TEC-0125

Learning to Populate Geospatial Databases via Markov Processes

Bruce A. Draper
J. Ross Beveridge

Colorado State University
601 South Howes Street
Fort Collins, CO 80523

December 1999

Prepared for:
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington, VA 22203-1714

Monitored by:
U.S. Army Corps of Engineers
Topographic Engineering Center
7701 Telegraph Road
Alexandria, Virginia 22315-3864

US Army Corps of Engineers
Topographic Engineering Center

T
E
C

20000310 089

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1999	3. REPORT TYPE AND DATES COVERED Final Technical April 1997 - July 1998

4. TITLE AND SUBTITLE Learning to Populate Geospatial Databases via Markov Processes	5. FUNDING NUMBERS DACA76-97-K-0006
6. AUTHOR(S) Bruce A. Draper J. Ross Beveridge	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Colorado State University Department of Computer Science 601 South Howes Street Fort Collins, CO 80523	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency 3701 North Fairfax Drive, Arlington, VA 22203-1714 U.S. Army Engineer Research and Development Center Topographic Engineering Center 7701 Telegraph Road, Alexandria, VA 22315-3864	10. SPONSORING / MONITORING AGENCY REPORT NUMBER TEC-0125

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report describes a 2-year project on learning to recognize an object using Markov decision processes. The underlying premise is that although the field of computer vision has made a great deal of progress during the last 20 years producing algorithms for specific subtasks (e.g., edge detection, model matching, stereo), it has produced very few end-to-end applications. This project investigated whether Markov decision processes and reinforcement learning might be used to automatically sequence and control vision algorithms to achieve specific tasks. In particular, we focus on learning object-specific recognition strategies using reinforcement learning, where the vision algorithms to be controlled are a set of 11 commonly available 2-D vision algorithms. Although the work reported here is only a first attempt at a complex problem we were able to automatically learn to recognize two different classes of objects (buildings and maintenance rails) in aerial images of Fort Hood. We also were able to learn to distinguish one style of house from four other styles of houses in the residential section of Fort Hood. With a slightly higher error rate, we were able to distinguish all five types of houses from each other, demonstrating an ability to learn to identify similar yet different classes of objects. Finally, we were able to show that for these tasks, the dynamic control policies learned by reinforcement learning were better than any possible fixed sequence of algorithms.

14. SUBJECT TERMS Object recognition, reinforcement learning	15. NUMBER OF PAGES 23
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UNLIMITED

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

PREFACE

This report was sponsored by the Defense Advanced Research Projects Agency (DARPA) and monitored by the U.S. Army Topographic Engineering Center (TEC), Alexandria, Virginia 22315-3864 under contract DACA76-97-K-0006, titled, Learning to Populate Geospatial Databases via Markov Processes. The DARPA Program Manager was Mr. George Lukes, and the TEC Contracting Officer's Representative was Ms. Lauretta Williams.

LEARNING TO POPULATE GEOSPATIAL DATABASES VIA MARKOV PROCESSES

1. Introduction

The goal of this contract was to develop underlying technology to help close the gap between the military's needs for comprehensive battlefield awareness and current image understanding (IU) capabilities. In particular, the primary goal was to develop machine learning technology to recognize **semantically meaningful features** such as roads, waterways, and military targets in aerial images, so that these features could be added to geospatial databases. In particular, we sought to eliminate three limitations of current IU technology: 1) **Number of Targets.** Most IU systems recognize a small number of object classes within a limited domain; 2) **Sensor Limitations.** Most IU systems interpret a single, fixed type of imagery (usually EO, but sometimes IR, SAR, or IFSAR); a few combine data from two sensors, for example EO and IFSAR data; and 3) **Automatic Systems.** Most current IU systems require some degree of operator assistance, whether it is to parameterize the system based on image and/or domain characteristics or to restrict its application to a region of an image.

To accomplish these goals, we suggest that IU should be approached as a **Markov Decision Process** (MDP). We believe that the technology base of IU procedures forms a library of discrete actions for image interpretation, and that intermediate data instances (2-D and 3-D images, points, lines, surfaces, etc.) form an infinite but structured search space of possible states. The process of object recognition can be modeled as a sequence of IU procedures applied to a series of intermediate states.

Unfortunately, this contract was terminated halfway through the scheduled life of the project. To explain what was accomplished and the state of the project at termination, this report is divided into three sections. Section 2 reviews the initial proposal; Section 3 outlines the modifications to the original plans made at the behest of Dr. Tom Strat, who was the DARPA program manager at that time; and Section 4 describes the accomplishments of the project at the point of termination. The first two major research publications about this work – *Bagging in Computer Vision,* which appeared in the IEEE Conference on Computer Vision and Pattern Recognition in June 1998, and *ADORE: Adaptive Object Recognition,* published at the International Conference on Vision Systems in January 1999– are included as appendices.

2. The Original Proposal

The original proposal was predicated on the belief that a Markov decision process formalism is a constructive framework for image understanding because it distinguishes between IU procedures and the control strategies used to integrate them, and because it provides a mathematical model of control in terms of policies. Formally, a **control policy** is a function that maps states (in this case, instances of intermediate data) onto actions (in this case, IU procedures); at each step of processing, the control policy selects the next action based on the properties of the data produced by the previous processing step.

In image understanding, control policies can be used to implement object-specific and task-specific recognition strategies. For example, the strategy for recognizing traditional, rectilinear buildings may be completely different from the one for recognizing Quonset huts. Modeling IU as an MDP allows the introduction of **reinforcement learning** (RL) techniques for training object-specific and task-specific control policies from examples. Reinforcement learning not only makes it possible to acquire large numbers of object recognition policies with less effort (and no reprogramming), it also produces well-motivated policies that maximize a utility function based on cost and accuracy. RL control policies are robust, in the sense that if a sensor is unavailable or an IU procedure fails, the control policy will react and select an alternative action.

We proposed to build a prototype system to learn control policies for 3-D **object recognition** using reinforcement learning, and to evaluate the system on the Ft. Hood and Kirkland/Sandia data sets. That system, that we now call ADORE[1], was to draw upon IU procedures from the **IUE**, **KBVision** and **Khoros** image libraries, as well as IU procedures developed locally at Colorado State University and the University of Massachusetts. Whereas we had already demonstrated some initial, limited success in learning control policies to identify objects in 2-D images prior to this proposal, our new work was to emphasize learning to extract **3-D representations** of objects from various types of sensor data (initially IFSAR and pairs of overlapping EO images).

Two methods of training were proposed for 3-D-AMORE. The first, lower-risk method uses 3-D CAD models of example object instances as the basis for the reward signal. This method applies when 3-D models are available, for example, from a partial site model or BRL/CAD models of military targets. The second, higher-risk method would exploit the redundant information in **overlapping 3-D images** without relying on pre-existing models. In this method, policies are trained by noting the position of objects in overlapping 3-D images, and extracting 3-D representations of the objects (in terms of grouped 3-D primitives) from one of those images. The reward signal for this method is based on how well 3-D representations extracted from one image predict the raw sensory data in another.

Finally, we proposed to study whether it is possible to **continually adapt** control policies over time without human intervention. In principle, it is possible to initially train control policies using a library of IU procedures, and then to add new procedures or sensors to the system afterward. By feeding a control policy's results back to itself as a training signal, the reinforcement learning algorithm should adapt the policy to take advantage of the new procedures/data.

This proposal had its intellectual roots in a long tradition of research into object-specific and task-specific control of computer vision [Hanson & Riseman 78; Draper, et al., 89], and further develops work begun at the University of Massachusetts on learning control strategies for object recognition [Draper 96a, 96b]. At the same time, this proposal extends these ideas in several new and exciting ways. First, the previous effort focused primarily on classifying objects (such as rooftops) in 2-D images. This proposal focused on extracting **3-D representations** of object

[1] ADORE: Adaptive Object REcognition

instances as well as identifying them, and will develop methods for learning to recognize objects in 3-D **without a-priori models,** as mentioned above. In addition, we propose to study how to continuously adapt control policies **during normal operation** (as opposed to during initial training). The idea of an IU system that continually improves itself during operation without human intervention was perhaps the most exciting one of this proposal. It implies, for example, that **new sensors** or **IU procedures** could be added to an operating IU system without reprogramming the system or even taking it off-line.

We believe that the impact of this work – had it been completed to fruition – would go far beyond the immediate deliverables. During the last 20 years, the field of image understanding has divided into 10-20 (or more) subfields, each with a narrowly-defined problem focus. Within each subfield, theories have been developed and tested and different solution methodologies have been adopted. As a result, there are now several good and improving algorithms for edge and line extraction (straight and curved), feature tracking, depth from motion (two-frame and multi-frame), camera calibration and 3-D pose determination, to name just a few of the areas in which progress has been made. Progress in 3-D vision has been particularly strong; the advent of 3-D IFSAR sensors and improvements in stereo processing now provide basic procedures for extracting and reasoning about 3-D information. One of the areas in which relatively little progress has been made, however, is the so-called "high-level" vision. We believe that this lack of progress results from the **lack of a theory of vision.** Without a common framework for discussing high-level image interpretation or a mathematical basis for comparing and analyzing high-level systems, progress in this area has stalled. We believe that the Markov Decision Process model provides the type of framework that is needed to enable progress not only in the automatic population of geospatial databases, but of many complex image understanding tasks.

3. Modifications to the Original Proposal

When we were notified that this proposal would be funded in January 1997, we were asked to make certain changes by Dr. Tom Strat, who was then the DARPA program manager. The most dramatic change was that the subcontract to the University of Massachusetts was dropped, and the money was instead used to add Dr. Ross Beveridge (of CSU) and one of his students to the project. This had a dramatic effect on the project, since UMass was to provide the optical stereo and IR vision procedures to the project. Dr. Beveridge, on the other hand, is an expert in target recognition and 2-D model matching, and brought procedures for these activities to the project.

The contractual changes dropping the University of Massachusetts and adding Dr. Beveridge's team to the contract were formally negotiated with DARPA. Unfortunately, the corresponding changes to the deliverables stemming from these changes were never put in writing. Instead, they were negotiated between Dr. Strat, a TEC representative, Dr. Beveridge, and Dr. Draper at the APGD kick-off meeting in California. At this meeting, it was agreed that the 3-D reconstruction and IR aspects of the project would be dropped, and the project would focus on learning 2-D recognition strategies, including strategies with embedded ATR routines. The reasoning was that the underlying technology with regard to learning, and the control of object recognition is the same whether the task is 2-D or 3-D, and that by concentrating the project's resources on 2-D recognition, progress in learning and control would be maximized. (Note that

2-D vision procedures are far more widely available through Khoros and the IUE than are 3-D procedures.)

At this meeting, the following list of milestones for the project was agreed to:

1. Six Months:
   a) Software infrastructure for control of vision procedures developed
   b) Experiment showing that open-loop strategies outperform closed-loop strategies
2. One Year:
   a) Recognition of two objects using automatically trained control policies
   b) Evaluation of those control policies.

In December 1997 we presented the first two milestones at the APGD workshop in Ft. Benning, GA. At this workshop, we reported that the first version of the software infrastructure for the project was complete. We reported the results of a study in which we compared the optimal closed-loop policy for a simple recognition task to the optimal open-loop policy for the same task. Unfortunately, this result was widely misunderstood by the audience as a report on 2-D roof recognition.

In fact, the intent of this study was to measure the relative improvement of closed-loop control vs. open-loop control for a simple 2-D recognition task. The motivation for this study (and the reason it was included in the scheduled milestones) is that the complexity of the Markov model is justified by the fact that it produces *open-loop* strategies. An open-loop strategy is defined as a strategy in which the sequence of vision procedures is not fixed before recognition begins; instead, new procedures are selected dynamically based on the results of previous actions. (Strategies in which a fixed sequence of procedures are selected a priori are called *closed-loop* strategies, and most other systems for learning visual control policies learn closed-loop strategies, e.g. [Au & Wang, Peng & Bhanu]).

Reinforcement learning models vision as a Markov decision process, and produces open-loop strategies. One of our first tasks was to justify this expense by showing that open-loop strategies outperform closed-loop strategies on even simple vision tasks. Logically, the more complex the task, the greater the benefit of closed-loop control over open-loop control should be ; therefore, we selected as our performance task the goal of finding rooftops in aerial images of a residential section of Ft. Hood, and collected a small library of simple (mostly pixel-based) vision procedures for accomplishing this task. We then exhaustively applied all combinations of our vision procedures to a set of training images, and by reasoning backwards measured 1) the average reward (measured as accuracy) of the best possible closed-loop strategy, and 2) the average reward of the best open-loop strategy. We also considered a third type of hybrid strategy (similar to the type being learned by [Maloof, et al]) in which a fixed sequence of procedures is selected a priori, but a dynamically trained classifier is inserted between each processing step.

The results of this study are shown in Figure 1. The closed-loop strategy performed an order of magnitude better than the open-loop strategy, as expected. The hybrid strategy (shown in the middle of Figure 1) performed well – within 10 percent of the closed-loop strategy – but was still non-optimal. We interpreted these results as justification for the Markov-based approach.

Closed vs Open (cont.)

Upper Bound: Comparing Optimal Open-Loop to Optimal Generate-and-Test & Optimal Closed-Loop

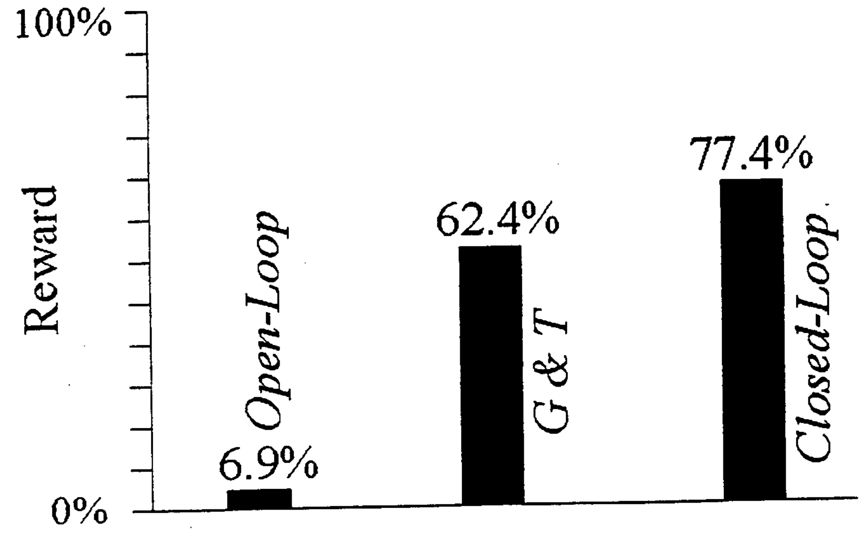


Figure 1: Results of Closed-loop vs. Open-loop Milestone

Unfortunately, many in the audience misinterpreted this study as a new method for recognizing rooftops. In fact, our intent was expressly *not* to introduce any new recognition techniques, but to use off-the-shelf vision procedures; moreover, we were limited to a small procedure library since our method was to exhaustively apply the procedures to the training images to guarantee (independent of inference method) that the optimal control strategies were found for each category of control policy. Perhaps this misunderstanding emphasizes the notion that this approach to object recognition is truly novel, and unfamiliar to this community. (Perhaps, too, we could have presented it more clearly.)

Shortly after the Ft. Benning meeting, there was a site visit from the new DARPA program manager, George Lukes. At this site visit, we demonstrated the software infrastructure, and an example of the system learning object recognition strategies for two objects: rooftops and maintenance rails (of the type used to maintain vehicles in the Ft. Hood motor pool). Although neither object was an overwhelmingly difficult target (the training rails, in particular, are quite distinctive), we thereby demonstrated the capability to 1) recognize two object classes (ahead of schedule), and 2) to learn to recognize two disparate object classes using a single system. To our knowledge, this later capability is unique.

At the site visit, Mr. Lukes told us that he was unaware that the project had been refocused by Dr. Strat. Mr. Lukes subsequently decided to terminate the contract early because we were not doing 3-D reconstruction. While it is true that we did not do 3-D reconstruction as called for in the original proposal with UMass, we note that 1) we had achieved all scientific and technical milestones agreed to with Dr. Strat at the kick-off meeting; 2) we were making strong progress toward our (mutually refocused) objectives; and 3) the technology we were developing is critical to both 2-D and 3-D object recognition.

While in Ft. Collins for the site visit, Mr. Lukes suggested that if we were going to do 2-D object recognition, and if we were going to recognize buildings as one of our target objects, then it would be more appropriate to find the footprint of the partially occluded buildings than to find the visible portion of the rooftop. Since finding the footprint of an occluded building requires recognizing the architectural plan of the building and distinguishing it from other, similar architectural plans, this gave us a set of similar object classes to distinguish from each other – the task on which we demonstrated our year-end progress.

4. Accomplishment: The ADORE system

The goal of the adaptive object recognition (ADORE) project is to avoid knowledge engineering by casting object recognition as a supervised learning task. Users train ADORE by providing training images and training signals, where the training signal gives the desired output for the training images. ADORE learns control strategies that dynamically select vision procedures in order to recreate the training signal as closely as possible from the training images. This control strategy can then be used to hypothesize new object instances in novel images.

To learn control strategies, ADORE models object recognition as a discrete control process. The state of the system is determined by data produced by vision procedures. For example, the state of the system might be a region of interest (ROI), a set of 2-D line segments, or a 2-D contour. The actions are vision procedures that change the state of the system by producing new data from current data. A control policy is a function that maps states onto actions. In the context of ADORE, control policies map data onto vision procedures, thereby selecting the next action at each step of the recognition process.

At a systems level, ADORE is most easily thought of as two distinct components: a run-time execution monitor that applies vision procedures to data, and an off-line learning system that trains control policies.

4.1) The Execution Monitor

The execution monitor is a run-time loop that begins when an image is presented to the system. On each cycle, it evaluates the control policy on the current data, thereby selecting a vision procedure. It then applies the vision procedure to the current data, producing new data. This loop continues until a vision procedure returns without producing any new data.

Of course, this simple description glosses over some important details. After every vision procedure, the execution monitor measures features of the output data to be used in training control policies. The execution monitor also measures the run-time of each procedure, to be used as a component in the reward function if the goal is to train control policies that make tradeoffs between time and accuracy. Finally, there are two special procedures – called accept and reject – that return no data. The accept procedure signals that an object instance has been found; during training, accepted data tokens are compared to the training signal and a reward or penalty is generated. The reject procedure signals that the current data should be rejected; during training, it always returns a reward of zero.

In the interests of generality, the execution monitor is independent of the vision procedure library. Each vision procedure is an independent unix executable; a library file tells the execution monitor the number and type of input arguments for each procedure, the number and type of output arguments, and the unix pathname. As a result, vision procedures can be added or removed from the system simply by editing the library file. Similarly, the execution monitor is independent of particular data representations, since all data tokens are kept in files. For each data type, the library file tells the execution monitor 1) the name of the data type (so the monitor can match data tokens with arguments to vision procedures); 2) the number of features measured (for function approximation – see below); and 3) the path of the unix executable for measuring features; thus new data types, like new vision procedures, can easily be added to the system. (Note that one of the original goals of the original proposal was to keep the systems and the vision procedure library separate, so that new vision procedures could be easily introduced into a working system.)

4.2) Control Policies

The execution monitor is a control loop for applying vision procedures to data. The control policy directs the execution monitor by selecting a vision procedure at each step. Since the choice of vision procedure depends in part on the target object class and image domain, a different control policy is learned for every object recognition task.

Control policies are functions that map data tokens (represented in terms of their features) onto vision procedures. In order to learn sound control policies, object recognition is modeled as a discrete control problem. In particular, the state of the system is always defined by the features of the current data. The vision procedures are the discrete actions, and these actions are non-deterministic in the sense that we cannot predict the features of the output based solely on the features of its input.

To train a control policy, we train a Q-function for every vision procedure. In control theory, a Q-function is a function that predicts the expected reward that will follow from applying a specific action to the current state. For example, in ADORE we train a Q-function for predicting the reward that will result from applying the segmentation procedure to an ROI, based on the features of the ROI. We also train Q-functions for predicting the rewards that will follow from applying the active contour procedure to an ROI and the correlation procedure from an image. Control policies are implemented by evaluating the Q-function of every vision procedure that can be applied to the current data (based on its datatype) and selecting the procedure with the highest Q-value.

It is important to note that the Q-function predicts the total reward that follows a procedure, not just the immediate reward. For example, in the experiments described below, the immediate reward for resegmenting an ROI is zero, since the segmentation procedure returns a 2-D contour while the training signal is given in terms of ROIs. Nonetheless, there can be a delayed reward for resegmenting an ROI, since the resulting contour can be transformed into another ROI through the Generalized Hough Transform procedure. The Q-function for the segmentation procedure predicts this delayed reward, not just the immediate reward from the segmentation procedure.

4.3) Off-line Learning

The control and artificial intelligence literatures contain many techniques for learning optimal Q-functions for control problems with discrete state spaces. If the transition probabilities associated with the actions are known (a so-called process model), dynamic programming can be used to estimate Q-values and produce an optimal control strategy. In the absence of a process model, reinforcement learning (most notably the temporal difference [Sutton, 1988] and Q-learning [Watkins, 1989] algorithms) have been shown to converge to optimal policies in a finite number of steps.

Unfortunately, the object recognition control problem, as defined here, depends on a continuous, rather than discrete, state space. Tesauro [Tesauro, 1995] and Zhang & Dietterich [Zhang, 1995] have shown empirically that neural nets can approximate Q-functions for continuous feature spaces within a reinforcement learning system and still produce good control policies. Unfortunately, their method required hundreds of thousands of training cycles to converge. ADORE has a sequence of continuous feature spaces, one for each data representation (images, ROIs, contours, etc.). This requires getting a sequence of neural nets to converge on a control policy. Although theoretically possible, we did not succeed in making this work.

Instead, we train Q-functions by optimistically assuming that the best control policy will always select the action that creates the highest possible reward for every data token. Strictly speaking, this assumption is not always true: a control policy maps points in feature space onto actions, and it is possible for two different tokens to have the same feature measurements and yet have different "optimal" actions. Nonetheless, the optimistic assumption is usually true, and it breaks the dependence between Q-functions, allowing each neural net to be trained separately.

In particular, we approximate Q-functions by training backpropagation neural networks. The training samples are data tokens extracted from the training images. For each training sample, we apply all possible sequences of procedures, in order to determine which procedure yields the maximum reward. A neural network is trained for each vision procedure using the data features as input and the maximum reward from the data created by the vision procedure as the output. In this way, the neural net learns to approximate the future reward from an action under the optimistic control assumption.

4.4) Bagging

One aspect of this formulation of the object recognition problem is that it requires highly accurate function approximation if the control policy is to be optimal. In order to maximize the accuracy of our neural nets, we turned to a new technique from the machine learning community called *bootstrap aggregation*, or *bagging*. The approach was originally developed by Breiman [94], and is similar to work by Dietterich on improving non-parametric classification [Dietterich & Bakiri 95, Kong & Dietterich 95 & 97] (see also [Heath, et al 93, Kwok & Carter 90]). The basic idea is to randomly subsample the training set and train multiple function approximators from these subsampled sets. The aggregated or "bagged" predictor is then the average of the predictions for a sample.

Statistically, bagging is a theoretically sound technique that reduces prediction error by reducing the *variance* component of the error. (The *bias* error is unchanged.) This technique has not been widely used in computer vision. In [Draper & Baek 98] we were able to show that bagging produced an even larger increase in accuracy for our computer vision control problem than had been reported by Breiman for other tasks. More importantly, as part of this project, we were able to show both theoretically and in practice that if you weight each component network by the inverse of its mean squared error (MSE) on a validation training set, then the variance error in the bagged predictor is reduced ever further. (This original contribution has been submitted to *IEEE Transactions on Pattern Analysis and Machine Intelligence*.)

5. Experiments

To test ADORE in a tightly controlled domain, we trained it to recognize styles of houses in aerial images such as the one in Figure 2. In the first experiment, the goal was to find duplexes of the type shown in Figure 3. The training signal is a bitmap that shows the position and orientation of the duplexes in the training images; Figure 4 shows the training signal matching the image shown in Figure 2. The reward function used to evaluate hypotheses during training is the size of the pixel-wise intersection of the hypothesis and the training signal, divided by the size of the union. This evaluation function ranges from one (perfect overlap) to zero (no overlap).

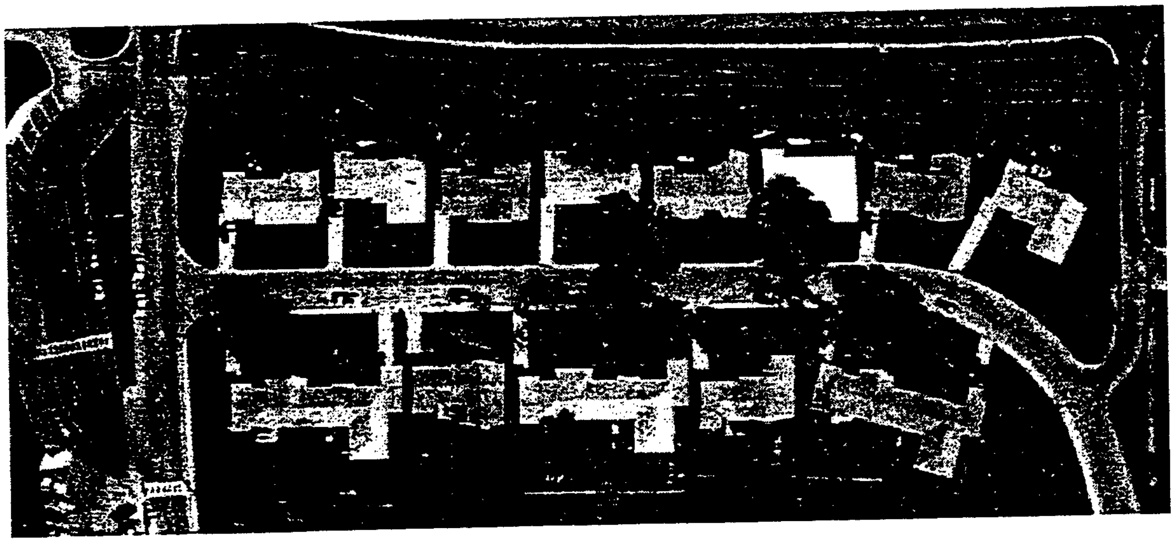


Figure 2: A nadir-view aerial image of the residential section of Fort Hood, TX.

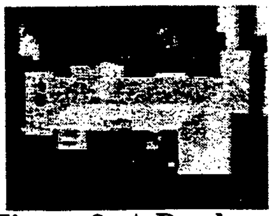


Figure 3: A Duplex

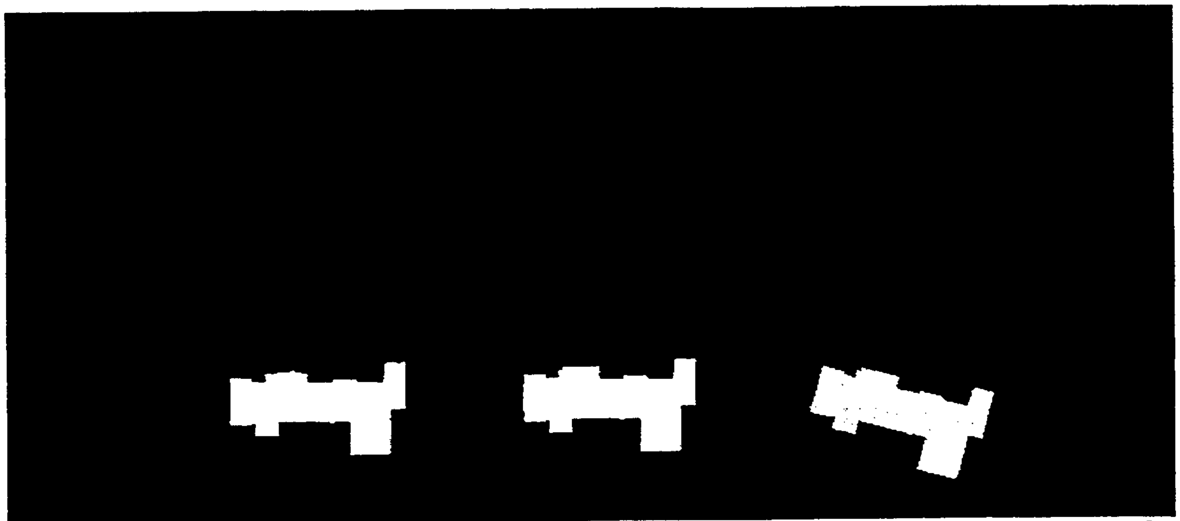


Figure 4: The training signal for Duplexes for the training image shown in Figure 2

5.1) The Vision Procedure Library

The vision procedure library contains 10 2-D vision procedures, as depicted in Figure 5, many of which are derived from ATR routines. Three of the procedures produce likelihood images (with orientation information) from intensity images and a template[2]. The rotation-free correlation procedure [Ravela, 1996] correlates the template at each position in the image by first rotating the template until the direction of the edge at the center of the template corresponds to the edge direction at the center of the image window. The TAStat procedure is a modification of the algorithm in [Nguyen, 1990]. For every image window it also rotates a mask of the object until it aligns with the local edge data, and then measures the difference between the intensity distributions of the pixels inside and outside of the mask. The greater the difference between the intensity distributions, the more likely the mask matches an object at that location and orientation in the image. Finally, the probing procedure also uses edge information to rotate the template for each image window, and then samples pairs of pixels in the image window, looking for edges that match the location of edges in the template.

[2] In all of our experiments, we assume that a template of the object is available.

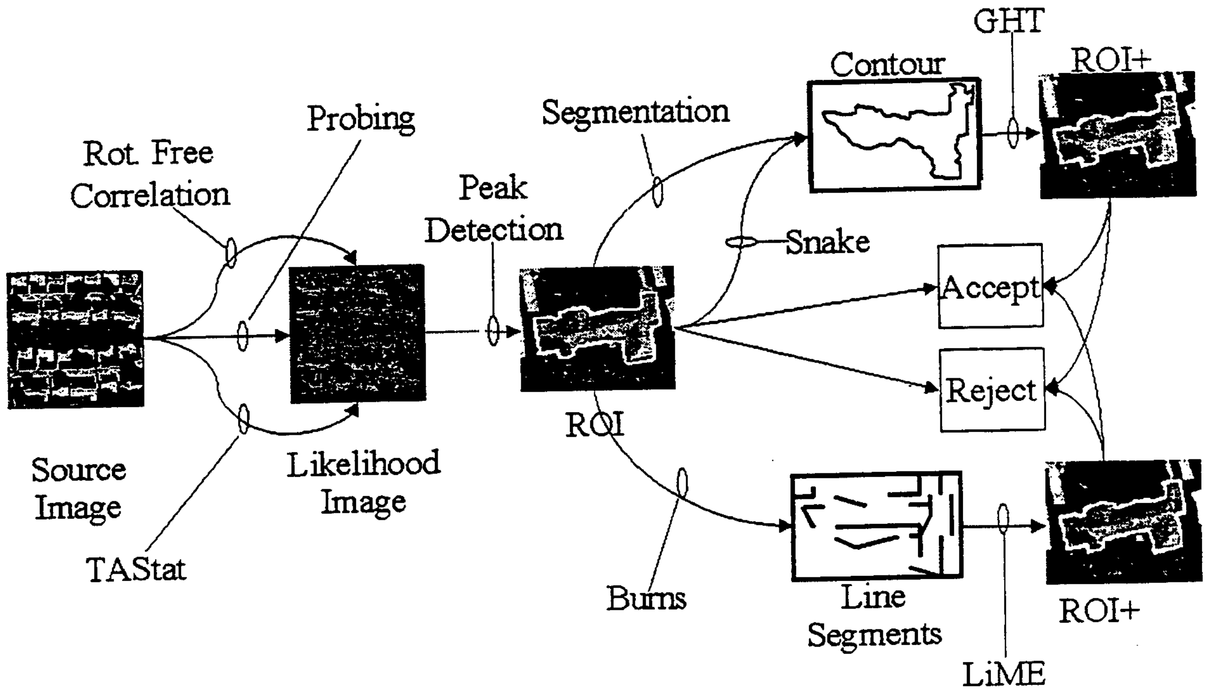


Figure 5: An iconic depiction of ADORE's current vision procedure library. The boxes represent intermediate image representations, such as likelihood images, image contours or "regions of interest" (image chips with masks). Depending on the quality of the photocopy, dark boxes are generally gray-scale images. Note that the peak detection procedure produces on the order of 20 ROIs each time it is called.

Regions of interest (ROIs) are chips from the original image that are hypothesized to correspond to object instances; each ROI also has a mask that details the hypothesized position and orientation of the object. ROIs can be extracted from likelihood images using a peak detection procedure, which finds the top N peaks in a likelihood image above a threshold. For these experiments, the peak detection procedure was parameterized to extract a maximum of 20 peaks from each likelihood image.

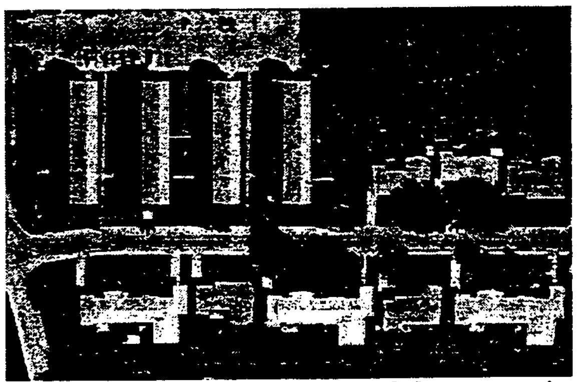
Five procedures can be applied to any ROI. Two of these actions are the special actions mentioned in Section 4.1, accept and reject. Accept and reject are terminal actions that end the recognition process. The other three options are: 1) an active contour procedure [Kass, 1988] that modifies the outline of an ROI mask until the contour lies along edges in the original image; 2) a segmentation procedure [Comaniciu, 1997] that extracts the boundary of a new region (as a 2-D contour) within the image chip; or 3) a straight line extraction procedure [Burns, 1986].

A Generalized Hough Transform procedure [Ballard, 1981] matches 2-D image contours to the contour of a template, thus creating a new ROI. A symbolic line matching procedure (LiME; [Beveridge, 1997]) finds the rotation, translation and scale that maps template (model) lines onto image lines, again producing a ROI. It should be noted that LiME transforms hypotheses in scale as well as rotation and translation, which puts it at a disadvantage in this fixed-scale domain.

5.2) Finding Duplexes

To test the system's ability to learn duplex recognition strategies, we performed N-fold cross-validation on the set of eight Fort Hood images. In other words, we divided the data into seven training images and one test image, trained ADORE on seven training images, and then evaluated the resulting strategy on the test image. We repeated this process eight times, each time using a different image as the test image. All the results presented in this paper are from evaluations of the test image.

Figure 6 shows the results of two tests, with the ROIs extracted by ADORE overlaid in red on top of the test image. As a crude measure of success, ADORE found 21 out of 22 duplexes, while producing six false positives. The only duplex not found by ADORE can be seen in the image on the right of Figure 6– it is the duplex that is half off the bottom right-hand corner of the image. Every duplex that lies completely inside an image was recognized. (The right side of Figure 6 also shows one false positive.)



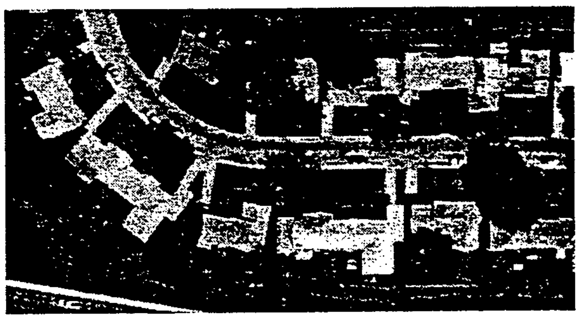


Figure 6: Duplexes extracted from two images. In the image on the left, all three duplexes were found. On the right image, a false positive appears on the upper right side. Also, the half-visible duplex to the bottom right was the only instance missed during testing – and only half of that duplex lies inside the image.

Analyzing ADORE in terms of false positives and false negatives is misleading. Much of the benefit of ADORE's dynamic strategies lies in its ability to refine imperfect hypotheses, not just make yes/no decisions. ADORE maximizes its reward function by creating the best hypotheses possible, given the procedure library. Table 1 gives a quantitative measure of ADORE's success. The left-most entry in Table 1 gives the average reward across all 22 positive duplex instances from the optimal strategy, where the optimal strategy is determined by trying all sequences of procedures and taking the best result. The second entry gives the average reward generated by the strategy learned by ADORE. As further points of comparison: the third entry in

Table 1 gives the average reward for duplex instances if no further processing is applied after hypotheses are generated; the fourth entry gives the average reward if every duplex ROI is segmented and the Generalized Hough Transform is used to reposition the ROI in terms of the region boundary; the fifth entry gives the average reward if the active contour procedure is applied (followed by the Generalized Hough Transform); and the sixth entry is the average reward if lines are extracted from hypotheses, followed by the LiME symbolic matcher.

Table 1. Comparison between the optimal policy, the policy learned by ADORE, and the four best fixed policies. The first row considers only positive instances; the second includes a penalty of –1 for each false positive. Note that there are 24 object instances.

	Optimal Policy	ADORE Policy	Fixed: Accept or Reject	Fixed: Segment	Fixed: Active Contours	Fixed: Line Extraction
Avg. Reward	0.8991	0.8803	0.7893	0.8653	0.7775	0.1807

Two conclusions can be drawn from Table 1. First, the strategy learned by ADORE for this task is within about 98 percent of optimal. Second, the dynamic strategy learned by ADORE, although not perfect, is better than any fixed sequence of actions. (The best fixed sequence of actions is to always segment ROIs and apply the Generalized Hough Transform.) This implies that dynamic strategies are worth learning.

5.3) Finding Smaller Houses

Having succeeded in finding a good strategy for finding duplexes, we expected to easily repeat the process for other styles of houses. Indeed, we expected the resulting policies to be very similar to the duplex policy; therefore, we repeated the experiment using the same methodology as above but with the house styles shown in Figure 7.

To our dismay, ADORE identified 18 of 19 instances of the house style *A* but generated 22 false positives. Combining the results from house styles A through D, ADORE found 47 out of 61 instances, while generating 85 false positives.

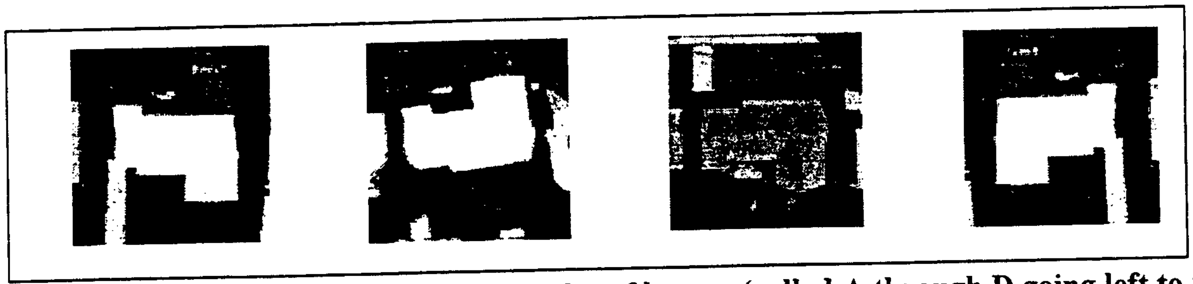


Figure 7: Templates of four other styles of houses (called A through D going left to right).

Why is this problem harder than finding duplexes? The most critical control decision in this domain comes after ROIs are extracted from likelihood images. At this point, ADORE has a choice of five procedures: segmentation, active contours, line extraction, accept or reject. Of these five, line extraction turns out to never be the optimal choice, but the four other options are optimal for some ROIs.

The control policy must select the optimal action based on features of the ROIs. By inspecting the weights of the neural net Q-functions trained for duplex recognition, we discovered that two of the eleven features dominated the control decision. The most important feature measures the average image edge strength along the boundary of the ROI. The second most important feature measures the percent of pixels outside the mask of the ROI that have roughly the same intensity value as pixels under the mask. (We informally refer to these as "missing" pixels, since their intensities suggest that they were accidentally left out of the hypothesis.)

Based mostly on these two features, ADORE learns a strategy for picking the next procedure. If we interpret the behavior of the Q-functions in terms of these two features, we can describe the control strategy learned by ADORE as follows (see Figure 8): ROIs with very high boundary edge strength and very few "missing" pixels should be accepted as is. (The points in Figure 8 correspond to training samples, color-coded in terms of the optimal action for each ROI. Dark blue points correspond to ROIs that receive approximately the same reward whether segmented or accepted as is.) If the edge strength is less high but relatively few pixels are "missing," then the ROI should be segmented. Although many false hypotheses – shown in yellow in Figure 8 because the optimal action is to reject them – are segmented according to this rule, there is another control decision after segmentation and the Generalized Hough Transform, where false hypotheses can be rejected and true ones accepted. This decision is made easier by the result of the segmentation procedure. There also is one small spot in the feature space where the active contour procedure is optimal. This is harder to explain and may result from the training set being a little too small. Finally, if the missing pixel count is high or the edge strength is low, reject the ROI. The solid boundaries in Figure 8 approximate our interpretation of the control policy's decision boundaries.

If we look at the edge strength feature for the other house recognition task, we find that it does not discriminate as well. It turns out that if you overlay the four templates shown in Figure 8 on top of each other, most of the boundaries are aligned. As a result, if a ROI for style A is incorrectly placed over an instance of styles B, C, or D, the edge's strength is still very high. (The same is true for ROIs of style B, C, and D). As a result, when looking for one style of house, false hypotheses created by instances of the other style have high edge strength, so the feature is not very discriminating. As a result, ADORE has a harder time learning a strategy that can distinguish between true instances and false ones, with the result that many more false hypotheses are created. In effect, the difference in feature space between one style and the next is too small to reliably distinguish between them.

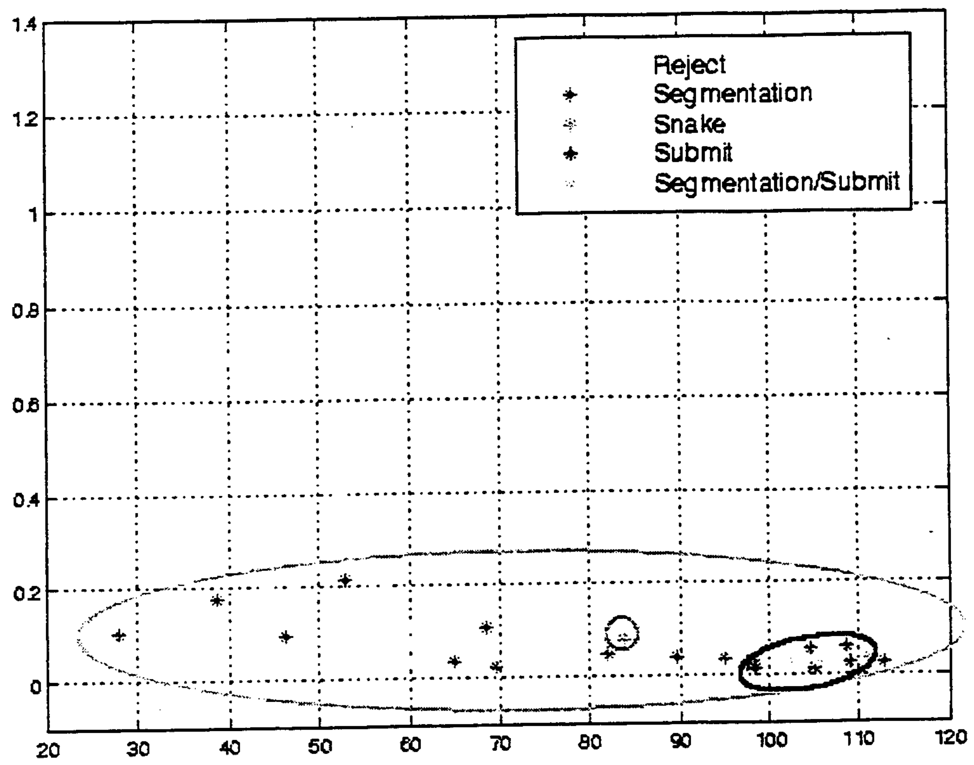


Figure 8: ROIs plotted in two dimensions of the eleven dimensional feature space. The color of the ROI point indicates the optimal control decision for that ROI. (Blue ROIs receive roughly the same reward whether they are submitted as is or refined through segmentation.) The ellipses correspond to our interpretation of the decision boundaries learned by the neural networks.

6. Conclusion

Under this contract, we built a prototype adaptive object recognition system that is capable of learning good object-specific recognition strategies when features of the intermediate data representations provide enough information to base intelligent control decisions on. Using this prototype, we successfully learned a dynamic control policy that outperforms any fixed strategy for a recognition task. Using the same prototype, we were then able to recognize other objects, some of which were similar to the first object type in the sense of being building designs, one of which was completely unrelated (the maintenance rails from the motor pool of Ft. Hood). In so doing, we met and exceeded our target milestones for the first year of the project.

At the same time, we have identified the primary weakness of our system. Without sufficiently discriminating features, it is not possible to learn effective control policies. (It should be noted that under these conditions, it is not possible to create effective control policies by hand, either.) We had just begun to look at the issue of dynamically selecting token features as well as vision procedures, and were beginning to implement a feature selection system based on the work of

[Jain & Zongker 97, Almuallim & Dietterich 92, Pudil, Novovicova & Kittler 94, Baker, Nayar and Murase 98, Vafaie & DeJong 93, Koller & Sahami 96].

7. Bibliography

Almuallim, H. and T. Dietterich, 1992. "Efficient Algorithms for Identifying Relevant Features," *Canadian Conference on Artificial Intelligence.*

Ballard, D., 1981. "Generalizing the Hough Transform to Detect Arbitrary Shapes," *PR*, 13(2):11-122.

Baker, S., S. Nayar, and H. Murase, 1998. "Parametric Feature Detection," *International Journal of Computer Vision,* 27(1):27-50.

Beveridge, R., 1997. *LiME Users Guide*. Technical report 97-22, Colorado State University Computer Science Department.

Burns, B., A. Hanson, and E. Riseman, 1986. "Extracting Straight Lines," *PAMI* 8(4):425-455.

Comaniciu, D. and P. Meer, 1997. "Robust Analysis of Feature Space: Color Image Segmentation," *CVPR*, pp. 750-755.

Draper, B., R. Collins, J. Brolio, A. Hanson, and E. Riseman, 1989. "The Schema System," *IJCV*, 2(2):209-250.

Draper, B., 1996. "Modelling Object Recognition as a Markov Decision Process," *ICPR*, D95-99.

Draper, B. and K. Baek, 1998. "Bagging in Computer Vision," *CVPR*, pp. 144-149.

Kass, M., A. Witken, and D. Terzopoulis, 1988. "Snakes: Active Contour Models," *IJCV* 1(4):321-331.

Koller, D. and M. Sahami, 1996. "Toward Optimal Feature Selection," *International Conference on Machine Learning,* Bari, Italy.

Maloof, M., P. Langley, S. Sage, and T. Binford, 1997. "Learning to Detect Rooftops in Aerial Images," *IUW*, 835-846.

Mundy, J., 1995. "The Image Understanding Environment Program," *IEEE Expert*, 10(6):64-73.

Nguyen, D., 1990. *An Iterative Technique for Target Detection and Segmentation in IR Imaging Systems*, Technical Report, Center for Night Vision and Electro-Optics.

Pudil, P., J. Novovicova, and J. Kittler, 1994. "Floating Search Methods in Feature Selection," *Pattern Recognition Letters*, 15:1119-1125.

Peng, J. and B. Bhanu, 1998. "Closed-Loop Object Recognition using Reinforcement Learning," *PAMI* 20(2):139-154.

Rasure, J. and S. Kubica, 1994. "The KHOROS Application Development Environment," In *Experimental Environments for Computer Vision*, World Scientific, New Jersey.

Ravela, S., B. Draper, J. Lim, and R. Weiss, 1996. "Tracking Object Motion Across Aspect Changes for Augmented Reality," *IUW*, pp. 1345-1352.

Sutton, R., 1988. "Learning to Predict by the Methods of Temporal Differences," *ML*, 3(9):9-44.

Tesauro, G., 1995. "Temporal Difference Learning and TD-Gammon," *CACM*, 38(3):58-68.

Vafaie, H. and K. DeJong, 1993. "Robust Feature Selection Algorithms," *International Conference on Tools with AI,* Boston.

Watkins, C., 1989. *Learning from Delayed rewards,* Ph.D. thesis, Cambridge University.

Zhang, W. and T. Dietterich, 1995. "A Reinforcement Learning Approach to Job-Shop Scheduling," *IJCAI.*

Appendix A: "IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 1998"

Bagging in Computer Vision

Bruce A. Draper

Department of Computer Science
Colorado State University
Fort Collins, CO. 80523
draper@cs.colostate.edu

Kyungim Baek

Department of Computer Science
Colorado State University
Fort Collins, CO. 80523
baek@cs.colostate.edu

Abstract

Previous research has shown that aggregated predictors improve the performance of non-parametric function approximation techniques. This paper presents the results of applying aggregated predictors to a computer vision problem, and shows that the method of bagging significantly improves performance. In fact, the results are better than those previously reported on other domains. This paper explains this performance in terms of the variance and bias.

1 Introduction

Function approximation and classification are ubiquitous problems in computer vision. The pattern recognition task of mapping from a set of (not necessarily independent) features to a predicted function value or label rises in object recognition, target recognition, visual navigation, and almost every other application of computer vision. For many years, the traditional approach to this problem was to estimate the parameters of the feature distributions, and apply maximum likelihood classification or related approximation techniques to predict labels/values. Unfortunately, this approach often performs poorly in the presence of unmodeled dependencies between features.

To counteract this problem, many researchers have turned to non-parametric techniques such as neural networks, decision trees, and nearest neighbor classifiers. While these techniques often provide improved results on computer vision data sets, they are all too often "black boxes" to the researchers who use them. As a result, many researchers do not know how to optimize the performance of these techniques short of extensive trial-and-error.

This paper presents a technique called *bagging*[1] for improving the performance of non-parametric function approximation techniques. This approach was developed by Breiman [1], and is similar to work by Dietterich on improving non-parametric classification [2, 4, 5]. (See also [3, 6].) The basic idea is to randomly sample the training set and produce multiple function approximations. The aggregated or "bagged" predictor is then the average of the component predictions on a given sample. The non-obvious result is that the bagged predictor is significantly better than any of the component predictors.

Since results from other domains do not always apply in computer vision, this paper applies bagging to the task of evaluating image regions via neural networks. Our results prove to be consistent with - if not better than - those presented by Breiman [1]. In addition, we predict which tasks will benefit from bagging by measuring the variance and bias of the component non-parametric function approximations (predictors).

2 Bagging

The basic idea behind bagging is simple: train multiple function approximations and average their results. To clearly explain when and why this works, however, we have to get a little more formal. A function approximation task can be defined in terms of a set W:

$$W = \{(F_n, y_n), n = 1, ..., \infty\},$$

where F_n is a feature vector describing a data instance, and y_n is the function value associated with instance n. The goal is to learn a function approximation φ such that $y_n \approx y'_n = \varphi(F_n)$ from a training set $T \subset W$.

Training algorithms (such as backpropagation) for non-parametric function approximation can be thought of as processes that map training sets onto functions (i.e. $T \rightarrow \varphi$). They are typically used to train a single approximation $\varphi = Train(T)$. In bagging, however, T is randomly subsampled (with replacement) to produce M subsets T_M, and each of

[1]Bagging is an acronym for bootstrap aggregating.

these subsets is used to train a function approximation $\varphi_M = Train(T_M)$. The bagged approximation can then be written as:

$$\varphi_{bag}(F_i) = \frac{(\sum_{k=1}^{M} W_k \varphi_k(F_i))}{(\sum_{k=1}^{M} W_k)}$$

where

$$W_k = \frac{1}{MSE(\varphi_k)}$$

(In other words, $\varphi_{bag}(F_i)$ is the weighted average of $\varphi_M(F_i)$.) The weighting term, W_M, gives more weight to predictors that have lower overall mean squared error (MSE) than others. (Breiman does not use a weighting term, but we found that it significantly improved performance.)

There are two bases for arguing the φ_{bag} should be a better approximation than $\varphi = Train(T)$. The first assumes that the training algorithm is unstable, in the sense that a small change to the training set T_M results in a large change to approximation φ_M. (This assumption is true for backpropagation, and Breiman argues that it is true for decision tree inference algorithms as well [1].) In this case, the function approximations $\varphi_1, ..., \varphi_M$ are largely independent of each other. If they are also unbiased, then $\varphi_1(F_i), ..., \varphi_M(F_i)$ can be thought of as M independent samples drawn from a distribution whose mean is y_i. As a result, the expected value of $\varphi_{bag}(F_i)$ is y_i, even if the errors in the individual approximations φ are large.

The second argument says that most training algorithms get caught in local optima, and as result produce approximations φ that are accurate over part but not all of the feature space. Once again, the difference among the subsampled training sets T_M suggest that the approximations φ_M will correspond to different local optima. If each φ_M is accurate for most of the feature space, then the majority of estimates $y'_{iM} = \varphi_M(F_i)$ will accurately reflect y_i. Moreover, we can once again assume that the outliers will be unbiased, so their expected value is zero. As a result, the bagged predictor should be accurate across the entire feature space, or at least a larger portion of the space than any component φ_M.

Both of these arguments can be cast in terms of variance and bias. In general, bias is the tendency of an approximation to overestimate or underestimate, while variance is (informally) the noise in the estimation process. In regression, it is a well-known theory that the expected squared error of an approximation algorithm on test data can be decomposed into bias and variance terms [7]:

$$Error(F_i) = Bias^2(F_i) + Var(F_i)$$

where

$$Bias(F_i) = \overline{y_i}' - y_i$$

$$Var(F_i) = \mathrm{E}_T(\overline{y_i}' - \varphi_T(F_i))^2$$

and

$$\overline{y_i}' = \mathrm{E}_T(y_i)$$

The general result of bagging is to reduce the variance, so that the error of an aggregate approximation approaches the bias error as the number of component approximations approaches infinity. If the component predictors are unbiased (which they should be overall, but may not be for specific samples F_i), then the aggregate error approaches zero.

3 The Test Problem

Our test case evaluates bagging on a region evaluation problem. As part of another project, we have an object recognition system that generates hypotheses in the form of labeled image regions. Working in an aerial image domain, we collected a data set of 3,374 house hypotheses, each represented by a mask and a corresponding image chip. The task is to estimate the quality of hypotheses from features describing the regions.

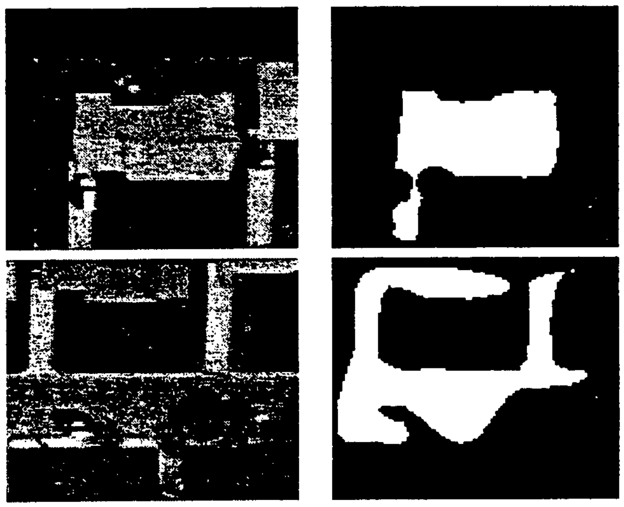


Figure 1: Image chips(left) and masks(right) for two house hypotheses. The top hypothesis gets a score of .87(the inclusion of the driveway is most of the penalty), while the lower hypothesis scores only .15 since it barely overlaps a house.

A total of twenty features are used to describe hypothesized regions. The features themselves are not unusual. They include shape features, size features, texture features, and brightness features. One feature

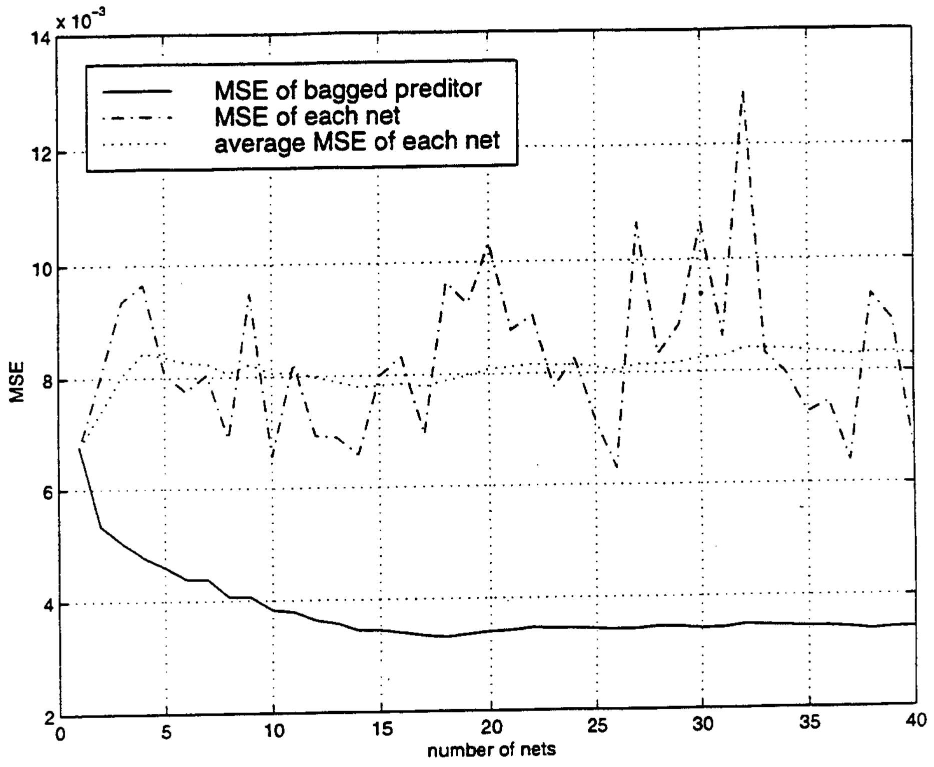


Figure 2: Mean Squared Error of Forty Nets vs. Bagged(Aggregated) Prediction

	Individual Nets	Bagged Net : 15 nets (decrease)	Bagged Net : 40 nets (decrease)
MSE average	0.0083	0.0036 (56.6%)	0.0034 (58.8%)
MSE min(best)	0.0063	0.0031 (50.8%)	0.0034 (45.8%)
MSE max(worst)	0.0129	0.0040 (69.0%)	0.0034 (73.6%)
Standard Deviation	0.0014	0.0002	–

Figure 3: MSE Comparison of individual nets and bagged net. The 3rd column is the result of bagging 100 times with 15 nets randomly selected from 40 nets in each time.

is a shadow measure that relies on knowing the sun angle. In many respects, this task is typical of function approximation tasks that arise in computer vision.

Part of what makes this task difficult is that the answer is not binary. Most of the hypotheses are good to the extent that they overlap the true position of a house, but they do not match it exactly. The quality of a hypothesis depends on how closely it matches the ground truth data, and is measured according to:

$$\frac{|H \cap T|}{|H \cup T|}$$

where H is the set of hypothesized pixels, and T is the true set of house pixels. (Note that this function - the size of the intersection of the pixel sets divided by the size of their union - is bounded between zero and one.) Figure 3 shows two hypotheses, one with a high score (.87) because it matches the true house region closely, the other with a lower score (.15).

4 Results

The goal of this experiment is to test whether bagging improves accuracy in computer vision problems, and if the improvement is in line with the results reported in [1]. Secondarily, we are also interested in whether the reason for the improvement (if any) matches the explanation given in Section 2.

We tested the bagging technique using backpropagation to train fully-connected neural networks with 30 hidden units, a sigmoidal squashing function and a momentum term of 0. In order to test the bagging technique, we split our total database of 3,374 samples into a training set T of 2,874 samples and a test set U of the remaining 500 samples. We then trained a total

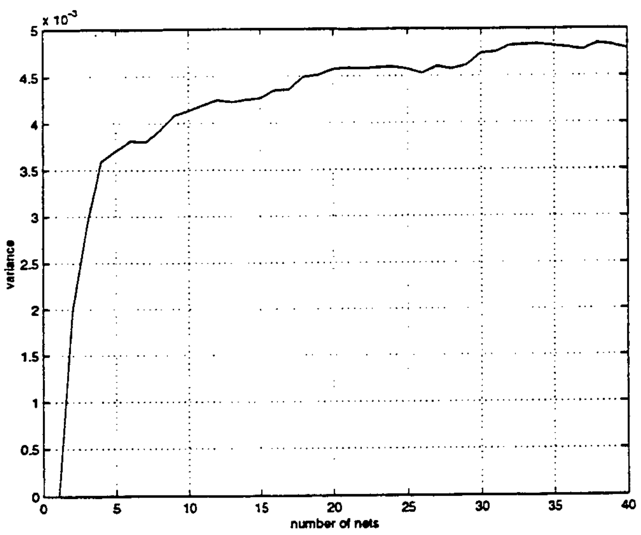


Figure 4: Cumulative Variance

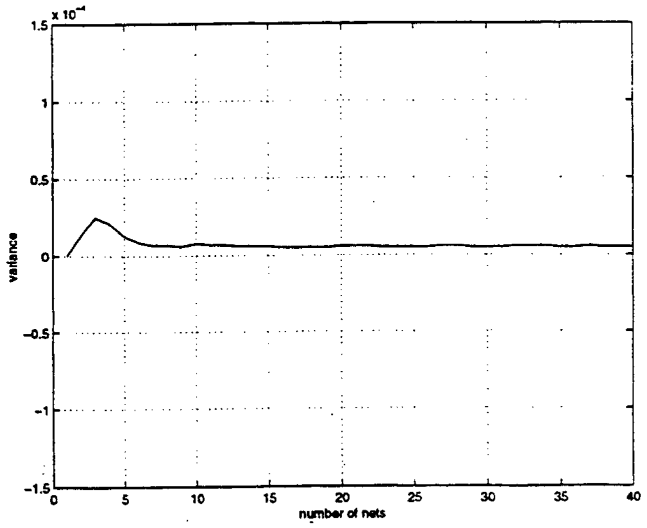


Figure 6: Variance of Bagged net

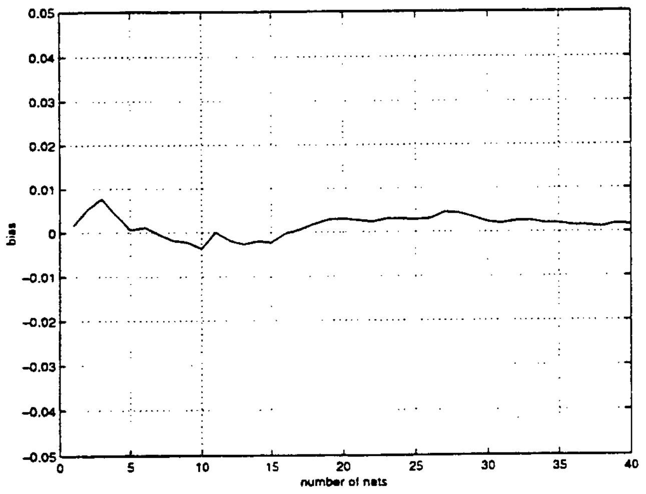


Figure 5: Cumulative Bias

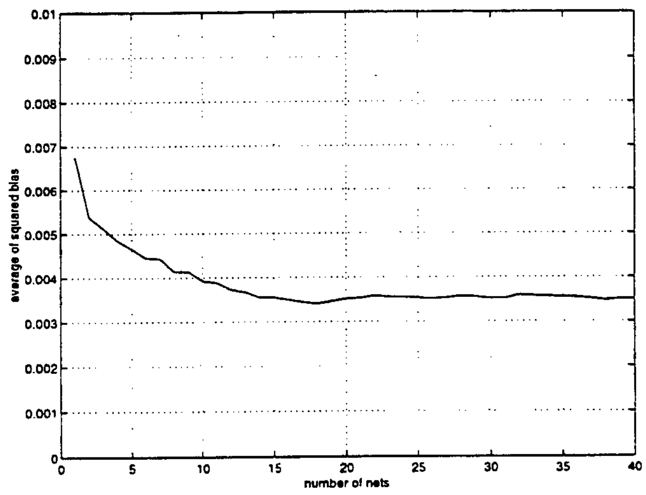


Figure 7: Bias Error

of forty networks, where network φ_i was training by randomly selecting 2,000 samples from T to form T_i, and then applying backpropagation to train network φ_i over data set T_i.

Figure 2 shows the result of applying these networks to the test set U, with and without bagging. The dash-dot(-.-) line shows the mean squared error(MSE) of the test set on each of the forty nets. The solid line is the MSEs that result from bagging, so that for example the MSE of bagging the first five nets is 0.0048, while the MSE from bagging ten nets is 0.0038. (Remember that the function being approximated has a range from 0 to 1.) For comparison's sake, the dotted line(..) is the running average of the MSEs of the nets.

The immediate conclusion from Figure 2 is that bagging leads to a striking improvement. The MSEs of the forty individual nets ranged from 0.0063 to 0.0129 with an average of 0.0083, while the MSE from bagging the forty networks is 0.0034 (Figure 3). Bagging therefore leads to an improvement of 58.8% over the average performance of the individual nets and an improvement of 45.8% over the best of the component nets. In addition, although bagging is clearly more expensive than other approaches (since it trains multiple nets), from Figure 2 we see significant improvement from bagging as few as five neural nets, and we see almost all of our improvement by the time fifteen nets have been bagged. The second column of Figure 3

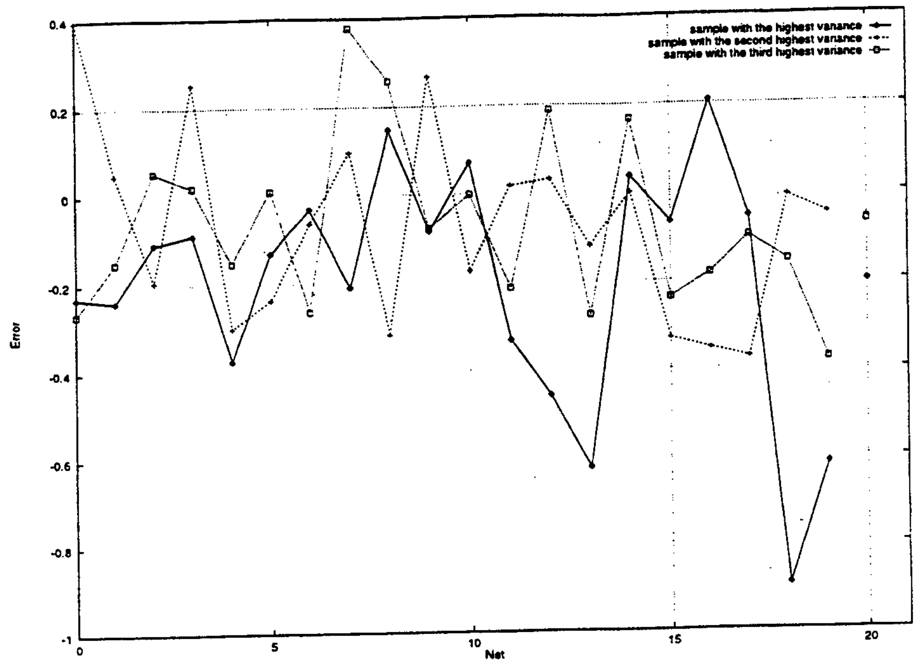


Figure 8: Prediction errors for the three samples with the highest variance across the first twenty nets(net #0 - net #19). The variance is clearly visible in the different errors produced by each net, while the average prediction(net #20) for each sample is reasonable.

shows it more clearly. The values have been drawn by bagging 15 randomly selected nets 100 times. The average error shows that the performance is very close to the results gained by bagging 40 nets.

Not only are these results consistent with those reported in [1], the improvement of 58.8% ranks it as the best. (Breiman's best was an improvement of 47% on the Heart dataset at the UCI dataset repository.) In the test reported here, the random sampling was done without replacement. However, the results of experiments with replacement are not much different from previous test. Although the MSE is slightly worse (ranging from 0.0075 to 0.0174 for individual nets, 0.0044 for bagging 40 nets), the average decrease in error is 64.2%.

Whether this is because of the domain, because we bagged neural nets as opposed to decision trees (as Breiman did), or because we added the weighting term, we cannot say. Either way, bagging is clearly beneficial in this domain. But does the reason for its success lie in the explanations posited in Section 2?

Since we observed a striking improvement by bagging, if the explanations from Section 2 hold, two other predictions should be met: 1) the bias of the neural nets (viewed as a set) should be low and the variance of the neural nets should be high. 2) bagging reduces most of the variance, so the error of bagged net should approach the bias error. Figures 4 through 7 do indeed show this to be true, which explains why there was so much improvement as a result of bagging. Over forty nets, the variance is almost 0.005, while the bias is 0.0015 (Figures 4 and 5). In Figure 6, we see that the variance of the bagged net is almost zero and the bias error (average of squared bias of each sample over multiple nets) in Figure 7 is almost same as the MSE of bagged net in Figure 2.

Finally, one intuitive way to look at these results is to plot the three test samples with the highest variance. As shown in Figure 8, all of these samples were estimated with reasonable accuracy by most of the nets. (We show just the first twenty nets and the bagged predictor in Figure 8, while the vertical dimension is the error between the true and predicted value.) Certain nets did very poorly on certain samples, however. For example, net #18 had an error of almost 0.9 (in a range from 0 to 1) on one sample. (Nets #13 and #19 also had trouble with this sample.) Another sample proved problematic for net #7, with an error of almost 0.4. Because the aggregate predictor averages across all forty nets, however, these occasional failures - which occur for every net on at least a few samples - are replaced by more reliable estimates, resulting in an aggregate predictor with far lower error than any of its components.

5 Conclusion

Aggregate or "bagged" predictors work as well or possibly even better on a typical computer vision dataset than they do on the non-visual UCI test suites for which results have previously been reported.

References

[1] L. Breiman, *Bagging Predictors.* Technical Report No. 421, Dept. of Statistics, University of California(Berkeley). Sept. 1994.

[2] T. Dietterich and G. Bakiri, "Solving Multiclass Learning Problems via Error-Correcting Output Codes," *Journal of Artificial Intelligence Research,* 2:263-286(1995).

[3] D. Heath, S. Kasif, and S. Salzberg, "K-dt: a Multi-tree Learning Method," 2nd *Workshop on Multistrategy Learning,* Chambery, France, 1993, pp.1002-1007.

[4] E.B. Kong and T. Dietterich, *Why error-correcting output coding works with decision trees.* Technical Report, Dept. of Computer Science, Oregon State University (Corvalis), 1995.

[5] E.B. Kong and T. Dietterich, "Probability Estimation via Error-Correcting Output Coding," *IASTED Conference on Artificial Intelligence and Soft Computing,* Banff, Canada, 1997.

[6] H. Kwok and C. Carter, "Multiple Decision Trees," in *Uncertainty in Artificial Intelligence 4,* Shachter, Levitt, Kanal and Lemmer (eds.), North-Holland, 1990, pp.327-335.

[7] E.B. Kong and T. Dietterich, "Error-Correcting Output Corrects Bias and Variance," *Machine Learning: Proceedings of the 12th International Conference,* pp.313-321.

Appendix B: "International Conference on Vision Systems, 1999" (a longer version will appear in Videre)

1) Introduction

As the field of computer vision matures, fewer and fewer vision systems are built "from scratch". Instead, computer vision systems are assembled out of standard procedures, including (but my no means limited to): image smoothing / image enhancement, edge and corner extraction, region segmentation, straight line and curve extraction, grouping, symbolic model matching (including Hausdorf matching, key feature matching, and heuristic search), appearance matching, pose determination, and depth from stereo, motion or focus. Separately, each of these procedures addresses a part of the computer vision problem. Sequenced together, they form end-to-end vision systems that perform specific tasks.

To help users build end-to-end systems, computer vision software environments provide libraries of image processing and computer vision procedures, and graphical user interfaces to sequence them together. In Khoros [Rasure, 1994 #25], programmers build applications by selecting procedures (called "glyphs") from a menu and graphically connecting the output of one procedure to the input of another. Unfortunately, the Khoros procedure library is mostly limited to 2D image processing routines; high-level grouping and matching procedures, for example, are missing. CVIPtools [Umbaugh, 1998 #30] is another software environment intended primarily for academic use. Although it also focuses mostly on image processing, it includes a slightly larger set of low-level vision procedures for tasks such as region segmentation and straight line extraction. The Image Understanding Environment (IUE) [Mundy, 1995 #21] has a more sophisticated procedure library for both 2D and 3D computer vision. The IUE is still under development, but a preliminary version is freely available.

Software tools such as Khoros, CVIPtools and the IUE make it easier for programmers to form and test sequences of vision procedures. Unfortunately, they do not help programmers with the underlying problem of how to select procedures for a specific task. Programmers are left to choose vision procedures based on intuition, and to refine sequences of procedures by trial and error.

The goal of the Adaptive Object Recognition (ADORE) project is to provide a theoretically sound mechanism for dynamically selecting vision procedures based on the task and the current state of the interpretation. We eventually seek to build a system that can adapt to any recognition task by dynamically selecting actions from among dozens (if not hundreds) of vision procedures. Not surprisingly, this ambitious goal exceeds our current grasp. This paper describes an initial prototype of ADORE that learns to find houses in aerial images using a library of ten vision procedures. While this system is clearly short of our ultimate goal, it is an example of an end-to-end system that adapts by learning to dynamically control vision procedures. This paper describes two experiments with the prototype version of ADORE – one in which ADORE succeeds in finding a good object recognition strategy, and one in which it fails.

2) Examples of Static and Dynamic Control

Before describing ADORE in detail, let us first illustrate the type of problem it is supposed to solve. Figure 1 shows a nadir-view aerial image. The task is to find instances of specific styles of houses, such as the duplex in Figure 2. To achieve this goal, ADORE is given access to ten vision procedures and a template of the duplex. (Descriptions of all ten procedures can be found in Section 5.1.) ADORE is also given training images and a training signal that gives the position and orientation of each duplex. ADORE's role is to dynamically select and execute procedures so as to produce duplex (and only duplex) hypotheses.

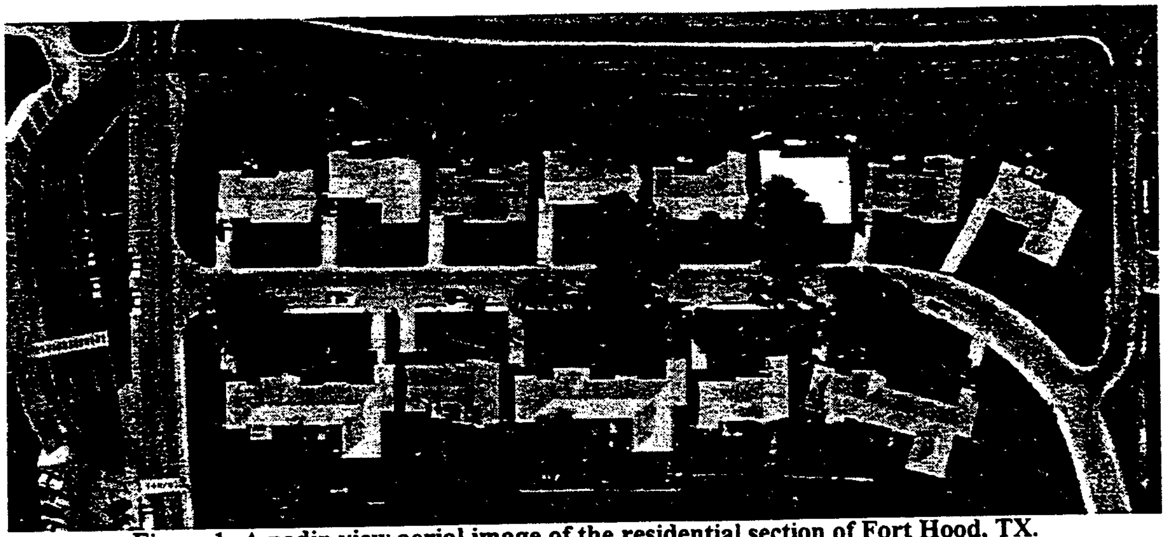


Figure 1: A nadir-view aerial image of the residential section of Fort Hood, TX.

ADORE finds duplexes by learning a control strategy that selects a vision procedure at each processing step. For example, ADORE's vision procedure library currently contains three procedures for producing regions of interest (ROIs) from images: a rotation-free correlation procedure, a statistical distribution test, and a probing routine. All three can be used to generate duplex hypotheses, but ADORE learns from the training data that for this problem – where the duplexes are almost identical to each other, and lighting differences and perspective effects are minimal – pixel-level correlation outperforms the other two procedures. ADORE therefore learns a recognition strategy that begins with correlation.

The next step is more complex. Figure 3 shows three ROIs produced by correlation. The ROI on the left of Figure 3 matches the position and orientation of a duplex very well. In fact, none of the procedures in the procedure library can improve this hypothesis, so the best action for

ADORE to take is to accept it as is. The ROI on the right in Figure 3, on the other hand, does not correspond to any duplex. The best action here is to reject it.

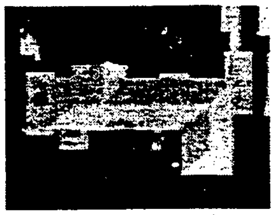


Figure 2: A Duplex

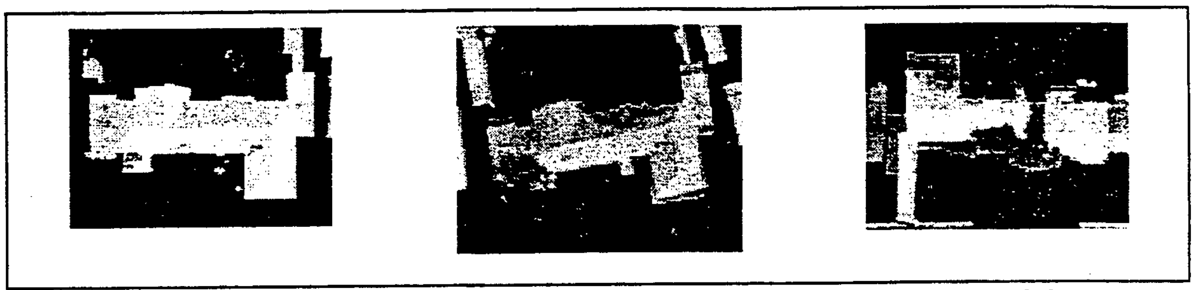


Figure 3: Three hypothesized regions of interest (ROIs). Although all three were created the same way, the best action for the one on the left is to accept it, while the one in the middle should be refined via the segmentation procedure, and the one on the right should be rejected.

The ROI in the middle of Figure 3, on the other hand, is more interesting. This ROI roughly matches a duplex, but the position and orientation of the duplex are skewed, probably because the duplex is partially occluded in the image . In this case, the best procedure is to refine the hypothesis by resegmenting the image chip (using [Comaniciu, 1997 #10]) and then applying a Generalized Hough Transform [Ballard, 1981 #5] to align the template with the extracted region boundary. Figure 4 shows the resulting hypothesis after these two procedures are applied.

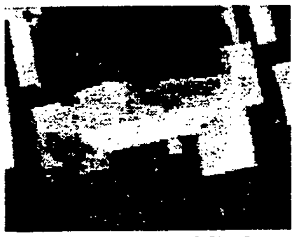


Figure 4: Refined ROI for middle hypothesis from Figure 3.

Examples like the ones in Figure 3 demonstrate the importance of dynamic control decisions. In all three cases, the first procedure was the same: correlation. The choice of the next procedure, however, depended on the quality of the ROI produced by the previous step. In general, control strategies should choose procedures based not only on static properties of the object class and image domain, but based also on data produced by previous procedures.

3) Related Work

At a theoretical level, researchers argued for specialized recognition strategies built from reusable low-level components long before the appearance of software support tools like Khoros. As far back as the 1970s, Arbib argued from psychological evidence for specialized visual "schemas" built from reusable components [Arbib, 1972 #4]. Ullman developed a similar theory, in which primitive "visual routines" are combined to form specialized recognition strategies [Ullman, 1984 #29]. Later, Aloimonos [Aliomonos, 1990 #2] and Ikeuchi & Hebert [Ikeuchi, 1990 #16] argued for specialized recognition strategies made from primitive vision operations in the context of visual robotics.

At a practical level, researchers have been building recognition systems with special-purpose recognition strategies for twenty years. In the late '70s and early '80s, researchers built AI-style production and blackboard systems that selected and sequenced vision procedures to achieve specific tasks. Nagao & Matsuyama's production system for aerial image interpretation [Nagao, 1980 #22] was one of the first, and lead to several long-term development efforts, including SPAM [McKeown, 1985 #20], VISIONS/SCHEMA [Draper, 1989 #11], SIGMA [Hwang, 1986 #15], PSEIKI [Andress, 1988 #3] and OCAPI [Clement, 1993 #9]. More recently, other researchers [Chien, 1996 #35][Lansky, 1995 #36][Jiang, 1994 #17] have applied AI-style planning technology to logically infer control decisions from a database describing the task and the available procedures.

Unfortunately, knowledge-based systems are often *ad-hoc*. Researchers formulate rules for selecting procedures based on their intuition, and refine these rules through trial and error. (See [Draper, 1992 #12] for a description of the knowledge engineering process in object recognition) As a result, there is no theoretical reason for believing that the control policies that emerge from these heuristics are optimal or even good, nor is there any way to directly compare these systems or evaluate their control policies.

Recently, researchers have tried to put the control of object recognition on a stronger theoretical foundation using Bayes nets (e.g. TEA1 [Rimey, 1994 #33] and SUCCESSOR [Mann, 1996 #19]). Unfortunately, designing Bayes nets can itself become an *ad-hoc* knowledge engineering process. Other researchers are trying to eliminate the knowledge acquisition bottleneck by learning control policies from examples. Researchers at Honeywell used genetic algorithms to learn target recognition strategies [Au, 1996 #1], while reinforcement learning has been used by Draper to learn sequences of procedures [Draper, 1996 #13] and by Peng & Bhanu to learn parameters for vision procedures [Peng, 1998 #24]. Maloof et. al. train classifiers to accept or reject data instances between steps of a fixed sequence of procedures [Maloof, 1997 #18].

4) Object Recognition as a Supervised Learning Task

The goal of the adaptive object recognition (ADORE) project is to avoid knowledge engineering by casting object recognition as a supervised learning task. Users train ADORE by providing training images and training signals, where the training signal gives the desired output for the training images. ADORE learns control strategies that dynamically select vision procedures in order to recreate the training signal as closely as possible from the training images. This control strategy can then be used to hypothesize new object instances in novel images.

To learn control strategies, ADORE models object recognition as a discrete control process. The state of the system is determined by data produced by vision procedures. For example, the state of the system might be a region of interest (ROI), a set of 2D line segments, or a 2D contour. The actions are vision procedures that change the state of the system by producing new data from current data. A control policy is a function that maps states onto actions. In the context of ADORE, control policies map data onto vision procedures, thereby selecting the next action at each step of the recognition process.

4.1) The ADORE prototype

At a systems level, ADORE is most easily thought of as two distinct components: a run-time execution monitor that applies vision procedures to data, and an off-line learning system that trains control policies.

4.1.1) The Execution Monitor

The execution monitor is a run-time loop that begins when an image is presented to the system. On each cycle, it evaluates the control policy on the current data, thereby selecting a vision procedure. It then applies the vision procedure to the current data, producing new data. This loop continues until a vision procedure returns without producing any new data.

Of course, this simple description glosses over some important details. After every vision procedure, the execution monitor measures features of the output data to be used in training control policies. The execution monitor also measures the run-time of each procedure, to be used as a component in the reward function if the goal is to train control policies that make tradeoffs between time and accuracy. Finally, there are two special procedures – called accept and reject – that return no data. The accept procedure signals that an object instance has been found; during training, accepted data tokens are compared to the training signal and a reward or penalty is generated. The reject procedure signals that the current data should be rejected; during training, it always returns a reward of zero.

In the interests of generality, the execution monitor is independent of the vision procedure library. Each vision procedure is an independent unix executable; a library file tells the execution monitor the number and type of input arguments for each procedure, the number and type of output arguments, and the unix pathname. As a result, vision procedures can be added or removed from the system simply by editing the library file. Similarly, the execution monitor is independent of particular data representations, since all data tokens are kept in files. For each data type, the library file tells the execution monitor 1) the name of the data type (so the monitor can match data tokens with arguments to vision procedures); 2) the number of features measured (for function approximation – see below); and 3) the path of the unix executable for measuring features. Thus new data types, like new vision procedures, can easily be added to the system.

4.1.2) Control Policies

The execution monitor is a control loop for applying vision procedures to data. The control policy directs the execution monitor by selecting a vision procedure at each step. Since the choice of vision procedure depends in part on the target object class and image domain, a different control policy is learned for every object recognition task.

Control policies are functions that map data tokens (represented in terms of their features) onto vision procedures. In order to learn sound control policies, object recognition is modeled as a discrete control problem. In particular, the state of the system is always defined by the features of the current data. The vision procedures are the discrete actions, and these actions are non-deterministic in the sense that we cannot predict the features of the output based solely on the features of its input.

To train a control policy, we train a Q-function for every vision procedure. In control theory, a Q-function is a function that predicts the expected reward that will follow from applying a specific action to the current state. For example, in ADORE we train a Q-function for predicting the reward that will result from applying the segmentation procedure to an ROI, based on the features of the ROI. We also train Q-functions for predicting the rewards that will follow from applying the active contour procedure to an ROI and the correlation procedure from an image. Control policies are implemented by evaluating the Q-function of every vision procedure that can be applied to the current data (based on its datatype) and selecting the procedure with the highest Q-value.

It is important to note that the Q-function predicts the total reward that follows a procedure, not just the immediate reward. For example, in the experiments described below the immediate reward for resegmenting an ROI is zero, since the segmentation procedure returns a 2D contour while the training signal is given in terms of ROIs. Nonetheless, there can be a delayed reward for resegmenting an ROI, since the resulting contour can be transformed into another ROI through the Generalized Hough Transform procedure. The Q-function for the segmentation procedure predicts this delayed reward, not just the immediate reward from the segmentation procedure.

4.1.3) Off-line Learning

The control and artificial intelligence literatures contain many techniques for learning optimal Q-functions for control problems with discrete state spaces. If the transition probabilities associated with the actions are known (a so-called process model), dynamic programming can be used to estimate Q-values and produce an optimal control strategy. In the absence of a process model, reinforcement learning (most notably the temporal difference [Sutton, 1988 #27] and Q-learning [Watkins, 1989 #32] algorithms) have been shown to converge to optimal policies in a finite number of steps.

Unfortunately, the object recognition control problem as defined here depends on a continuous, rather than discrete, state space. Tesauro [Tesauro, 1995 #28] and Zhang & Dietterich [Zhang, 1995 #31] have shown empirically that neural nets can approximate Q-functions for continuous feature spaces within a reinforcement learning systems and still produce good control policies. Unfortunately, their method required hundreds of thousands of training cycles to converge. ADORE has a sequence of continuous feature spaces, one for each data representation (images, ROIs, contours, etc.). This requires getting a sequence of neural nets to converge on a control policy. Although theoretically possible, we have not yet succeeded in making this work.

Instead, we train Q-functions by optimistically assuming that the best control policy will always select the action that creates the highest possible reward for every data token. Strictly speaking, this assumption is not always true: a control policy maps points in feature space onto actions, and it is possible for two different tokens to have the same feature measurements and yet have different "optimal" actions. Nonetheless, the optimistic assumption is usually true, and it breaks the dependence between Q-functions, allowing each neural net to be trained separately.

In particular, we approximate Q-functions by training backpropagation neural networks. The training samples are data tokens extracted from the training images. For each training sample, we apply all possible sequences of procedures, in order to determine which procedure yields the maximum reward . A neural network is trained for each vision procedures using the data features as input and the maximum reward from the data created by the vision procedure as the output.. In this way, the neural net learns to approximate the future reward from an action under the optimistic control assumption. (Complicating the picture somewhat, we "bag" the neural nets to reduce variance; see [Draper, 1998 #14].)

5) Experiments

To test ADORE in a tightly controlled domain, we trained it to recognize styles of houses in aerial images like the one in Figure 1. In the first experiment, the goal is to find duplexes of the type shown in Figure 2. The training signal is a bitmap that shows the position and orientation of the duplexes in the training images; Figure 5 shows the training signal matching the image shown in Figure 1. The reward function used to evaluate hypotheses during training is the size of the pixel-wise intersection of the hypothesis and the training signal, divided by the size of the union. This evaluation function ranges from one (perfect overlap) to zero (no overlap).

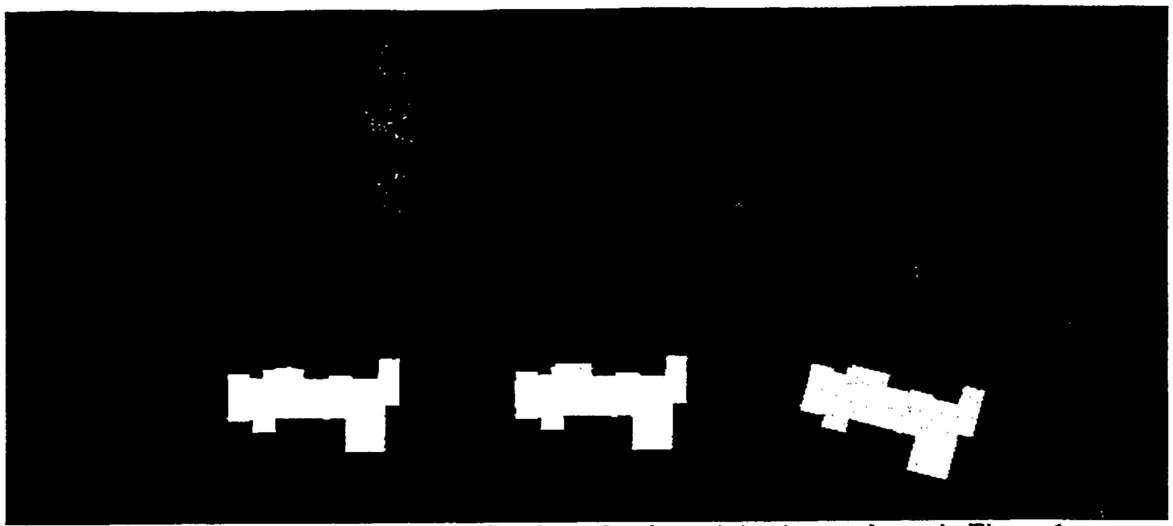


Figure 5: The training signal for Duplexes for the training image shown in Figure 1

The Vision Procedure Library

The vision procedure library contains ten 2D vision procedures, as depicted in Figure 6. Three of the procedures produce likelihood images (with orientation information) from intensity images and a template[1]. The rotation-free correlation procedure [Ravela, 1996 #26] correlates the template at each position in the image by first rotating the template until the direction of the edge at the center of the template corresponds to the edge direction at the center of the image window. The TAStat procedure is a modification of the algorithm in [Nguyen, 1990 #23]. For every image window it also rotates a mask of the object until it aligns with the local edge data, and then measures the difference between the intensity distributions of the pixels inside and outside of the mask. The greater the difference between the intensity distributions, the more likely the mask matches an object at that location and orientation in the image. Finally, the probing procedure also uses edge information to rotate the template for each image window, and then samples pairs of pixels in the image window, looking for edges that match the location of edges in the template.

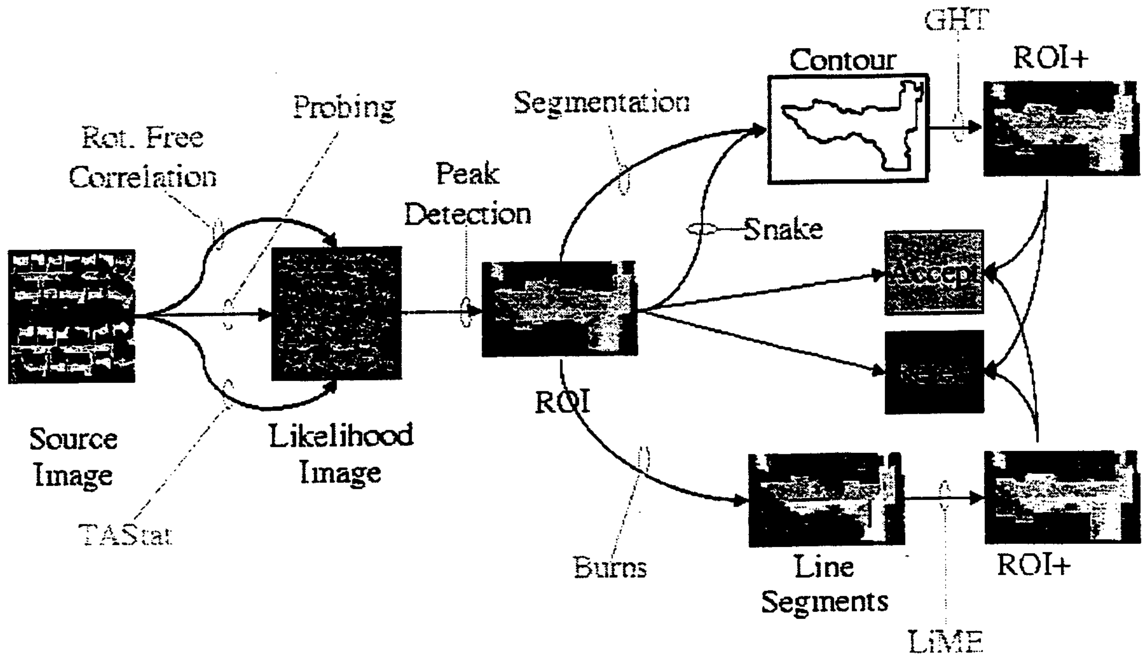


Figure 6: A visual depiction of ADORE's current vision procedure library. Note that the peak detection procedure produces on the order of twenty ROIs each time it is caled.

Regions of interest (ROIs) are chips from the original image that are hypothesized to correspond to object instances; each ROI also has a mask that details the hypothesized position and orientation of the object. ROIs can be extracted from likelihood images using a peak detection procedure, which finds the top N peaks in a likelihood image above a threshold. For these experiments, the peak detection procedure was parameterized to extract a maximum of twenty peaks from each likelihood image.

Five procedures can be applied to any ROI. Two of these actions are the special actions mentioned in Section 4.1, accept and reject. Accept and reject are terminal actions that end the recognition process. The other three options are: 1) an active contour procedure [Kass, 1988 #34] that modifies the outline of an ROI mask until the contour lies along edges in the original image; 2) a segmentation procedure [Comaniciu, 1997 #10] that extracts the boundary of a new region

[1] In all of our experiments, we assume that a template of the object is available.

(as a 2D contour) within the image chip; or 3) a straight line extraction procedure [Burns, 1986 #7].

A Generalized Hough Transform procedure [Ballard, 1981 #5] matches 2D image contours to the contour of a template, thus creating new a ROI. A symbolic line matching procedure (LiME; [Beveridge, 1997 #6]) finds the rotation, translation and scale that maps template (model) lines onto image lines, again producing an ROI. It should be noted that LiME transforms hypotheses in scale as well as rotation and translation, which puts it at a disadvantage in this fixed-scale domain.

5.2) Finding Duplexes

To test the system's ability to learn duplex recognition strategies, we performed N-fold cross-validation on the set of eight Fort Hood images. In other words, we divided the data into seven training images and one test image, trained ADORE on seven training images, and then evaluated the resulting strategy on the test image. We repeated this process eight times, each time using a different image as the test image. All the results presented this paper are from evaluations of the test image.

Figure 7 shows the results of two tests, with the ROIs extracted by ADORE overlaid in red on top of the test image. As a crude measure of success, ADORE found 21 out of 22 duplexes, while producing 6 false positives. The only duplex not found by ADORE can be seen in the image on the right of Figure 7– it is the duplex that is half off the bottom right-hand corner of the image! Every duplex that lies completely inside an image was recognized. (The right side of Figure 7 also shows one false positive.)

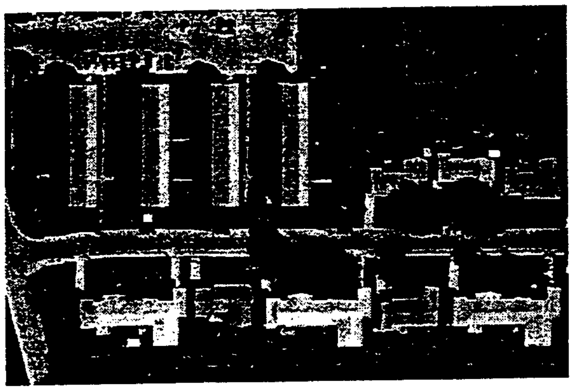
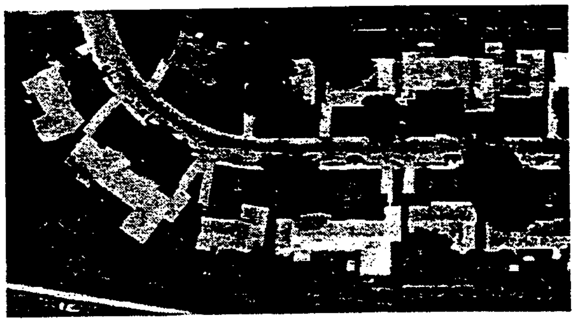



Figure 7: Duplexes extracted from two images. In the image on the left, all three duplexes were found. On the right image, a false positive appears on the upper right side. Also, the half-visible duplex to the bottom right was the only instance missed during testing – and only half of that duplex lies inside the image.

Analyzing ADORE in terms of false positives and false negatives is misleading, however. Much of the benefit of ADORE's dynamic strategies lies in its ability to refine imperfect hypotheses, not just make yes/no decisions. ADORE maximizes its reward function by creating the best hypotheses possible, given the procedure library. Table 1 gives a quantitative measure of ADORE's success. The left most entry in Table 1 gives the average reward across all 22 positive duplex instances from the optimal strategy, where the optimal strategy is determined by trying all sequences of procedures and taking the best result. The second entry gives the average reward generated by the strategy learned by ADORE. As further points of comparison: the third entry in Table 1 gives the average reward for duplex instances if no further processing is applied after hypotheses are generated; the fourth entry gives the average reward if every duplex ROI is segmented and the Generalized Hough Transform is used to reposition the ROI in terms of the region boundary; the fifth entry gives the average reward if the active contour procedure is applied (followed by the Generalized Hough Transform); and the sixth entry is the average reward if lines are extracted from hypotheses, followed by the LiME symbolic matcher.

	Optimal Policy	ADORE Policy	Fixed: Accept or Reject	Fixed: Segment	Fixed: Active Contours	Fixed: Line Extraction
Avg. Reward	0.8991	0.8803	0.7893	0.8653	0.7775	0.1807

Table 1: Comparison between the optimal policy, the policy learned by ADORE, and the four best fixed policies. The first row considers only positive instances; the second includes a penalty of –1 for each false positive. Note that there are 24 object instances.

Two conclusions can be drawn from Table 1. First, the strategy learned by ADORE for this (admittedly simple) task is within about 98% of optimal. Second, the dynamic strategy learned by ADORE, although not perfect, is better than any fixed sequence of actions. (The best fixed sequence of actions is to always segment ROIs and apply the Generalized Hough Transform.) This implies that dynamic strategies are worth learning.

Finding Smaller Houses

Having succeeded in finding a good strategy for finding duplexes, we expected to easily repeat the process for other styles of houses. Indeed, we expected the resulting policies to be very similar to the duplex policy. We therefore repeated the experiment using the same methodology as above but with the house styles shown in Figure 8.

To our dismay, ADORE identified 18 of 19 instances of the house style *A* but generated 22 false positives. Combining the results from house styles A through D, ADORE found 47 out of 61 instances, while generating 85 false positives.

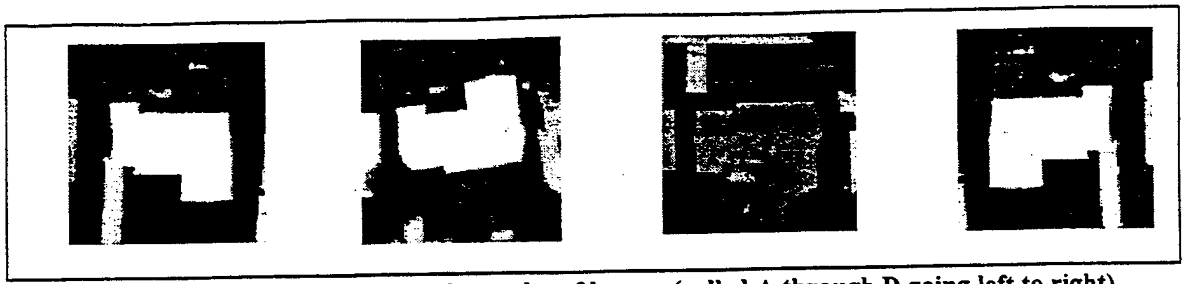


Figure 8: Templates of four other styles of houses (called A through D going left to right).

Why is this problem harder than finding duplexes? The most critical control decision in this domain comes after ROIs are extracted from likelihood images. At this point, ADORE has a choice of five procedures: segmentation, active contours, line extraction, accept or reject. Of these five, line extraction turns out to never be the optimal choice, but the four other options are optimal for some ROIs.

Therefore the control policy must select the optimal action based on features of the ROIs. By inspecting the weights of the neural net Q-functions trained for duplex recognition, we discovered that two of the eleven features dominated the control decision. The most important feature measures the average image edge strength along the boundary of the ROI. The second most important feature measures the percent of pixels outside the mask of the ROI that have roughly the same intensity value as pixels under the mask. (We informally refer to these as "missing" pixels, since their intensities suggest that they were accidentally left out of the hypothesis.)

Based mostly on these two features, ADORE learns a strategy for picking the next procedure. If we interpret the behavior of the Q-functions in terms of these two features, we can describe the control strategy learned by ADORE as follows (see Figure 9): ROIs with very high boundary edge strength and very few "missing" pixels should be accepted as is. (The points in Figure 9 correspond to training samples, color coded in terms of the optimal action for each ROI. Dark blue points correspond to ROIs that receive approximately the same reward whether segmented or accepted as is.) If the edge strength is less high but relatively few pixels are "missing", then the ROI should be segmented. Although many false hypotheses – shown in yellow in Figure 9 because the optimal action is to reject tem – are segmented according to this rule, there is another control decision after segmentation and the Generalized Hough Transform, where false hypotheses can be rejected and true ones accepted. This decision is made easier by the result of the segmentation procedure. There is also one small spot in the feature space where the active contour procedure is optimal. This is harder to explain and may result from the training set being a little too small. Finally, if the missing pixel count is high or the edge strength is low, reject the ROI. The solid boundaries in Figure 9 approximate our interpretation of the control policy's decision boundaries.

If we look at the edge strength feature for the other house recognition task, however, we find that it does not discriminate as well. It turns out that if you overlay the four templates shown in Figure 8 on top of each other, most of the boundaries are aligned. As a result, if an ROI for style A is incorrectly placed over an instance of styles B, C or D, the edges strength is still very high. (The same is true for ROIs of style B, C and D). As a result, when looking for one style of house, false hypotheses created by instances of the other style have high edge strength, so the feature is not very discriminating. As a result, ADORE has a harder time learning a strategy that can distinguish between true instances and false ones, with the result that many more false hypotheses are created. In effect, the difference in feature space between one style and the next is too small to reliably distinguish between them.

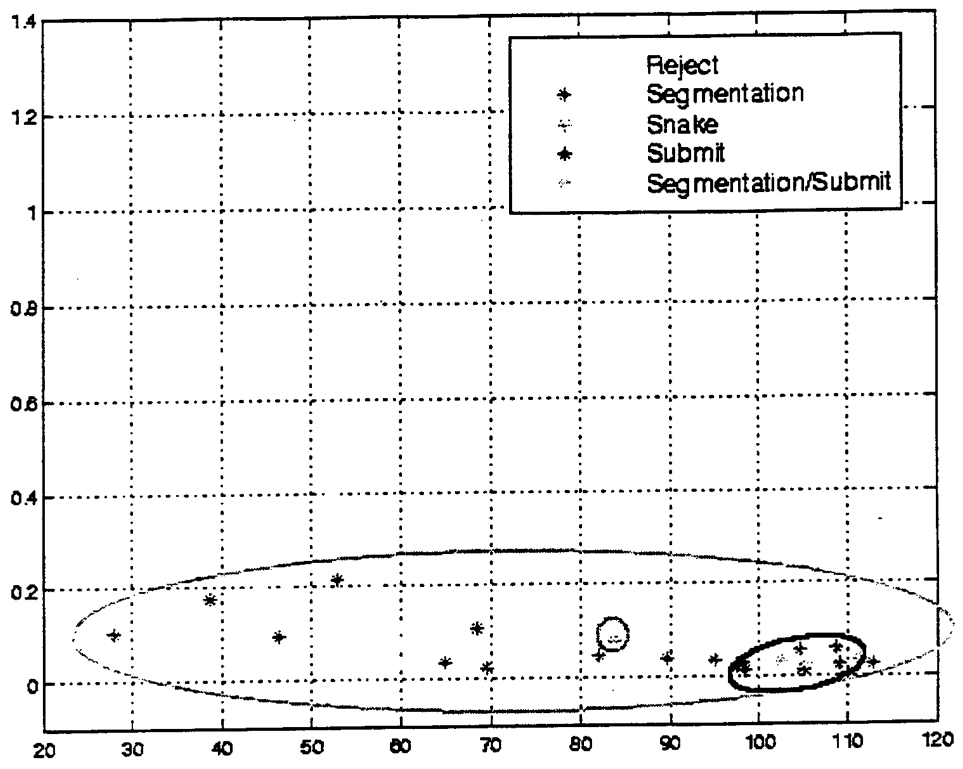


Figure 9: ROIs plotted in two dimensions of the eleven dimensional feature space. The color of the ROI point indicates the optimal control decision for that ROI. (Blue ROIs receive roughly the same reward whether they are submitted as is or refined through segmentation.) The ellipses correspond to our interpretation of the decision boundaries learned by the neural networks.

6) Conclusion

We have built a prototype adaptive object recognition system that is capable of learning good object-specific recognition strategies if and only if features of the intermediate data representations provide enough information to base intelligent control decisions on. Using this prototype, we have successfully learned a dynamic control policy that outperforms any fixed strategy for one recognition task. On the other hand, we were unable to learn a recognition strategy for another task when the target object class and background objects were too similar to be distinguished using ADORE's current features.

7) Bibliography

(see Final Report)