green, blue and neutral density filters provide information required to compute brightness, hue and saturation. The neutral density (i.e., black and white) television image of a scene is shown in Figure 22-a. This set is complemented by a range finder image of the same scene. This range picture, consisting of measurements of the distance from the range finder center to each elementof the image, and associated calibration matrices, allows the system to compute the X,Y,Z (i.e., world) coordinates of each point in the image. From these coordinates, the system obtains height (Z) and local surface orientation.

The range finder[5] is a scanned device which measures the phase difference between the reflection of a modulated laser beam and a reference beam. This provides the time-of-flight information required to compute the distance to the reflecting point. Figure 22-b shows a range picture displayed so that near points appear darker, and more distant ones, lighter.

In addition to computing range, the instument also measures the amplitude of the reflected signal. After a

---

* Also referred to as "brightness" images.



(c) (d)

Figure 22. Television and Range Finder Images

normalization to eliminate inverse square law variations[**], the amplitude measurement provides an accurate measurement of reflectivity of the point at the laser wavelength[*] (see Figure 22-c). The amplitude picture can be treated as another brightness (i.e., TV) picture, but one with significant advantages over normal television images. There is perfect registration with the range picture (the television and range images are not perfectly registered since they are taken from two slightly different views of the scene). There are also no shadows since the transmitting and receiving optics are the same. In addition, the dynamic range of the laser system is much better than that of the television system. Consequently, the black and white television picture was replaced with the laser amplitude picture.

The two separate optical systems employed for the television and range pictures require that the images be registered. This is accomplished by transforming the television picture into the coordinate system of the range finder (as shown in Figure 22-d), using the associated calibration matrices calculated before taking the pictures. Calibration of the instruments, which is done before the pictures are taken, allows the system to compute the X,Y,Z

** This normalization, and another for Lambertian reflection are described in a forthcoming paper by Nitzan and Duda[6]

* He-Ne at .6328 microns.

coordinates of every image point in the range array, and the picture coordinates of each point in the scene. These transformations are used later to compute windows in the image.

We would prefer that the images be perfectly registered, but as we see in Figure 22-d), this is not the case. The centers of the two optical systems are separated by eighteen inches (with the objects about about ten feet away), resulting in some parallax discrepancies. In addition, the areas viewed by the two devices do not exactly overlap. The white strip on the right side of the image is that part of the scene visible to the rangefinder, but not to the camera. We will confine our attention to the overlapped area, and therefore have both range and brightness information available. In addition, since we are generally interested in surfaces, rather than isolated points, parallax errors and slight misregistrations do not bother the program*.

In the remainder of this chapter, examples will be illustrated using amplitude pictures. These are more distinct and easier to interpret (in black and white) than the television pictures. Points in the image will be indicated as small light or dark (depending on the background) squares. Regions will be shown as strongly outlined areas in the image.

---

* Even when the program filters point sets, there are generally several points related from any given surface. By loading at these, the average characteristics of the point set tend to climate, and slight discrepancies will tend to cancel.

2. Operators

a. Detectors

The detectors used in these experiments are indicated in Table I the attributes measured include brightness, hue, saturation, height and local surface orientation. The table gives the average cost (in milliseconds) of measuring the value for a single sample*. A detector is simply a predicate that determines whether a measured attribute value lies within a specified interval. The detector returns true (T) when the value is within the limits, false (NIL) otherwise.

Detector	Cost
BGHT (brightness)	76
HUE	101
SAT (saturation)	95
RANGE	99
HEIGHT	110
ORIENT	145
EDGEP (edge op)	70

Table I. Available Detectors

Cost is normally defined as the expected execution time required to take the measurement. For these examples, we have

---

* A sample generally implies a single point, although for computing orientation, it will mean a small patch of points.

used cost values for range derived attributes that are far below actual current costs. Taking a range finder picture is a lengthy process. Before the measurement at a point is taken, the laser is allowed to settle for a half second. Therefore, taking a 128 by 128 range picture means that over two hours is spent waiting to take data. In addition, the measurment time is proportional to strength of the signal return (to improve the signal-to-noise ratio of the device), resulting in a total time for a range picture between three and five hours. Obviously, if this remained the case, no sensible system would use range, except as a last resort. However, since we expect the whole process (which is still experimental) to be greatly improved, we elected to compute cost as the average time to take the range measurement (not including settling time) plus the time requested for any associated computation.

The confidence of a detector is computed from the attribute value histograms for each object, as described in Appendix I. The system chooses between competing alternatives on the basis of the score computed from cost and confidence as described in Chapter IV.

b. Primitive Operators

To apply detectors to an image, the system has a number of primitive programs available. These are listed in Table II categorized by function. Briefly, the system can use

Acquisition	Validation	Bounding
FILTER-WINDOW	VALIDATE	HRB
SCAN-UP	DISTINGUISH	VRB
SCAN-DOWN		GROW-REGION

Table II. Program Primitives

FILTER-WINDOW and SCAN for acquisition. FILTER-WINDOW examines a set of points in a window of the picture, and returns all those that are accepted by a particular detector. SCAN* requires a previously located object for a starting point, and sequentially examines points on a line segment, searching for the target object.

To verify acquisition, the system uses DISTINGUISH OR VALIDATE. DISTINGUISH classifies each point as belonging to one of the objects that might be ambiguous with respect to the acquisition detector. If a point is classified as the target object with sufficient likelihood (currently the classification likelihood must be greater than .8), the point is retained, otherwise it is deleted from the set. VALIDATE checks remaining (i.e. unchecked) attributes to ensure that their values fall within range for the object.

* In these experiments, the system can use SCAN-UP or SCAN-DOWN.

After the object has been located, the system can compute its outline in a variety of ways. The horizontal or vertical rectangle bounders (HRB and VRB) are useful for appropriately shaped objects. These programs scan for an edge of the object (using the edge operator), predict and locate the perpendicular edges, and then locate the last edge of the boundary. Alternatively, the system may elect to use GROW-REGION to extend the initially acquired points to the boundary, and then generate the boundary with the convex hull routine, HULL*.

3. Relationships

In these experiments, the system knows about a small number of object relationships. These include, ABOVE, BELOW, BEHIND, IN-FRONT-OF, SUPPORTS, SUPPORTED-BY, AND ADJACENT**.

Associated with the relationships are the likelihoods that the relation holds. A typical relationship is TELEPHONE SUPPORTED-BY TABLE. Now if TELEPHONE is in the scene, the probability that it is supported by TABLE is supplied by the likelihood on the relationship.

----

* The use of HULL, while not always appropriate is motivated by the fact that most of the objects in our experimental world are convex, and HULL provides a simple, cheap way of generating this outline (see Appendix 2).

** BEHIND, SUPPORTS, and SUPPORTED-BY imply ADJACENT.

These relations serve two particular purposes in our system. First, they are used to decide if an object is pictorially adjacent to another. That is, whether, in the image, the objects are adjacent. This is useful for SCAN and GROW, both of which normally want to know what objects are adjacent to the one they are interested in.

Relations are also used to compute windows for an object provided by another object. For example, after finding the table, a window consisting of a volume of space above the table can be computed. This window then constrains the location of objects likely to be on the table top. From this window, the system also computes the set of objects likely to be in the window. For example, the window associated with the table top contains all objects likely to be on the table top, plus such pictorially adjacent things as the wall.

These relations are quite simple, and designed to provide useful information from a scene with a certain "normal" viewing position. If the scene were viewed from another angle, they would not be sufficient. There is no conceptual reason for not using a model which could be geometrically transformed to account for any viewing angle. However, this was more complicated and would have been a diversion, without significantly changing the essence of our work.

4. Objects

Table III lists a few of the objects known to the system when these examples were run. The attribute values listed represent the extremes of those values, for each object. The system actually makes its choice of detectors based on histograms of attribute values for each object. Relations and associated likelihoods are also listed for each object.

These object characterizations were generated in an interactive session with the system. The objects were indicated to the system (in various images), by manually outlining them on the display. The system then measured the attributes of the indicated regions, and computed the corresponding histograms. After computing these histograms, the system interrogated the user about related objects, and stored all relationships specified.

Object	Description	Relations
TABLETOP	BGHT: 18 to 26 HUE: 26.0 to 58.0 SAT: .23 to .32 HEIGHT: 26.0 27.5 ORIENT: -7.0 to 5.5	SUPPORTS TELEPHONE .6 SUPPORTS BOOK .4 IN-FRONT-OF WALL 1.0
TELEPHONE	BGHT: 4 to 8 HUE: 72.0 125.0 SAT: .15 to .22 HEIGHT: 5.0 to 6.0 ORIENT: -20. to 95.0	SUPPORTED-BY TABLETOP 1.0 IN-FRONT-OF WALL 1.0
PICTURE	BGHT: 7 22 HUE: 15.9 85.6 SAT: .15 .398 HEIGHT: 35.0 47.0 ORIENT: 82.0 92.0	IN-FRONT-OF WALL 1.0 ABOVE TABLETOP 1.0
SEAT	BGHT: 7 to 12 HUE: 110.0 to 146.0 SAT: .42 to .47 HEIGHT: 14.0 15.0 ORIENT: -15.0 to 10.0	BELOW BACK1 .4 BELOW BACK2 .6 IN-FRONT-OF WALL 1.0
BACK1	BGHT: 6 to 9 HUE: 115.0 to 130.0 SAT: .4 to .45 HEIGHT: 18.0 to 28.0 ORIENT: 75.0 to 90.0	ABOVE SEAT 1.0 IN-FRONT-OF WALL 1.0
BACK2	BGHT: 8 to 14 HUE: 75.0 to 240.0 SAT: .24 to .31 HEIGHT: 18.0 to 28.0 ORIENT: 60.0 90.0	ABOVE SEAT 1.0 IN-FRONT-OF WALL 1.0

Table III. Partial List of Objects

B. Operation With Unknown Object

In this first example, the system is requested to locate the telephone the scene shown in Figure 22-c. At this point, the system has no knowledge of notebooks.

A plan is computed for finding the telephone, that contains alternatives at several points. The plan can be summarized as follows:

```
Find the telephone by acquiring, validating, and
bounding the telephone;

  Acquire the telephone by direct or indirect
  means;

    Acquire the telephone directly by filtering
    and distinguishing;

    Acquire the telephone indirectly by finding
    the tabletop, windowing and filtering within
    the window;

      Find the tabletop by acquiring, validating,
      and bounding the tabletop;

        Acquire the tabletop ....
```

When the plan is scored, the best score is provided by the path through "Acquire the telephone indirectly" to "Acquire the tabletop directly" to "Sample the scene at a density of .01

and Filter the scene with the predicate: (HEIGHT 27.0 28.0)* . and distinguish the table top from objects: wall, books, telephone, and tapeholder." This is a reasonable plan, since it is explicitly recognizing the relative sizes and distinguishabilities of the tabletop and the telephone. In addition, the detector for tabletop is taking advantage of its main characteristics, that it is a collection of points at a particular height.

In Figure 23, the system has sampled the scene at the prescribed density. These samples are filtered with the indicated height detector, resulting in the set of points shown in Figure 24. The program has also selected a number of points from other objects. These objects were known to be ambiguous with respect to the detector, and an appropriate step to distinguish them from the tabletop was included in the plan. After distinguishing the tabletop points from the imposters (using (AND (ORIENT -5.0 5.0) (BGHT 19 24))), as shown in Figure 25, the system progresses to the bounding phase of the strategy for locating the tabletop.

This step provides a choice between using the HRB program or region growing. The likelihood of either working correctly is fairly large (close to 1.0), with the region grower slightly more likely to succeed. The real choice here depends on the expected cost.

----

* This detector, which is a simplified representation of a LISP program, requires the HEIGHT of a point to be between 27 and 28 inches.

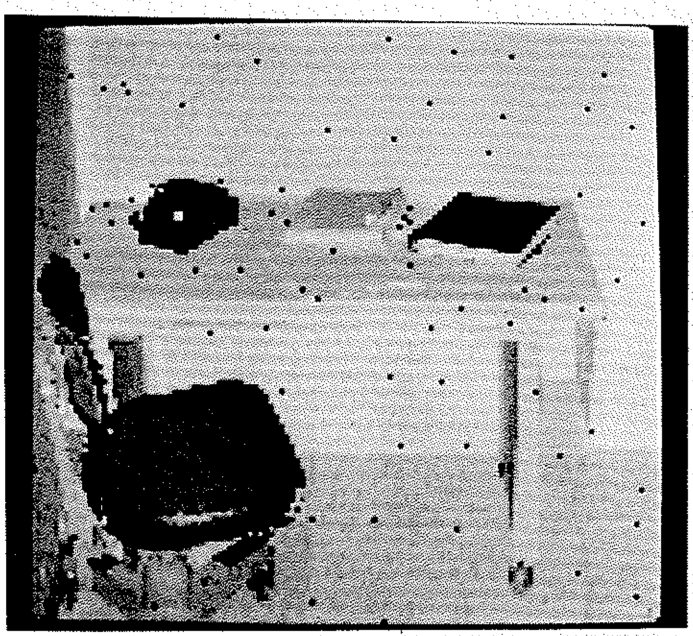


Figure 23. Scene Sampled for Table Top

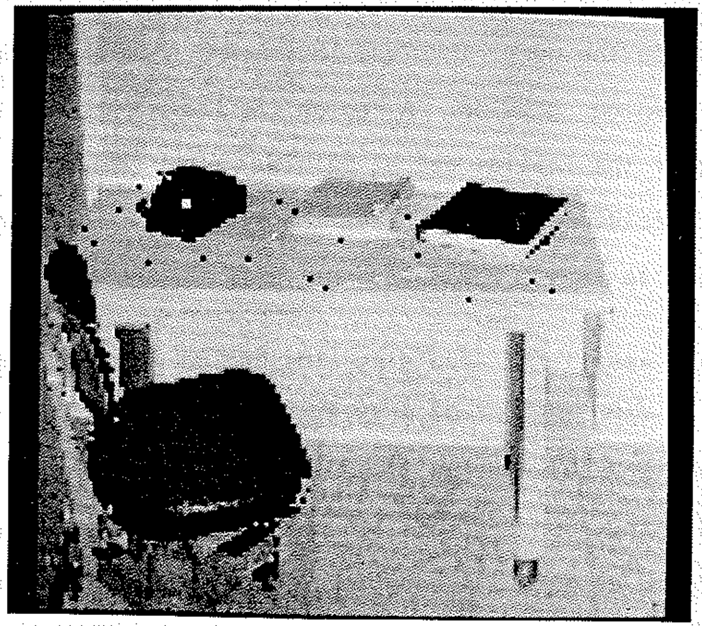


Figure 24. Points at Height of Table Top

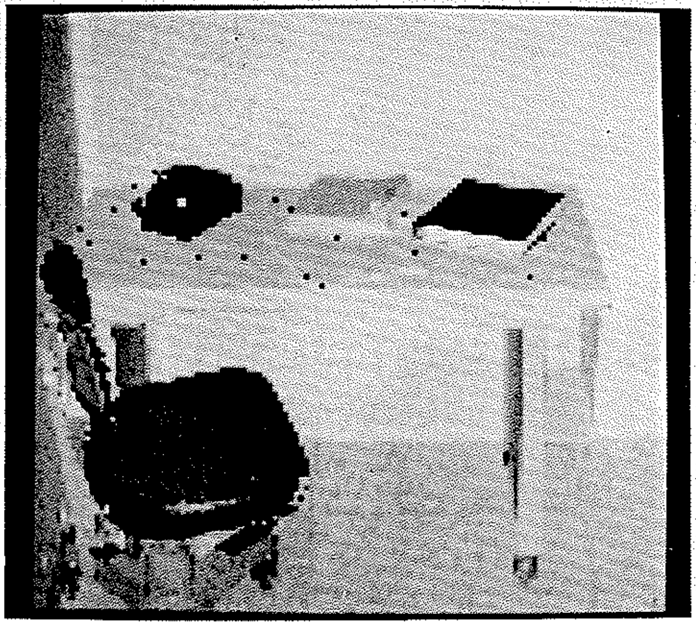


Figure 25. Verified Table Top Points

Due to the work involved in the HRB program, the system opts for region growing. A region is grown outward from the boundary of the initial set of points. Since the set is close to the expected region in size, the system anticipates little extra work to locate the boundary.

The results of applying the region grower, using the predicate, (AND (HEIGHT 27.0 28.0) (ORIENT -5.0 5.0)), to check the height and orientation of each point, and then calling HULL to compute the boundary, are shown in Figure 26. The system is now ready to compute a window for locating the telephone. The window is computed from the boundary of the object that provides the constraint (in this case, the tabletop), the

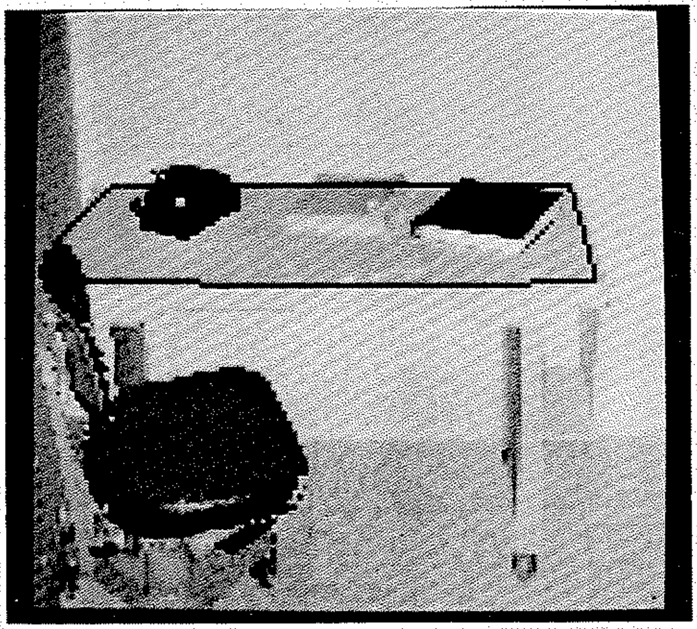


Figure 26. Table Top Outline

direction in which the window extends, the spacing between the region and the window, and the extent of the window. In particular, for locating the telephone, a window is computed upward from the tabletop, with a spacing of zero (since the two objects are adjacent), and extending approximately six inches above the tabletop. The window's projection in the image is computed by finding the X,Y,Z coordinates of the window vertices, and then computing their projections in the image. The region of the scene so defined is shown in Figure 27, overlaid on the defining region to emphasize the derivation. Another important piece of information is the set of objects that might be found within the window. This set is computed at planning time, and consists of all other objects having the

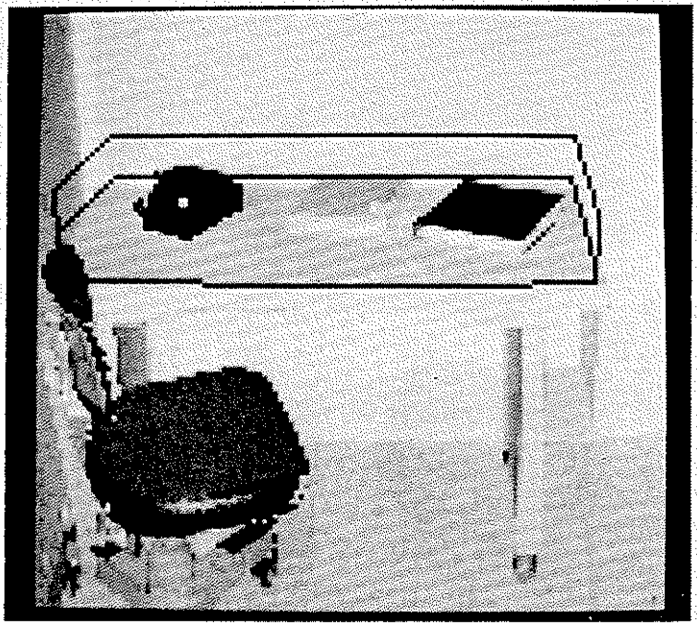


Figure 27. Projected Window for Telephone

same relationship to tabletop as telephone (i.e. SUPPORTED-BY).

'The next step in the plan is to filter the new window for telephone points. The window is sampled, as shown in Figure 28, and filtered selecting all points passing the brightness detector (BGHT 4 7) as shown (in white) in Figure 29. Based on its current knowledge of objects in the environment, the system has mistaken points from the notebook as telephone. In the course of validation, the system is unable to detect its error, and retains all the acquire points. It does notice that the area covered by the points is greater than the size of a telephone, and splits the region, treating the two new regions

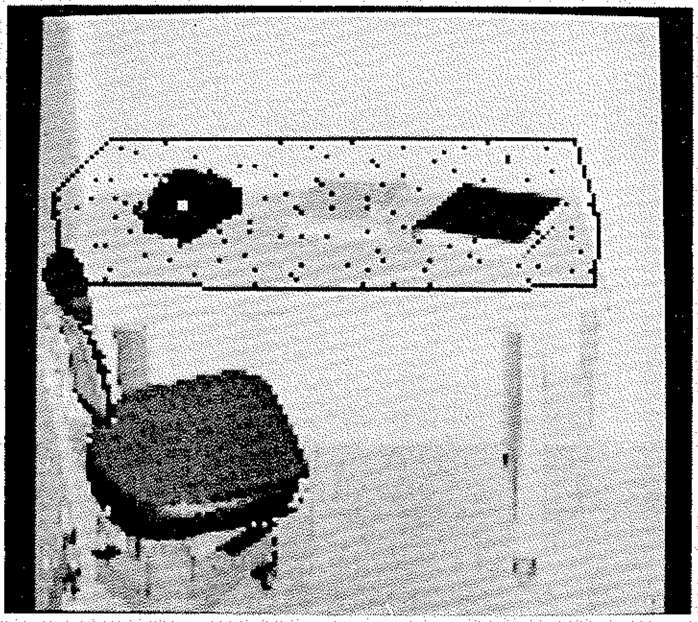


Figure 28. Sampled Window for Telephone

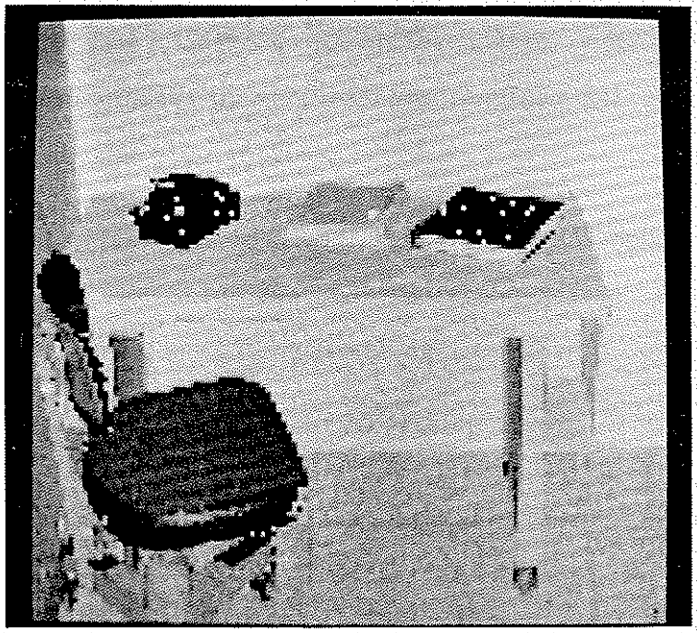


Figure 29. Initial Telephone Acquisition Points With Errors

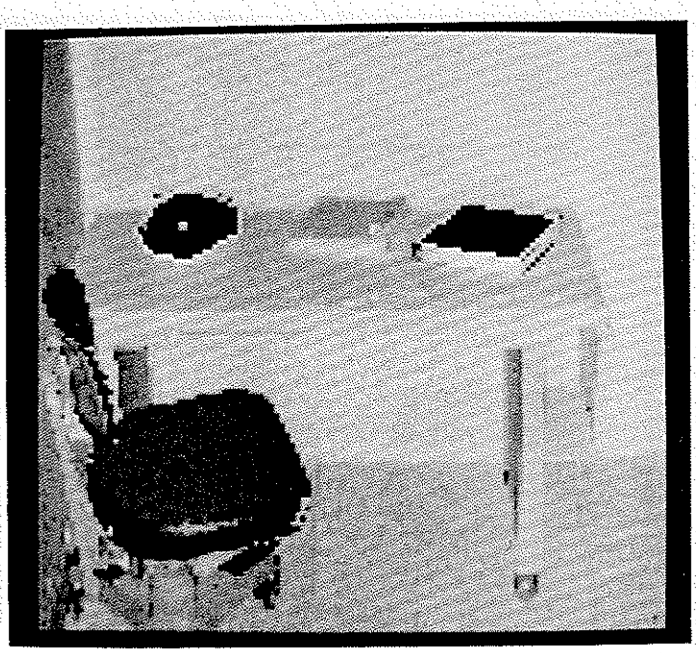


Figure 30. Regions Identified as Telephone

as if they were each possible telephones. In Figure 30 the system grows the regions, and finishes by indentifying them as telephone.

Although the result was incorrect, it really was not an error. The system produced the best plan for the available information. Therefore, to improve the results, the system needs additional knowledge about notebooks. The first step in teaching the system about notebooks is to indicate the region of the image occupied by the notebook to the program, and to supply the appropriate relationships, allowing it to characterize the notebook, exactly as was originally done for the initial set of objects.

Now, armed with a fresh description of a notebook, the system is again requested to locate the telephone. The plan is virtually the same as before (the main difference in the overall plan is to make the direct acquistion of the telephone even less attractive, since now the detector for telephone must check more things, thus requiring even longer execution times). The plan is executed up to the point where the window provided by the tabletop is created and sampled.

This time, the system is aware that notebooks are likely to be in the window, and generates the acquisition predicate, (AND (BGHT 4 7) (ORIENT 60.0 90.0)), to guard against making another mistake. Since the notebook example it has seen had a horizontal surface orientation, the detector for telephone requires points to be off the horizontal. This test effectively eliminates all but points on the telephone as shown in Figure 31. The final result is again grown out to produce the correct telephone outline in Figure 32.

The error commited by the system in identifying the notebook as a telephone is the type apt to be committed by any system which must rely on a set of rules and information insufficient to the task. What is crucial, however, is the fact that the system was able to incorporate the new knowledge so quickly and gracefully. This incremental acquisition of knowledge is very important to a an intelligent system. It allows the system itself to demonstrate what information it needs, and only that information needs to be added.

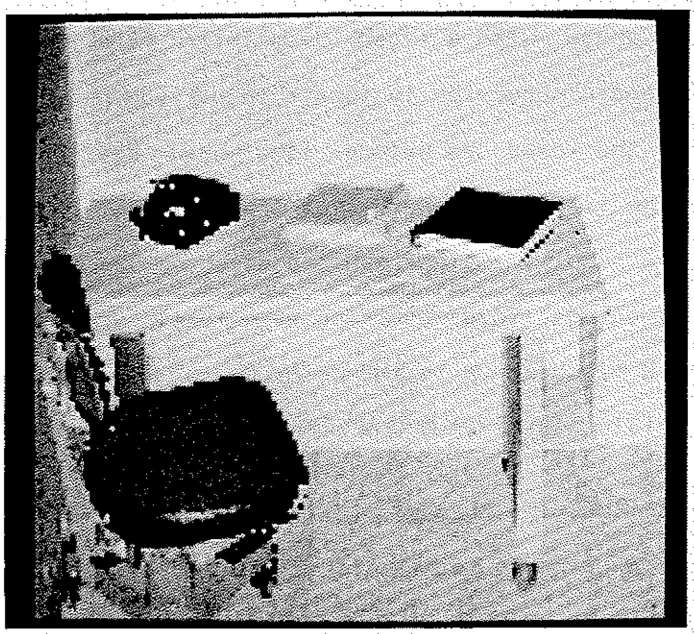


Figure 31. Correctly Acquired Telephone Points

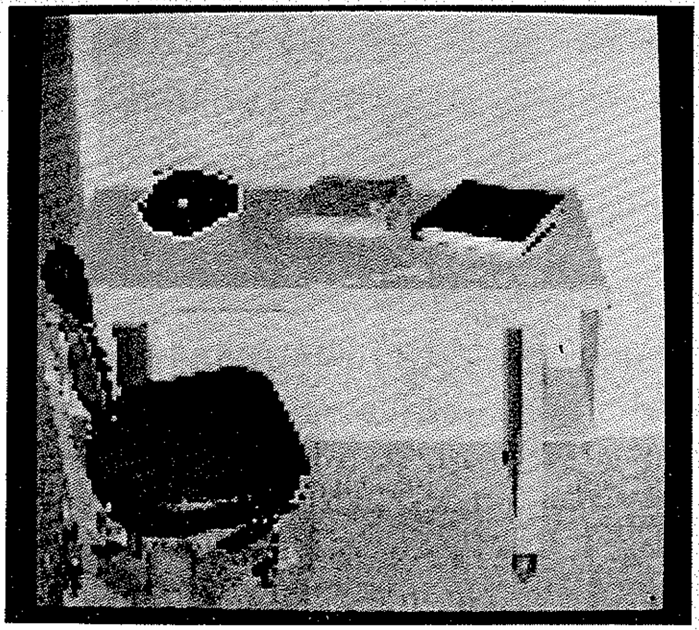


Figure 32. Correct Telephone Outline

The system is incrementally modifiable due to the fact that it generates plans as needed. This allows it to take full advantage of new information. It makes the assimilation of new information into its strategies a natural and automatic operation.

Of course, we must realize the system's knowledge of notebooks comes from a single example, a black notebook with horizontal surface. If the system were again requested to locate the telephone, in a scene containing the notebook standing up, it would probably repeat its earlier mistake.

C. Locating Several Other Objects

1. Chair

In this example, the system is requested to locate the chair in the image in Figure 33. This means finding the seat and the back of the chair. The plan (which is similar to that diagrammed in Figure 16) allows the program to decide which part to look for first. Once one part is located, an indirect strategy can be used for finding the other part.

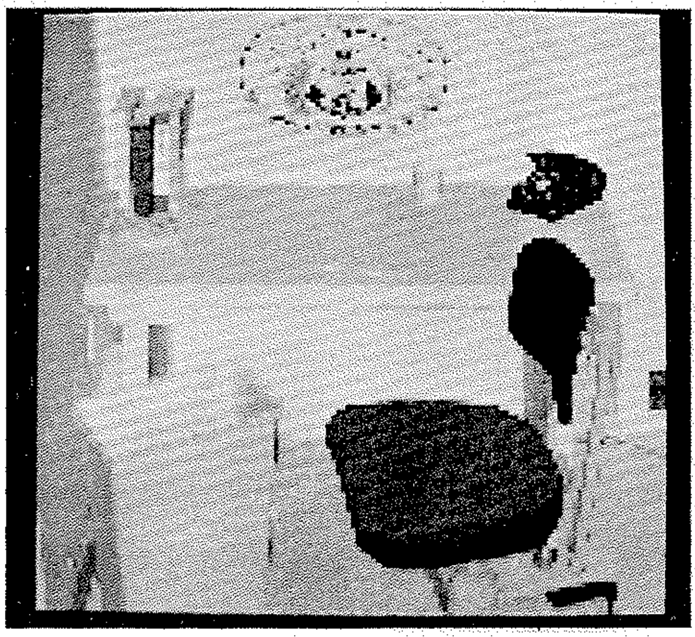


Figure 33. Scene for Locating Chair

The system decides to find the seat first. The difference in scores (38200 for finding the seat, 45300 for the back) implicitly represents a number of differences between the objects. Since, the seat is a horizontal surface, its appearance does not change much as a result of a change in the orientation of the chair. But the appearance of the back changes dramatically with orientation (from back1 to back2). Equally as important, however, is the fact that the seat is more easily distinguished from other objects in the domain than the back.

The acquisition phase begins by sampling the scene as shown in Figure 34. These samples are filtered with a height detector, (HEIGHT 14.0 15.0). The resulting points are shown in Figure 35. The set of acquired points is further distinguished from wall with the detector, (HUE 110.0 146.0), which requires the hue to be blue-green, the known range for the chair seat. The resulting set of points, shown in Figure 36, is grown out to produce the boundary indicated in Figure 37.

Now, the system must decide how to locate the back. It has three alternatives: first to look for the back directly (with a score of 45300); second, to generate an appropriate window and filter it (score of 36600); third, to scan upwards from the seat to try and find the back (with a score of 34000). The third alternative, requiring the least work, is selected.

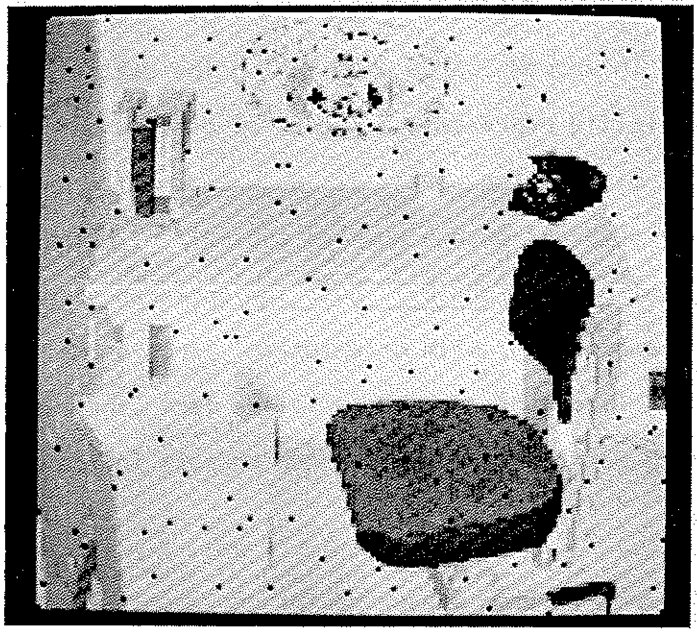


Figure 34. Scene Sampled for Chair Seat

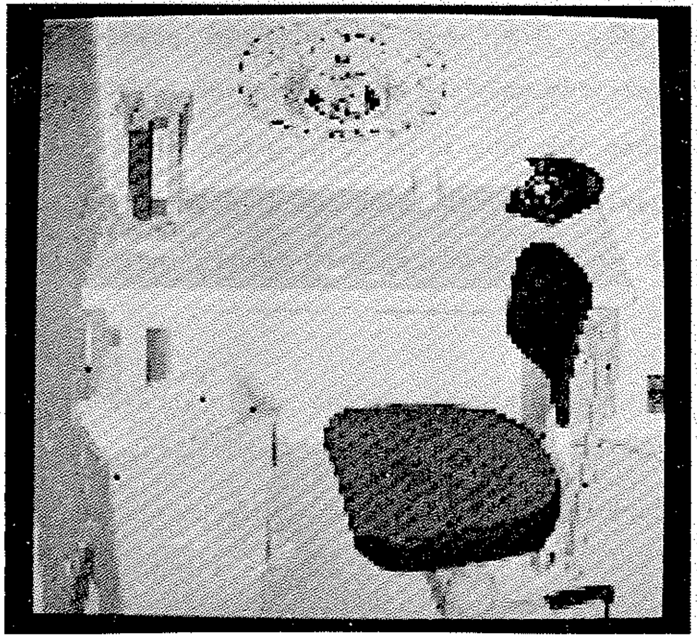


Figure 35. Samples at Height of Chair Seat

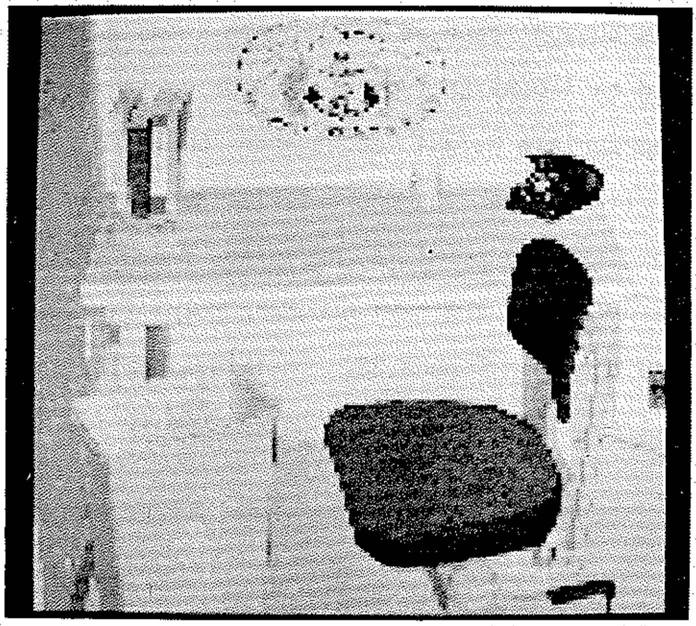


Figure 36. Validated Chair Seat Samples

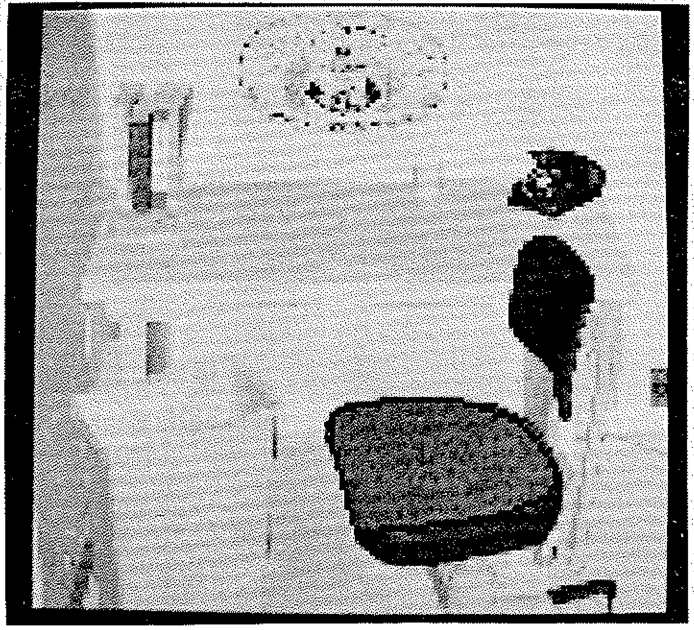


Figure 37. Outline of Chair Seat

The scan is cheaper because the program knows exactly where the back should be (in height). All it need do is to check a point at the appropriate height above each boundary point for the seat. If it matches the characteristics of the back, it is kept, otherwise the point is rejected, that scan terminated, and the point above the next boundary point is examined. The results are shown in Figure 38. In this example, the scans are shown as if they tested many points -- this is for illustrative purposes.

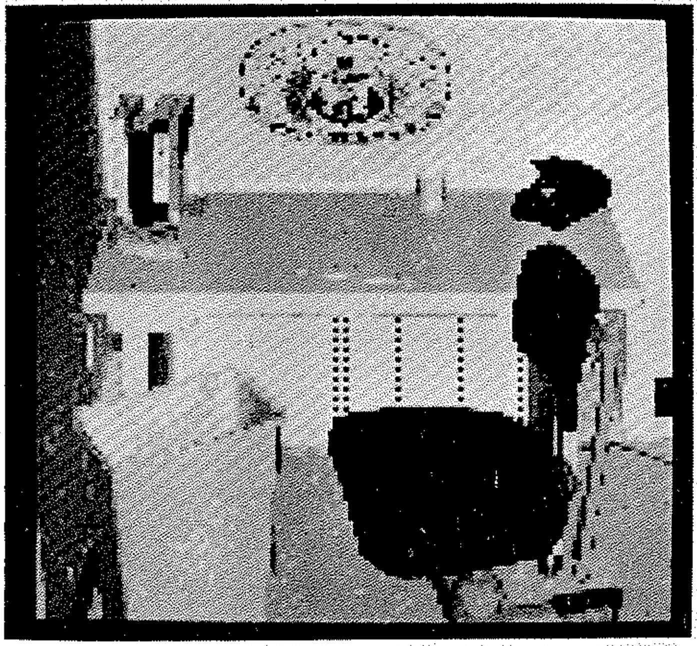


Figure 38. Scanning for Chair Back

After acquiring and verifying the chair back points, the boundary is grown outward, and a convex hull is fitted to it. As can be seen in Figure 39, this boundary is not precise, since it includes area outside the chair back. As mentioned

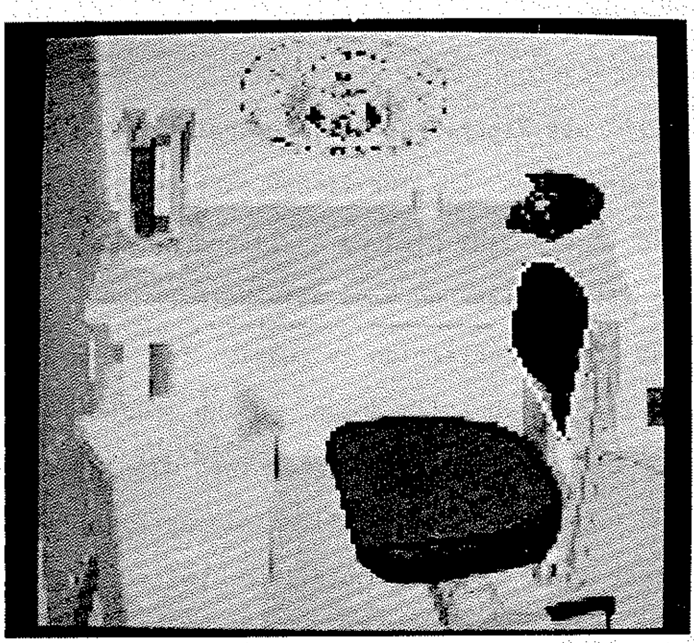


Figure 39. Final Outlines of Chair Seat and Back

previously, the boundary could be made more precise, but for most purposes, the convex hull program quickly generates quite satisfactory boundaries.

This example required the system to find an object with parts, and allowed it to choose the order. It chose the least expensive, most reliable approach. However, had its first attempts failed, it might then have elected to try to find the back directly. In any situation with options, the system will try the most promising one first. If that should fail, it can fall back some of the more difficult approaches.

2. More Objects

We will use Figure 40 to illustrate a few more object finding strategies. In the first strategy, the system is required to locate the picture. It knows what the picture looks like (i.e., its surface attributes), and its general height. The height criterion is useful for constraining the location of the picture (as shown by the acquired samples in Figure 41), but to extract it from the wall, the system is required to use hue and saturation information. The samples retained from the initial acquisition set are grown out to produce the boundary shown in Figure 42.

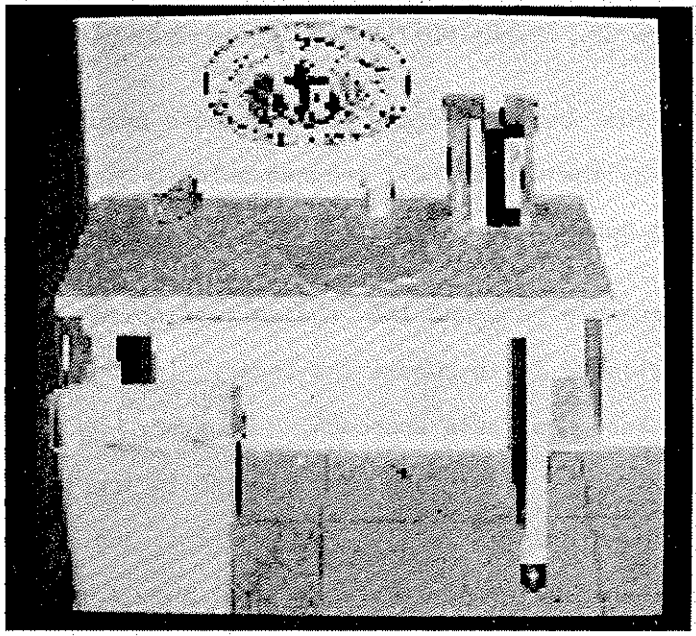


Figure 40. Scene With Several Objects

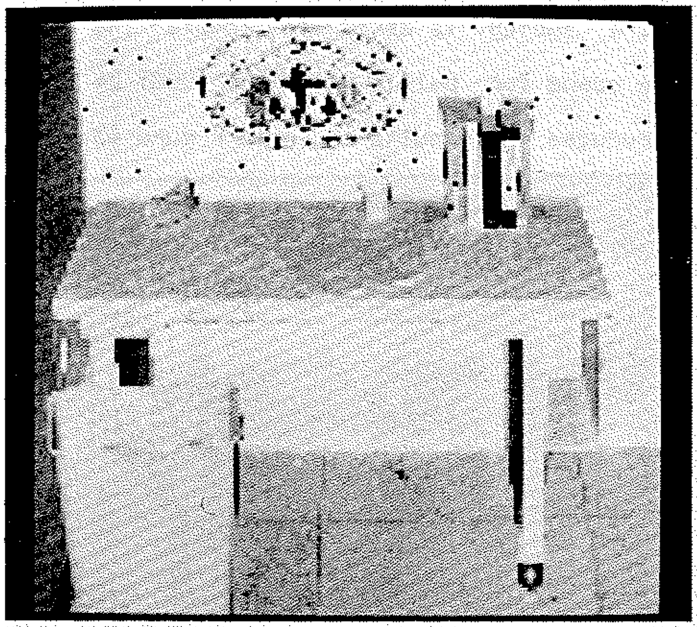


Figure 41. Samples at Approximate Height of Picture

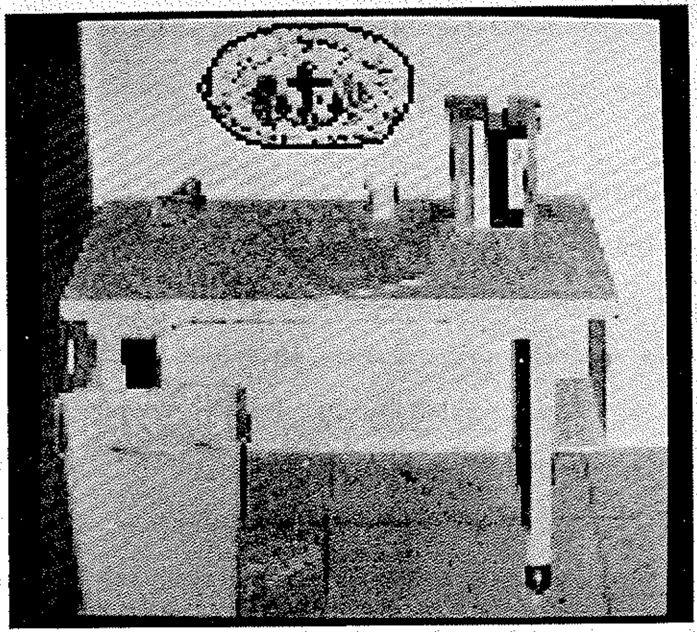


Figure 42. Boundary of Picture

Next the system is asked to locate the wastebasket. Filtering the scene for points with (AND (BGHT 24 28) (HEIGHT 0.0 18.0)) produces the results shown in Figure 43.

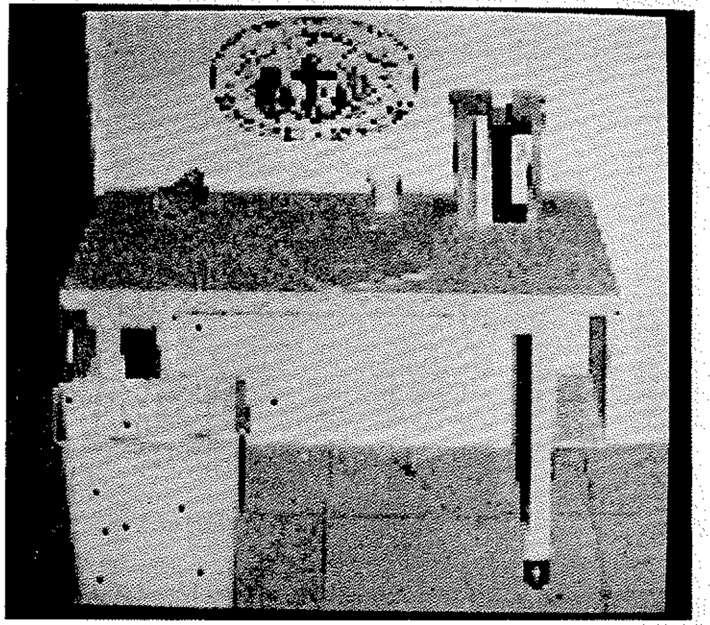


Figure 43. Initial Acquisition Samples for Wastebasket

Unfortunately, there is no easy way of distingushing the wastebasket from the wall on the basis of point attributes. Instead, the system opts to check the horizontal extent of the surface, and accordingly, scans laterally from each point looking for a range discontinuity (with a positive change). When the discontinuity is found, the point is retained, otherwise it is deleted. The points selected are shown in Figure 44. Finally, the surface is grown out with a predicate that checks for height and brightness in the specified range, and range within limits computed from the validated samples.

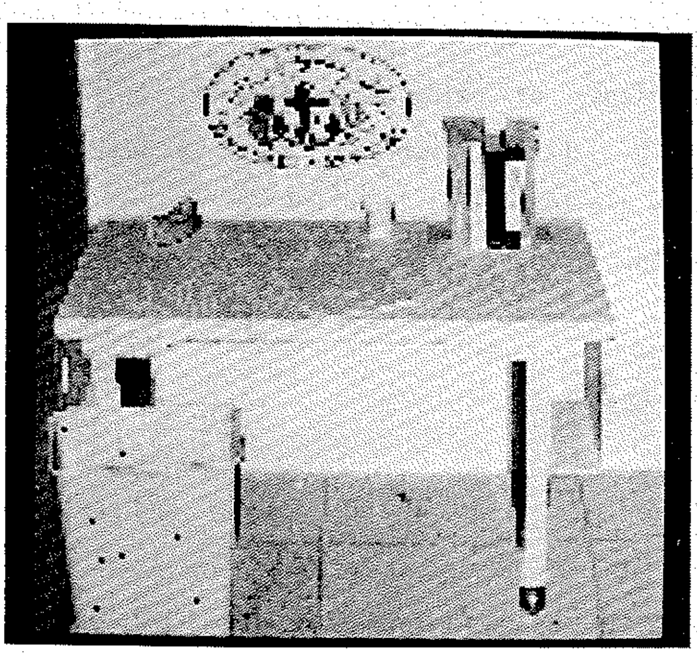


Figure 44. Wastebasket Points Distinguished From Wall Points

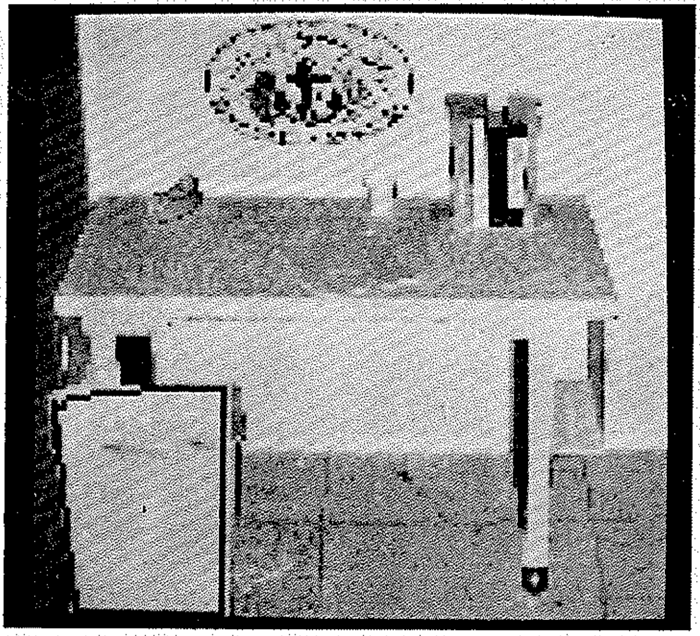


Figure 45. Outline of Wastebasket

The results are shown in Figure 45. This example demonstrates the advantages of being able to use multi-sensory data effectively. It is important to know when local surface attributes are sufficient, and when more global tests must be resorted to.

3. Other Scenes.

Figure 46 shows an example of one of our earlier pictures. These pictures predated the range-finder, and hence used simulated range information. The scenes served as the developing grounds for our system, and therefore should be mentioned here.

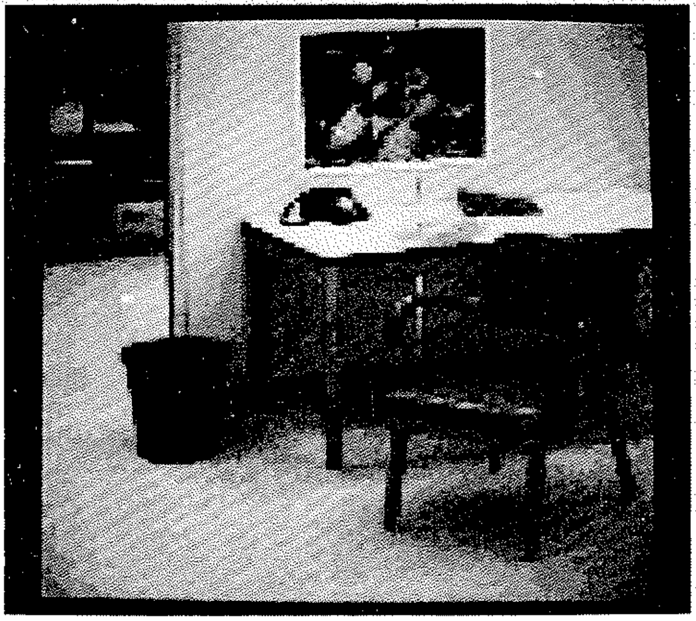


Figure 46. Early Experimental Scene

In these pictures, color information played a much stronger role than in the current set. Detectors tended to be much more complicated, and take longer to run (these strategies are discussed in a paper by Garvey and Tenenbaum[7]). In these pictures we were able to find such things as chairs, table, pictures, telephones, and many other objects. The relative ease of switching from that domain to the current one by merely characterizing the new set of objects is another indication of the flexibility and power of the system.

4. Errors

An interesting question about a system such as this is, "How does it fail?" We have demonstrated one particular type of error that occurs when the system has insufficient knowledge to plan a good strategy. The system sometimes fails to distinguish points on one object from those on another, due to an inadequacy in detectors. In both of these cases, the failures are not inherent within the system, but rather due to inadequate information. The system can be supplied with better information and operators, and it will perform better.

VI CONCLUSION

A. Background and Discussion

1. Introduction

We have described a system that plans and executes cost-effective strategies for locating specified objects. While we feel this is an important contribution to scene analysis research, we are well aware that locating objects in known environments is a somewhat specialized function. In this section, we will compare our work with that of others in an attempt to show how various systems may complement each other in a complete perceptual system. We will point out both strengths and weaknesses in our approach.

2. Robot Vision

Chapter VI

CONCLUSION

The original motivation for this research was to provide a vision system for a robot that could operate in approximately real time. This required a set of specialized perceptual capabilities. In addition, the system had to be fast, and it had to take advantage of the inherent goal-directed nature of the task. The existing systems'[8] approach of analyzing a single exhaustively segmented television picture was unsatisfactory for a number of reasons. First, it was much too slow. Second, images were analysed in their entirety, whether or not the image was similar or identical to one previously processed. Third, segmentation was performed independently of what was likely to be in the scene.

We first thought to remedy this situation by taking a new image every few inches, as the robot moved. This approach promised several advantages. The first image could be analysed fairly completely using available techniques and any unresolved areas could be left for later analysis. A map (model) of the scene would be created over a period of time. Each new image would be scanned rapidly to ensure that "known" areas had not changed. Then major computing resources could be devoted to processing "new" areas. After several images had been processed, it was expected that a good map would be produced, and very little effort would be required to incorporate new information. Furthermore, motion parallax was expected to provide three dimensional information. A form of this type of visual processing -- which we called "incremental vision", due to the gradual way new information was acquired -- later came to be known as "verification vision"[9]. The success of a verification vision system rests on its ability to confirm expectations about an image with minimal processing. The object finding strategies generated by our system seem well suited for this purpose.

3. Goal Directed Vision

A fundamental question in perception is that of control. Organic systems seem to utilize a combination of data directed (i.e., bottom-up) and goal directed (i.e., top-down) control strategies.

Our robot used perception primarily in support of specific tasks (e.g., navigation or object location). It could thus use goal orientation to be intelligent about how it processed sensory information, and focus attention on areas of the scene containing material relevant to those specific tasks. When support for robot research waned, we deemphasized those aspects of our research concerning mobility and concentrated instead on investigating goal-directed, top-down vision.

Top-down systems had been previously developed in various scene domains. Kelly[10] and Kanade[11] had developed systems for face recognition. Duda and Hart[12] had created a system using decision trees to parse room scenes. Shirai[13] described a top-down system for directing the location of edges in scenes of blocks. Recently, Lieberman and Bajczy[2] generated a system for analyzing several landscape scenes. These systems had a common shortcoming: they all executed rigid sequential strategies, recognizing things in fixed order.

These inflexible systems were not satisfactory for our work for three important reasons. First, they could not tolerate errors; if a step early in the process failed, the entire process was likely to fail. Second, they were difficult to modify; the addition of a single new object would often invalidate the original strategy. They were tailored for particular contexts, and could not be used in others without major reprogramming. Third, the systems could not use restricted contexts to their advantage; without this type of dynamic computational limitation, the programs were restricted to limited, static environments.

There are two major approaches to overcoming some of these disadvantages. The first is that of global optimization. Instead of pursuing each piece of information sequentially, all possible information is considered simultaneously in an attempt to arrive at the best possible interpretation of the scene, within a set of constraints. Fischler[14] used this approach in a system for recognizing faces. He achieved flexibility by treating the face descriptions as data to a general optimization procedure. In another system for analyzing natural scenes, Yakimovsky[1] globally evaluated scene interpretations semantically control a region growing system to produce an optimal partition of the picture. Montinari[15] has investigated the theoretical properties of optimization methods in satisfying a network of contraints.

The second approach, which we adopted, is to dynamically generate sequential plans for solving a particular problem. The use of a sequential procedure allows initial steps in a plan to provide information for use by subsequent steps. In particular, it allows the concentration of resources on successively refined areas of the image. By creating plans designed to operate in the current context, a flexible control structure results, enabling the system to deal effectively with dynamic environments.

Our research was based on the concept of "vision by distinguishing features." Object recognition via distinguishing features is performed by looking only for those features that differentiate the target object from other known objects in a particular context. Most of the image can be quickly eliminated from consideration. Only those areas that have the distinguishing characteristic need be processed further.

The distinguishing features approach eliminates the problems mentioned above. The actual features used in a strategy are computed when needed, by comparing object descriptions and noting the differences. This allows both for easy addition of new objects and straightforward extension to new sensory modalities. The detectors for the selected features are dynamically ordered to choose a good, cost-effective sequence for reducing context. This results in a strategy where simple features (in the current context) are checked first, and more expensive tests deferred until the context is sufficiently restrictive to warrant their use. Finally, if unexpected problems or errors occur, the system can replan the strategy using any new information.

4. Perceptual Strategies

At the time this work was begun, there was a great deal of interest in languages containing specific features for problem solving. Some of these languages, such as PLANNER[16] and QLISP[17], had interesting data structures and control mechanisms, but the available implementations were too inefficient and cumbersome (due mainly to a level of generality in the system which we did not require for our work), and, additionally, were not completely debugged.

More importantly, however, there are several points that differentiate perceptual strategies from other types of problem solving strategies. First, they are primarily aimed towards information gathering, and therefore are often required in situations where insufficient data is available to completely plan the strategy. It is often necessary to resort to incremental planning, where an incomplete plan is generated, executed, the results monitored, and the strategy either extended or replanned, based on the results. Second, as opposed to other types of problem solving, where any solution is acceptable, cost-effective solutions are required in a perceptual system. A perceptual system can conceivably generate a vast number of execution states; ways to quickly eliminate most of these become critical.

Chapter VI CONCLUSION

No available system was able to intersperse planning and execution effectively. Furthermore, there were no systems that allowed organization of plans based on cost-effectiveness. We, therefore, designed and implemented our own system, including such concepts as pattern directed function invocation, antecedent theorems, backtracking, and procedural embedding of knowledge, which were relatively easy to include in LISP.

Other perceptual systems utilizing cost-effective concepts of planning and inference have been devised. Turner[18] demonstrated a scene analysis system that assigned processes to each part of an object to be recognized and performed a global reordering of tasks in order to choose the best thing to do next. This reordering was not strictly cost-effective, but was done on the basis of such facts such as whether the module needed preconditions satisfied, and how many other modules were awaiting its results. Any complete system would have to take these into account. Sproull[19] and Coles[20] are investigating the problem of planning in an uncertain environment, where one must decide when it is advantageous to seek new information. This is an important question when the cost of new information is non-zero, as in a robot system.

In addition, this type of processing may arise in other areas. For example, decision analysis[21] is concerned with rational decision making in uncertain situations. Typically, one important decision is whether to seek new information. This is handled by estimating the value of the information, both in the best case, and in the expected case. Another area where cost-effective planning could prove useful is medical diagnosis. It is often important to estimate the relative utility of tests, particularly as they become more expensive. A technique such as ours which tries to maximize the return from inexpensive tests before resorting to more complicated ones could prove very useful.

5. Distinguishing Features Representation

A fundamental problem in perception is that of representation. Previous work in this area (e.g. see Guzman[22], Waltz[3] or Agin and Binford[23]) has emphasized precise characterization of a few particular attributes, such as shape, texture, or structural descriptions. In a cluttered domain containing a variety of objects, such as ours, these are not cost-effective to measure. The lack of adequate shape descriptions is currently a great limitation. In addition, it is often difficult to characterize natural objects (e.g., a tree) with these kinds of descriptions. In contrast, our distinguishing features representation relies on many redundant features which can be loosely specified, as opposed to one very precise description. These descriptions need not be complete (and, in fact, usually are not), but must only be sufficient to distinguish between objects. A set of distinguishing features is usually cheaper to measure than a single precisely specified one. More importantly, however, our representation is usually more robust, since it is easier to match a number of coarsely specified features to an object (which could appear in a variety of situations and attitudes, or even be partially obscured by other objects) in an image, than a more precise model. Since features for recognizing an object are computed as needed, they can provide the strongest discrimination for the context.

Chapter VI CONCLUSION

Although our implementation concentrates mainly on the use of local surface attributes, this is not an intrinsic limitation of the distinguishing features concept. Distinguishing representatives of more complicated features also fit well into the representation. For example, a function that measures P^2/A (i.e., perimeter squared divided by area) provides a shape test that will distinguish squares from rectangles (or notebooks from telephones).

Although it is often easy to see things based on a number of simple attributes, this is not always the case. There will always be cases when the available, simple features will not suffice, and more complicated representations must be used. When the only attribute that distinguishes between two objects is their shape, then shape must be measured, and the finer the distinction, the more precise the measurement must be. However, this is the appropriate point to make this measurement, when all other features have been exhausted.

It is important to realize that the usefulness of distinguishing features representation is strongly dependent on the state of knowledge of the context. Representations are created to distinguish objects from others known to the system. It is easy to fool the system by adding objects to the scene that are similar in appearance to known objects. In this situation, it is likely that the system will create representations that will not discriminate between these objects.

6. Multi-sensory Systems

Discrimination is much easier when inputs from many sensory channels are used, instead of attempting to extract all the information from a single source. However, it is important to be able to coordinate their use effectively, otherwise, the sensory system will be overloaded. We have shown that the distinguishing features representation provides a good framework for this coordination.

This system was originally designed to provide sensory inputs for a robot. These are typically visual-type inputs, but they could come from a variety of sources, such as auditory, tactile, or electromagnetic stimuli. The telephone, for example, might be distinguished from other black, desk-top objects as an emitter of sound; a cup of coffee as an emitter of heat; a person would be distinguished from other large objects in a room by his movement. Each of these attributes is crude and easy to measure, given appropriate sensors. The coordination of inputs of this sort would be handled by our system precisely as the more readily obtainable color and range inputs were.

7. Interactive Vision

In a research area as complex as that of interpretation of natural scenes, it is impossible to completely formulate programs and strategies in advance. Perception is still an experimental science, and, although central ideas can be established, verifiction of the actual details are possible only through trial. A key to experimentation in machine perception is to be able to see data as the machine does. This allows the experimenter to perform operations on the data and view the results directly, thus eliminating a level of interpretation normally required. By providing a set of perceptual operations, the user is elevated above the level of specifying details, thereby facilitating communication between him and the machine, and allowing him to concentrate on the task. The major function of our interactive system is this facilitation of communication between the man and the machine.

B. Extensions

Many possible extensions are possible (and necessary) to allow this system to function as part of a complete perceptual system. We will first discuss direct extensions to the work reported here, and then look into areas that, while interesting, would have diverted us from our principal research topics.

1. Extensions to the Existing System

a. Window Operators

Most of our perceptual operators look only at small (typically one to nine pixel) image samples. By extending the concept of a detector to include arbitrary functions of surface data (i.e., intensity and range) over a window, we could anticipate adapting our system to use very complex attributes. Texture detectors could be written using statistical measures or Fourier operators. These detectors might then be used to locate surfaces which would appear too noisy to be adequately discriminated by our current detectors. Shapes of varying complexity could be recognized with special templates. It should be possible to use an edge operator such as the Heuckel operator[24] as an acquisition tool, for example. Functions to detect variations in shading could provide clues about surface structure (for example, a highlight detector might provide a distinguishing test for a metallic surface).

For acquisition, our system now measures specific attributes of single sample points. If these samples were small patches of points over which average attribute values were computed, the values would be more representative of the actual surface attributes of objects. A, perhaps better, approach would be to use the average attribute values of regions resulting from a crude, bottom-up segmentation, as acquisition data.

b. Strategy Extensions

Although our cost-effective strategies have proved quite effective, we see many ways in which they could be improved. For example, we have been interested in more equitable ways for distributing the cost of a subgoal over all interested goals. Now, the cost is borne by the first goal to require the subgoal's execution. For planning purposes, however, it would seem better to divide the cost of the subgoal by the number of interested goals, and pass the partial costs back individually.

One problem with this is deciding which of the parent goals will actually be executed (since many goals are generated to provide redundancy). A modification to overcome the problem might distribute costs only among those goals that will probably be executed (that is, the ones appearing most promising). Unfortunately, whether a goal will probably be executed depends partly upon its cost, so we cannot determine one without the other. The actual assessment of who should pay for what becomes quite complicated.

While it is clear that cost-effective execution is an important concept*, it is not sufficient to handle all the problems faced by a perceptual system. We would prefer to use the cost-effective information as part of a set of parameters governing plan execution. Other parameters could include an interest measure. This would reflect the relative importance of the goal to the solution of the main goal. In turn, the main goal might have an interest level commensurate with its level of importance to the overall system. This would allow the main system utilizing the perceptual system to provide a set of goals to be worked on as resources were available. Information gathering tasks where the results were not immediately required, could then be set up as goals with a low interest level. If resources later became available, these goals could be worked on. Alternatively, interest levels could

---

* The cost-effective approach usually chooses an order of execution close to what the user would choose.

be raised if the goal suddenly became important, or lowered (perhaps to zero) if the goal was determined to be less important.

Another parameter might be a budget that was allocated to subgoals (recursively). The criteria for allocation could include the interest level of the goal, and its expectation of success. We could even anticipate modules that produced varying effort levels for varying levels of "funding."

A problem with cost-effective planning is that it is sometimes difficult to estimate the required parameters with any degree of confidence. While it is usually the case that any estimates are likely to allow the system to perform better than it would with no information at all, clearly the better its guesses, the better it will work. We have relied heavily on models of processes for our probabilistic estimates. While this is obviously of value for certain simple processes, as programs become more complex, or as we add programs where we have only sketchy models for their operation, it is more difficult to find reliable parameters. As a result, we must depend more heavily on measurements of program properties. Unfortunately, the brute force method of executing a program many times on a representative scene will not work, as it fails to take into account the various contexts in which it will be required to operate. The problem of modelling perceptual (and, in general, computer) processes is clearly an area that needs further exploration.

CONCLUSION

Although, people often learn more from failure than success, very few problem solving programs are able to utilize this source of information. Just the fact that a module failed, for example, should suggest something new about the world, but failure modes are typically less well explored (and understood) than modes of success. We see two possible ways in which our system (which also cannot use information about failures) might benefit from failures. The first would be to have a module perform further analysis of the data that caused (or allowed) it to fail. For example, although a filter module failed to find points satisfying the predicate provided to it, it might try to classify the points it examines (which would be costly), or possibly create histograms of attributes of points examined for later use. The system could even flag points that "almost" passed, indicating which attributes caused it to be rejected.

Another way in which failure information could be utilized would be to model the possible failure modes probabilistically. An example utilizing a model (of the process of sampling and filtering) to update a priori likelihoods following a failure is provided in Appendix 3.

Probably the most consistent way to use failure information is treat it the same as success. That is, allow it to alter a cumulative confidence on the the subgoal. This would allow the decision about when to consider a module as having failed to be left up to a higher level program that might be able to bring other information to bear on the question. In addition, the system could then be making decisions about whether to attempt to raise or lower the confidence of a goal.

c. Additions to ISIS

Although ISIS has been an effective tool, our work has indicated several directions for improvement. A major requirement is the upgrading of the data structures. Our region boundary representation, which is useful for describing large polygonal structures (such as we find in our room scenes), is not as useful for describing long, irregular boundaries. The convex hull routine, used to generate boundaries from samples is inadequate except when the region is known to be convex. In our next version of ISIS, we will include boundary following routines. The resulting boundaries will probably be stored in arrays, and utilize some form of chain encoding.

Although INTERLISP[25] provides a rich set of facilities enabling programs to be written and debugged quickly, there are penalties associated with its use. Large amounts of data cause serious space problems. One possible alleviation of the problem involves using forks as additional data areas. A much worse problem, however, is that facilities for arithmetic in INTERLISP are grossly inadequate. CLISP[25] alleviates some of the notational difficulties by providing algebraic notation and control facilities, but the real problems are due to the vast resources (both time and space) required for arithmetic in LISP. One response to this has been to use an algebraic language system, running in a lower fork, to perform as much arithmetic as possible. This has been a workable compromise, but forces us to relinquish some of the flexibility provided by INTERLISP.

Other extensions to ISIS would include the addition of new region routines, such as the semantic region grower developed by Tenenbaum and Weyl[26]. With this program, we can produce a complete segmentation of the scene, if desired. Other areas of ISIS to be upgraded will include the display and interactive portions.

d. New Sensors

As we have shown, the more sensors, the easier it is to see things. To facilitate representing cetain objects, we plan to add new sensors to our repertoire. The first is an extension of the laser range-finder used to provide three dimensional scene information. Range information is provided by measuring time-of-flight of a laser beam. Until recently, the amplitude of the return signal was ignored (primarily due to the low signal-to-noise ratios in the system). Current work, however, is using the amplitude of the return to measure the reflectivity of the surface (at the laser wavelength). The result, after compensation for reflection modes (primarily Lambertian), is an excellent quality grey-level image. This appears to be a useful augmentation of our television images, and may sometimes supplant them.

An extension of this idea is to use tunable lasers, operating at a variety of wavelengths, to provide reflectivities in other wavelengths, and therefore color information about surfaces on which it impinges. It has been suggested[27] that other (micro) surface structure information may also be provided by this technique.

Other possible sensors include acoustic detectors, infrared detectors, and ultrasonic motion detectors. These sensors, which could be interfaced directly, would provide distinguishing tests for many objects.

2. Role in complete perceptual system

Our system was designed for finding specified objects in a known context of other objects. This, of course, examines only one aspect of perception. We would now like to look into possible roles of purposive vision (as exemplified by object location) in more complete perceptual systems.

There are several directions in which the system could be extended to provide a complete scene description, the simplest of which would be to merely instruct the system to locate all known objects. But, this is an undesirable approach, since for any reasonable number of objects, the system will spend most of its time searching for objects that are not even in the image. Instead, we need to look at ways of focussing attention upon areas of the image where the context can be limited to small sets of objects.

The problem of describing a complete scene is probably best accomplished by generating an initial description from preliminary bottom-up analysis. This initial description would then be refined in a top-down manner. Three main advantages would be provided by this combination. First, the system could efficiently direct its resources to constrained areas of the image. Second, an object could "catch its eye". This is not possible in the normal, top-down-only mode of operation of the system, since it only sees objects for which it is looking. Third, the system could bound objects (based on local attributes) that were not known a priori; this, again, is something our current system cannot handle.

An initial segmentation and identification process would supply cues a la Riseman[28] for subsequent stages of processing. These cues would direct attention to key objects, which, in turn, could restrict the context for the location of other objects. New objects located, or new attributes measured would shift attention to themselves, further restricting the context, and further refining the cues for other objects. Cues could also be located by instructing the system to first find major objects, which would then provide cues to focus attention for finding subsequent, harder objects.

The integration of bottom-up with top-down analysis might be accomplished in other ways. For example, we could integrate our system with MSYS[29], a system that utilizes contextual information to constrain relations among objects. By obtaining globally best embeddings of objects in images, it has analyzed room scenes and natural, outdoor scenes. The two systems could cooperate in a variety of ways. MSYS could disambiguate objects that are not distinguishable on the basis of local attributes, but only through contextual relations. Better estimates of confidence in object identifications would be provided by MSYS's analysis of relational constraints. MSYS, in turn, could call upon our system to locate or identify related objects or parts of objects in order to establish the identification of some other object.

3. Concept Learning

We were interested in the idea of concept learning using techniques similar to those exploited by Winston[30] to infer structural descriptions of complex objects. It was likely that perceptual concepts* could be inferred by generalizing descriptions from many examples and counterexamples of objects. These descriptions would emphasize essential features, and eliminate variable ones. By learning from symbolic descriptions, Winston avoided the problem that a model is needed to describe the object used as an example for creating the model**. We would break that cycle, by having our system learn incrementally through interaction with a teacher. Initially, an object would be outlined for the system, which would then make a model based on whatever descriptions of the object it could generate. This model would then be used to distinguish other examples, with the instructor interacting when the system failed. By comparing new objects with models of past objects, the system would isolate those attributes which were criterial to the object from those that were specific to certain instances of the object, thus creating models based on criterial attributes in a natural manner.

---

* For example, although tables may have many different shapes and sizes, the important item in a description of a table is the horizontal, plane surface in a particular height range.

** Also known as the chicken/egg problem.

4. Pointing

It is often difficult to precisely manually outline complex objects (such as trees or telephone cords) in an image. A better approach would be to point to these objects and have the system perform the detailed outline work. One means of accomplishing this is to have the instructor point to examples of an object and immediately adjacent objects. The system would then create an initial model of the object based on characteristics both of the area surrounding the example location and the adjacent areas. The same programs used to distinguish objects from other objects then would be used to distinguish examples from non-examples. Using its model, the system would attempt to bound the object. The instructor would indicate errors, and the system would use this information to create successively refined models.

Initial experiments with a system such as this would probably require the user to draw coarse outlines of the object, thereby providing more information about the object, and requiring less inference from the system. As the descriptions got better, the system would take on more of the burden, allowing the instructor to point to fewer examples.

C. Other Applications

Obviously, the applicability of this system extends beyond the domain of office scenes. This choice of domain was motivated by two facts: first, objects in rooms are often distinguishable on the basis of local attributes only; and, second, relations between objects are generally well specified. Our system would be relevant in analysis of other domains which also share these properties. Although our primary interest is in areas involving other types of imagery, non-imagery domains (such as decision analysis and diagnosis, mentioned earlier) where scarce resources must be allocated economically also provide potential applications.

1. Maps from Aerial Imagery

We have made preliminary investigations into the use of an interactive system to aid a cartographer in the task of making maps from aerial imagery. In the example photograph of Figure 47, there are two prominent lakes in the upper part of the image. In the next image (a sub-window of the first

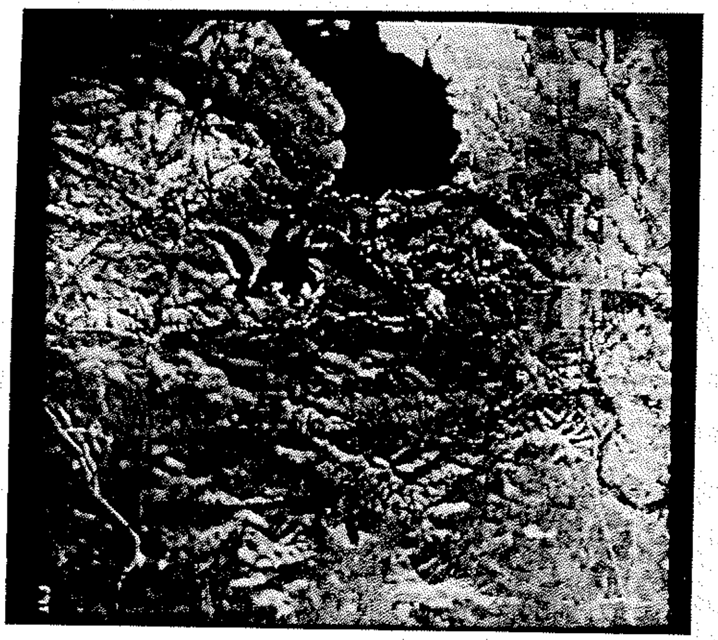


Figure 47. Aerial Photograph Showing Two Prominent Lakes

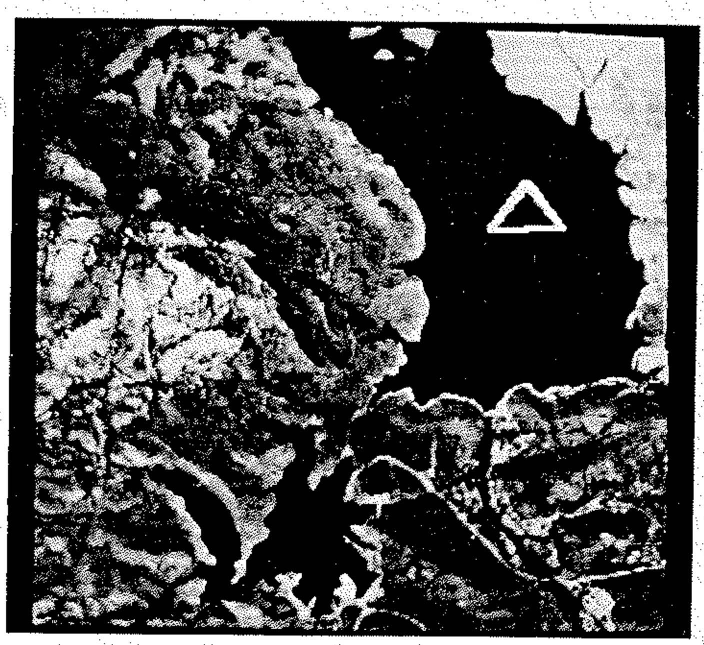


Figure 48. User Interactively Indicates Larger Lake

photograph), Figure 48, the user* has indicated a point in the larger of the two lakes. As outlined above under the discussion of "pointing", the system makes a model of the lake based on brightness characteristics (the only data available for this example) of nearby samples. Using this model, an initial lake boundary is located by the system. This boundary, shown in Figure 49, has two types of defects. Points were included that are not part of the lake, and points that are part of the lake were missed. The user indicates both types of defects to the system in Figure 50 and also indicates that a larger operator size should be used in subsequent processing. The system refines its model, and with the larger operator size preventing boundary "leakage", generates the final boundary shown in Figure 51. Now, the model created for the first lake can be used to outline the second lake Figure 52.

We now have a simple model for lakes in this type of image, which could be used in an automatic mode for finding all lakes in similar images. The model is admittedly simple, using only brightness information and operator size in its description. However, it is possible to obtain multi-spectral aerial imagery, where classification of land areas with respect to use is relatively straightforward. Additionally, elevation data is often available. Furthermore, the system had no knowledge of typical terrain features surrounding the lakes,

---

* The user's input is indicated in white; the system's response in black.

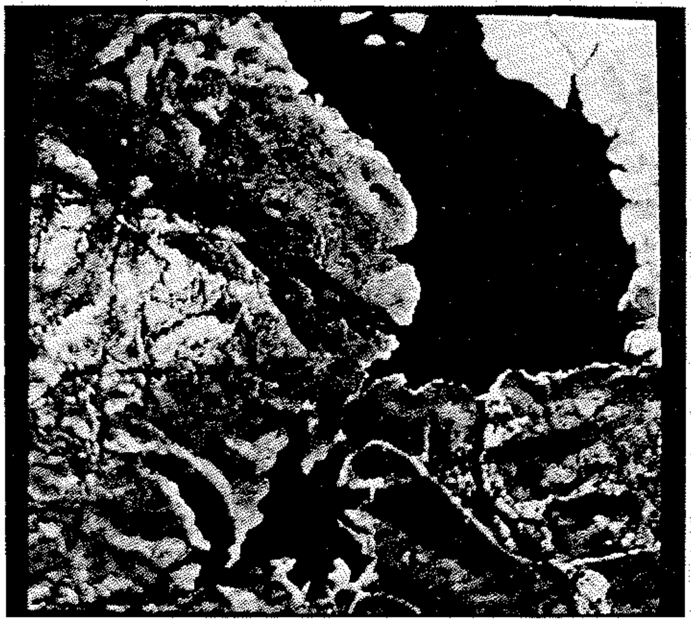


Figure 49. Initial Boundary with Defects

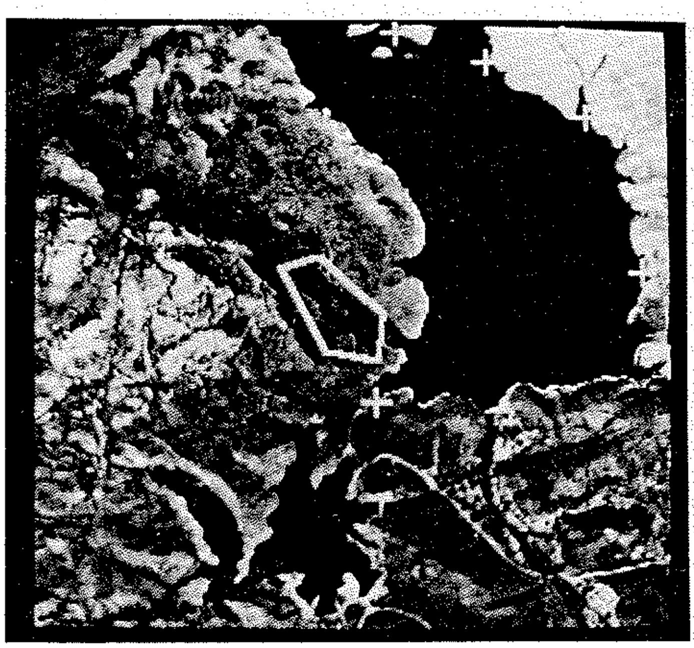


Figure 50. User Indicates Errors

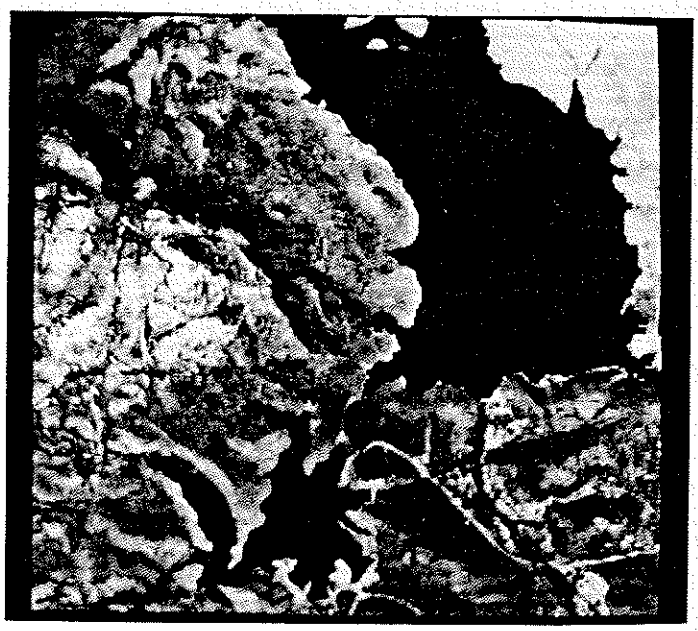


Figure 51. Final Boundary of Larger Lake After Updating Model

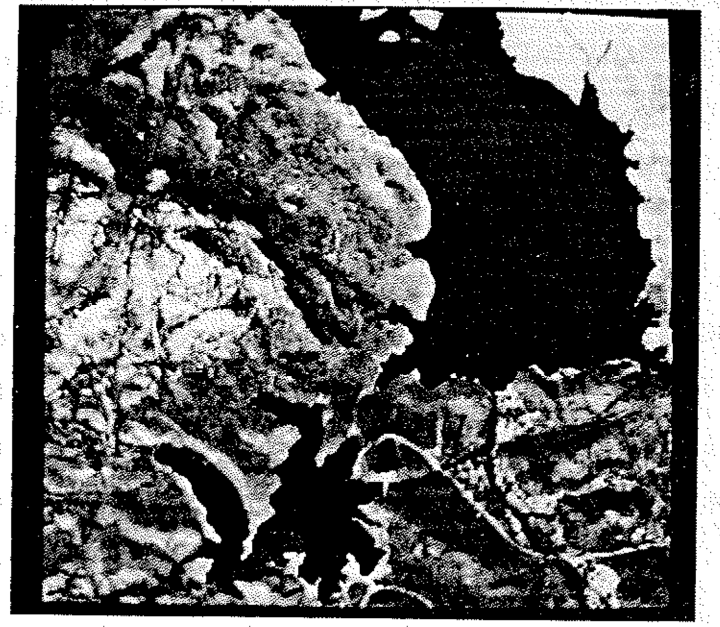


Figure 52. Final Boundaries of Both Lakes

information which is often valuable for restricting possible identifications. Armed with this extended information, it would be possible to use a system like ours to create fairly detailed maps, first in an interactive mode, and later as new routines and models were developed, in a more automatic fashion.

In addition, it should be possible to use existing maps as a source of knowledge, in much the same way the user now provides knowledge to the system. That is, the map could be used to constrain the location of cartographic features in aerial photographs.

A further potential application is the use of the system to answer questions about a data base composed of maps and imagery. A typical question might be, "What is the best route for traffic through Palo Alto?" To answer this question, the system might first have to locate Palo Alto on a map, and select the major roads through the city. It might then use the map to look for these roads in a recent photograph of the area, in order to determine areas under construction, or whether new additions (such as traffic islands) might affect traffic flow. This would obviously require an inferential capability beyond that which our system now possesses, but certainly within the capabilities of existing question answering systems.

2. Biomedical Imagery

Biomedical imagery is generated in vast quantities every day. Most of it is scanned by medical personnel searching for particular artifacts. A distinguishing features approach, based on attributes of both artifacts and healthy tissue should be useful in eliminating some of this labor. Applications which show the most immediate promise include mass screening of breast thermograms for cancer or cancer warning signs, and the examination of retinal photographs for various types of lesions. In both of these areas, it is possible to describe the artifacts being sought in such a way that a program could locate them. Additionally, most biomedical imagery has standard formats, allowing the system to be constrained as to where to look. There are usually standard cues used by specialists when examining these pictures, which could also be utilized by our system. In addition, there are areas (such as breast thermogram screening) where a program is likely to do better than many humans[31]. This is due to the fact that many diagnoses can be made more reliably by the use of quantitative data, from the image. Although, making relative, quantitative measurements from a picture is quite difficult for a human, it is quite simple for a machine.

APPENDIX 1

COST AND CONFIDENCE

This appendix defines the concepts of cost and reliability (or confidence) used in evaluating detectors, and derives their computational formulas.

Confidence

In the course of planning a strategy, or examining a picture interactively, an important question frequently arises: "Given a particular set of attribute measurements on an item (a region or sample), what set of objects could it possibly belong to?" If the measurements have already been taken from some region (or sample), the answer to the question allows the system to "classify" the item as belonging to some element of a set of objects. If the system is considering generating a detector for taking the measurements, then the answer to the question allows an estimate of how well the detector should work.

In addition to knowing the set of possible objects which could provide the measurements, it is also important to know the relative likelihood of each object in the set. For example, if the system knows that the local (surface) orientation of a sample is horizontal, then it should know not only that it belongs to either tabletop or floor, but also that it is more likely to belong to the floor since the floor (usually) takes up much more of the image than does the tabletop. If the system also measures the height of the sample to be 2 1/2 feet, then it should realize that the sample has to belong to tabletop, since that is the only object which has both properties. Finally, if the measurements span a wide range of values, then several objects' attribute ranges may overlap the measured range. In this case, the system should take into account the amount of overlap.

We can formulate a set of requirements for a system which answers our original question. The system should account for a priori probabilities of the item belonging to an object, should be able to handle combinations of attribute measurements, and should be able to measure and use the degree of attribute range overlap.

The program that satisfies these requirements, CONF, measures the confidence that a sample belongs to an object based on a set of detector outcomes. A recursive Bayes procedure[12] is used in the computations. Attribute measurements are taken as independent events which are linked together through object descriptions. That is, if no object descriptions were available, the measurements would be truly statistically independent events. However, having object descriptions allows the system to compute dependent probabilities.

Before beginning the discussion of the procedure, some notation is required. The set of outcomes of the application of a set of detectors, $\{D_1, D_2, \ldots, D_n\}$ will be written, D^n. A detector, D_k, implies a range of attribute values and a program that can measure the attribute for an item, "accepting" the item if the value is within the range, and "rejecting" it otherwise. Thus, the only two possible outcomes of the application of a detector to a sample* are either accept or reject. Confidence is formally defined as $P(s \in O_j | D^n)$. That is, confidence is the probability that a sample, s, belongs to object, O_j, given the outcomes of the set of detectors, D^n. The phrase, $s \in O_j$, will be shortened to O_j in the following discussion. Therefore, confidence is written, $P(O_j | D^n)$.

----

* From now on, we will say sample, with the understanding that the discussion applies equally to regions.

Using the recursive Bayes procedure, the expression for confidence is expanded,

$$P(O_j|D^n) = P(O_j|D_n, D^{n-1})$$

$$= \frac{P(D_n|O_j, D^{n-1})P(O_j|D^{n-1})}{P(D_n|D^{n-1})} \qquad (1)$$

Since the outcome of a detector is affected only by the objects it is conditioned on, and not by the outcomes of other detectors, the term $P(D_n|O_j, D^{n-1})$, is reduced to $P(D_n|O_j)$ and the expression becomes:

$$P(O_j|D^n) = \frac{P(D_n|O_j)P(O_j|D^{n-1})}{P(D_n|D^{n-1})} \qquad (2)$$

This is the usual way of writing the recursive Bayes expression. $P(D_n|O_j)$ is the probability that a random sample from object, O_j, will have outcome D_n when the detector is applied. This probability is computed from stored data by summing the samples that would produce the given outcome, and dividing by the total number of samples measured. The data used for this step come from characterizations generated by the system from examples of the object.

The term, $P(O_j|D^{n-1})$, is computed recursively, terminating with $P(O_j|D_1)$ which is expanded in the normal way,

$$P(O_j|D_1) = \frac{P(D_1|O_j)P(O_j)}{P(D_1)} \quad (3)$$

$P(O_j)$ is the a priori likelihood of a randomly selected sample belonging to O_j. This is usually computed by comparing the expected projected two-dimensional area of the object with the area of the image. $P(O_j)$ also reflects the set of objects expected to be present. By setting $P(O_k)$ to zero, the object, O_k, is effectively eliminated from consideration.

To compute $P(D_1)$, $P(D_1|O_j)$ summed over all objects.

$$P(D_1) = \mathrm{SUM}_{O_j}[P(D_1|O_j)P(O_j)] \quad (4)$$

Now, with the derivation of $P(O_j|D_1)$, and therefore the derivation of $P(O_j|D^{n-1})$, the final details for the original computation of $P(O_j|D^n)$ can be completed.

The last term to be expanded is $P(D_n|D^{n-1})$. This is where the detector outcomes are linked through the object descriptions. As in Equation 4, the term will be expanded over all objects.

$$P(D_n|D^{n-1}) = \mathrm{SUM}_{O_j}[P(D_n|D^{n-1}, O_j)P(O_j|D^{n-1})]$$

As mentioned previously, the outcome of a detector depends on objects, not on other detectors except insofar as they select objects. Therefore, the term $P(D_n|D^{n-1}, O_j)$ is reduced to $P(D_n|O_j)$, and the expression is rewritten as

$$P(D_n|D^{n-1}) = \mathrm{SUM}_{O_j}[P(D_n|O_j)P(O_j|D^{n-1})] \qquad (5)$$

This expression is Equation 4, conditioned on the remaining set of detectors, D^{n-1}.

Several points need to be emphasized. It is important to be able to limit the set of objects under consideration. Limiting this set allows the system to reduce its window into the scene, from a full image, to a selected subimage. This reduction typically comes about when the system locates some object and then looks in the immediate vicinity for other objects. The initial object then provides a new window into the scene, usually with an appreciably smaller subset of objects that need to be considered.

Another important point is that the derivations use the outcomes of the set of detectors, and do not depend on the value of the outcomes. That is, the computations do not require that all the detectors accept the sample, but only that there is some outcome. This fact allows for the possibility of generating decision trees where one branch of the tree is taken for an acceptance, and the other for a rejection. Although the system does not currently generate decision trees (but only conjunctions that require that all outcomes are accepances), there are no theoretical barriers.

Appendix 1 COST AND CONFIDENCE

Cost

In this discussion, cost will be the anticipated cost (measured in milliseconds of CPU time) of applying a detector to a sample (regions are not applicable here). In the discussion of confidence, the order of application of detectors was immaterial, since all outcomes were needed. In this discussion, detectors will be limited to composite detectors, i.e., conjunctions of simple detectors applied sequentially. The cost of application of a composite detector to an image is highly order dependent, since a rejection by one detector in a sequence implies that the remaining detectors in the sequence need not be applied.

Most of the notation required here was defined in the preceding section. Let $C_s(D^n)$ be the per sample cost of applying detector sequence, D^n, and let $C_s(D_i)$ be the per sample cost of the single detector, D_i. The single detector costs are measured previously by the system.

The cost of applying the detector sequence, D^n, is

$$C_s(D^n) = C_s(D_1) + C_s(D_2)P(D_1) + C_s(D_3)(P(D_1, D_2) + \ldots + C_s(D_n)P(D^{n-1})$$

$$= \mathrm{SUM}_{i=1}^{n}[C_s(D_i)P(D^{i-1})]. \qquad (6)$$

D_1 is always applied. A fraction of the samples tested with D_1 (i.e., those accepted) will also be tested with D_2. The percentage of samples passed by D_2 and D_2 will also be tested by D_3, and so on.

The cost of application of a detector to the image (or to a window) is the per sample cost of the detector times the number of samples to be tested. Since the number of samples to be tested also depends on the detector, that number is derived here. It is assumed that a certain predetermined number, N_D, of samples on the target object should be accepted by the detector. From N_D and some object data, we compute a sampling density, f, which will provide a sufficient number of samples on the object, such that N_D of them could be expected to pass the detector. N_W, the total number of samples in the window, is equal to the sampling density times the area of the window, S(W). That is,

$$N_W = fS(W). \tag{7}$$

We let N_D be the expected number of samples from the window that are both on the object, and acceptable to the detector. Or,

$$N_D = N_W P(O,D^n). \tag{8}$$

And,

$$N_D = fS(W)P(O,D^n) . \tag{9}$$

Therefore,

$$f = \frac{N_D}{S(W)P(O,D^n)} .$$

Or,

$$f = \frac{N_D}{S(W)P(D^n|O)P(O)} .$$

Since the system does not retain the correlated data necessary to determine $P(D^n|O)$, it instead uses the minimum of these set of probabilities, $\{P(D_1|O), \ldots, P(D_n|O)\}$.

The total cost now is

$$C(D^n) = N_W C_s(D^n).$$

Or,

$$C(D^n) = \frac{N_D}{P(O,D^n)} * C_s(D^n).$$

APPENDIX 2

ISIS

Introduction

In this appendix, we describe the operation of ISIS (Interactive Scene Interpretation System). ISIS allows a user to define regions by indicating points in the region, or vertex points on the boundary. The system can compute various attributes (such as hue or height) of either regions or samples.

ISIS is an integrated set of INTERLISP functions facilitating the development and testing of pictorial representations. The LISP functions communicate with FORTRAN subprograms residing in an inferior fork. The system consists of an image file, a library of primitive feature extraction functions, a means for applying selected primitive operators to graphically designated areas of a scene, and an iconic data structure for retaining pictorial examples.

Iconic regions are represented as an explicit list of picture samples and/or as a list of vertices describing a closed polygonal boundary. A bounding rectangle is also determinable for each region. Efficient routines exist for determining whether a given picture element is contained within a bounded region, for obtaining a set of samples over a bounded region, and for fitting a boundary around a set of samples.

Images are stored in a rectangular array (usually 120 x 120 picture cells on a side), with each picture cell characterized by tv brightness through red, green, blue, and neutral filters and by a range measurement (the range measurements are output of a time-of-flight range finder, or simulated output of this device which is still undergoing improvement). High resolution vector displays and an MOS-refreshed color video monitor with a low resolution vector overlay serve as graphic output devices. Both displays allow the user to select points on the screen with a cursor.

Definitions

We begin the discussion of the system by defining some basic data representations and concepts.

<sample>

a point in the image. A sample is represented by the dotted pair of its X and Y (i.e. horizontal and vertical) components, i.e. (X . Y).

<samplelist>

a list of samples of the form: (SAMPLE <sample> ... <sample>).

<boundary>

a list of samples of the form: (<sample> <sample> ... <sample>), where the first and last samples in the list are EQUAL. The samples form the vertices of a closed polygon.

<region>

a <sample region> or a <boundary region>. Efficient functions exist to convert between the two.

<sample region>

a region composed of the set of samples in an associated samplelist.

<boundary region>

a region consisting of the area delimited by the polygon represented by an associated boundary list.

<region number>

the canonical representation for a region. The region number is a unique integer between 1 and the value of the global, RCT (the maximum number of regions created up to that point).

<alias>

a non-unique atomic name for a region. An alias may be associated with at most one region, but a region may have several aliases.

<current region>

initially region number 1, the entire picture. The current region is the most recent region created by NEWREGION (see below).

<region designator>

is one of the following:

<region number>

<alias>

<sample list> - either refers to an existing region with an EQ samplelist, or a new sample region is created.

<boundary list> - either refers to an existing region with an exactly identical boundary list, or a new boundary region is created.

MOUSE or M - allows the user to select a list of sample points with the mouse (the display cursor), choosing points with the left button, verifying with the middle button, and terminating the selections with the right button. If a single point is selected no new region is defined, but the point will be processed as if it was a region.

<sample> - specifies a single point. No new region is defined, but the point will be processed as if it was a region.

TTY: - LISP goes into a break, and the user should return a region designator.

NIL - specifies the current region.

In all cases, the region designator is mapped into a region number, either for an existing region, or for one created during the mapping.

<bounding rectangle>

a rectangle defined implicitly by the maximum and minimum values of X and Y for the samples of a region.

ISIS Function Descriptions

All ISIS functions described below are LAMBDA functions (i.e., their arguments are evaluated when the function is called), unless otherwise specified. In general, functions which access properties of a region (such as its samples, boundary list, or hue) first check to see if the property has been previously computed. If it has been, the previous value is returned. Otherwise, the value is computed and saved in a special hash array to be looked up on the next access. This results in considerable time savings.

Starting ISIS

ISIS exists as a SAVE file on directory <SUBSYS>. Therefore, to run ISIS you need only type ISIS to TENEX. ISIS initially loads the FORTRAN file, <ISIS>ISISF.SAV, into a lower fork. You may replace that file with some other, if you wish, by using the function ISISFORK. A picture is loaded using RDPIC. After running a while, you can save your work up to that point with the function, SAVESYS.

ISISFORK [<forkname>]

This function creates an inferior fork (after first killing any existing ISIS fork), and loads the SAVE file specified by <forkname> into it. It sets the global variable, ISISFORKNAME to <forkname>, and will use this as a default value of <forkname>, if it is not supplied in later invocations of the function. The file <ISIS>ISISF.SAV will be the most current default ISIS fork.

RDPIC [<pic id>]

RDPIC loads an image file into the ISIS fork. If <pic id> is a number, it references the picture indexed by <pic id> on the index file, <VISION>INDX.DAT. If <pic id> is a filename, that file is loaded.

SAVESYS [<filename>]

saves anything you have added to ISIS on the TENEX file specified by <filename>. To restore the system to its saved state, type the filename to TENEX. It will automatically start ISIS and restore you to where you were. SAVESYS sets the global variable, SAVESYSNAME to the filename specified. Thereafter, you need not supply the file name; it will use SAVESYSNAME as a default. When a SAVESYS file is restored, the ISIS fork in use when it was saved is restarted, and any picture previously loaded is reloaded.

> When a SAVESYS file is restarted, initializations for that session can be performed by the system. The variable, SESSIONVARS, is a list of forms to be evaluated. If the format of the form is a list with a single element which is also a list, that element is evaluated when the system is restarted. If the form consists of a list of two elements, the first of which is an atom, then the value of the second replaces the top-level value of the atom.

To save the region datastructures as a symbolic file, execute the LISP form: MAKEFILE(DATASTRUC.ext), where ext is an extension which you provide. They can be loaded into another incarnation of ISIS with the LISP function, LOAD (i.e. LOAD(DATASTRUC.ext)).

Region Functions

Regions are defined and accessed using the following functions. ISIS is initialized with the first region defined as a boundary region of the whole picture with the alias, SCENE.

NEWREGION [<region designator>; <type>]

Defines a new region and makes it the current region. <type> should be S or SMP for a sample region, B or BND for a boundary region, and NIL will cause the user to asked which type of region is being supplied.

RESAMPLE [<region designator>; <sample density>]

Resample the specified region. Create a new region consisting of the sample points. If <sample density> is specified, it causes RNDREGSZ to be rebound to its value (see description of RNDSMP below).

SAMPLES [<region designator>]

Returns the sample list for a region. A sample list is created and saved if it does not already exist.

SAMPLESP [<region designator>]

Returns the sample list if it exists, otherwise, it returns NIL. SAMPLESP will not cause the region to be sampled if no samples exist.

BLIST [<region designator>]

Returns the boundary list for a region. A convex hull is computed for the region samples and stored as the boundary if the region does not already have a boundary list.

BLISTP [<region designator>]

Returns the boundary list if one exists, NIL otherwise. This function will not compute a new boundary list.

ALIAS [<alias>; <region number>]

Defines an alias in terms of a region number. If no region number is given then the region number associated with the alias is returned if the alias has been previously defined. If the alias has not been previously defined and no region number is given the alias is associated with the current region.

XREGION [<region designator>]

deletes the specified region from the region data structure.

CLRLDS [ ]

clears the region data structure, and

reinitializes it so that region 1, alias SCENE, consists of the entire image.

Predicates

LIMITP [<form>; <arg>; <min>; <max>]

This predicate returns T if the (numeric) value of <form> applied to <arg> lies in the closed range of values specified by the minimum, <min>, and the maximum, <max>.

WITHIN [X; Y; <boundary list>]

This program checks to see if the point (X . Y) is inside the region defined by <boundary list>. It returns NIL if the point is external to the region, 1 if internal, 2 if a boundary point, and 3 if a vertex of the boundary.

Primitive Functions

HULL [<sample region>]

This generates the convex hull of <sample region>. This is typically used as a default region boundary for a sample region.

RNDSMP [<boundary region>]

This function selects random samples over <boundary region>, and sets the region's samples to this collection. The global variable, RNDREGSZ, sets the number of points to be selected. If RNDREGSZ is an integer, that number will be used as the number of points to be selected; if it is a floating point number (which must be between 0.0 and 1.0), the number is used as a fractional density; that fraction multiplied times the 2-dimensional area of the region (in picture points) gives the number of samples to be returned. For example, if RNDREGSZ has the value .25, RNDSMP will randomly select one fourth of the samples of the region and set the region's samples to be this collection. RNDREGMAX and RNDREGMIN are globals which control the maximum and minimum number of samples selected. RNDSMP will always select at least RNDREGMIN samples, and at most RNDREGMAX samples.

Region and point attribute functions

Region properties are determined by the following functions:

BGHT [<region designator>]

average brightness of a region. The brightness of a completely black region is zero; a completely white region is (currently) 31.

HEIGHT [<region designator>]

average height (z-value) of a region in feet. The floor has a height of zero.

HUE [<region designator>]

average hue of a region in degrees from red (see below). The hues of the three primary colors, red, blue, and green, are zero, 120, and 240 degrees respectively. Regions with other colors will have hue values somewhere in between these.

ORIENT [<region designator>]

average orientation in degrees of the vector normal to the region with respect to the z-axis. A perfectly horizontal region would have an orientation of 0 degrees.

RANGE [<region designator>]

average distance from the camera to the region in feet

SAT [<region designator>]

average color saturation of a region (defined below)

To each of the above functions the following suffixes may be added to obtain different statistics:

EXT - return the highest and lowest observed values

SD - return the standard deviation of this feature

S - return the value of the feature for a specific sample point, bypassing the code to check what type of region designator is given. Use of the S functions will speed up predicates.

Other region property functions are:

AREA [<region designator>]

returns the total number of picture points within a region.

RGBR [<region number>]

returns the average RGB coordinates of picture points within a region.

BRECT [<region designator>]

returns the bounding rectangle for the specified region.

Point properties can be obtained by using the above functions, or by using the "Q" functions which interrogate the low level data structure. Useful Q functions are listed below:

SORQ [<sample>]

returns the parameters of a plane fit to the 3 X 3 matrix surrounding a point.

XYZQ [<sample>]

returns the real world coordinates of a picture point in the form (X Y Z), measured in feet.

SURFACEQ [<sample>]

returns surface orientation and plane intercepts for the 3 X 3 matrix surrounding a point.

IJQ [<real world point>]

returns the picture coordinates corresponding to a point in the real world specified in the form (X Y Z) measured in feet.

Display Functions

As a default ISIS assumes that no display is assigned; to obtain a display use the function:

```
USEDISPLAY [<display name>]
        <display name> = NIL to use the default display
                                (currently RAMTEK)
                         ADAGE
                         HP
                         RAMTEK
        The default display is reset to that specified by
        <display name>.
```

To display the picture, use the function GRADIENT. This will display a color picture if the RAMTEK is currently selected, or a gradient picture if the ADAGE or the HP is the current display.

```
GRADIENT [<picture selector>; <scale>; <threshold>]
        <picture selector> =
                    BWPIC -- display picture taken
                      through neutral density filter.
                    REDPIC -- use picture taken
                      through red filter.
                    GREENPIC -- use picture taken
                      through green filter.
                    BLUEPIC -- use picture taken
                      through blue filter.
```

NIL -- use default picture designated by the global variable GRDFLTPIC (initially set to REDPIC).

<scale> = IDISPFS - full scale display
IDISPQS - quarter scale display

<threshold> = gradient threshold, defaults to the global IGRTHR which is initially set to 7. This is only for the ADAGE or the HP.

SHOW [<region designator>; BLINK]

Displays the region's samples and (if it exists) the region boundary. If BLINK is not NIL, the display will blink.

SHOWCOLOR [<region designator>]

Displays region points in the color triangle.

BOUNDARY [<region number>; BLINK]

Displays only the region boundary. If BLINK is not NIL, the display will blink.

Other ISIS functions

Other useful ISIS functions are documented below:

SMPRNT [<region number>; <form>]

prints out the value of the form applied to each sample of the region. e.g., SMPRNT (8 HUES) would print out the HUE of each sample in region 8.

FILTER [<predicate>; <region designator>]

A new current region is created, consisting of all samples in the old current region that make the predicate true. If the first argument is NIL then the user will be asked to provide predicate clauses, (which will refer to the sample being tested as the variable, X) one at a time. When NIL is returned as a clause, clauses entered up to this point are combined conjunctively, and the conjunction is used as the predicate.

GROW [<region designator>; <predicate>;

<window given as a list consisting of min x, max x, min y, max y>; <growth increment>; <8-connectedness>] The opposite of filter, points are grown out from the boundary of the specified region. Points within the window are considered at a spacing given by <growth increment>. Points

are extended in a 4-connected sense if <8-connectedness> is NIL, and in an 8-connected sense otherwise. Points are accepted if they make the predicate true. After coarse growing is complete, the new boundary is extended by growing in smaller increments if the global variable GROWP is non-nil. The default for <growth-increment> is the global GROW-DFLTDELTA which is initially set to 8.

Procedures

WITHIN

The procedure used by WITHIN to decide whether a point is internal or external to a region is to count the number of boundary segments crossed by a line extending from the point to another point outside the region. An odd number of crossings means the point is inside the region. Special checks are be made to see if the point is a vertex or lies on the boundary. For a point, (XP . YP) (as shown in Figure 53), the external point selected is (XP . 0). The program sequences through the boundary segments ignoring any segment ((X1 Y1) (X2 Y2)) unless XP lies in the range, X1 to X2, and YP is within the range Y1 to Y2, since these are the only segments for which an intersection could exist. For these segments, the intersection (XC . YC) is computed and a crossing occurs if YC < YP. If YC = YP then the point is on the boundary. In order to eliminate errors and to increase efficiency, no divisions are performed by WITHIN.

HULL

In order to generate the convex hull of a sparse set of points, HULL connects the points having extreme values in X and

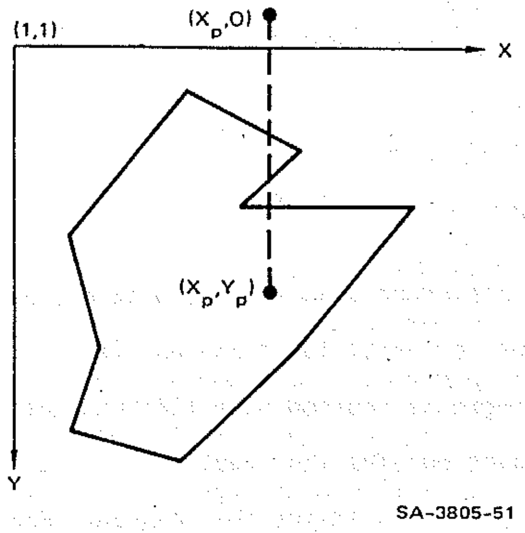


Figure 53. Boundary and Internal Point

Y for its first guess at the boundary. It then checks each remaining point to see whether it is inside or outside that boundary. Any inside point is removed from the list of points which are boundary candidates. The points remaining are split into two sets, those above and those below the line segment connecting the point with minimum X to the point with maximum X. The two sets are ordered first in X and then in Y so that they can be joined into a counterclockwise boundary. Since this boundary will generally have concavities, the final step of the procedure is to iteratively traverse the boundary, removing concavities until none remain. A point is defined as a point of concavity if it is to the left of the vector from the point preceding it in the boundary to the point succeeding it. When the process terminates, the ordered list of points remaining will form the convex hull surrounding the initial set of points.

Procedure for HULL

1) Select maxima and minima in X and Y (Figure 54), and generate initial counterclockwise boundary. Special cases can arise here. The cases dealt with include those where the initial set comprises only one or two points, and those where the set of points containing the maxima and minima in X and Y comprises only two or three points. We will not consider these cases here, mentioning only that they affect only the efficiency of the algorithm, and not the efficacy.

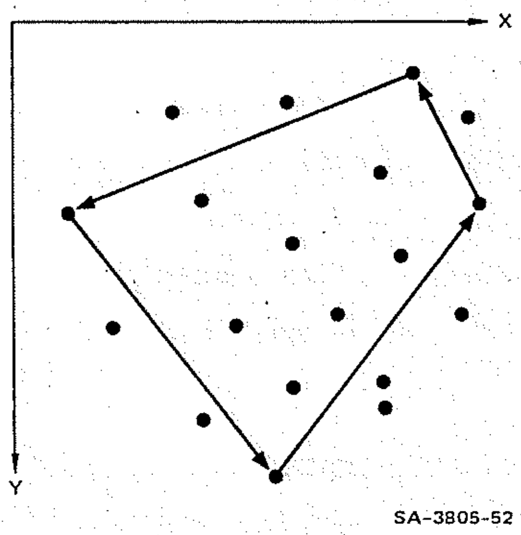


Figure 54. Maxima and Minima in X and Y of Set of Points

2) Check all remaining samples to determine whether inside or outside current boundary. If inside, remove from set of points under consideration as shown in Figure 55.

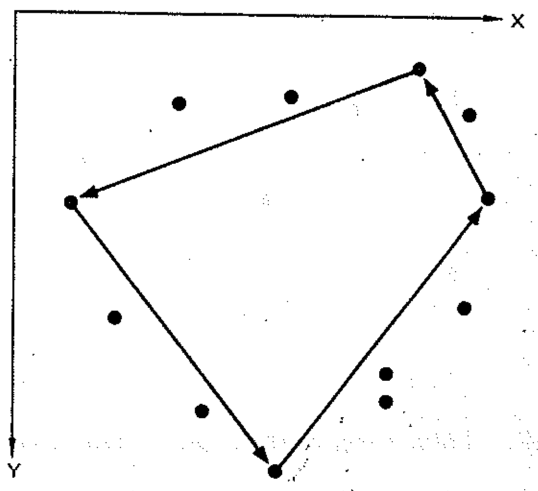


Figure 55. Point Set After Removal of Internal Points

3) Using LEFTP, a predicate which returns true for three points if the third is to the left of the vector from the first to the second, separate the remaining points into two sets, those above the line segment from the point with minimum X to the point with maximum X (i.e., those above the directed line segment), and those below the line segment (i.e., those to the right). Call these two sets TOP and BOTTOM, respectively (Figure 56).

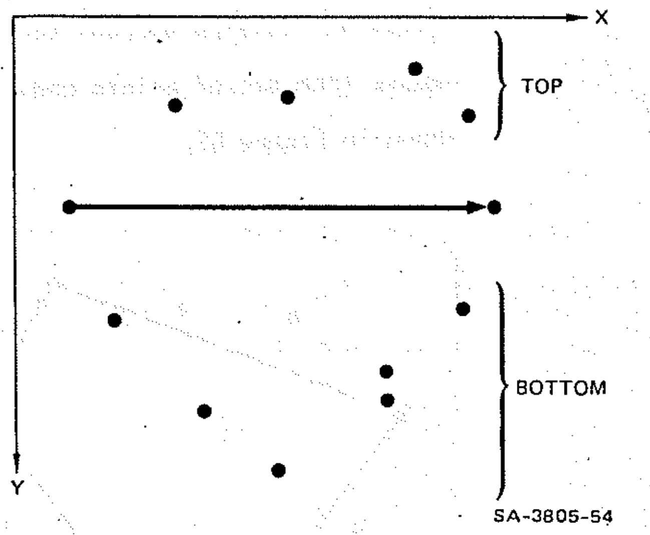


Figure 56. Remaining Points Separated Into TOP and BOTTOM Sets

4) Sort TOP so that points with lesser X precede those with greater X, and for points with the same X value, delete all but the point with minimum Y. These points are then joined into a length of boundary. Since our coordinate system is left-handed (i.e., Y increases in a downward direction), the segment of boundary will be ccw in direction.

5) Similarly sort BOTTOM. This time, order by increasing X, and where points have the same X value, retain only the point with maximum Y. Join the remaining points, and then join the TOP

boundary segment to BOTTOM, resulting in a ccw, closed boundary (Figure 57). If either TOP or BOTTOM is null, then no points are inserted for that set.

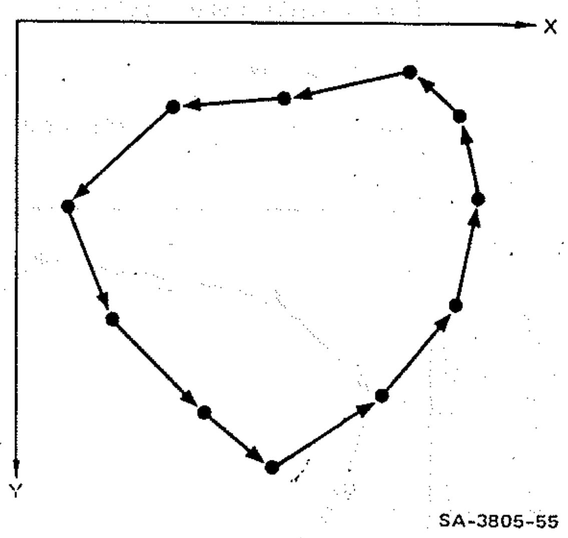


Figure 57. Initial CCW Boundary

6) Proceed around the boundary, checking three points at a time. If the second point is to the left of the vector from the preceding to the succeeding point (using LEFTP), it is a point of concavity, and is deleted by physically modifying the list. If the point is not removed, the process steps to the point just examined, in order to consider its neighbor. The process does not step to the next point when a point is

deleted. The deletion has the effect of bringing the point following the deleted point to the correct position for consideration, and so no stepping is required.

7) If no points were deleted in step 7, return the remaining set of points as the convex hull (Figure 58). Otherwise, return to step 6.

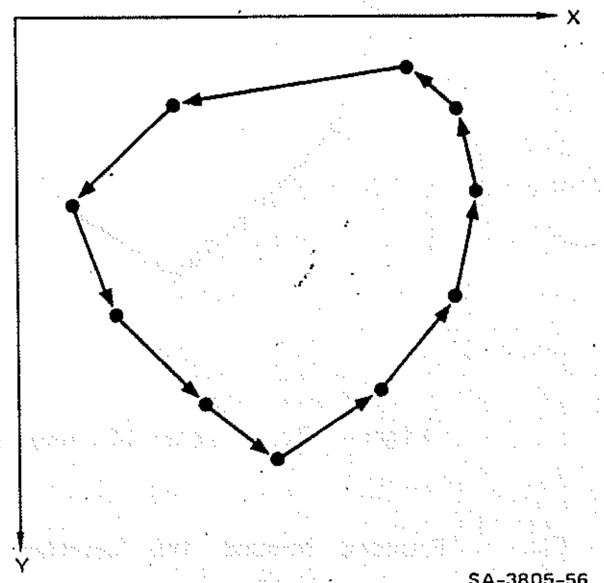


Figure 58. Final Convex Hull

This routine is coded in LISP, with only LEFTP being hand-coded in LAP (LISP assembly language). It operates very quickly, typically requiring about .3 seconds to compute a boundary for 150 randomly selected points.

RNDSMP

RNDSMP works by computing an appropriately spaced grid over the bounding rectangle of the region being sampled. The grid spacing is determined by the density of samples dictated by RNDREGSZ. A point is selected randomly from each grid cell (using the LISP random number generator, RAND), and checked to see if it is WITHIN the boundary region. If so, it is added to the list of samples being collected. The advantage of this approach is that the samples are uniformly distributed over the region, and the program need never check to see if the sample has already been generated. While the sampling is not completely random over the region, it is completely random within the grid cells.

HUE and SATURATION

Prior to computing hue and saturation from color intensities, R, G, and B, obtained from TV images taken through red, green, and blue filters, these raw values are normalized. First, the raw brightness value from each color filter is divided by the corresponding brightness of a known achromatic (e.g., white) color sample, observed through the same filter, thereby compensating for colorations in the illumination source and for unequal filter densities.

$R' = R/R_W$, $G' = G/G_W$, $B' = B/B_W$ A second normalization by the sum of all intensities removes variations in source brightness.

$$r = \frac{R'}{R' + G' + B'}, \quad g = \frac{G'}{R' + G' + B'}, \quad b = \frac{B'}{R' + G' + B'}$$

All points (r, g, b) now lie in the first quadrant of the plane $r + g + b = 1$. This triangular region bounded by the points (1,0,0), (0,1,0), and (0,0,1) is the well-known color triangle, and is shown in Figure 59 color sample maps into the point (1/3, 1/3, 1/3) at the center of this triangle.

In the color triangle of Figure 60 point p is represented as the angle θ, from the vector $\bar{R}-\bar{W}$ to the vector $\bar{P}-\bar{W}$. Saturation is defined as the ratio of the length of $\bar{P}-\bar{W}$ divided by the length of $\bar{S}-\bar{W}$ (the extension of $\bar{P}-\bar{W}$ to the closest leg of the color triangle).

To compute θ, first define $\bar{R} = \bar{R} - \bar{W}$

We have

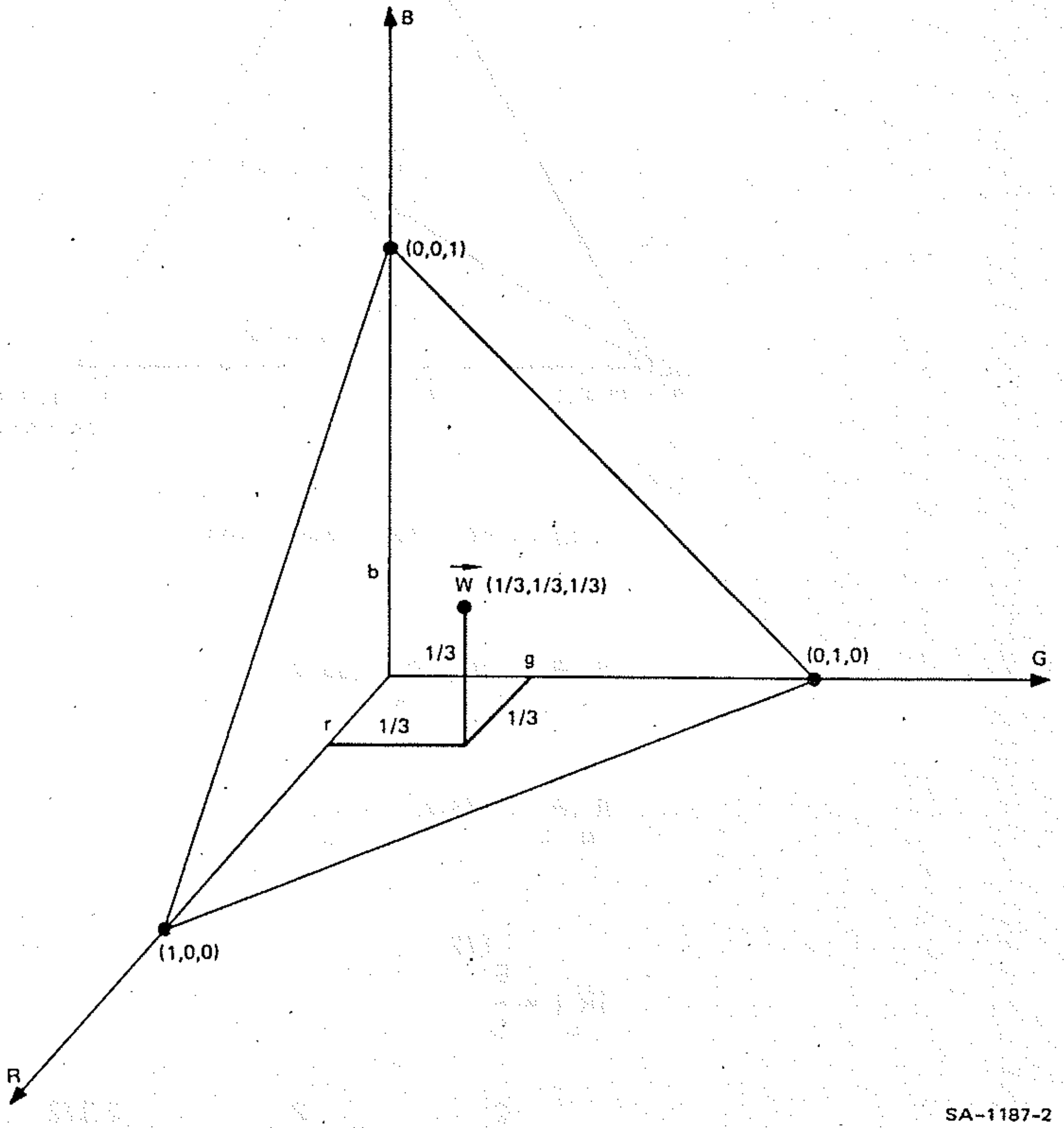


Figure 59. Normalized Color Triangle

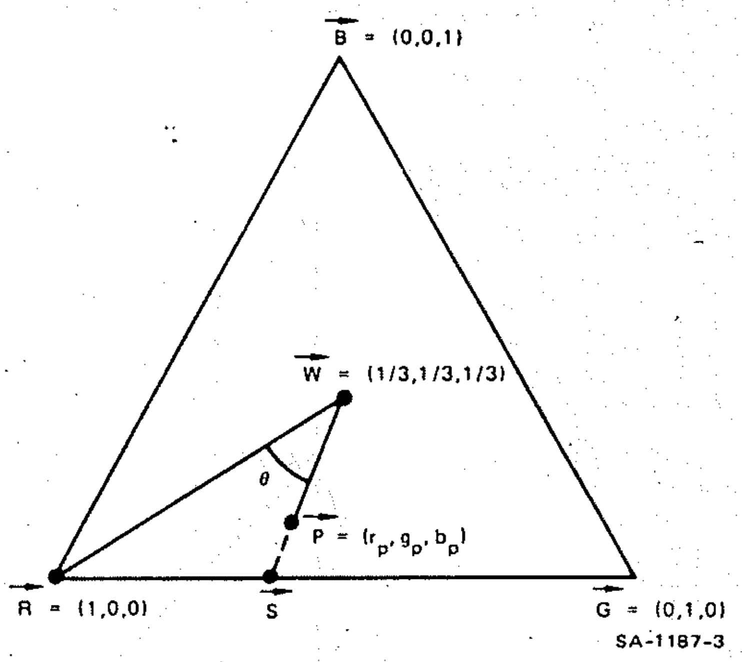


Figure 60. Hue Computation

$$\bar{R}_W . \bar{P}_W = |R_W||P_W| \cos \theta$$

$$\bar{R}_W . \bar{P}_W = 1/3(2r_p - b_p)$$

$$|R_W| = \frac{6^{1/2}}{3}$$

$$|P_W| = [(r_p - 1/3)^2 + (g_p - 1/3)^2 + (b_p - 1/3)^2]^{1/2}$$

Solving for θ gives the desired value modulo 180.

$$\theta = \arccos \frac{(2r_p - g_p - b_p)}{6^{1/2}\,[(r_p - 1/3)^2 + (g_p - 1/3)^2 + (b_p - 1/3)^2]^{1/2}}$$

To obtain the final value of θ, b_p is compared with g_p. If $b_p > g_p$, then the point, P, must lie closer to B than to G, and therefore the angle from R must be greater than 180 degrees. Accordingly, if $b_p > g_p$ then set θ equal to 360 - θ.

The computation of saturation will be carried out for the point P in Figure 61. The nearest side of the color triangle (in this case the R-G side) is S. T is the projection of W onto the RG plane. Q is perpendicular from P to the line WT.

We have

$$\mathrm{SAT(P)} = \frac{\overline{WP}}{\overline{WS}}$$

Since triangle WPG is similar to triangle WST,

$$\frac{\overline{WP}}{\overline{WS}} = \frac{\overline{WQ}}{\overline{WT}}$$

However, $\overline{WT} = 1/3$ and $\overline{WQ} = 1/3 - \overline{QT} = 1/3 - b_p$. Therefore,

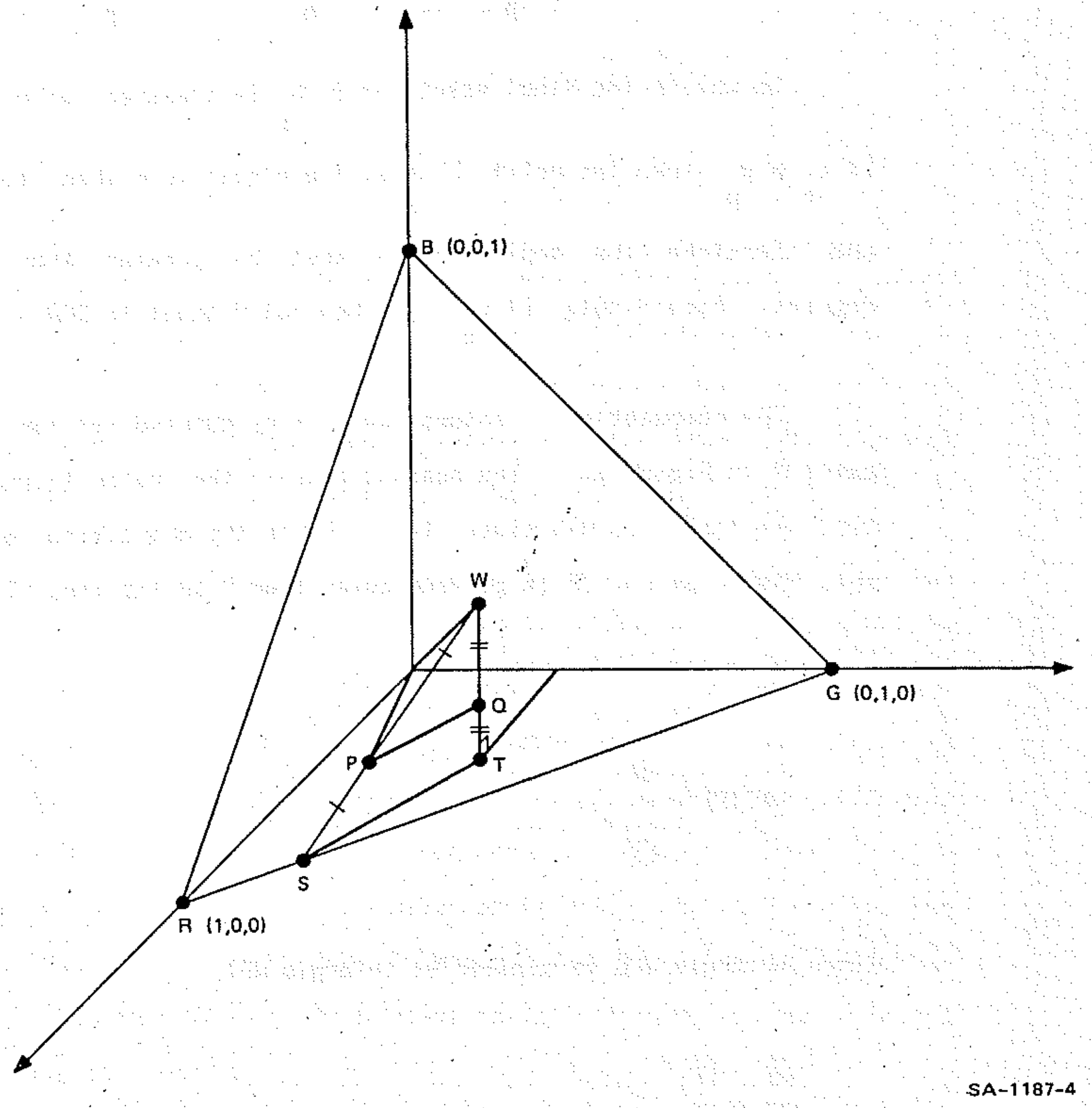


Figure 61. Saturation Computation

$$SAT(P) = \frac{\overline{WQ}}{\overline{WT}} = \frac{1/3 - 3b_p}{1/3} = 1 - 3b_p .$$

Since the minimum component of a point (r_p, g_p, b_p) determines the closest side of the color triangle, we have in general

$$SAT(P) = 1 - 3 * \min (r_p, g_p, b_p).$$

APPENDIX 3

USE OF FAILURE INFORMATION

Our current system makes no use of information gained from module failures. One way in which failure information could be utilized would be to model the possible failure modes probabilistically. For example, we could model the complete process of sampling and filtering as follows.

The process of sampling and filtering assumes that the target is in the image with some prior probability. A sampling density that will guarantee a certain number of samples will be on the object (again with some prior probability) is calculated. This sampling density is selected so that some fraction of the object samples will also be accepted by a detector. Finally, the detector is applied to each sample, and the process succeeds if enough samples match the detector characteristics. The process fails if no samples are selected by the detector.

We will let P_O be the prior probability that the object is in the image and $P_{S,O}$ be the prior probability that the object is in the image and the sampling module will get n samples on the obect. The prior probability that the object is in the image, that the sampler hit the object, and the detector, D, will pass one of the samples on the object will be $P_{D,S,O}$. Finally, P_F is the probability of failure.

We can write

$$P_F = P_{\bar{O}} + P_{O,\bar{S}} + P_{O,S,\bar{D}} .$$

That is, the process can fail (i.e., return no object samples) because the object is not in the image, the sampler failed to hit the object, or the detector did not accept any of the samples on the object. Now, if the module should fail, we can compute some a posteriori probabilities,

$$P_{O|F} = \frac{P_{F|O} * P_O}{P_F} .$$

Where

$$P_{F|O} = P_{O,\bar{S}} + P_{O,S,\bar{D}} .$$

And, therefore, we have the a posteriori probability of the object being in the image,

$$P_{O|F} = \frac{(P_{O,\bar{S}} + P_{O,S,\bar{D}}) * P_{O}}{P_{F}} .$$

Similarly,

$$P_{S|O,F} = \frac{P_{O,F|S} * P_{S}}{P_{O,F}} = \frac{P_{F|O,S} * P_{O|S} * P_{S}}{P_{O|F} * P_{F}} = \frac{P_{F|O,S} * P_{S|O} * P_{O}}{P_{O|F} * P_{F}}$$

$$= \frac{P_{F|O,S} * P_{S,O}}{P_{O|F} * P_{F}} .$$

Now, the only way we could have a failure, given that the object is in the image and the sampler hit it, is for the detector to fail. Therefore,

$$P_{F|O,S} = P_{O,S,\bar{D}} = 1 - P_{O,S,D} .$$

And, so,

$$P_{S|O,F} = \frac{P_{O,S,\bar{D}} * P_{O,S}}{(P_{O,\bar{S}} + P_{O,S,\bar{D}}) * P_{O}} .$$

Of course,

$$P_{D|O,S,F} = 0.$$

That is, the a posteriori probability of the detector accepting the object samples selected by the sampler, after it rejected them is zero.

Now, it could be possible to model many of the execution processes in this fashion, and thereby gain from our failures.

REFERENCES

1. Y. Yakimovsky and J. A. Feldman, "A Semantics Based Decision Theory Region Analyzer," Proc. Third Joint Conference on Artificial Intelligence, pp. 580-588,Stanford University, Stanford, California

2. R. Bajcsy and L. I. Lieberman, "Computer Description of Real Outdoor Scenes," Proc. Second International Joint Conference on Pattern Recognition,pp. 174-179,Copenhagen, Denmark (August 1975)

3. D. Waltz, "Understanding Line Drawings of Scenes With Shadows," in P. H. Winston, The Psychology of Computer Vision,pp. 19-92 (McGraw-Hill Book Company, San Francisco, 1972)

4. A. Rosenfeld, Picture Processing by Computer, Academic Press, New York, 1969

5. N. J. Nilsson et al, "Artificial Intelligence--Research and Applications," Annual Technical Report to ARPA,Contract DAHC04-72-C-0008, pp. 135-166, Stanford Research Institute, Menlo Park, California(May 1974)

6. D. Nitzan and R. O. Duda, Private Communication

7. T. D. Garvey and J. M. Tenenbaum, "On the Automatic Generation of Programs for Locating Objects in Office Scenes," Proceedings of the Second International Joint Conference on Pattern Recognition, pp. 162-168, Copenhagen, Denmark (August 1974)

8. C. R. Brice and C. L. Fennema, "Scene Analysis Using Regions," Artificial Intelligence, 1, No. 3, pp. 205-226 (Fall 1970)

9. M. Minsky, "A Framework for Representing Knowledge," in P. H. Winston,The Psychology of Computer Vision, pp. 211-278, (McGraw-Hill Book Company, San Francisco, 1972)

10. M. Kelly, "Visual Identification of People by

Computer," Artificial Intelligence Laboratory, Memo AI-130, Stanford University,Stanford, California (July 1970)

11. T. Kanade, "Picture Processing System by Computer Complex and Recognition of Human Faces," Department of Information Science, Kyoto University, November 1973

12. R. O. Duda and P. E. Hart, Pattern Classification and Scene Analysis, John Wiley & Sons, New York (1973)

13. Y. Shirai, "Analyzing Intensity Arrays Using KnowledgeAbout Scenes," in P. H. Winston, The Psychology of Computer Vision, pp. 93-114, (McGraw-Hill Book Company, San Francisco, 1975)

14. M. Fischler and R. A. Elschlager, "The Representation and Matching of Pictorial Structures," IEEE Transactions on Computers, Vol. c-22, pp. 67-72 (January 1973)

15. U. Montinari, "Optimization Methods in Image Processing," Proceedings IFIP Congress, Vol. 74, Stockholm, Sweden, 1974

16. C. Hewitt, "Description and Theoretical Analysis (using schemata) of PLANNER: A Language for Proving Theorems and Manipulating Models in a Robot," Ph.D. Thesis, Dept. of Mathematics, MIT, Cambridge, MA (1972)

17. R. Reboh and E. Sacerdoti, "A Preliminary QLISP Manual," SRI Artificial Intelligence Center, Tech. Note 81, Stanford Research Institute, Menlo Park, CA (August 1973)

18. K. Turner, "Computer Perception of Curved Objects Using a Television Camera," Ph. D. Thesis, University of Edinburgh, 1974

19. R. Sproull, Ph. D. Thesis, Stanford University, Stanford, (forthcoming)

20. L. S. Coles, et al, "Decision Analysis For an Experimental Robot With Unreliable Sensors," Proc. of Fourth International Joint Conference on Artificial Intelligence,pp. 749-757, Tblisi, USSR (September 1975)

21. R. Howard, J. Matheson, et al, "Readings in Decision Analysis," Stanford Research Institute, Menlo Park, California

22. A. Guzman, "Computer Recognition of Three-dimensional Objects in a Visual Scene," Ph.D. Thesis, MIT, Cambridge, Massachusetts, (1968)

23. G. J. Agin and T. O. Binford, "Computer Description of Curved Objects," Proc. of Third International Joint Conference on Artificial Intelligence, pp. 629-640, Stanford University, Stanford, CA

24. M. Heuckel, "An Operator Which Locates Edges in Digitized Pictures," Artificial Intelligence Memo 105, Stanford University, Stanford, CA, 1969

25. W. Teitelman, "INTERLISP Reference Manual," Xerox PARC, Palo Alto, California, (October 1974)

26. J. M. Tenenbaum and S. Weyl, "A Region Analysis Subsystem For Interactive Scene Analysis," Proc. of Fourth International Joint Conference on Artificial Intelligence, pp. 682-687, Tblisi, USSR (September 1975)

27. C. A. Rosen, Private communication

28. E. M. Riseman and A. R. Hanson, "Design of a Semantically Directed Vision Processor," COINS Technical Report 74C-1, University of Massachusetts, Amherst Massachusetts (January 1974)

29. J. M. Tenenbaum, "Artificial Intelligence Center Tech Note," Stanford Research Institute, Menlo Park, CA (forthcoming)

30. P. H. Winston, "Learning Structural Descriptions from Examples," Technical Report MAC TR-76, MIT, Cambridge, Massachusetts, 1970

31. D. Falconer, "Digital Processing of Breast Thermograms," SRI AI Tech Note 96, Stanford Reasearch Institute, Menlo Park, CA