Learning Grouping Strategies for 2D and 3D Object Recognition*

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

Abstract

The Schema Learning System (SLS) automatically assembles task-specific object recognition programs from existing IU algorithms. SLS brings together two emerging technologies – image understanding and machine learning – to automatically build customized procedures for recognizing and extracting specific object classes in constrained contexts. This paper describes the representations and algorithms underlying SLS, and presents an example of SLS learning to recognize rooftops in aerial images of Ft. Hood. This task is the first of several tasks from the ARPA/ORD sponsored RADIUS project [6] that SLS is intended to learn *without human interaction.* In later experiments, SLS will be tasked to automatically construct 3D models of buildings and other objects of interest from overlapping aerial images.

1 Introduction

Although the field of image understanding (IU) has made significant advances over the past twenty years, we have not yet developed a theoretical or practical understanding of how the many components of vision are combined into coherent, functioning systems. As a result, there are few applications of image understanding technology in the real world, even though the library of available IU techniques keeps growing. The problem is the labor and expertise required to select the right set of IU algorithms for a specific task, and to combine them into a single, smoothly-functioning system.

Much of what makes the integration problem difficult is that the most effective combinations of algorithms are often object, context or task dependent. Some objects, for example, have distinct colors that can be used to focus attention on particular parts of an image, while others have easily identifiable substructures, repetitive textures, or other properties that help us to recognize them and place them in space. Unfortunately, the specific features and techniques needed vary from object to object and context to context, so that most visual tasks require specialized solutions, even within constrained domains such as aerial image interpretation. This limits the general use of image understanding technology, because successful vision systems must be redesigned and/or hand-tuned for each new application.

At the same time, control engineers have long modeled the control of discrete events (such as algorithms) as Markov Decision Problems (MDPs). Although the traditional control-theoretic techniques for solving MDPs (i.e. Dynamic Programming) require a more detailed process model than is generally available for IU applications, we believe that recent advances in reinforcement learning [15, 17, 16, 18] and in function approximation (including, but not limited to, backpropagation neural networks [13, 11]) make it possible to learn near-optimal control policies for image understanding. In principle, one should be able to transform the task of constructing a new vision application to one of training the system with a set of representative input-output examples relevant for the task. Given a library of available IU algorithms and representations, the goal is to automatically select sequences of algorithms and intermediate representations to optimize specific applications while minimizing the involvement of the user.

1.1 The Need for Learning in Complex IU Applications

In many ways, the stage has been set for learning control strategies for image understanding by the research of the past twenty years. Computer vision researchers have been dividing naturally - without any global consensus or

*This work was supported by the Advanced Research Projects Agency (via USAF Rome Laboratory) under contract number F30602-94-C-0042.

mandate - into 10 or 20 subfields with small, well-defined problems. This has led to the development of specific mathematical theories and algorithmic techniques for each subdiscipline. There are now several good and improving algorithms for camera calibration, edge and line extraction (straight and curved), stereo analysis, tracking, depth from motion (two-frame and multi-frame), shape recovery, and 3D pose determination, to name just a few. Indeed, computer vision researchers have made more progress than most outside the field (and many inside) are aware of. This state of affairs is due primarily to our inability to easily produce highly visible results in the form of integrated task-specific systems.

The need for robust and flexible techniques that adapt to the user without requiring extensive explicit programming and customization is particularly apparent in image understanding problems that exploit context and/or models, both of which can be expected to change over time. Vision systems that *learn and adapt* are one of the most important directions in IU research right now. This reflects an overall trend – to make intelligent systems that do not need to be fully and painfully programmed. It is the only way for us to develop vision systems for the military that are robust and easy to use in many different tasks.

1.2 Learning Strategies for 2D and 3D Building Reconstruction

The ARPA/ORD RADIUS project is an interesting example of both the importance of IU technology and the problems with it. Current military doctrine is to achieve dominant battlefield awareness by digitizing the battlefield, which implies that the number of images collected and interpreted will have to increase by orders of magnitude. Unfortunately, the number of image analysts available to interpret this data is expected to remain the same or even decrease, meaninn that each analyst will have to become far more productive, presumably by automating or semi-automating portions of their task.

To this end, the RADIUS project has sought to develop IU tools to automate analysis tasks, such as (2D) building detection, (3D) building reconstruction, change detection and road detection. As part of this program, several universities have developed new and original algorithms for achieving all or parts of these tasks. Because of the practical nature of the RADIUS project's goals, however, these universities have had to craft not just isolated algorithms, but complete, functioning systems. Although many of the underlying algorithms are generally useful, changes in the problem statement – such as new image domains and/or new types of sensors – have often meant that the overall system had to be retuned, if not overhauled entirely, in order to work on the new task.

This is exactly the type of problem that SLS is meant to address, so we have adopted the RADIUS data and task statements as a test domain for SLS. This paper presents some early results of a major experiment in using SLS to accomplish RADIUS-project tasks such as building detection and reconstruction. The first of these tasks (described below) is to recognize the image positions of rooftops in aerial photographs. Several other universities have previously addressed this problem [7, 9] developing hand-crafted strategies for finding rooftops by grouping line segments, analyzing shadows, and exploiting other 2D image cues. Our goal for SLS is to automatically learn an equivalent (or better) strategy based on the same type of information.

Ideally, SLS's library should contain the same subroutines used in other rooftop recognition projects. Unfortunately, SLS's procedures must be executable UNIX modules, while many of the subroutines developed for RADIUS are Lisp functions embedded in RCDE [10]. Therefore, a small set of RADIUS-style vision routines (recoded as stand-alone C or C++ modules) have been used for this experiment, many although not all from UMass. In this paper, the challenge for SLS is to find a control policy for applying these algorithms that maximizes system performance on the (2D) roof recognition task.

The strength of SLS is not only that it produces effective control policies, but that it becomes possible to reconfigure the system for new tasks as they arise. Currently, SLS has access to only a small set of IU algorithms, most of which extract 2D information from a single image. In the next few months, however, we expect more algorithms to be added to this library, including 3D algorithms for computing digital elevation maps (DEMs) from stereo image pairs and algorithms for fitting planes and other surfaces to the DEM data. As each new algorithm is added, SLS can learn a new control policy that learns how best to take advantage of the new routine in conjunction with the 2D and 3D algorithms already in its library. SLS can also learn new control policies to adapt to changing goals, as we expand the system from finding flat-roofed buildings to constructing 3D building models of flat-roofed buildings and eventually to constructing models of buildings with multi-level roofs, peaked roofs, curved roofs, and/or large structures (such as air conditioning or water storage units) on the roof.

2 Markov Decision Problems

Control engineers have long modeled the control of discrete processes as a Markov Decision Problem (MDP). Although even a brief tutorial on MDPs is beyond the scope of this article, MDPs can be pictured in terms of systems (similar to finite state machines) having a discrete set of states and making discrete transitions between states as a result of actions. Initially, the system starts in state s_0; in response to an action (call it a_0), the system is advanced to a new state (call it s_1); the system is also given a reward (or penalty) for making the transition from state s_0 to state s_1. The next action a_1, applied at state s_1, advances the system to state s_2 at time t_2 (and gives it another reward or penalty), and so on until the system reaches a terminal state. The goal of a Markov Decision Problem is to select a sequence of states and actions $s_0, a_0, s_1, a_1, ..., s_n, a_n$ that maximizes the total reward of reaching a terminal state. (MDPs can also maximize the total reward as $t \rightarrow \infty$, but for the purposes of this paper we will limit the discussion to tasks with terminal states.)

Two features of the MDP formalism are particularly important. One is that the MDP formalism is *stochastic*; each action a_i has a probability function associated with it that gives the probability of transitioning to state s_k from state s_j (written as $P(s_j|a_i, s_k)$). The other is that the solution to a Markov Decision Problem is a *control policy*, often expressed as a table, that maps actions onto states so as to maximize the expected total reward. (This is necessary since the outcomes of actions are stochastic.)

Many readers may be familiar with Dynammic Programming, a set of techniques for computing optimal control policies for MDPs, given that the transition probabilities and rewards are known a-priori for each action/state pair. Unfortunately, such process models are often unavailable, and reinforcement learning is a branch of machine learning research that seeks to learn optimal control policies for MDPs without knowing the transition probabilities in advance, generally by developing empirical estimates of these probabilities on-line.

Another property of the MDP formalism are the $V^\pi(s)$ and $Q^\pi(s, a)$ functions. Intuitively, $V^\pi(s)$ is the expected reward from starting in state s and following control policy π until a terminal state is reached; the $V^\pi(s)$ function is also sometimes called the state value function. $Q^\pi(s, a)$ is the expected reward from starting in state s, applying action a, and then following control policy π thereafter until a terminal state is reached; the Q function is sometimes called the state/action value function. The basis of Dynammic Programming algorithms is that given a process model, they compute $V^*(s)$ and/or $Q^*(s, a)$ for every state or state/action pair, where $*$ is the optimal policy. The $V^*(s)$ and $Q^*(s, a)$ functions can then be used to generate the optimal policy table by selecting, for every state s, the action a with the highest $Q^*(s, a)$ value.

3 The Schema Learning System

The Schema Learning System (SLS) uses reinforcement learning and neural networks to automatically assemble computer vision algorithms into working special-purpose computer vision systems. It accomplishes this by casting image understanding as a Markov Decision Problem (MDP), in which the reward function is a measure of the accuracy of the final object hypothesis[1]. The control policies learned by SLS reason across multiple levels of representation, selecting the next action at each step as a function of the "knowledge state" of the system. The overall goal of SLS is to learn policies that produce accurate object hypotheses, thereby maximizing the total reward.

3.1 Actions

The levels of representation in SLS are a product of the visual procedure library. Each procedure is declared to have an input data type(s) and an output data type(s). For example, an edge extraction procedure is applied to images and produces edges, while an edge grouping operator is applied to edges and produces either straight lines or curves. The library therefore defines both the visual procedures and the levels of representation that SLS can reason across.

Not surprisingly, the visual procedures in the library are the actions of the MDP. The states of the system correspond to the data items (called *tokens*) produced by visual procedures. For example, in the rooftop recognition scenario, the initial state of the system corresponds to an image. If the action selected is an edge operator,

[1] In general, the reward functions may trade off accuracy for efficiency, but we will not consider that here.

then this will produce a set of edges which becomes the new state of the system. Thus the edge operator action transitions the system from the initial image state to the edge state.

3.2 State Spaces

Clearly, not all sets of edges are the same; neither are all images, polygons, etc. In order to learn sophisticated control policies, it is necessary to distinguish good (high quality) data from bad. Unfortunately, it is not obvious how to divide any given level of representation into a discrete set of states a-priori (although in the past we have learned policies by training decision trees to divide each representation's space of tokens into discrete states [5]).

As an alternative, we note that Markov Decision Problems can be defined over infinite state spaces. In this formulation, the control policy becomes a function that maps points in the (now infinite) state space onto actions, and the $V^{\pi}(s)$ and $Q^{\pi}(s, a)$ functions similarly range over infinite state spaces. In particular, in SLS we define a state space for every level of representation, so that an action maps points in one space to points in another space, depending on the level of representation of its input and output. To return to the edge operator as an example, it maps points in image space onto points in edge space.

Each level of representation therefore has a state space associated with it. This space is defined by the set of measurable features defined for that representation. For example, in the current experiment images have four measurable features: average intensity, standard deviation, edginess and speckle. The image state space is therefore defined as a four dimensional space with each feature as one dimension. Any particular image is represented as a point in this space. As another example, one of the representation used heavily in the rooftop experiments is the parallel line pair (i.e. two lines that are parallel to each other). The parallel line pair space is five dimensional, corresponding to the five features that we have defined for parallel line pairs: relative angle, extent of overlap, average contrast, shadowness (whether a shadow lies just to the outside of either line), and the variance of the intensity (pixel) values between the lines.

Mathematically, this is a clean formulation of the problem. In practice, it only works if we can learn estimate for the $Q^{*}(s, a)$ or $V^{*}(s)$ functions over these infinite state spaces from a finite number of samples. Inspired by Tesauro [16] and Zhang and Dietterich [18], we use backpropagation neural networks to learn approximations to the $Q^{*}(s, a)$ function for each action, as described below in Section 3.4. The control policies learned by SLS are therefore represented as a set of neural networks, each approximating the Q function for a state space / action pair. At run-time, the best action to apply to a state (i.e. data token) is the action with the highest estimated Q value given its feature vector.

3.3 Backtracking and the State/Action Queue

As a first pass, SLS can therefore be visualized as a system that begins with a data token s_0, typically an image. SLS evaluates the function $Q(s_0, a)$ for every action a that can be applied to token (and state) s_0, and selects the action with the highest estimated total reward. The action (a_0) is then applied to data s_0, creating a new data token s_1. SLS then selects the action with the highest estimate for $Q(s_1, a)$ and applies it, creating state s_2. This process repeats itself, creating a sequence of states and actions $s_0, a_0, s_1, a_1, \ldots$, until a data token at the target level of representation (for example, 2D roof hypothesis) is created. Such a token represents a terminal state for the system.

The problem with this simplified version of SLS is that many IU procedures – particularly matching procedures – do not produce a single data item as output. One of the visual procedures in the library for recognizing rooftops, for example, is a graph matching procedure that looks for a rectangle given a set of line segments. Depending on the number of rectangles in the image, the matching procedure may produce zero, one or many rectangle hypotheses. Consequently, when this action is applied to a data state s_i, it may produce many (or no) possible successor states s_j.

SLS therefore maintains a state/action queue. When an action produces multiple results $s_{j1}, \ldots, s_{jn}$, it evaluates the Q function for each new state/action pair, and creates a state/action queue sorted by the estimated Q values. It then applies the highest rated state/action pair, producing zero, one or more new data states, which are then added to the state/action queue.

The state/action queue[2] does more than just allow SLS to accomodate visual procedures that return an indeterminate number of arguments. It also allows SLS to backtrack during the interpretation process, if necessary. When SLS selects an action, it does so because the esti-

[2] We refrain from calling it the Q-Queue.

mated reward for that action/state pair is high (meaning that it expects it to lead to a good goal-level hypothesis). Sometimes, however, this estimate is erroneous, and the selected action creates low-quality data (i.e. data with low estimated $V(s)$ values). In this case, SLS will not pursue the bad hypothesis. Since new action/state pairs made from the bad hypothesis will have lower Q estimates than some of the unexecuted state/action pairs already on the queue, SLS will backtrack to try one of these previously unexecuted actions. In essence, unlike most MDP applications, SLS is maintaining a complete search tree, and using the Q and V functions as heuristics to select which nodes to expand, in a manner similar to A^* search.

3.4 Computing Q and V

There are several reinforcement learning algorithms for estimating Q or V without a-priori process models, with TD(λ) [15] and Q-Learning [17] being the best known. These algorithms build successively better approximations of Q and V based on experience gained by running the system; Tesauro [16] and Zhang and Dietterich [18] have used these techniques with neural network function approximators to learn Q values for systems with continuous state spaces.

One problem with these techniques is that they have to execute tens of thousands of actions to converge to good Q and V estimates – even more when continuous state spaces are used. In an application domain in which each action may take a minute or more to execute, we do not have enough time to directly apply these techniques.

Fortunately, very few sequences of visual procedures are very long; in general, it only takes between five and ten IU prodecures to get from an image to an object hypothesis. As a result, although the search trees associated with object recognition problems have high branching factors, they are typically not very deep. It is possible to do an exhaustive search of a limited number of training images and use this data to estimate Q and V. (In the rooftop recognition experiment, the exhaustive search took about 8 hours per training image; less when the known positions of rooftops in the training images were used to constrain the set of hypotheses).

In particular, for every data instance created from a training image during exhaustive search, we can compute the reward of the best goal-level token that can be created from it. These reward values are samples of a function $Q^{opt}(s,a)$, which represents the best reward that can be computed starting from a state/action pair. $Q^{opt}(s,a)$ should not be confused with $Q^*(s,a)$, the estimated reward for the optimal control policy. It may not be possible to select the best action at every state because of ambiguity in the feature vectors, so $Q^{opt}(s,a)$ is an *optimistic estimate* of $Q^*(s,a)$ (hence the term Q^{opt}), with the property that $Q^{opt}(s,a) \geq Q^*(s,a)\,\forall s,a$. In fact, as the data features become more discriminating, $Q^{opt}(s,a)$ approaches $Q^*(s,a)$ from above. Most importantly, $Q^{opt}(s,a)$ can be computed from far fewer training samples than $Q^*(s,a)$, because each $Q^{opt}(s,a)$ is independent of every other, so the neural nets can be trained separately.

3.5 Properties of SLS

Readers who are familiar with MDPs may want a clarification of certain technical points:

- **Markov State Properties.** The theory behind MDPs, Dynammic Programming and Reinforcement Learning relies on the so-called "Markov Assumption" that the transition probabilities for a state/action pair depend only on the state (and the action), and not on any previous states. Formally, it assumes that:

  $$\forall i,j,k \;\; P(s_k|s_i,a_j) = P(s_k|s_1,...,s_i,a_j)$$

  Clearly, this is only approximately true for the states in SLS, as represented by the feature vectors associated with data instances. (Strictly speaking, SLS's states should be called *situations*.) Consequently, the formal proofs that TD(λ) and Q-Learning converge to the optimal control policy do not apply.

- **Optimality of Q^{opt}.** Even if SLS had true states (instead of situations), a greedy policy with respect to Q^{opt} would not be optimal. It is easy to construct artificial examples in which $Q^{opt}(s,a) \gg Q^*(s,a)$ for some state/action pairs and not for others, leading to inefficient behavior. However, because Q^{opt} is an optimistic estimate and SLS uses a state/action queue, applying $Q^{opt}(s,a)$ will always produce the optimal answer, even it sometimes executes unnecessary actions along the way (assuming true states and perfect function approximators, and therefore an accurate estimate of $Q^{opt}(s,a)$).

Representation	Features
Image	Avg. Intensity, SD Edginess, Speckle
LinePairSet	Count, Avg Contrast
Parallel Line Pair	angle, overlap, shadowness surface fit, distance
Corner (L-Junction)	angle, gap, shadowness surface fit, scale
Line Group	Count, Parallel Count Corner Count
Polygon	Edge Cover % Worst Edge Cover Avg Perimeter Contrast Worst Side Contrast Plane fit error (intensity data) scale, shadowness

Table 1: Features defined for each level of representation in the visual procedure library for recognizing rooftops.

4 Experiment – Detecting Rooftops

As has already been discussed extensively, we assigned SLS the task of finding rooftops in aerial images of Ft. Hood, a task that was copied from the RADIUS project task domain. On each trial, the system is given a subimage containing one or sometimes two buildings, and a set of 3D line segments computed as described in [4]. SLS is also given a visual procedure library that defines eight levels of representation and nine visual procedures. The levels of representation correspond to images, sets of 3D line segments, parallel line pairs, corners (i.e. vertices of orthogonal lines), line groups and polygons. Because much of the power of SLS lies in its ability to distinguish good data from bad based on feature measurements, Table 1 gives the set of features defined for each level of representation.

4.1 The Visual Procedure Library

The visual procedures employed are meant to approximate some of the techniques being used by researchers in the RADIUS project. The 3D line segments were computed off-line as described in [4], and filtered according to the known height of the ground plane. Eight other visual procedures are available. The rectilinear line grouping (RLGS) procedure is an updated version of [12] that computes relationships between nearby line segments; it was updated to use information about the camera pose (available for all RADIUS images) to remove the effects of perspective distortion from orthogonal and parallel relations. The SelectParallel and SelectCorner procedures select parallel line pairs and corners out of the relations computed by RLGS.

The grouping procedures (GroupParallel and GroupLJnct) take a pair of parallel lines (or a corner) and form a group out of all the lines that share a significant relation to one of the lines in the original pair. This results in a small group of line segments in which the Graph Matching procedure can search for a rectangle of lines. Alternatively, given a pair of parallel or orthogonal line segments, the Par2Polygon and Corner2Polygon algorithms go back to the original image data and apply edge extraction and edge linking operators top down, in order to look for evidence of additional lines that might complete the rectangle. Finally, the Polygon2Roof procedure serves no purpose other than to give SLS a way to designate a particular polygon as a roof.

At first glance, the visual procedure library would appear to have only four paths to the goal, which would make SLS's task fairly easy. The procedure for selecting corners, however, typically finds on the order of fifty to one hundred corners per image, while the procedure for finding parallel line pairs typically finds twice that many. As a result, there are approximately five hundred polygons that SLS might detect in most images, and most of the work that SLS does is in selecting which hypotheses – in terms of which corners, parallel pairs, and polygons – to pursue.

4.2 Detecting Rooftops

SLS was tested on a set of ten image fragments taken from the RADIUS project's images of Ft. Hood. The training and evaluation was carried out using the ground truth data developed for the (self) evaluation of UMass's hand-crafted building detector and 3D reconstruction system [4].

SLS was tested using a "leave one out" methodology. On each trial, SLS was be trained on data from nine images, and then the control policy it learned would be applied to the tenth image. This was repeated ten times, each time holding a different image out of the training set for testing. Figure 1 shows one of the rooftops extracted by SLS. Over ten trials, SLS found a polygon that corresponded to a true roof surface nine times; in the tenth trial, SLS confused part of the roof boundary with shadow line that was near to (and parallel with) the true edge of the roof, creating an incorrect roof hypothesis.

Figure 1: One of the ten aerial images of buildings at Ft. Hood, and the roof hypothesis (shown in white) SLS learned to find in it.

The contol policies learned by SLS did not always take a straight path to the solution. Although they always prefered finding corners to parallel lines, they would often select one corner as being the most promising, use it to generate a polygon hypothesis, and then decide that the polygon was not as good as it thought it would be. (In other words, the V value of the hypothesis was lower than the Q value of the action that created it.) The system would then backtrack, find the next most promising corner, and try again. In general, the system created ten to twenty polygon hypotheses (out of several hundred possible ones) before finding the polygon it finally selects to be the rooftop.

Significantly, SLS can adapt quickly to the introduction of new procedures or features. The first time we tested the system on the Ft. Hood data, it succeeded in only about half the trials. Looking at its mistakes, we realized it was often confusing shadows with the edges of the buildings that cast them. We then added a shadowness feature to the parallel pair, corner and polygon representations, and immediately SLS's performance improved. In general, we suspect that this is how SLS will be used – as a system to which people add knowledge until it is able to accomplish the assigned task. Ironically, it could therefore be viewed as a very good knowledge engineering tool.

5 Future Experimental Plans: 3D Building Reconstruction

The goals of the RADIUS project go beyond simply recognizing objects in aerial images and determining their image position. One of the goals of RADIUS is to provide the image analyst with tools that automatically construct 3D models of buildings from overlapping aerial views. Although more thorough testing of SLS on the 2D recognition task is also planned, the primary emphasis in the future will be on getting SLS to learn control policies for 3D building reconstruction.

Although there are clues to 3D structure in monocular images that SLS is not currently taking advantage of (such as the sun angle and length of shadows), we believe that what will make 3D building reconstruction far more effective is the depth information provided by overlapping aerial views. The UMass terrain reconstruction system [14] constructs dense digital elevation maps (DEMs) accurate to within a meter from pairs of images, even when those images were taken with wide baselines at highly oblique angles. This type of dense, 3D data, in combination with the 3D lines computed in [4], should make it possible to reconstruct highly accurate building models. These procedures, along with additional procedures for fitting planes and complex surfaces to DEM data, should give SLS many alternative strategies for constructing 3D building models.

SLS's task will be to combine the new 3D procedures with the previous 2D set to form accurate and efficient control policies. Although the 3D reconstruction results are not available at the time of printing, we invite readers to visit our URL site to see how the experiments are going:

```
http:\\vis-www.cs.umass.edu
\projects\SLS3D.html
```

6 Conclusions

Over the last twenty years, image understanding researchers have developed many effective algorithms for solving visual subproblems, such as edge and line extraction, stereo analysis, tracking and pose determination.

Unfortunately, we have not developed a comprehensive theory of how these algorithms might be put together into functioning systems, with the result that many advances in IU have yet to see their way into military or civilian applications.

The Schema Learning project is an attempt to understand both the science and the practice of combining IU algorithms into special-purpose vision system, by casting the control of image understanding as a Markov Decision Problem. At an abstract level, this requires defining state spaces for IU and algorithms for calculating the Q and V functions; more practically, it requires building a system capable of integrating many different IU algorithms.

The Schema Learning System (SLS) is a first pass at such an algorithm and system. This paper presents the results of an early experiment in which SLS was able to learn control policies that successfully found rooftops in aerial images in nine out of ten trials. The near-term future goal, however, is to learn control policies for automatically constructing 3D building models.

Acknowledgements

Many people have contributed their time, effort and code to the Schema Learning System. Madi Das created the neural net C++ object, based on code published in [11]. Bob Collins created the algorithm for computing 3D line segments, as well as constructing the ground truth models. Lionel Gaucher updated the 2D line extraction code of [2], thereby making the Par2Polygon and Corner2Polygon procedures possible. Shashi Buluswar contributed the shadow feature measurement program. Gokhan Kutlu and Robert Heller programmed large parts of ISR3, including SLS's graphics interface. Frank Stolle calculated the ground heights for each image, and Chris Jaynes extracted the camera positions from RCDE.

References

[1] A. Barto, S. Bradtke and S. Singh. "Learning to Act using Real-Time Dynamic Programming," *Artificial Intelligence*, 1995.

[2] M. Boldt, R. Weiss and E. Riseman. "Token-Based Extraction of Straight Lines," in *IEEE Trans. on Systems, Man and Cybernetics,* 19(6):1581–1594, 1989.

[3] R. Collins, A. Hanson, E. Riseman, C. Jaynes, F. Stolle, X. Wang and Y. Cheng. "UMass Progress in 3D Building Model Acquisition," *ARPA Image Understanding Workshop,* 1996.

[4] B. Draper. "Learning Control Strategies for Object Recognition," in *Symbolic Visual Learning,* K. Ikeuchi and M. Veloso (eds.), Oxford University Press, New York, to appear 1996.

[5] D.Gerson and S.Wood, "RADIUS Phase II – The RADIUS Testbed System," *Arpa Image Understanding Workshop,* Monterey, CA, November 1994, pp. 231–237.

[6] A. Huertas, C. Lin and R. Nevatia. "Detection of Buildings from Monocular Views Using Perceptual Grouping and Shadows," *ARPA Image Understanding Workshop,* Washington, DC. 1993.

[7] R. Mohan and R. Nevatia. "Using Perceptual Organization to Extract 3D Structures," *IEEE Trans. on Pattern Analysis and Machine Intelligence,* 1989.

[8] J.Mundy, R.Welty, L.Quam, T.Strat, W.Bremner, M.Horwedel, D.Hackett and A.Hughes, "The RADIUS Common Development Environment," *Arpa IUW,* San Diego, CA, Jan 1992, pp. 215–226.

[9] Yoh-Han Pao. *Adaptive Pattern Recognition and Neural Networks.* Addison-Wesley, 1989.

[10] G. Reynolds and R. Beveridge, "Searching for Geometric Structure in Images of Natural Scenes", *ARPA Image Understanding Workshop,* Los Angeles, CA, Feb. 1987, pp. 257-271.

[11] D. Rumelhart, G. Hinton and R. Williams. "Learning Internal Representations by Error Propagation," in Parallel Distributed Processing, Vol 1, Rumelhart and McClellan (eds), MIT Press, Cambridge, MA.

[12] H. Schultz. "Terrain Reconstruction from Widely Separated Images," *SPIE: Integrated Photogrammetric Techniques with Scene Analysis and Machine Vision II,* Orlando, FL, April 1995, pp. 113-123.

[13] "Learning to Predict by the Method of Temporal Differences", *Machine Learning* 3:9-44.

[14] G. Tesauro. "Temporal Difference Learning and TD-Gammon" *Communications of the ACM*, 38(3):58-68

[15] C. Watkins. *Learning from Delayed Rewards,* Ph.D. thesis, Cambridge University, 1989.

[16] W. Zhang and T. Dietterich. "A Reinforcement Learning Approach to Job-Shop Scheduling," *Int. Joint Conference on Artificial Intelligence,* 1995.