1 Learning Control Strategies for Object Recognition

Bruce A. Draper
Dept. of Computer Science
University of Massachusetts
Amherst, MA., USA. 01003
bdraper@cs.umass.edu

Abstract

This paper presents a system for learning object-specific recognition strategies from training images and libraries of image understanding routines. The motivation for this work is that thirty years of computer vision research has produced hundreds of algorithms for visual subtasks ranging from edge detection to pose determination, but very few complete vision systems. The Schema Learning System (SLS) addresses this problem by casting object recognition as an control problem: for every object to be recognized, it learns a sequence of algorithms that will find it quickly and robustly.

More formally, SLS learns control policies under supervision. For every task, a user specifies the target representation (e.g. 2D image position or 3D world position), and provides a set of training images and the locations of the target objects. SLS then applies a three-step process of search, learning from examples and graph optimization to produce a recognition graph that expresses a control policy for invoking image understanding routines to recognize the object.

1. Introduction

Although the field of computer vision has has made significant advances over the past 30 years, there remains one notable exception – are area whose major unsolved problems and lack of advancement are severely limiting the application of computer vision technology to problems of practical importance. We have gained little or no insight – at either a theoretical or practical level – into how the many aspects of vision are integrated into functioning systems.

The most dramatic advances in the field have come not from researchers building task-focused systems, but rather from those who concentrate on better-defined subproblems which admit to "clean" solutions. As a result, although the catalog of image understanding algorithms keeps growing, relatively few complete computer vision systems are built, and when they are it is almost always a major effort that is costly in terms of design, integration and development time. Unfortunately, we will not see the benefits of breakthroughs in the component technologies until we understand, at both a theoretical and practical level, how the many components of a complex vision system are integrated and controlled. It is as though we have the supplies and tools to build a house, but lack the architectural drawings.

This paper presents a first step toward a practical understanding of how the components of vision might be integrated[1]. It presents a system, called the Schema Learning System (SLS), that learns object-specific recognition strategies from training images and a library of image understanding routines. A user (or teacher) marks the position of an object in training images, and SLS searches its IU library for combinations of operators that reliably locate that object.

The principle underlying SLS is that vision is a *goal-oriented* process in which visual algorithms or skills are combined in task-specific ways to generate percepts. This idea in itself is not new: Arbib [3], Aloimonos [1], Ballard [4], and Ikeuchi and Hebert [21] have all advocated goal-oriented vision, with Arbib and Aloimonos putting particular emphasis on the integration of independent visual modules (similar to Ullman's *visual routines* [30]).

The difference between these earlier position papers and this work is that we present algorithms for constructing goal-directed systems automatically. SLS demonstrates that goal-directed vision can be cast as a control problem. Most visual tasks can be achieved by some combination of existing techniques, such as edge extraction, vanishing point analysis or graph matching. SLS automatically develops control policies for achieving goals, thus avoiding the cost of handcrafting systems [17]. The resulting object recognition systems should be immediately useful in such emerging technologies as intelligent vehicles and flexible manufacturing systems, where predictable environments invite the use of special-purpose recognition strategies.

In addition, SLS gives support to the notion of goal-directed vision, which has been criticized on the grounds that the goal of computer vision research is not just to create object recognition systems, but to put forth a coherent and parsimonious theory of vision. Some researchers claim that by modeling vision as a loose (to be critical, *ad-hoc*) collection of special-purpose recognition modules, proponents of goal-directed vision abandon that goal. SLS puts forth a counterclaim by example, however; a claim that special-purpose recognition strategies do not have to be *ad-hoc* or unstructured, that they can arise through predictable and scientific mechanisms in response to a viewer's environment. Indeed, the criticism can be turned around: given that special-purpose strategies can be acquired through experience, it seems unnecessary and unjustified to assume that all visual goals must be met by a single general-purpose mechanism.

[1] Most of this material is from [15]

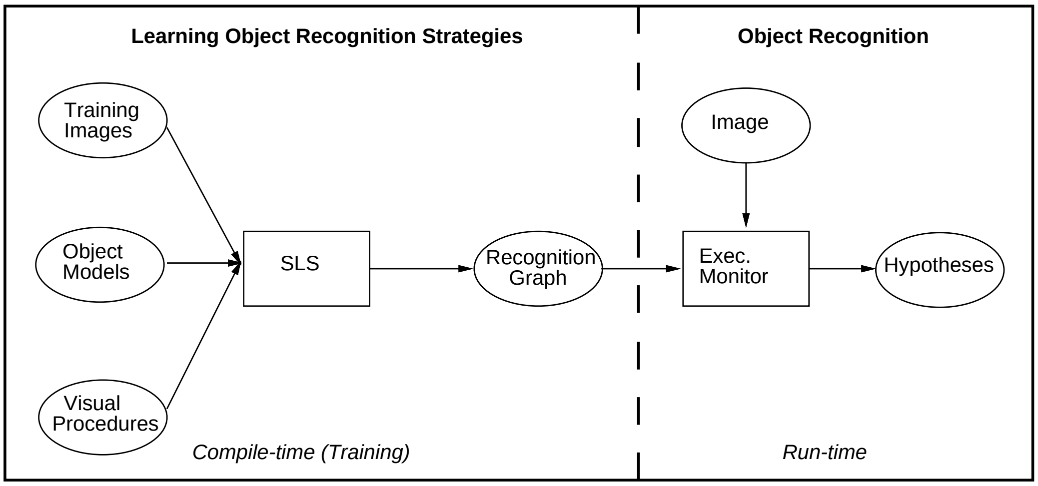

Figure 1: *Top-level view of SLS architecture*

2. The Schema Learning System

The Schema Learning System (SLS) learns special-purpose recognition strategies from training images. A teacher provides a training signal indicating the target (or goal) in each training image. SLS searches its library of image understanding algorithms for combinations of operators that locate the target in the training images. The resulting control policy is then available to application programs such as autonomous vehicles or manufacturing robots any time they need to recognize an instance of the object or object class. SLS is therefore a compile-time (or "off-line" or "batch") system that learns strategies in advance of the run-time application that will use them, as shown in Figure 1.

2.1. Recognition Goals

More specifically, SLS learns control policies (which we will call *recognition strategies*) to satisfy *recognition goals.* In SLS, a teacher provides a recognition goal specifying the object to be recognized, the target representation (e.g. 2D image position or 3D object pose) and corresponding accuracy thresholds. For example, a recognition goal might be to recognize the (3D) position of the UMass engineering building, or to identify the centroid of the image projection of a tree. In both cases, accuracy thresholds would be included; for example, the image position of the tree might have to be accurate to within three pixels.

2.2. Visual Procedures

To satisfy such goals, SLS models vision in terms of *visual procedures* (VPs) and *hypotheses.* Visual procedures are algorithms from the computer vision literature, such as edge extraction, vanishing point analysis or model matching. VPs are thus analogous to knowledge sources in a blackboard system (e.g. [18, 16]) or Ullman's visual routines [30], in the sense that they are the procedural primitives used to build larger strategies. Hypotheses are intermediate-level data items, such as edges or surfaces (or sets thereof). At each step in the recognition process, a VP is applied to one or more hypotheses and either 1) measures a feature of the hypothesis or 2) generates new, higher-level hypotheses. (Feature measurement procedures are referred to as FMPs, while transformational procedures are called TPs[2].)

Figure 2 shows the template for declaring a visual procedure. The template contains only enough syntactic information about a VP to allow SLS to apply it to training images; any other information, such as the expected cost, much be estimated by SLS. The VP template specifies how many hypotheses are required as (run-time) arguments, the level of representation of each argument, any prerequisite features, and a Lisp S-expression for invoking the VP. In addition, TP declarations include the type of hypothesis generated, while FMP declarations include the number of discrete feature values the FMP might return.

2.3. Recognition Graphs

Recognition strategies are represented by *recognition graphs*, which are a generalization of decision trees to multiple levels of representation. Recognition graphs control hypothesis generation as well as hypothesis verification, as shown in Figure 3. The underlying premise is that image data should not be matched directly to object models. Instead, a sequence of more and more abstract descriptions of the image data, represented as intermediate-level hypotheses, are built up under constraints provided by the object model, until eventually goal-level hypotheses are generated. Recognition graphs therefore model vision as a sequence of representational transformations interleaved with hypothesis verifications. Each level of the recognition graph corresponds to one type of intermediate-level hypothesis (in blackboard terminology, one level of abstraction), with the decision tree at that level determining which hypotheses can be definitively rejected based on their feature values. Hypotheses that cannot be rejected are said to be verified, and verified hypotheses are transformed into more abstract hypotheses, continuing the cycle until goal-level hypotheses are generated.

[2] Although TPs are described as transformation procedures, the word 'transformation' should not be construed as implying a one-to-one mapping between old and new hypotheses. TPs can combine information from multiple hypotheses (e.g. stereo) and may generate an arbitrary number of new hypotheses (e.g. segmentation). In addition, TPs do not consume their arguments, so multiple TPs may be applied to a single hypothesis. Some readers may therefore find it helpful to think of TPs as procedures that generate new hypotheses from old hypotheses, rather than as transformation operators.

VP Declaration Template

VP Name:	*VP Name*
Type:	*Transformation of Feature Measurement*
Arguments:	*Number of run-time arguments (hypotheses)*
Levels:	*Level of representation for each argument*
Prerequisites:	*List pf required hypothesis features*
Result Level:	*Level of representation of resulting hypothesis (TPs only)*
Feature Value:	*List of discrete feature values (FMPs only)*
S-Expr:	*Lisp s-expr to invoke procedure.*

Figure 2: *VP Declaration Templates. Each VP declaration in the library includes enough syntactic information for SLS to apply the VP to training images. This includes the name of the VP, its type, the number of run-time arguments (hypotheses), the level of representation (and any prerequisites) of each argument, and either the discrete feature values (for a FMP) or the type of hypothesis generated (for a TP).*

2.3.1. Decision Trees

Each level of the recognition graph is a decision tree directing how hypotheses at that level are verified. Borrowed from the field of operations research, decision trees are trees of alternating *choice nodes* and *chance nodes* designed to help managers make decisions about actions with uncertain outcomes [19]. Choice nodes in a decision tree represent decisions over which the agent (typically a business manager) has control; chance nodes represent events the agent cannot control but whose likelihoods can be estimated. Using decision trees, managers estimate the probabilities of potential consequence of a decision or series of decisions before any action is taken. For example, a manager might consider investing in a new manufacturing facility. If the investment is made and the product sells there will be a profit, but there is some possibility that the product will not sell and the investment will be lost. This scenario can be represented by a decision tree with a choice node at the root representing the option to invest or not, and a chance node representing whether or not the product sells. In AI terminology, decision trees can be thought of as state-space representations similar to game trees with probabilistic opponents.

(Readers familiar with AI-style decision trees such as ID3 [26] will note that the choice nodes in such systems are omitted. These systems make all their choices while learning, leaving only the chances nodes in the tree. SLS does the same, pruning away every option but one at each choice node. Nonetheless,

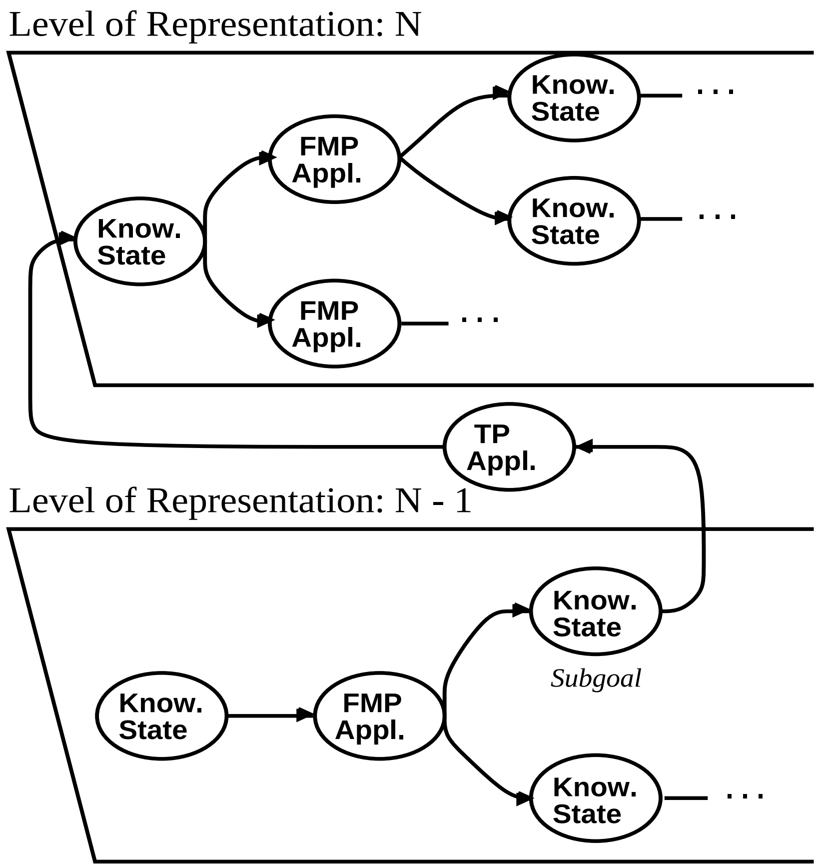

Figure 3: *A recognition graph. Levels of the graph are decision trees that verify hypotheses using feature measurement procedures (FMPs). Hypotheses that reach a subgoal are transformed to the next level of representation by transformation procedures (TPs).*

it is convenient to leave the choice nodes in the formalism for describing the optimization algorithm that produces minimum-cost trees.)

In SLS, decision trees represent the process of verifying or rejecting hypotheses. Choice nodes in the tree are hypothesis knowledge states, represented by sets of features, while chance nodes correspond to FMP invocations. The agent in this scenario is the control program that decides which feature to calculate next (i.e. which FMP to apply) based on the knowledge state of a hypothesis. The uncontrollable events are FMP invocations that return discrete features according to estimated distributions. Verification is a cycle in which the control strategy selects a FMP, the FMP returns a feature, and the control strategy selects another FMP. This cycle is represented in a decision tree as a progression from a choice node to a chance node and on to a new choice node. Eventually the process leads to a leaf node, corresponding to features that either verify or refute a hypothesis.

Figure 4 shows a complete SLS-style decision tree. Hypotheses begin at the start state with no computed feature values, leaving the control program to choose which feature to compute. In the example shown in Figure 4 the choice is between two FMPs, A & B. Whichever FMP is selected will return a feature, advancing the hypothesis to a new knowledge state. (The reader may note that duplicate knowledge states can be joined, since the same knowledge state results from applying A and then B as B and then A. This converts SLS's decision trees into directed acyclic graphs.)

Ultimately, the goal behind the decision tree formalism is not just to represent options and outcomes, but to aid in decision making. SLS constructs efficient verification strategies by determining at compile-time which options minimize the expected cost of verification. By making these decisions at compile-time, SLS eliminates the need for complex dynamic scheduling and permits the run-time control mechanism to be implemented as table-lookup.

2.3.2. Goal-level Classification

Each level of a recognition graph can be viewed as a classifier for distinguishing hypotheses that lead to good goal-level hypotheses from those that do not. An unusual feature of these classifiers is that they are allowed to produce false positive results but not false negatives, since verifying a poor hypothesis merely causes it to be transformed to a higher level of representation and retested, while rejecting a valid hypothesis may cause the strategy as a whole to fail.

The exception to this rule is at the goal level. Depending on the application, rejecting a valid goal-level hypothesis may or may not be as damaging as verifying a false one. Consequently, the best criterion function for training a goal-level classifier is task-specific. Goal-level classification techniques also depends on whether the recognition goal is to find a single object or to find multiple members of a class of objects. If the goal is to find a single item, only one hypothesis should be verified per image; otherwise, many hypotheses may be correct.

Goal-level classification is therefore unique. When a single hypothesis is re-

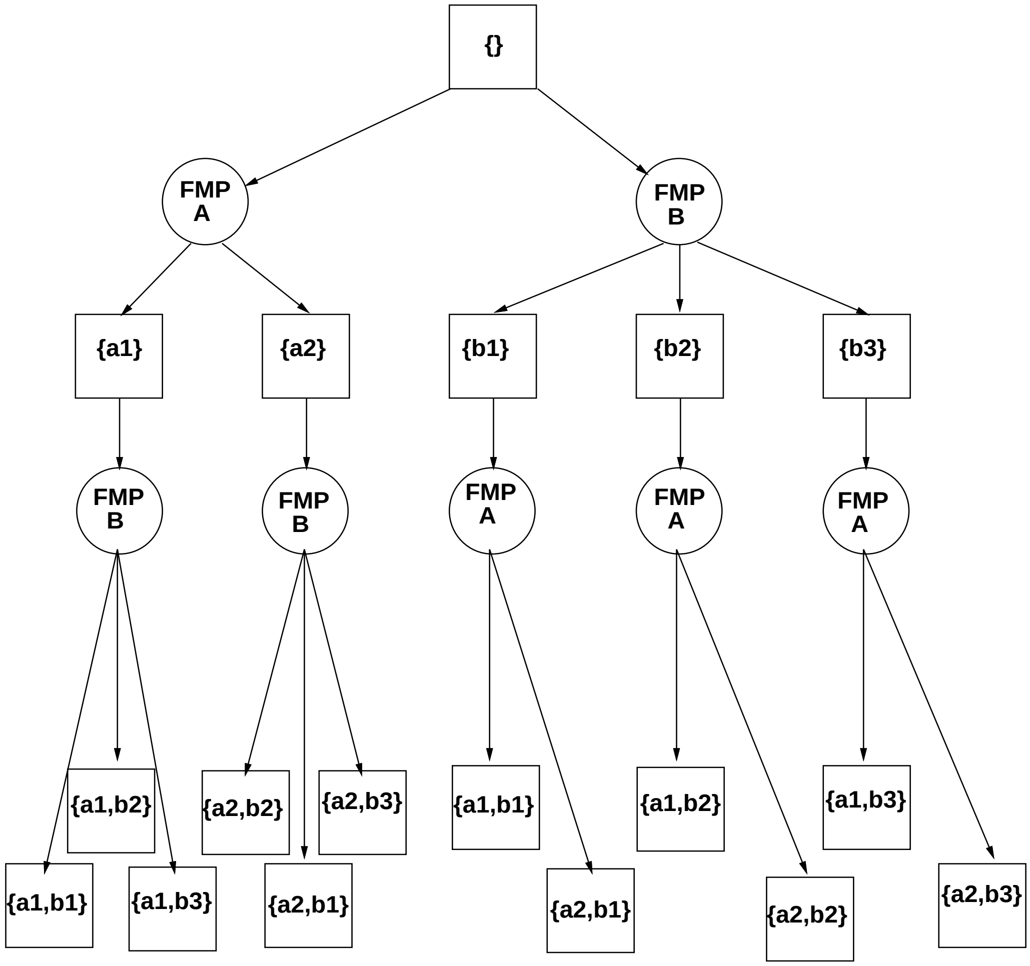

Figure 4: *A Decision Tree. The squares indicate choice nodes, where the agent chooses which action to take, and the circles indicate chance nodes representing actions with probabilistic outcomes. In SLS, the agent is the run-time control program, choice nodes are hypothesis knowledge states corresponding to sets of discrete feature values, and chance nodes are FMP invocations to determine feature values. (For efficiency, the implementation joins duplicate nodes, creating a decision graph rather than a decision tree.*

quired, run-time classifiers that compare hypotheses directly to each other and select the best are used. In the experiments presented in this paper, for example, the goal is always to find a single object, and a minimum-distance classifier is used to compare goal-level hypotheses and select the best. When multiple goal-level hypotheses may be correct, on the other hand, classifiers that do not compare hypotheses directly to each other are more appropriate.

2.3.3. Capabilities and Limitations of Recognition Graphs

So far, object recognition has been described as a "bottom-up" process starting with an image and ending with an abstract representation of an object. Although we will continue to use bottom-up terminology, it should be noted that recognition graphs can also represent "top-down" strategies and even mixed bottom-up and top-down strategies. "Bottom-up" strategies are created from TPs that create more abstract hypotheses from less abstract ones; top-down strategies are constructed from TPs that reduce abstract hypotheses to more concrete ones. Many strategies are mixed, using TPs that produce both more and less abstract hypotheses. The only constraint enforced by SLS on recognition graphs is that the VP library should not contain any loops, where hypotheses of type A are created from hypotheses of type B and *vice-versa.*

At the same time, recognition graphs are not capable of representing strategies based on relative strengths of hypotheses. Traditional blackboard systems can use heuristic schedulers that apply a knowledge source to the top N hypotheses at a level of representation, but such strategies cannot be embedded in recognition graphs. Recognition graphs can represent strategies that apply VPs to hypotheses with specific sets of features, but not to the N best hypotheses in an image. (This is why a minimum distance classifier is used to enforce the constraint that only one goal-level hypothesis by verified per image.)

SLS's strategies compare run-time hypotheses to training-time hypotheses. If training-time hypotheses with similar features led to correct goal-level hypotheses, then a hypothesis is pursued further; if not, it is rejected. SLS strategies base their control decisions not on the relative strengths of hypotheses from a single image, but on the relative strength of run-time hypotheses when compared to the larger (but less specific) pool of training hypotheses.

3. The Three Algorithms of SLS

At the heart of SLS are algorithms that create recognition graphs from training images. SLS learns recognition graphs through a three step process of *search, learning from examples,* and *graph optimization,* as shown in Figure 5. The search algorithm looks for sequences of transformation procedures that produce correct goal-level hypotheses; in the process, it also estimates the costs and likelihoods associated with VPs and FMPs. The learning from examples algorithm inspects the operator sequences identified by the search algorithm and infers a generalized

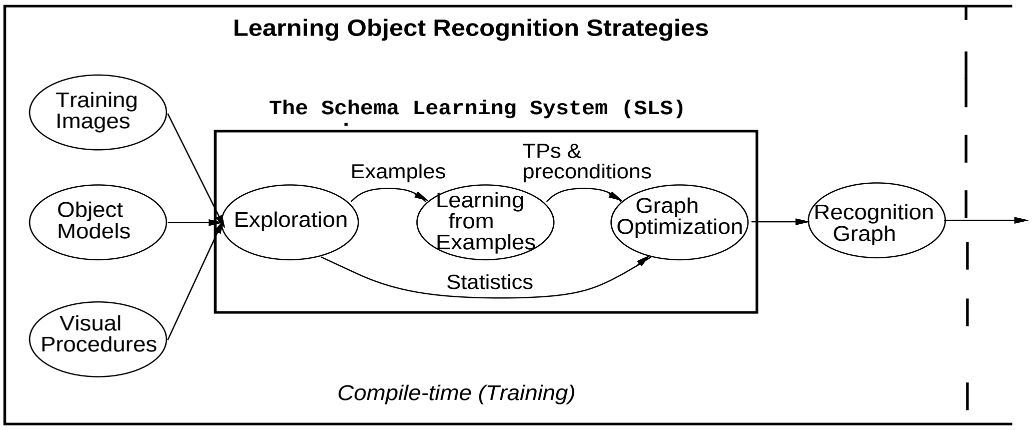

Figure 5: *The Three Algorithms of SLS. This figure expands the left-hand side of Figure 1 to show the search, learning from examples, and graph optimization modules that are the heart of SLS.*

concept of how correct goal-level hypotheses are generated. Typically it will discover that in order to recognize an object reliably, several (possibly redundant) operators must be applied to certain types of hypotheses. Finally, the graph optimization algorithm creates decision trees at each level of the recognition graph that minimize the expected cost of verification. The result is a multi-level recognition graph representing an efficient and reliable strategy for identifying the target object in terms of the specified goal (e.g. 2D or 3D, approximate or exact).

3.1. Search

The search algorithm applies visual procedures to training images and to intermediate-level hypotheses generated from training images. It begins by applying TPs to the images, producing intermediate hypotheses such as regions, lines, and points. The properties of these hypotheses are measured by FMPs, and then they are transformed by TPs into still more abstract hypotheses. The search algorithm exhaustively expands the tree of hypotheses in this way until no new hypotheses can be generated.

There are two reasons for exhaustively searching the space of hypotheses that can be generated from training images. The first is to provide correct hypotheses for the LFE algorithm. The training signal provided by the user distinguishes correct goal-level hypotheses from incorrect ones, but it does not indicate how goal-level hypotheses can be generated from images through sequences of intermediate-level hypotheses. To learn to generate goal-level hypotheses, SLS needs examples of how correct hypotheses are created. By exhaustively generating all possible hypotheses, the search algorithm is guaranteed to create as many correct hypotheses as possible, and it saves a record of how each hypothesis was generated.

The second reason for exhaustively searching images is to estimate the costs and benefits of VPs. In order to optimize the verification process, SLS has to know the probability of a feature given a hypothesis, as well as the expected cost of measuring that feature. Unfortunately, SLS's VP library does not include any information about the costs of FMPs or the probabilities of each discrete feature value. SLS therefore has to build up a statistical characterization of the FMPs by applying them to training images.

Although SLS is designed to maximize run-time, rather than compile-time, efficiency, there are many situations where exhaustively expanding the tree of possible hypotheses is not feasible. In such cases, the cost of exploration can be heuristically reduced by not exploring hypotheses that do not satisfy spatial constraints derived from the training signal. For example, if the recognition goal is to recover the three dimensional position of an object, any region hypotheses that do not overlap the object's projection can be rejected without being explored further. Similarly, points, lines, planes, and other types of geometric hypotheses can be rejected if they fail to overlap the correct solution or its projection. In this way, the combinatoric nature of exploration is damped, but the positive examples required by the LFE algorithm are still generated.

The disadvantage of this heuristic is that negative examples are used in SLS 1) by the LFE algorithm, to select the minimal cost DNF subterm (see Section 3.2.3.), and 2) to estimate the costs and probabilities associated with features. At the risk of a less efficient strategy, both tasks can be accomplished by exploring only a subset of negative hypotheses and extrapolating the results. We are currently experimenting with this and other heuristics for minimizing the search cost; nonetheless, the experiments in this paper were run using exhaustive search.

3.2. Learning from Examples (LFE)

SLS's learning from examples (LFE) algorithm analyses correct hypotheses produced during exploration and infers from them an efficient scheme for generating accurate goal-level hypotheses. The approach reflects the idea that recognition is a series of transformations interleaved with verifications. By looking at the histories of how correct hypotheses develop, SLS learns how to generate goal-level hypotheses from images through series of intermediate-level hypotheses. At the same time, it learns which features of intermediate hypotheses indicate that a hypothesis should be pursued, and which imply that a hypothesis should be abandoned.

3.2.1. Learning from Examples: A Definition

In the machine learning literature, the term *learning from examples* refers to algorithms that learn rules for evaluating examples. Following the terminology in the *AI Handbook* [12], learning from examples problems are defined in terms of *instance spaces* and *rule spaces*. The instance space is the set of possible examples or *instances* that might be encountered, either during training or testing. The rule space is the set of possible inference rules for evaluating instances. Learning from examples algorithms search rule spaces for the best methods of evaluating instances.

In SLS's LFE algorithm, the task is to generate correct goal-level hypotheses from images through sequences of intermediate representations. Instances are strings of hypotheses and TPs that lead from images to correct goal-level hypotheses. The rule space is composed of (sets of) features and TPs: the features determine which hypotheses should be pursued (at each level of representation), and the TPs indicate how they should be transformed. The goal of the LFE algorithm is to select a set of TPs and features that will generate a correct hypothesis for every target object instance in the training set, while generating as few false hypotheses as possible.

3.2.2. Dependency Trees

Inside the LFE algorithm, instances of correct hypotheses are represented as *dependency trees.* A dependency tree is an AND/OR tree recording the TPs and intermediate-level hypotheses on which a goal-level hypothesis depends. For example, a 3D pose hypothesis can be created by a geometric matching algorithm that finds the pose that minimizes the error of projecting a (3D) object model onto a set of (2D) image line segments. If so, the pose hypothesis is dependent on the geometric matching TP and the image line segments, as well as the TPs and hypotheses needed to generate the image line segments, as shown in Figure 6. In general, dependency is recursive, with 'AND' nodes resulting from TPs that require multiple arguments (and are therefore dependent on more than one hypothesis), and 'OR' nodes occurring when more than one TP redundantly generates the same hypothesis.

Each dependency tree represents the different methods for generating a specific hypothesis. In the example in Figure 6, pose-10 can be generated either by applying the geometric matching TP (to an image lineset hypothesis) or the planar distance TP[3], but at least one of the two is required. Furthermore, if the geometric matching TP is used, it must be applied to 2D-lineset-2. Alternatively, if the planar distance TP is used instead, it must be applied to region-14 and orientation-3. (The planar distance TP takes two image-based arguments.)

Dependency trees like the one in Figure 6 apply to specific hypotheses generated during the search phase of SLS. The first step in inferring a more generalized scheme for generating goal-level hypotheses is to replace specific hypotheses with their feature vectors, as shown in Figure 7. The rationale for the substitution is that TPs have preconditions associated with them that select which hypotheses they should be applied to. If a TP needs to be applied to hypothesis H to ensure that a goal is met, then only features of H should be considered as preconditions

[3]The Planar Distance TP converts a 3D orientation hypothesis into a plane hypothesis by scaling it relative to a region or set of image points.

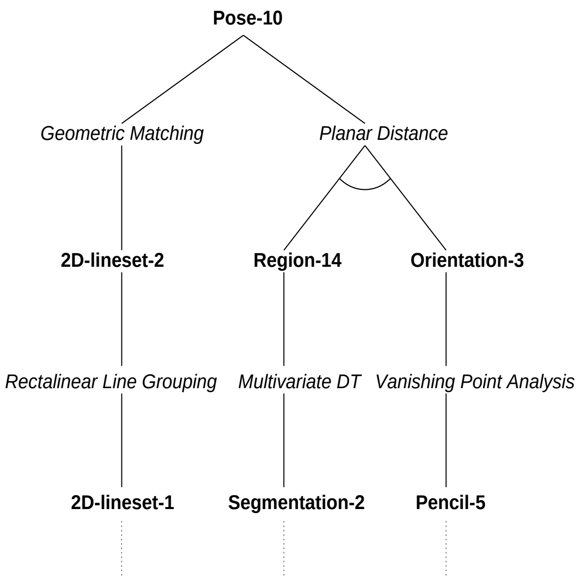

Figure 6: *An example of a dependency tree showing the different ways that one correct pose hypothesis can be created during training. The names of transformational procedures are in italics, while hypotheses (data instances) are in bold.*

for the TP.

In general, a hypothesis is guaranteed to be created by any set of preconditioned TPs that "satisfies" its dependency tree. A dependency tree DT is satisfied by a set of TPs G (with affiliated preconditions P) if: 1) the root of DT is an AND node, and every subtree of DT is satisfied; 2) the root of DT is an OR node, and at least one subtree of DT is satisfied; or 3) the root of DT is a leaf node with TP g and preconditions P such that g is in G and the preconditions of g either match or are a superset of P.

3.2.3. LFE: A DNF-based Algorithm

The algorithm for finding optimal sets of TPs and preconditions is deceptively simple:

1. Convert the generalized dependency tree of a correct goal-level hypothesis to disjunctive normal form (DNF)[4].
2. For every other correct goal-level hypothesis:
   (a) Convert its generalized dependency tree to DNF.
   (b) "AND" together the new DNF expression with the previous DNF expression.

[4]The disjunctive normal form of a logical expression is an OR of ANDs of monomial expressions, for example $(A \wedge B) \vee (A \wedge C)$.

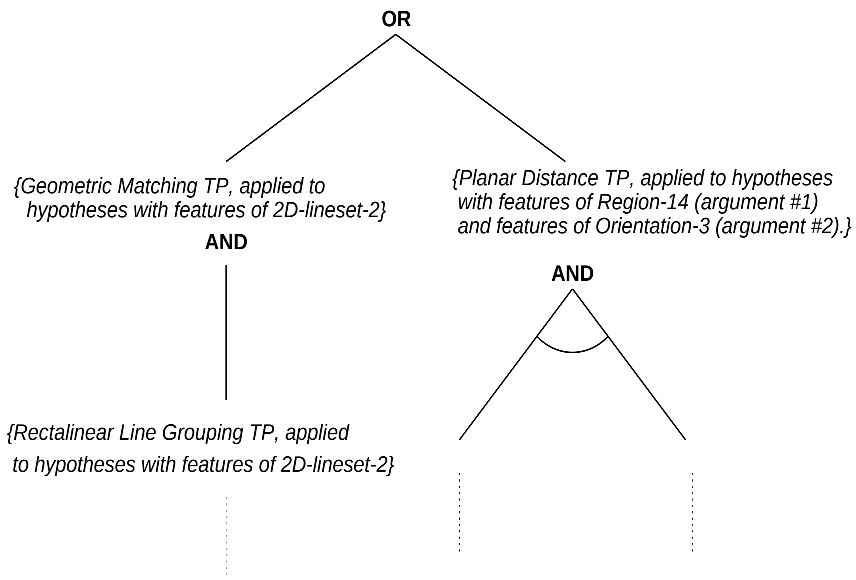

Figure 7: *A generalized dependency tree created by replacing the hypotheses in Figure 4.3 with their feature values.*

   (c) Convert the resulting 'AND' tree to DNF[5].

3. Select the conjunctive subterm that generates the fewest total hypotheses.

By the logic of the dependency relation, the TPs and preconditions in any conjunctive subterm of the final DNF expression are sufficient to re-generate all correct goal-level hypotheses from the training images. By selecting the minimal term, SLS chooses the best method for generating correct hypotheses.

AND/OR dependency trees are converted to DNF by a standard algorithm that first converts every subtree to DNF and then either merges the subterms, if the root is an OR node, or takes the symbolic cross product[6] of the subterms, if the root is an AND node. If a TP is ANDed with itself when taking the cross product, the resulting preconditions are the intersection of the preconditions of the two instances being ANDed.

This basic algorithm is altered slightly to improve efficiency. Because SLS seeks to find the minimal term (measured as the number of hypotheses generated) of the DNF expression rather than every term, any conjunctive subterm that is a logical superset of another can be pruned, reducing the total number of terms considered. A second modification is to sort the correct goal-level hypotheses

[5]Logically, this algorithm is equivalent to the simpler two-step process of ANDing all the dependency trees together and converting the result to DNF. However, iteratively adding each new dependency tree to an evolving expression simplifies the probabilistic analysis given in [15].

[6]Symbolic cross product: $\{A, B\} \times \{C, D\} = \{AC, AD, BC, BD\}$.

according to the size of their dependency trees and to iterate in step two from the simplest dependency trees to the most complicated. This reduces the size of the interim DNF expressions without affecting the final result.

3.3. Graph Optimization

As was stated earlier, recognition graphs interleave verification and transformation, using FMPs to measure properties of hypotheses and TPs to transform them to higher levels of representation. By building dependency trees from the training samples, converting them to DNF and picking the minimal subterm, SLS learns which TPs to use to transform hypotheses from one level to the next. Just as important, it learns which preconditions a hypothesis must meet before it should be transformed. These preconditions are the subgoals of the recognition process at intermediate levels of representation.

The optimization algorithm builds decision trees for each level of representation that minimize the expected cost of reaching a subgoal or, conversely, of deciding that a hypothesis cannot satisfy a subgoal and should be rejected. The decision trees are constructed by first building a graph representing all possible sequences of FMP applications, and then optimizing the graph by determining which options at each choice node minimize the overall cost of recognition, and removing the other options. The final result is a decision tree at each level of representation that minimizes the expected cost of verification.

3.3.1. Estimating FMP properties

A preliminary step to building efficient decision trees is to characterize the performance of FMPs. In particular, SLS estimates:

- **Expected Cost (VP, F),** the expected cost of applying a VP to a hypothesis with the feature values F;
- **Feature Likelihood (FMP, f1, F),** the likelihood of a FMP returning feature value f1 when applied to a hypothesis with feature values F.

In general, these values are estimated from applications of FMPs to similar hypotheses during training. When an insufficient number of similar hypotheses (i.e. hypotheses with feature values F) are generated during training, the dependency on F is dropped and the values are estimated across all hypotheses.

Unfortunately, these statistical measures cannot be inferred directly from the search data, because the probabilities and costs associated with features depend on the quality of the hypotheses being measured. The search algorithm, which exhaustively explores the space of possible hypotheses, generates more hypotheses of lesser quality than SLS's run-time recognition strategy will. The exploration hypotheses are therefore drawn from a different statistical distribution than the run-time hypotheses will be.

As a result, the estimations of FMP performance are delayed until after the LFE algorithm has been run. The results of LFE are used to prune the exploration data, removing those hypotheses that are merely artifacts of exhaustive search and would not be generated using the VPs and preconditions selected by the LFE algorithm. Once the search data has been pruned, the remaining hypotheses are used to characterize the performance of VPs.

3.3.2. Graph Layout

For each level of representation, a directed acyclic graph is constructed representing all possible sequences of FMP applications. The graph starts from a single knowledge state, corresponding to a newly generated hypothesis for which no features have been computed. The start state, like all knowledge states, is a choice node, since the control program gets to choose which FMP to apply first. FMP applications nodes are therefore added for every feature that can be measured of a hypothesis in the start state. These FMP application nodes lead to new knowledge states (one for each possible feature value), which in turn have more FMP applications attached to them, and so on. The expansion of the graph continues until it reaches either a subgoal knowledge state or a knowledge state that is incompatible with every remaining subgoal (i.e. a failure state).

For example, Figure 8 shows the initial graph for a level of representation with two FMPs and a subgoal of {a1,b1}. Graph construction begins with the start state and expands by adding a chance state for each FMP. The FMPs lead to a total of five new knowledge states, but three of them are failure states that are incompatible with the subgoal {a1,b1}. The other two states each have one more FMP to be applied, leading to four more knowledge states, one of which is the subgoal state and three of which are failure states.

More formally, we refer to subgoal states and failure states as the *terminal* states for each level of the recognition graph. The cost of promoting a hypothesis from knowledge state n to a terminal state is called the Expected Decision Cost (EDC) of knowledge state n, and the expected cost of reaching a terminal state from state n using FMP v[7] is the Expected Path Cost (EPC) of n and v. Since features are discrete, we denote the possible outcomes of a FMP v as a set $R(v)$, and the probability of a particular feature value f being returned as $P(f|v,n)$, $f \in R(v)$.

The EDC's of knowledge states can be calculated starting from the terminal states and working backward through the recognition graph. Clearly, the EDC of a subgoal or failure state is zero:

$$EDC(n) = 0, \quad n \in \{terminal\ states\}.$$

The expected path cost of reaching a terminal state from a FMP application node is:

$$EPC(n,v) = C(v) + \sum_{f \in R(v)} (P(f|n,v) \times EDC(n \cup f))$$

[7] v is an awkward abbreviation for a feature measurement procedure, but f will be used for feature values and p would look like a probability value. Since FMPs are a subclass of VPs, v is therefore used.

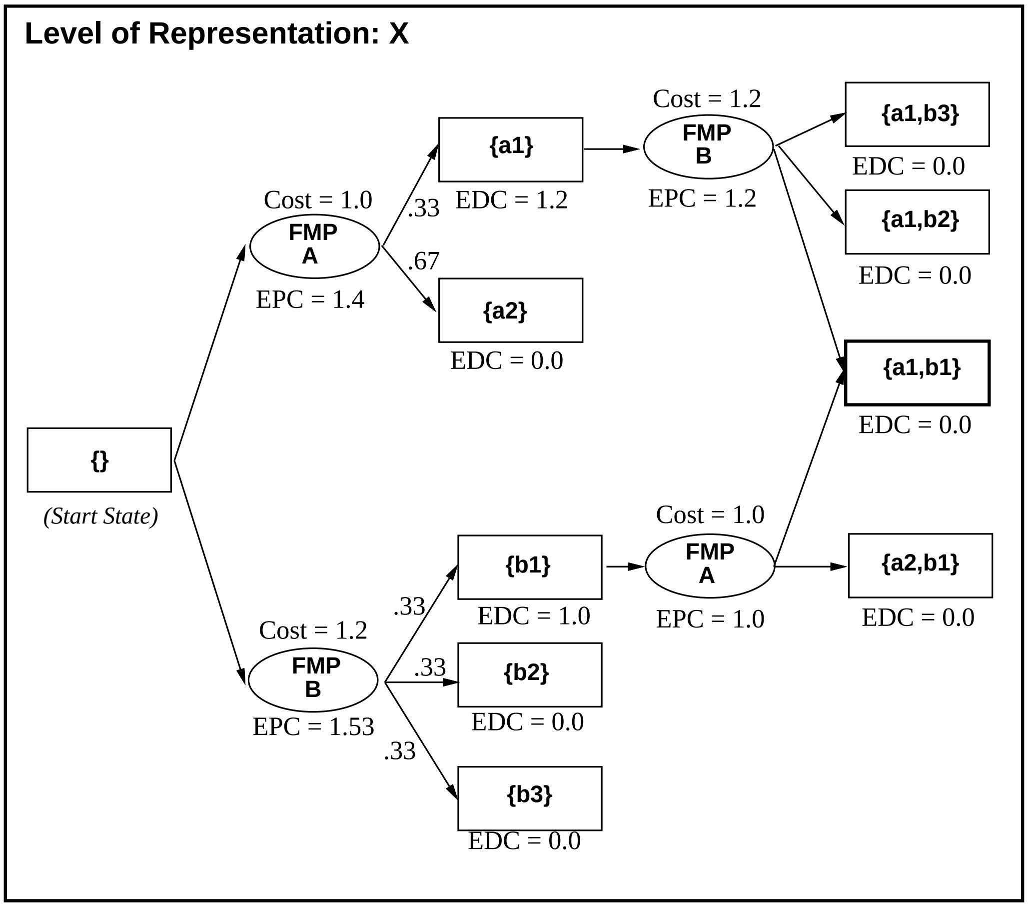

Figure 8: *An initial decision graph. Choice nodes, shown as rectangles, correspond to knowledge states of a hypothesis. Chance nodes, shown as ovals, represent FMP applications. Starting from an empty knowledge state, the system adds a chance node corresponding to each FMP. Since FMPs measure feature values, they lead to new knowledge states, where new FMPs can be selected. The graph expands until it reaches either a verification state, or a state that is incompatible with the features of a verification state.*

where n is a knowledge state expressed as a set of feature values, $n \cup f$ is the knowledge state that results from FMP v returning feature value f, and $C(v)$ is the estimated cost of applying v.

The EDC of a knowledge state, then, is the smallest EPC of the FMPs that can be executed from that state:

$$EDC(n) = \min_{v \in VP(n)} (EPC(n, v))$$

where $VP(n)$ is the set of FMPs applicable at node n. The minimal-cost decision tree is created by making a single pass through the directed acyclic graph, starting at the terminal nodes and working backward toward the start state. At each knowledge state, the pruning process calculates the EPC of every FMP that can be applied from that state, and removes all FMP application nodes except the one with the smallest EPC. The final result is a minimal-cost decision tree.

Figure 9 shows the result of pruning the initial graph shown in Figure 8. Starting at the terminal nodes and working backward, the first choice states the pruning algorithm considers are $a1$ and $b1$. These states have only one option each, however, so selecting the minimum-cost option has no effect. The next choice node encountered is the start state $\{\}$, where there are two options, since the system can choose to compute feature A or feature B. However, as depicted in Figure 8, the expected cost (EPC) of verifying a hypothesis if feature B is computed first is 1.53, while the cost of verifying a hypothesis by computing feature A first is only 1.4. Consequently, the optimization algorithm prunes option B from the start node in Figure 8, leaving the optimized decision tree shown in Figure 9.

3.3.3. Estimating Total Cost

The equations above establish a mutually recursive definition of the expected decision cost of a knowledge state. The EDC of a knowledge state is the EPC of the optimal FMP application from the state; the EPC of a FMP application is the expected cost of applying the FMP plus the expected EDC remaining after the FMP has been applied. The recursion bottoms out at terminal nodes, whose EDC is zero. Since every path through the object recognition graph ends at either a subgoal or a failure node, the recursion is well defined.

Furthermore, the total cost of recognition can be estimated from the EDCs of start states and the expected costs of the TPs selected by the LFE algorithm. The EDC of the start state for a level of representation estimates the expected cost of verifying or rejecting hypotheses at that level. By estimating the total number of hypotheses generated at each level by the preconditioned TPs and multiplying it by the EDCs of the start states, the total cost of verification can be estimated. Since the expected number of times a TP will be executed can also be estimated from the LFE algorithm's results, the total expected cost of recognition can be obtained easily.

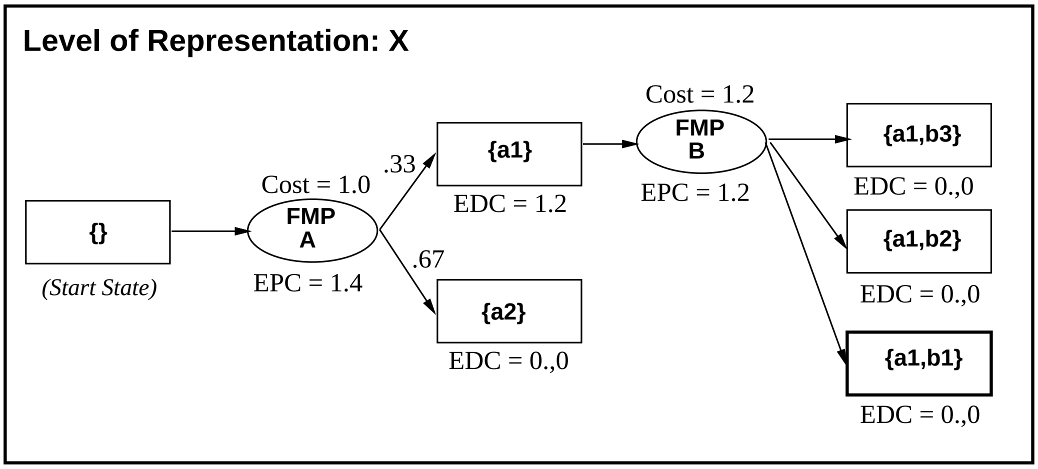

Figure 9: *A pruned decision graph. This Figure shows the graph depicted in Figure 8 after it has been pruned by the graph optimization algorithm. All actions which either do not lead to the subgoal state or which are not on the most efficient path to the subgoal have been removed.*

4. Experiments

We present an example of SLS learning to recognize the (3D) position and orientation of a building in images taken from an approximately known location. As described in [15], SLS has also been used to recognize the (2D) image position of a tree from an approximately known viewpoint and the (3D) position and orientation of another building from an unknown viewpoint. Taken together, these exercises suggest that SLS can recognize both natural and man-made objects, can recognize them from either known or unknown viewpoints, and can do so in either two dimensions or three. In this paper, however, we shall limit ourselves to the one demonstration of finding the pose of the Marcus Engineering building from an approximately known viewpoint.

4.1. Training Images

The training data is selected from a set of twenty-one images collected along a hundred foot stretch of a footpath on the UMass campus. Figures 10 and 11 show the first and last images of the sequence. The images were taken level to gravity ($\pm 1°$) and from approximately four feet above the ground, although the ground rises and falls over the course of the sequence. The camera was also subjected to small rotations in pan from one image to the next. As a result, the pose of the camera has four degrees of freedom, with large variations in position in the ground plane and smaller deviations in camera height and pan.

The "ground truth" positions and orientations of the Marcus Engineering

Figure 10: *The first of twenty-one training images. The images were taken along a hundred-foot section of the path, with the camera level to gravity.*

Figure 11: *The last of twenty-one training images. The pose of the camera has four degrees of freedom, with large variations in position in the XZ (ground) plane and small differences in camera height and pan.*

building were determined by manually matching image points to model points and applying Kumar and Hanson's algorithm [23] to determine the building's pose relative to the camera. The training signal is therefore composed of errorful pose estimates, rather than true positions. However, Kumar and Hanson's results suggest that, with correct correspondences, their algorithm produces pose estimates that are extremely accurate when compared to the relatively lax error thresholds in the recognition goal. The estimated poses can therefore reasonably be used as a training signal.

4.2. Recognition Goal

The recognition goal is to find the pose of Marcus Engineering relative to the camera. Pose hypotheses are represented as rotation matrices with translation vectors, in the traditional

$$P' = RP + T$$

representation, where R and T are the rotation matrix and translation vector that transform a set of points P in the model's coordinate system into a set of points P' in the camera's coordinate system. Unfortunately, errors expressed in terms of R and T tend to be unintuitive, since if an object is rotated slightly about its center, this will be represented as a rotation about the focal point, counteracted by a large translation[8]. It is helpful, therefore, to express the error tolerances in a different representation.

Since the pose of the building has only four degrees of freedom, the tolerance thresholds in the recognition goal are expressed in terms of scale, image position, and object angle. These parameters reflect the fact that the pose of the building can be expressed as a vector from the focal point to any known point on the building, plus a horizontal rotation about the known point (remember that the building has no tilt or roll relative to the camera). Errors in the positional vector are expressed as an error in length, measured as a percent of the true camera-to-object distance, and an error in position, measured as an angle. (Since we are interested in the magnitude of the orientation error, not its direction, this can be written as a scalar.) Errors in the rotation of the object are also represented as an angle, this time about the axis of gravity.

The error thresholds for this exercise require the position of the building to be correct to within one degree of image angle and ten percent depth, while the orientation of the building must be correct to within five degrees around the axis of gravity. This implies that the hypothesized building poses should be highly accurate with respect to image position and reasonably accurate in orientation, but only approximate in depth. Figure 12 shows an example of a pose that satisfies these criteria, in this case the building pose identified by SLS's strategy in the first of twenty-one trials.

[8] The size of the counteracting translation is a function of both the extent of the rotation and the distance from the object center to the focal point.

Figure 12: *A correct pose from one trial of the Marcus recognition strategy. The pose shown here was generated and verified on the first of twenty-one trials, and is off by 1.8% in depth, 4.79 degrees in orientation about the axis of gravity, and 0.16 degrees in image location.*

4.3. The Visual Procedure Library

The visual procedure library used in this experiment is shown in Table 1. It includes many procedures for extracting and grouping two-dimensional representations such as points and lines. Lines can be extracted using the edge-linking algorithm of Boldt and Weiss [7], and regions can be extracted by the algorithm described in Beveridge, et. al. [5]. Regions can also be created by fitting a convex hull to a set of line segments. Regions that match an expected color and texture can be selected from a region segmentation by a multivariate decision tree [8, 14]. (This algorithm is included twice in the library with two different parameterizations, one designed to select red brickface regions, the other highly textured window regions.) Nearby regions can be grouped by a region merging TP, while another TP groups lines that intersect a given region. Nearby lines that are parallel, collinear or orthogonal can be grouped according to the relations defined by Reynolds and Beveridge [28]. (All of the grouping VPs are implemented using the facilities of the ISR database system [9].) Image points are extracted by finding trihedral junctions of lines.

The procedure library also includes routines that create or consume three-dimensional representations. Orientation hypotheses represent the orientation, but not location, of a plane in space, while planar surface hypotheses specify both the orientation and location of a plane. Most importantly, transformation hypotheses represent a coordinate transformation from one coordinate system to

Transformational Proc.	**Input**	**Output**	**Ref**
Line Extraction	Image	2D Lines	[7]
Region Segmentation	Color Image	Segmentation	[5]
Multivariate Decision Tree	Segmentation	Regions	[8, 14]
Region Merging	Regions	Region	
Interest Operator (Anandan)	Image, Region	2D Points	[2]
Interest Operator (Moravec)	Image, Region	2D Points	[24]
Min. Dist. Classification	Region	Label	
Polygonal Approximation	Region	2D Lines	
Line Extension	2D Lines	2D Line Groups	
Rectalinear Line Grouping	2D Lines	2D Line Groups	[28]
Pencil Extraction	2D Lines	Line Pencils	[13]
Vanishing Point Analysis	Line Pencils	3D Orientation	[13]
Convex Hull	Line Pencil	Region	
Trihedral Jnct Finder	2D Lines	Trihedral Jnct	
Trihedral Angle Analysis	Trihedral Jnct	3D Orientation	[22]
Planar Distance (Scale)	Region or 2D Points, 3D Orientation	3D Pose	
Subgraph Isomorphism	2D Line Group	Correspondence	[29]
Image Resection	2D Points, Correspondence	3D Pose	
Perspective Projection	3D Pose	2D Points	
Geometric Matching	2D Lines	3D Pose	[6]

Table 1: *Transformational Procedures (TPs). Note that the Input column lists only image-based arguments. Model-based arguments are not listed for Subgraph Isomorphism, Geometric Matching and Planar Distance TPs.*

another, represented as a rotation matrix and a translation vector. Transformation hypotheses determine the pose of a modeled object by giving the transformation from the object model coordinate system to the camera coordinate system, and are goal-level hypotheses in this demonstration.

Three dimensional hypotheses are generated and manipulated by many visual procedures. Collins and Weiss [13] provide an efficient TP for grouping line segments into *pencils*, which are sets of lines that meet at a common point of intersection. Vanishing point analysis [13] infers the orientations of planes in space by assuming that the image lines in a pencil are the projections of parallel lines in space. Another approach to inferring the orientation of an object in space is to find trihedral junctions of line segments first, and then use the perspective angle equations of Kanatani [22] to infer the orientations of the planes, assuming the lines form right angles, like the corner of a building.

The distance from an object to the camera can be estimated when the size of the object is known; in the case of Marcus Engineering, the size of the building (and its wire-frame model) was extracted from its blueprints. Two parameterizations of the scaling TP are available in the VP library, one that estimates distance based on the apparent width of a window and the estimated angle of the building face, and a second that estimates distance from the height of the building using a direct inverse relationship of size to distance. (Note that since the images have zero tilt, the orientation of the building face is not needed to estimate distance from the building's height). Of course, since any two points on the object model can serve as compile-time parameters to a scaling TP, many other parameterizations of the scaling TP could be included in the library.

Although the visual procedure library shown in Table 1 is sufficient for the purposes of this experiment, it includes just a few of the computer vision algorithms described in the literature. Unfortunately, the current Lisp implementation of SLS has proved an impediment to building a larger library, since source code for most visual procedures is available only in C. We are currently reimplementing SLS in C++, in part so that it can access libraries of computer vision routines such as those included in Khoros [27] and KBVision [31] (and someday the IUE [25]). Already the C++ version of SLS can access over fifty visual procedures, including almost all Khoros and KBVision routines. Unfortunately, no experimental results are yet available that use the new implementation.

4.4. Testing Methodology

Because of the relatively small size of the training set, SLS was tested with a "leave one out" methodology, in which strategies are trained on twenty images and tested on the twenty-first. The process is repeated twenty-one times, each time with a different image "left out" of the training set and used as the test image. Each trial tests whether a strategy learned over twenty training images satisfies the recognition goal on the twenty-first. In addition to testing for robustness, the suite of twenty-one trials also tests SLS's ability to predict the reliability and average cost of its strategies.

4.5. Reliability Results

Table 4.5. summarizes the results of twenty-one trials of learning to recognize the pose of Marcus Engineering from an approximately known viewpoint. The right side of the table shows the errors in the best goal-level hypothesis generated, even if this hypothesis was never verified, while the right side shows the errors in the goal-level hypothesis verified by the minimum distance classifier. The verified pose for trial number one, which is also the best pose generated for that trial, was shown earlier in Figure 12.

Pose errors in Table 4.5. are measured in terms of the length and orientation of a vector from the focal point to the corner of the building, and the rotation of the building. More precisely, the error in the position of the building is measured as 1) the error in the distance to the building, measured as a percentage of the true distance, and 2) the image position of the building, measured by the angle between the true vector from the focal point to the building corner and the estimated vector (labeled "Im Pos" in Table 4.5.). The error in the building's orientation is measured as the angle about the gravitational axis between the estimated orientation of a building face and its true orientation (labeled "Rot." in Table 4.5.).

The most striking feature of Table 4.5. is the result of trial sixteen. The strategy learned by SLS in trial sixteen did not generate a single goal-level hypothesis, either correct or incorrect, for the test image. An *a posteriori* analysis reveals that in twenty of the twenty-one images, the corner of the building is marked by a trihedral junction of image lines. In one image, however, noise eliminates one of the three lines. As a result, when the image without the trihedral junction is removed from the training set (and used as the test image), SLS learns a strategy that relies entirely on finding trihedral junctions. The strategy does not succeed in finding any trihedral junctions in the test image, however, and therefore generates no goal-level hypotheses. Ironically, in the other twenty trials, the training sets include the case in which trihedral junctions fail, and therefore the other twenty strategies all include redundancy to account for the possibility of trihedral failure, a redundancy that is never needed for the test images to which they are applied.

Trial sixteen is the only case in which the strategy learned by SLS fails to generate a correct goal-level hypothesis, giving it a success rate at generating 3D building hypotheses of 95%. In trials fifteen and twenty, however, SLS verified the wrong hypothesis, giving it an over-all success rate of 86%.

4.6. Timing

Table 4.5. addresses the robustness of the strategies learned by SLS, but not their efficiency. Table 4.6. shows SLS's expected run-time (in seconds, rounded to the nearest whole second) for the strategy learned in each trial as compared to the actual run-time when the strategy was applied to a test image. On any given

Table 2: *Results of twenty-one trials of learning to recognize the pose of the Marcus Engineering Building. The left side of the table shows the errors in the best goal-level hypothesis generated for each trial, while the left side shows the errors in the hypothesis verified by the minimum distance classifier. Errors are specified in terms of the parameters discussed in Section 4.2., namely: 1) error in distance from the object to the camera, expressed as a percentage of the true distance from the object to the camera; 2) error in orientation about the axis of gravity; and 3) error in image position, measured in degrees.*

	Best Generated Pose			Selected Pose		
Trial	Dist.	Rot.	Im Pos	Dist.	Rot.	Im Pos
1	1.81	4.79	0.16	1.81	4.79	0.16
2	1.34	0.82	0.05	1.34	0.82	0.05
3	1.18	1.33	0.11	2.74	1.33	0.09
4	0.69	1.40	0.13	1.97	1.40	0.13
5	2.64	2.33	0.10	2.64	2.33	0.10
6	0.73	6.95	0.15	0.73	6.95	0.15
7	0.27	1.16	0.05	6.58	1.16	0.07
8	2.33	0.07	0.08	2.33	0.07	0.08
9	1.01	3.58	0.20	1.69	3.58	0.20
10	1.39	1.65	0.04	2.07	1.65	0.05
11	0.50	3.27	0.08	2.37	3.27	0.25
12	2.24	4.48	0.25	2.24	4.48	0.25
13	2.07	1.54	0.04	2.07	1.54	0.04
14	0.36	1.63	0.09	0.36	1.63	0.09
15	2.13	2.58	0.21	11.23	2.58	0.21
16	–	–	–	–	–	–
17	4.44	6.21	0.13	4.44	6.21	0.13
18	1.07	1.75	0.09	1.77	1.76	0.16
19	0.41	3.83	0.07	1.07	3.83	0.07
20	0.92	2.50	0.03	14.64	2.50	0.03
21	1.18	4.95	0.13	2.26	4.95	0.13

Table 3: *Timing results for the twenty-one Marcus Engineering trials. The first row shows the expected cost (in seconds, rounded to the nearest whole second) of applying the strategy, as predicted by SLS. The second row shows the actual cost.*

Trial	1	2	3	4	5	6	7	8	9	10	11
Exp.	82.9	84.7	84.2	78.6	85.1	85.0	84.9	85.0	85.2	85.3	84.6
Act.	107.3	88.4	79.4	113.7	88.9	74.8	66.2	71.7	72.6	67.4	74.5

Trial	12	13	14	15	16	17	18	19	20	21	Avg.
Exp.	86.2	85.1	85.4	83.2	61.3	79.6	85.5	84.9	80.1	85.7	83.0
Act.	61.4	89.4	73.2	82.7	45.5	62.3	74.4	69.2	76.6	57.4	79.3

trial, the discrepancy between the predicted and actual run-times is quite large. On average, however, the predicted run-times are within five percent of the actual run-times. This reflects the average-case nature of expected costs. The actual cost of recognizing an object in an image depends critically on the contents of the image, but as long as the training images are indicative of the test domain the average cost of recognition can be estimated. Indeed, an *a posteriori* analysis of the data shows that the five percent overestimate of the expected run time was caused by VPs executing more quickly during testing than during exploration, due primarily to variations in paging.

5. Conclusions

The primary contribution of SLS is that it automatically learns special-purpose recognition strategies under supervision. There has been earlier work on learning shape-based recognition strategies from CAD/CAM models [20, 10], and on learning to recognize two-dimensional objects from features that can be measured directly in the image (without using intermediate representations) [11]. None of these systems, however, can do what SLS can do: learn to recognize artificial or natural objects in complex images by integrating cues from shape, color, context and other types of knowledge.

SLS is able to achieve these goals because it reasons across multiple levels of representation, and takes advantage of the wealth of available computer vision procedures. By casting object recognition as a discrete control problem, SLS is able to leverage off 30 years of computer vision research, selecting those representations and techniques that are appropriate for a given task. Moreover, because it learns control policies automatically, it can generate systems capable of recognizing specific objects for a variety of applications, using already available components.

A more mundane, but still important, contribution is that SLS is the first system to supply a user (or application program) with an estimate of the expected cost of satisfying a recognition goal. This information can be critical for planning and resource allocation in robotic systems that rely on computer vision. Just as importantly, if the library of visual procedures is incapable of robustly achieving a recognition goal, SLS warns the user that the goal will not be met.

6. Future Work

SLS remains the focus of a great deal of research. The top two priorities are 1) reducing the cost of the initial search phase and 2) converting the system to incremental, rather than batch, learning. Effort is also being invested to reimplementing SLS in C++, partly to make it run faster but mostly to allow it to access available libraries of C and C++ computer vision algorithms, including those available within the Khoros [27] and KBVision [31] systems.

The relationship between SLS, dynamic programming (DP) and reinforcement learning (RL) is also being explored. Although there are many problems in applying DP or RL to SLS's domain, the potential benefits in terms of reliable incremental systems make the possibilities worth pursuing.

Finally, SLS assumes a library of executable procedures, when in fact most computer vision algorithms must be parameterized before they can be used. These parameters vary from simple sensitivity settings to complex geometric structures (such as might be given to a graph matching algorithm). So far, all such parameters have been specified by hand, and at times different parameters are used for different tasks. For SLS to completely eliminate human knowledge engineering, it must learn to parameterize visual procedures as well as control them. This, too, is an area of current research.

References

[1] J. Aloimonos. Purposive and qualitative active vision. In *Proceedings of the DARPA Image Understanding Workshop*, pages 816–828, Pittsburgh, PA, Sept 1990.

[2] P. Anandan. A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision*, 2:283–310, 1989.

[3] M. A. Arib. *The Metaphorical Brain: An Introduction to Cybernetics as Artificial Intelligence and Brain Theory.* Wiley Interscience, New York, 1972.

[4] D. H. Ballard. Animate vision. *Artificial Intelligence*, 48:57–86, 1991.

[5] J. R. Beveridge, J. Griffith, R. R. Kohler, A. R. Hanson, and E. M. Riseman. Segmenting images using localized histograms and region merging,. *International Journal of Computer Vision*, 2:311–347, 1989.

[6] J. R. Beveridge, R. Weiss, and E. M. Riseman. Optimization of 2-dimensional model matching. In *Proceedings of the DARPA Image Understanding Workshop*, pages 815–830, Palo Alto, CA, June 1989.

[7] M. Boldt, R. Weiss, and E. M. Riseman. Token-based extraction of straight lines,. *IEEE Transactions on Systems, Man and Cybernetics*, 19(6):1581–1594, 1989.

[8] C. E. Brodley and P. E. Utgoff. Multivariate decision trees. *Machine Learning.* to appear.

[9] J. Brolio, B. A. Draper, J. R. Beveridge, and A. R. Hanson. The isr: an intermediate-level database for computer vision. *Computer*, 22(12):22–30, 1989.

[10] O. I. Camps, L. G. Shapiro, and R. M. Haralick. Premio: The use of prediction in a cad-model-based vision system. Technical Report EE–ISL–89–01, University of Washington, Dept. of Electrical Engineering, Seattle, WA, 1989.

[11] C.H. Chen and P. G. Mulgaonkar. Automatic vision programming,. *CGVIP: Image Understanding*, 55(2):170–183, 1992.

[12] P. R. Cohen and E. A. Feigenbaum, editors. *The Handbook of Artificial Intelligence*, volume 3. William Kaufman, Inc., Los Altos, CA, 1982.

[13] R. T. Collins and R. S. Weiss. Vanishing point calculation as a statistical inference on the unit sphere. In *Proceedings of the International Conference on Computer Vision*, pages 400–405, Osaka, Japan, December 1990.

[14] B. Draper, C. Brodley, and P. Utgoff. Goal-directed classification using linear machine decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(9):888–893, 1994.

[15] B. A. Draper. *Learning Object Recognition Strategies.* PhD thesis, University of Massachusetts, Amherst, MA, May 1993.

[16] B. A. Draper, R. T. Collins, J. Brolio, A. R. Hanson, and E. M. Riseman. The schema system,. *International Journal of Computer Vision*, 2:209–250, 1989.

[17] B. A. Draper and A. R. Hanson. An example of learning in knowledge-directed vision,. In P. Johansen and S. Olsen, editors, *Theory and Applications of Image Analysis*, pages 237–252. World Scientific, 1992.

[18] A. R. Hanson and E. M. Riseman. Visions: A computer system for interpreting scenes. In A. R. Hanson and E. M. Riseman, editors, *Computer Vision Systems*, pages 303–333. Academic Press, New York, 1978.

[19] F. S. Hillier and G. J. Lieberman. *Introduction to Operations Research*, chapter 15. Holden-Day, Inc., San Francisco, CA, 1980.

[20] K. Ikeuchi. Generating an interpretation tree from a cad model for 3d-object recognition in bin-picking tasks. *International Journal of Computer Vision*, 1:145–165, 1987.

[21] K. Ikeuchi and M. Hebert. Task oriented vision. In *Proc. Internatiional Conference on Intelligent Robots and Systems*, Raliegh, NC, July 1992.

[22] K. Kanatani. Constraints on length and angle,. *Computer Vision, Graphics, and Image Processing*, 41:28–42, 1988.

[23] R. Kumar and A. R. Hanson. Determination of camera position and orientation,. In *Proceedings of the DARPA Image Understanding Workshop*, pages 870–881, Palo Alto, CA, May 1989.

[24] H. P. Moravec. *Robot Rover Visual Navigation.* UMI Research Press, Ann Arbor, MI, 1981.

[25] J.L. Mundy and IUE Committee. The image understanding environment: Overview. In *Proceedings of the DARPA Image Understanding Workshop*, pages 283–288, Washington, D.C., April 1993.

[26] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[27] J. Rasure and S. Kubica. The khoros application development environment. In H. Christensen and J. Crowley, editors, *Experimental Environments for Computer Vision and Image Processing*, pages 1–32. World Scientific, 1994.

[28] G. Reynolds and J. R. Beveridge. Searching for geometric structure in images of natural scenes. In *Proceedings of the DARPA Image Understanding Workshop*, pages 257–271, Los Angeles, CA, February 1987.

[29] J. R. Ullman. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1):31–42, 1976.

[30] S. Ullman. Visual routines. *Cognition*, 18:97–106, 1984.

[31] T. Williams. Image understanding tools. In *Proceedings of the International Conference on Pattern Recognition*, pages 606–610, Atlantic City, NJ, June 1990.

Acknowledgements

I would like to gratefully acknowledge the financial support of ARPA and Rome Labs, who supported this work under contracts F30602-91-C-0037 and F30602-94-C-0042. I would also like to thank Ed Riseman, Al Hanson, Paul Utgoff and J. MacGregor Smith for serving on my thesis committee.