REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Service, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT DATE	3. DATES COVERED *(From - To)*
06-13-2003	Final	20 September 2000 - 14 March 2003

4. TITLE AND SUBTITLE
Topics in Evolutionary Computation

5a. CONTRACT NUMBER

5b. GRANT NUMBER
N00173-00-1-G013

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
John J. Grefenstette

5d. PROJECT NUMBER
02-C176-02

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
George Mason University
4400 University Drive
Fairfax, VA 22030-4444

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
NRL Code 5510
Naval Research Laboratory
4555 Overlook Avenue, S.W.
Washington, DC 20375-5320

10. SPONSOR/MONITOR'S ACRONYM(S)
NRL Code 5510

11. SPONSORING/MONITORING AGENCY REPORT NUMBER

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release

13. SUPPLEMENTARY NOTES

20030916 040

14. ABSTRACT

Autonomous robotic systems are expected to play a significant role in a wide range of areas including surveillance, deep space and undersea exploration and construction, urban search and recovery, mining, and hazardous waste cleanup. Systems that need to operate for extended periods of time out of range of human control should be adaptable to changing or unexpected conditions. This work examines some possible designs for such adaptive autonomous robotics systems, focusing on the adaptation to component failures in autonomous mobile robots. Adaptation is defined as the ability to continue to perform a task, perhaps at a degraded level, despite the loss of some of the robot's original sensor and effector capabilities. The project addresses the problem of adaptation through an approach called Continuous Embedded Learning. Simulation and experimental results are reported.

15. SUBJECT TERMS
Adaptive systems, evolutionary algorithms, mobile robots

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPONE NUMBER *(Include area code)*
U	U	U	UU	27	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI-Std Z39-18

Topics in Evolutionary Computation: Final Report for Grant No. N00173-00-1-G013

PR Number: 02-C176-02,
Disbursing Code: N68892, AGO Code: N66020, CAGE Code: 7X764

PI: John Grefenstette, Ph.D.
School for Computational Sciences
George Mason University
MSN 4E3
10900 University Drive
Manassas, Virginia 20110

June 13, 2003

Abstract

Autonomous robotic systems are expected to play a significant role in a wide range of areas including surveillance, deep space and undersea exploration and construction, urban search and recovery, mining, and hazardous waste cleanup. Systems that need to operate for extended periods of time out of range of human control should be adaptable to changing or unexpected conditions. This work examines some possible designs for such adaptive autonomous robotics systems, focusing on the adaptation to component failures in autonomous mobile robots. Adaptation is defined as the ability to continue to perform a task, perhaps at a degraded level, despite the loss of some of the robot's original sensor and effector capabilities. The project addresses the problem of adaptation through an approach called Continuous Embedded Learning. Simulation and experimental results are reported.

1 Executive Summary

This is the Final Technical Report for Grant No. N00173-00-1-G013, *Topics in Evolutionary Computation*, performed by John Grefenstette at George Mason University in response to NRL BAA 005. The general research topic addressed by this project concerned the applicability of genetic algorithms to robotic learning. The specific research objectives were as follows:

- To improve the understanding of evolution as a process that produces robust organisms that are well-adapted to complex environments;
- To improve methods for designing autonomous systems that exhibit robust and adaptive behavior.

This project contributed to ongoing work at NRL on new principles for the development of intelligent, mobile robots performing complex tasks in unpredictable environments. In the behavior-based approach to robot design, the overall performance of the robot arises through the interaction of multiple, relatively simple, behaviors. The manual design of multiple interacting behaviors is difficult, labor-intensive and error-prone. One way to reduce the effort in the design of behavior-based robots is to develop an evolutionary approach in which the various behaviors, as well as their modes of interaction, evolve over time. Evolution may also provide a basis for the development of strategies for multiple-robot environments, for example, environments in which a robot is expected to adapt its behavior based on the current behavior of other agents or environmental conditions which themselves are changing over time.

This work was performed in collaboration with the Adaptive Systems Group (Code 5514) at the Navy Center for Applied Research in Artificial Intelligence at NRL. Principles of co-evolutionary design were investigated in the context of co-evolving competitive and cooperative behaviors in mobile robots.

2 Background

Autonomous robotic systems are expected to play a significant role in a wide range of areas including surveillance, deep space and undersea exploration and construction, urban search and recovery, mining, warehouse operations, hazardous waste cleanup, and maintenance of inaccessible areas such as fuel storage tanks and sewer systems. Systems that need to operate for extended periods of time out of the range of human maintenance, for example, in deep space, should be adaptable to changing or unexpected conditions. This work examines some possible designs for such adaptive autonomous robotics systems.

This work focuses on the adaptation to component failures in autonomous mobile robots. In this context, adaptation is defined as the ability to continue to perform a task, perhaps at a degraded level, despite the loss of some of the robot's original sensor and effector capabilities. It is common for designers of autonomous systems to consider various failure modes. In some cases, it may be decided to address certain failure modes through redundant or backup components. In other cases, it may be possible to design software to adapt to partial system failures. However, it is generally infeasible to plan for all possible combinations of failure modes at system design time. This project tries to address this limitation through an approach called *Continuous Embedded Learning* (CEL).

Continuous and Embedded Learning is a general approach to designing agents that continuously learn in a changing environment. The agent's learning module continuously tests new strategies against a simulation model of the task environment, and dynamically updates the knowledge base used by the agent on the basis of the results. The execution module controls the agent's interaction with the environment, and includes a monitor that can dynamically modify the simulation model based on its observations of the environment. When the simulation model is modified, the learning process continues on the modified model. The learning system is assumed to operate indefinitely, and the execution system uses the results of learning as they become available.

This report describes simulation studies illustrating the CEL approach applied to autonomous mobile robots. The approach is evaluated by comparing the robot's adaptation ability under a variety of learning regimes. The results suggest that an autonomous mobile robot using CEL can successfully learn to adapt to unanticipated combinations of failures in both sensors and effectors.

In a previous project, we examined the CEL model in the face of sensor failures, specifically, how a robot can learn to adapt to failures in its sensor capabilities over time. We showed that a robot can adapt to the partial loss of its sensors and learn to use different sensors to continue to perform a door traversal task. Both simulation studies and experiments on an actual mobile robot showed that the approach yields effective adaptation to a variety of partial sensor failures. The robot used in these experiments was a Nomadic Technologies Nomad 200 mobile robot, a three-wheeled, synchronized-steering vehicle used in experimental studies at NRL. The internal simulation used by the robot for learning approximated the Nomad robot's sensors and effectors. In simulation studies, the robot initially learned to improve its performance of the task from 25% to 63% with all seven sonar sensors operating. Upon the failure of three sonars (front, front right, and right), performance initially dropped to 37%, but then rebounded to over 60% as the monitor identified the failed sonars, modified the simulation, and the learning system adapted to the new simulation model. These results were verified by repeating the same rules on the Nomad robot, both with and without

sensor disabled. These results indicated that the Continuous and Embedded Learning model is a promising approach to adapting to partial sensor failures. Combined with other previous work showing adaptation to changing environments and actuator failures [1, 4, 3], this work indicated the generality of the CEL model for the design of robust autonomous robot systems. These results were presented at the 2000 SPIE Symposium on Unmanned Ground Vehicle Technology. [8]

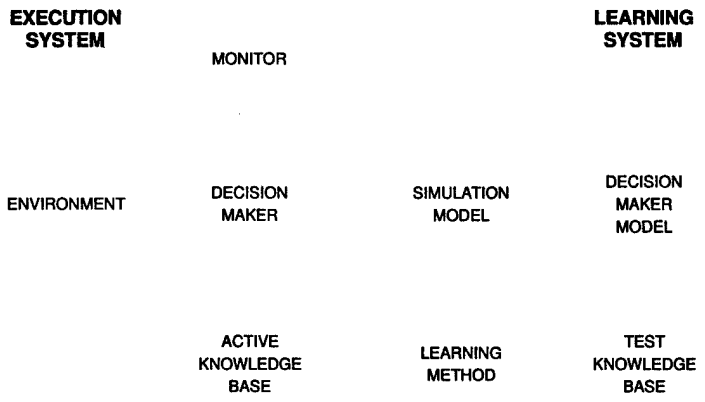


Figure 1: The Continuous and Embedded Learning Model

3 Methods

3.1 Continuous and Embedded Learning

The Continuous and Embedded Learning model addresses the problem of adapting a robot's behavior in response to changes in its operating environment and its capabilities. The outline of the approach is shown in Figure 1. There are two main modules in the CEL model. The *execution module* controls the robot's interaction with its environment. The *learning module* continuously tests new strategies for the robot against a simulation model of the environment. When the learning module discovers a new strategy that, based on simulation runs, appears to be likely to improve the robot's performance, it updates the rules used by the execution module. The execution module includes a *monitor* that measures aspects of the operational environment and the robot's own capabilities, and dynamically modifies the robot's internal simulation model based on these observations. When the monitor modifies the simulation because of an environmental change, it notifies the learning system to restart its learning process on the new simulation.

This general architecture may be implemented using a wide variety of execution modules, learning methods, and monitors. The key characteristics of the approach are:

- Learning continues indefinitely. This is unlike most machine learning methods, which employ a training phase, followed by a performance phase in which learning is disabled. This lifetime learning is what allows the system to be adaptive after being fielded.
- The learning system experiments on a simulation model. For most real-world robotic applications, experimenting with the physical robot may be time-consuming or dangerous. Using a simulation models permits the safe use of learning methods that consider strategies that may occasionally fail.
- The simulation model is updated to reflect changes in the real robot or environment. This is a secondary type of learning of the model. Making such modifications of simulation on

an automatic Al basis is a major research issue in its own right. In this study it is assumed that the simulation model is parameterized such that certain changes in the environment or in the robot itself can be reflected in the model by setting the appropriate parameters. This work does not address the larger issues of automatically making unforeseen changes to the simulation model. Here we assume only that it is possible to monitor the condition of the robot's sensors and actuators. For our purposes, it is not necessary to diagnose the cause of any detected failure, only the symptoms. We assume that the simulation has been constructed to allow the instantiation of sensor and actuator failure modes.

This final point reflects our assumption that the robot designer generally has at least partial knowledge of the robot and the environment. Knowledge that is relatively certain can be embodied in the fixed part of the simulation. Such knowledge might include certain fixed characteristics of the physical environment (e.g., gravity), as well as some aspects of the robot's design and performance. On the other hand, the robot designer should also identify those aspects of the environment and the robot's capabilities that are uncertain, and include these as changeable parts of the simulation module. Additional details on each module in Figure 1 are provided below in connection with the particular case study described in this report.

3.2 Case Study

The project conducted a case study of a robot's ability to adapt to combinations sensor and effector failures, while performing a specified task. The following section describes the performance task.

3.2.1 Performance Task

This task requires a robot to go from one side of a room to the other, passing through an opening in a wall placed across the room, as illustrated in Figure 2.

In each trial, the robot is placed at a random position along the starting line four feet in front of the back wall, facing in a randomly selected direction from -90 to 90 degrees (with 0 degrees facing the goal). The center of the front wall is located 12.5 feet from the back wall. The room is 25 feet wide. The location of the six foot opening in the front wall is also randomly selected each trial. The robot must then reactively navigate through the opening reaching the goal line one foot beyond the wall, by learning a set of rules which map the current sensors to the actions to be performed by the robot, at a one hertz decision rate. The robot has a limited time to perform the task. Exceeding the time limit, or having a collision with any of the walls, ends the current trial. The robot used in this study is a model of the Nomadic Technologies Nomad 200 Mobile Robot, a three-wheeled, synchronized-steering vehicle. The internal simulation used by the robot for learning approximated the robots sensors and effectors.

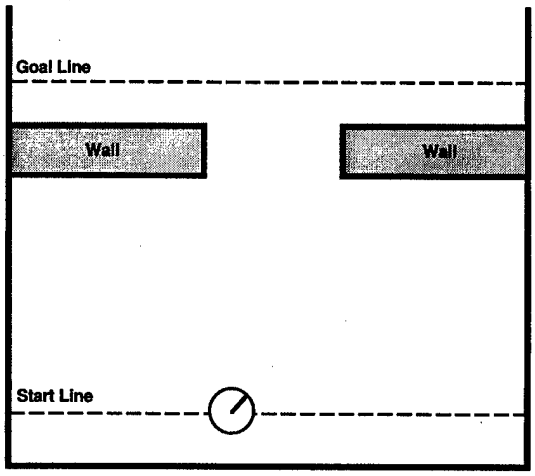


Figure 2: The door traversal task.

3.3 Execution Module

The execution module for the robot includes a rule-based system that operates on reactive (stimulus-response) rules. A typical rule might be:

> IF range = [35, 45] AND *front_sonar* < 20 AND *right_sonar* > 50 THEN SET turn = -24 (Strength 0.8)

Each decision cycle, the execution system compares the left hand side of each rule to the current sensor readings, selecting the best rule (after conflict resolution). That rule's action is then executed causing the robot to move. This is repeated until the robot succeeds or fails at the task.

In this task, the robot uses its seven front most facing sonars. Each sonar has a angular resolution of 22.5 degrees, with the front most sonar facing directly ahead, giving the robot a total sonar coverage of 157.5 degrees. The sonars are designated as *far_left*, *left*, *front_left*, *front*, *front_right*, *right*, and *far_right*. Each sonar has a maximum range of approximately 14 feet. The sonar values are all discretized and are partitioned into intervals of 24 inches. In addition to the sonars, the robot also has a sensor that gives the range to the goal from 0 to 14 feet in 5 inch intervals, and the robots heading within the room from 0 to 359 degrees in 22.5 degree intervals. The learned actions are velocity mode commands for controlling the translational rate and the steering rate of the robot. The translation rate is given as -4 to 10 inches/sec in 2 inch/sec intervals. The steering command is given in intervals of -10 degrees/sec from -30 to 30 degrees/sec. At each decision step, the system must choose a turning rate and steering rate based on the current sensors.

The rule *strength* is set by the learning system to estimate the quality of the rule. The execution module uses rule strengths to resolve conflicts among multiple rules that match the current sensors readings, but suggest different actions. In such cases, rules with higher strength are favored. See [2] for details.

3.3.1 Monitor

In this study, the monitor is assumed to periodically measures the output from the sonars, and compare them to recent readings and to the direction of motion. If the robot is moving forward, and the value of the sonar reads zero repeatedly, that particular sonar is marked as being defective. The monitor is also assumed to measure periodically the robot's maximum turn rate to each side, and the robot's range of speeds. These readings are stored in a *capability vector* which indicates the current status of the robot's sensors and effectors. When the capability vector changes, the monitor then modifies the simulation used by the learning system to correspond to current capabilities. It is important to note that the monitor is required only to identify symptoms of problems, not the causes.

3.4 Learning Module

The learning module uses SAMUEL[2], a learning program that uses genetic algorithms and other competition-based heuristics to improve its decision-making rules. Each individual in SAMUEL's genetic algorithm is an entire rule set, or strategy, for the robot. We have previously reported on using SAMUEL to learn simple robot behaviors such as navigation and collision avoidance [6, 5], robot herding [7], and in other complex domains.

3.4.1 Simulation Model

The simulation model used by the learning system includes a model of the robot's sensors and effectors, as well as a model of the operating environment. The code for the simulation model is in file *env.c*. The simulation model includes the possibility that various sensors and effectors may be faulty. Faulty sonars are modeled as returning a minimum-value reading. Effector faults are modeled by using the SAMUEL *constraint* mechanism, which overrides the actions selected by the rule-based system if the action violates a constraint. That is, if a rule issues a command to move forward at 30 inches per second (ips), but the constraints are set to a maximum of 10 ips, the the effect of the rule's action is limited to 10 ips.

3.4.2 Case Base

In previous studies, the case base was designed as a small set of plans associated with the learned behavior for each case. In this study, the case base was expanded to store the entire final population of the SAMUEL learning system for each case. This change enables SAMUEL to continue from precisely where it left off, should an identical case occur in the future.

3.4.3 Re-initialization Rules

The CEL method incorporates a *case-based approach* to re-initializing the population, as first proposed in [4]. The learning system re-initializes the population of strategies in the genetic algorithm by finding nearest neighbors from the case base consisting of previously learned strategies. Strategies in the case base are indexed by the capability list in place at the time the strategy was learned. Using previously learned strategies to initialize the population allows the system to very quickly adapt to situations that are similar to those seen before. Re-initialization is performed as follows (see the function `offline_update_init()` in file `offline.c` for details):

1. Compute the distance between the current case (as described by the monitor) and all cases in the case base.
2. Find the k nearest neighbors.
3. For each of the k nearest neighbors, allocate a portion of the new population proportional to the similarity between the current case and the neighboring case.
4. Copy the selected number of plans from the population associated with the previous case to the new population. Plans are selected in decreasing order of fitness.

In the first step above, the distance between two cases is defined by first computing the distance along each dimension of the capability vector. In the current study, the capability vector consists of a triple

$$C_i = \{turn,\ speed,\ sensors\}$$

where *turn* represents the range of turning capability, *speed* represents the range of speeds achievable, and *sonar* represents the vector of currently functional sonars. The *distance* in a given capability between two cases is defined as 1.0 minus the fraction of the total capability that is shared by each case. The distance between two cases is defined as the Euclidean distance induced by the vector of capability distances. See function `case_distance()` in `offline.c` for complete details.

3.5 Experimental Design

The goal of this study was to evaluate the effectiveness of the Continuous Embedded Learning approach in the context of an autonomous mobile robot that may experience failures in both sensor and effector subsystems. The approach here consists of a *lesion study* in which individual components of the CEL approach are systematically disabled in order to assess their contributions to the overall performance of the learning system. The lesion approach resulted in our alternative approaches: *Case-based Anytime Learning (CL)*, *Adaptive Learning (AL)*, *Simple Learning (SL)*, and *No Learning (NL)*, described below.

3.5.1 Case-based Anytime Learning (CL)

The Case-based Anytime Learning (CL) regime implements the full CEL method and includes the ability to store and use a growing case-base of plans learned during the operational lifetime of the robot. When the monitor detects a change in the capability of the robot, it alters the simulation model accordingly. Then the population in the learning system is reinitialized using plans learned during similar, previously observed cases. The CL regime is expected to provide an advantage when combinations of system failures occur that are similar to previously encountered cases, since the learning system will be able to start learning based on previously learned plans.

3.5.2 Adaptive Learning (AL)

The Adaptive Learning (AL) regime is derived from the CL regime by eliminating the use of the case base of previously learned plans. As in CL, the monitor alters the simulation model based upon the environmental changes. Unlike CL, the AL regime reinitializes the population of the SAMUEL learning system using a default set of initial rules. In effect, the AL regime corresponds to a continuation of the training regime over the operational lifetime of the robot.

3.5.3 Simple Learning (SL)

The Simple Learning (SL) regime is derived from the AL regime by eliminating the alteration of the simulation model used for learning during the operational lifetime of the robot. SL provides a base level of learning by continuing to run the SAMUEL learning system throughout the operational life of the robot. The SL regime is not expected to adapt very well to the changing operational environment, but it does provide data for accounting for the effect of continuous learning.

3.5.4 No Learning (NL)

The baseline approach to environmental changes is provided by the No Learning (NL) regime, which continues to use the best plan learned during the training regime for the no-fault scenario. That is, NL does not learn and ignores any indication of changes in the operational environment.

3.5.5 Training

In order to provide an initial basis for the CL method, the robot was subjected to a training regime consisting of selected failure modes. This training regime is meant to model the ordinary process of testing autonomous systems under various conditions before being placed in its operational environment, perhaps with a bias toward the most predictable kinds of systems failures. In this study, 15 training cases were defined, each case corresponding to a particular sensor of effector failure. SAMUEL was executed for 25 generations on each of training case, starting from a fixed

Case	left_turn	right_turn	min_speed	max_speed	sonar	description
0	-30	30	-4	10	0	*No Faults*
						Single Sensor Failures
1	-30	30	-4	10	1	one sonar blocked
2	-30	30	-4	10	2	one sonar blocked
3	-30	30	-4	10	4	one sonar blocked
4	-30	30	-4	10	8	one sonar blocked
5	-30	30	-4	10	16	one sonar blocked
6	-30	30	-4	10	32	one sonar blocked
7	-30	30	-4	10	64	one sonar blocked
						Turn Faults
8	-30	10	-4	10	0	limited right turn
9	-10	30	-4	10	0	limited left turn
10	-10	10	-4	10	0	limited turn
						Translation Faults
11	-30	30	0	10	0	no backing up
12	-30	30	-4	4	0	slow speed
13	-30	30	4	4	0	fixed slow speed
14	-30	30	10	10	0	fixed fast speed

Table 1: Fault scenarios used as training cases

initial set of rules. The final population for each training cases was stored in the case-base used by the CL regime.

The training cases used in the case study are shown in Table 1. The training cases were selected from four categories: (1) No Faults; (2) Sensor Faults; (3) Turn Faults and (4) Speed Faults.A sensor failure meant that a given sonar was disabled (turned off). An effector failure meant that a limitation was applied to either the range of speeds allowed (e.g., the robot could not move with a velocity greater than 10 ips) or the range of turning rates (e.g., the robot could not turn right).

As the table shows, fault cases are described by five parameters:

- *left_turn:* indicates the maximum turn rate to the left (between -30 and 0). The robots used in this study were set to have a maximum turn rate of 30 degrees per second, so a left_turn capability of -30 corresponds to a maximum left turn capability.
- *right_turn:* indicates the maximum turn rate to the right (between 0 and 30).
- *min_speed:* indicates the minimum speed of the robot (between -4 and 10 inches per sec).
- *max_speed:* indicates the maximum speed of the robot (up to 10 ips).
- *blocked_sonar:* indicates the current set of disabled sonars. Each of the seven front facing sonars may be individually disabled by setting one of seven bits in the blocked sonar field to 1. That is, a value of 0 indicates that all sonars are operating, a value of 1 indicates that the left-most sonar is blocked, a value of 2 indicates the next sonar is blocked, a value of 3 indicates that the 2 left most sonars are blocked, and so on. Blocked sonars return a constant minimum return reading.

For each training case, the SAMUEL learning was run for 25 generations on the given case, starting with a fixed initial rule set. The final populations produced during the 15 training cases were stored in the initial case base used by the CL learning regime. The AL regime reinitializes its population of plans with the same initial population used in the training phase. The SL regime never reinitializes its population, but starts with the final population produced in training case 0 (No Faults).

3.5.6 Testing

Unfortunately, it is not feasible to conduct tests of adaptation by actually running a robot in its operational environment until some fault occurs. Such an approach would not be likely to produce repeatable results, and may provide a very sparse coverage of interesting fault scenarios. In order to provide for a more systematic comparison, we defined a simulation-based *operational lifetime* for the robot consisting of a set of fault scenarios, or *test cases*. Each test case consists of a given combination of sensor and effector failures. The identical operational lifetime (i.e., set of test cases) was repeated multiple times for each learning regime, thus allowing comparative analysis of different adaptation methods.

The 20 test cases used in the study are shown in Table 2. The test cases consist of one case with no faults, 3 cases with single faults (repeating cases with had appeared in the training phase), 14 cases with double faults, and 2 cases with triple faults. This set of cases was selected to represent both cases that had been seen previously during training, as well as combinations of system faults. This set reflects the expectation that in practice, it is usually an unanticipated combination of system faults that is likely to lead to trouble for an autonomous system.

The next section describes the results of the simulation study.

Case	left_turn	right_turn	min_speed	max_speed	sonar	description
0	-30	30	-4	10	0	*No Faults*
						Single Failures
1	-30	30	-4	10	8	single sonar blocked
2	-10	10	-4	10	0	limited turn
3	-30	30	10	10	0	fixed fast speed
						Double Failures
4	-30	10	-4	10	8	limited right turn with sensor failure
5	-30	10	-4	10	64	limited right turn with sensor failure
6	-10	30	-4	10	1	limited left turn with sensor failure
7	-10	30	-4	10	8	limited left turn with sensor failure
8	-10	10	-4	10	2	limited turn with sensor failure
9	-10	10	-4	10	8	limited turn with sensor failure
10	-10	10	-4	10	32	limited turn with sensor failure
11	-30	30	4	4	4	fixed slow speed with sensor failure
12	-30	30	4	4	16	fixed slow speed with sensor failure
13	-30	30	10	10	1	fixed fast speed with sensor failure
14	-30	30	10	10	64	fixed fast speed with sensor failure
15	-30	10	10	10	0	limited right turn, fixed fast speed
16	-10	30	4	4	0	limited left turn, fixed slow speed
17	-10	10	10	10	0	limited turn, fixed fast speed
						Triple Failures
18	-10	10	-4	10	65	limited turn with two sensor failures
10	-30	30	-4	10	73	triple sensor failures

Table 2: Test cases in robot's operational lifetime.

4 Results

4.1 Training Phase

For each training case, SAMUEL was run for 25 generations starting with a default rule set. The results during the training phase are shown in the following figures. In most cases, SAMUEL is able to learn a high performance plan that successfully adapts to the given fault scenario, generally achieving a success rate of over 90% within the 25 generations. The more difficult cases include Case 4 (forward sonar blocked), Cases 8-10 (limited turning rates), and Case 14 (fixed fast speed).

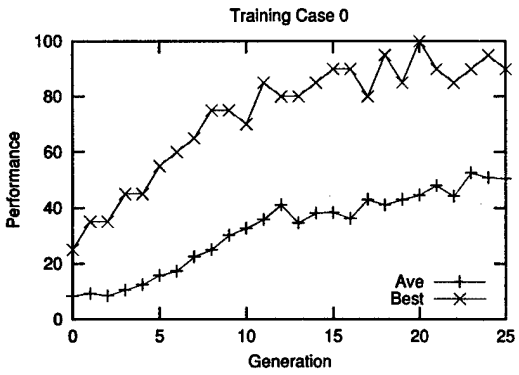
The training results provide a wealth of data concerning the kinds of plans that SAMUEL is able to learn under different conditions. One way to gain some insight into the relationships among the training cases is to evaluate the best plan learned in each training case in all of the other training cases. These results are shown in Table 3. In this table, each row shows the result of using the best plan from one of the training cases on each of the 15 fault scenarios in the training set. The diagonal elements are in bold face. One can infer from this data that SAMUEL learns effective plans for each training case that are specific for the particular fault involved. In every row, the plan performed best when tested on the specific case under which it was evolved. Furthermore, with one exception, the plan learned in a given training case performs better than any plan learned under a different training scenario. (The exception is that the plans learned for the two very similar cases,

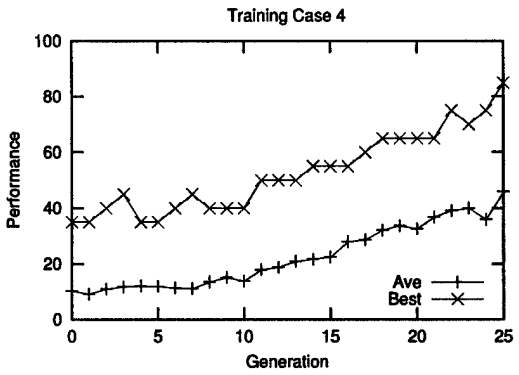
cases 12 and 13, seem to be essentially interchangeable.)

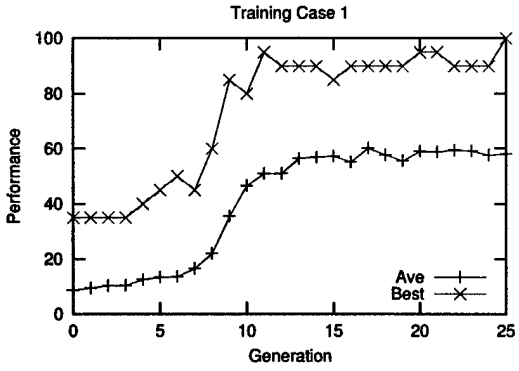
At first, some of these results may seem counterintuitive. For example, why do the rules learned in Case 1 (rightmost sonar blocked) performed more poorly in Case 0 (no faults)? Upon inspection of the rules in action, the answer becomes clear: The most useful rules learned in Case 1 only match perfectly when the rightmost sonar returns a value of 0 (since this was the case during learning). When applied to the no-fault case, the rightmost sonar returns values other than 0, and the matching/conflict-resolution mechanisms in SAMUEL end up selecting rules that were used infrequently in Case 1. That is, the rules learned in any particular case are tuned to the conditions present during learning. This tendency of SAMUEL to learn plans that are tuned to specific fault scenarios is a desirable feature of a learning system, since tends to use all available sensors to select the best available action. However, in a dynamic environment, the tendency to specificity must be coupled with the ability to adapt to changing conditions. The ability to adapt is examined in the next section.

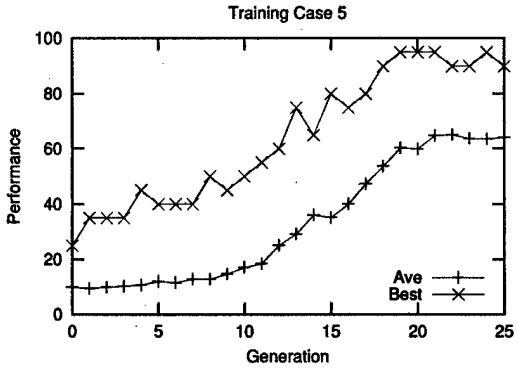
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	**0.90**	0.18	0.22	0.37	0.38	0.42	0.00	0.02	0.00	0.00	0.18	0.82	0.16	0.17	0.48
1	0.52	**0.88**	0.00	0.40	0.40	0.40	0.43	0.40	0.66	0.05	0.04	0.54	0.41	0.39	0.37
2	0.13	0.08	**0.80**	0.08	0.08	0.08	0.08	0.09	0.07	0.16	0.11	0.13	0.23	0.23	0.15
3	0.49	0.00	0.42	**0.79**	0.32	0.32	0.40	0.43	0.02	0.05	0.06	0.49	0.52	0.43	0.30
4	0.34	0.22	0.13	0.04	**0.63**	0.34	0.28	0.19	0.06	0.14	0.19	0.29	0.22	0.30	0.28
5	0.33	0.37	0.38	0.17	0.32	**0.84**	0.39	0.31	0.26	0.30	0.34	0.35	0.32	0.38	0.26
6	0.36	0.13	0.37	0.39	0.37	0.25	**0.79**	0.14	0.33	0.00	0.02	0.42	0.36	0.33	0.22
7	0.51	0.00	0.17	0.20	0.13	0.13	0.13	**0.92**	0.06	0.34	0.12	0.50	0.40	0.50	0.33
8	0.70	0.36	0.00	0.31	0.31	0.30	0.12	0.00	**0.76**	0.41	0.39	0.70	0.00	0.69	0.39
9	0.62	0.09	0.03	0.13	0.10	0.08	0.08	0.01	0.00	**0.60**	0.00	0.62	0.61	0.60	0.38
10	0.49	0.09	0.06	0.06	0.06	0.00	0.04	0.00	0.53	0.41	**0.53**	0.49	0.47	0.48	0.30
11	0.88	0.00	0.80	0.80	0.80	0.83	0.00	0.01	0.00	0.58	0.00	**0.88**	0.00	0.38	0.57
12	0.70	0.16	0.16	0.16	0.17	0.16	0.16	0.16	0.32	0.05	0.13	0.72	**0.78**	0.75	0.57
13	0.76	0.00	0.61	0.59	0.67	0.50	0.67	0.15	0.18	0.05	0.17	0.76	0.79	**0.79**	0.40
14	0.60	0.01	0.02	0.41	0.40	0.00	0.00	0.33	0.21	0.00	0.15	0.66	0.00	0.44	**0.69**

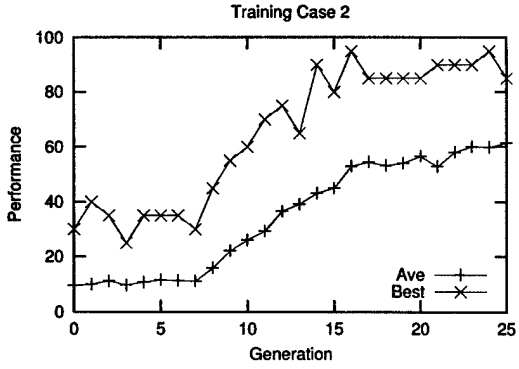
Table 3: Results of exercising best plan learned during each training case on all training cases. Each row represents a fixed plan, and each column shows the performance of that plan on a given training case. Diagonal items show performance of plan on the case for which it was learned.

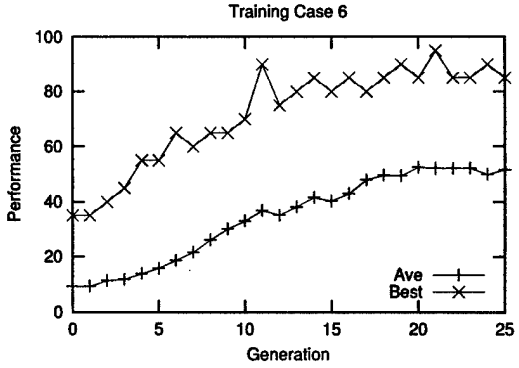

Training Case 0
Performance
Generation
Ave
Best

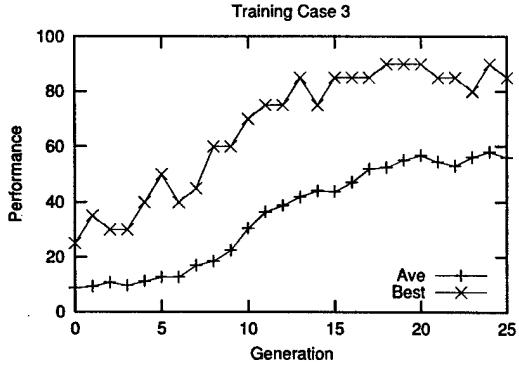

Training Case 4
Performance
Generation
Ave
Best

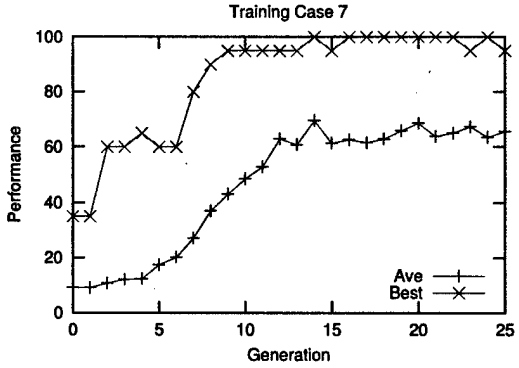

Training Case 1
Performance
Generation
Ave
Best

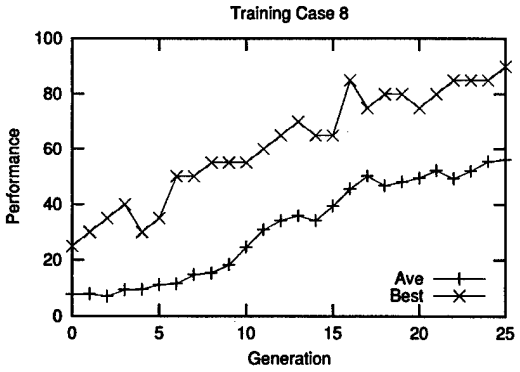

Training Case 5
Performance
Generation
Ave
Best

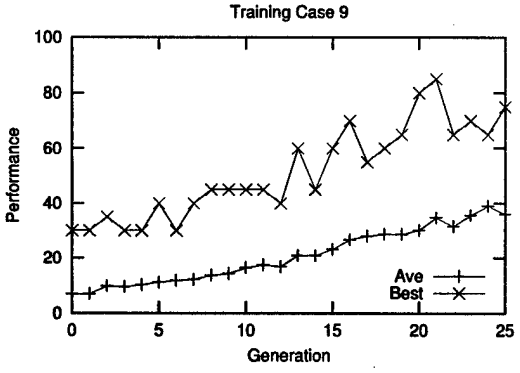

Training Case 2
Performance
Generation
Ave
Best

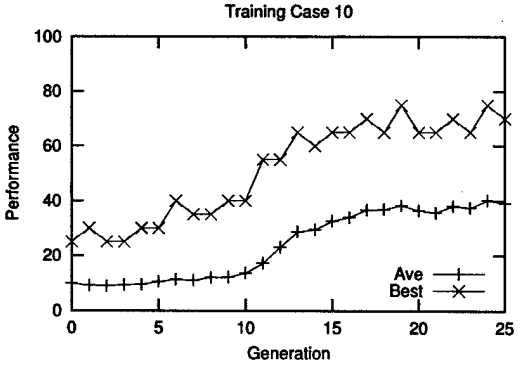

Training Case 6
Performance
Generation
Ave
Best

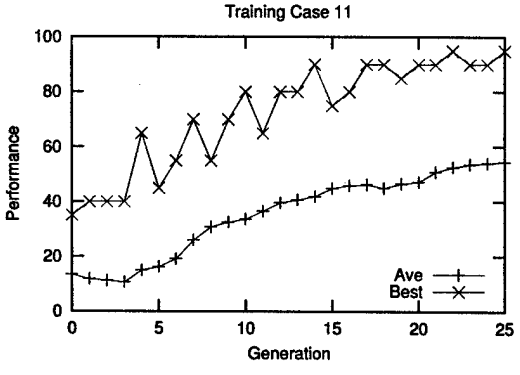

Training Case 3
Performance
Generation
Ave
Best

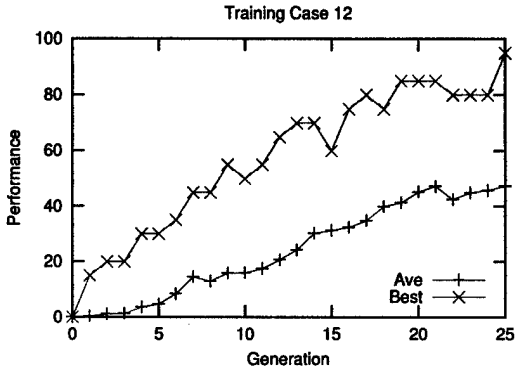

Training Case 7
Performance
Generation
Ave
Best

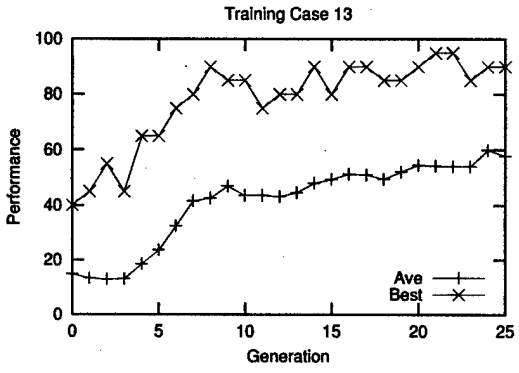

Training Case 8
Performance
Generation
Ave
Best

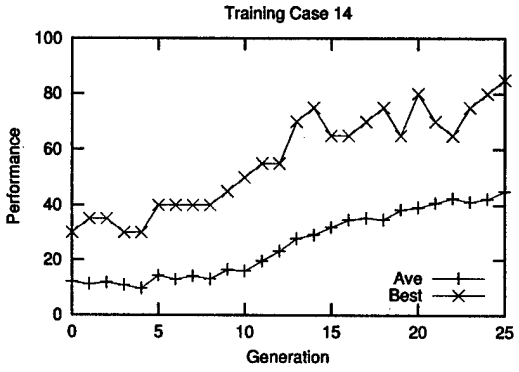

Training Case 9
Performance
Generation
Ave
Best


Training Case 10
Performance
Generation
Ave
Best


Training Case 11
Performance
Generation
Ave
Best


Training Case 12
Performance
Generation
Ave
Best


Training Case 13
Performance
Generation
Ave
Best


Training Case 14
Performance
Generation
Ave
Best

4.2 Comparisons of Learning Methods

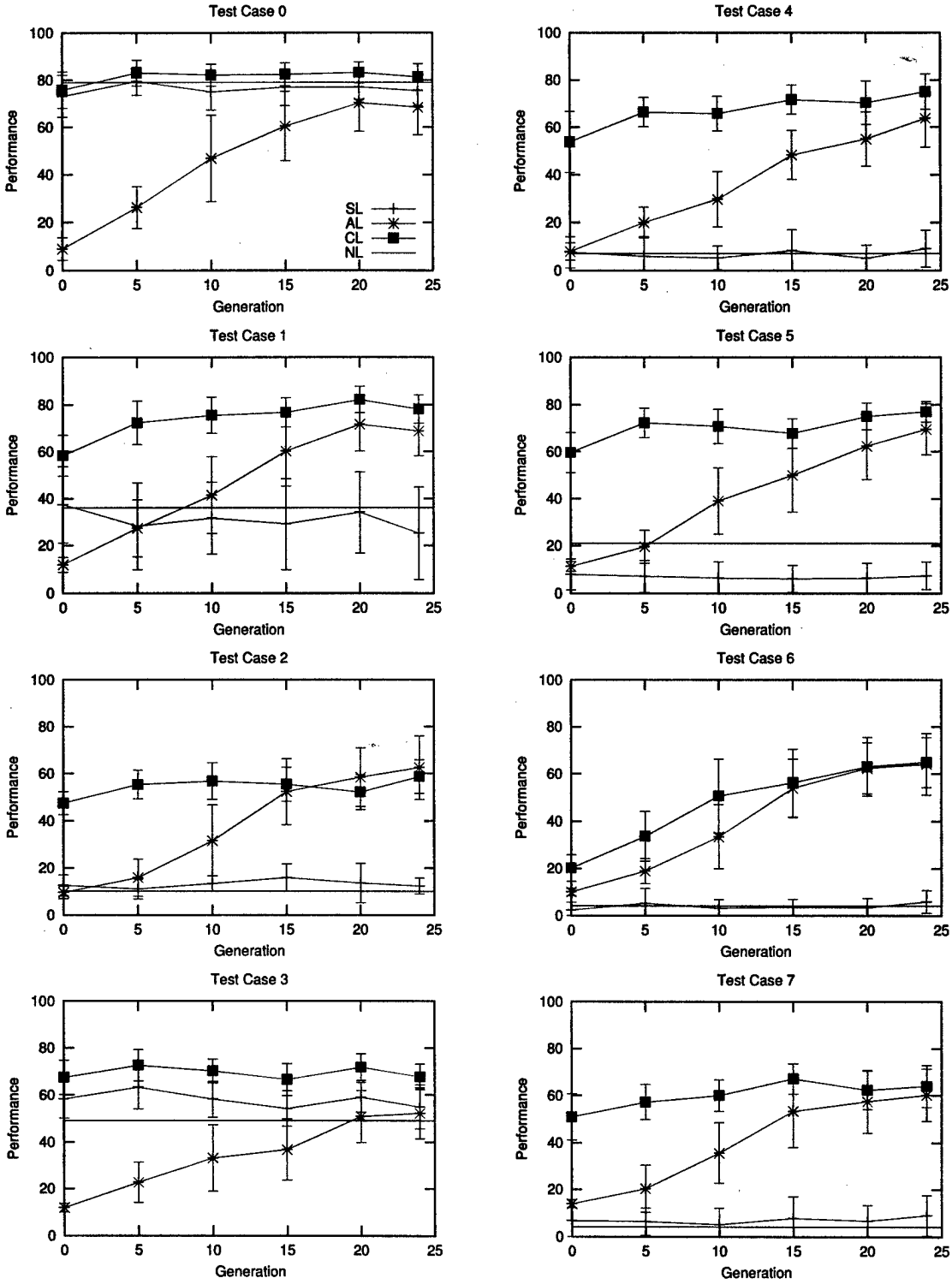
The four regimes NL, SL, AL and CL were compared over a simulated operational lifetime of the robot performing the door traversal task. The operational lifetime spanned the equivalent of 500 generations of the offline evolutionary algorithm.[1] After each generation, the robot updated the active rule set using the best learned rule set, as in Figure 1. (The exception of course was NL, which used the same rule set throughout the operational lifetime.) Every 25 generations of the offline learning system, a new fault scenario was introduced, following the order listed in Table 2. The operational lifetime was repeated 20 times for each regime, and the error bars in the graphs below correspond to one standard deviation across the 20 repetitions.

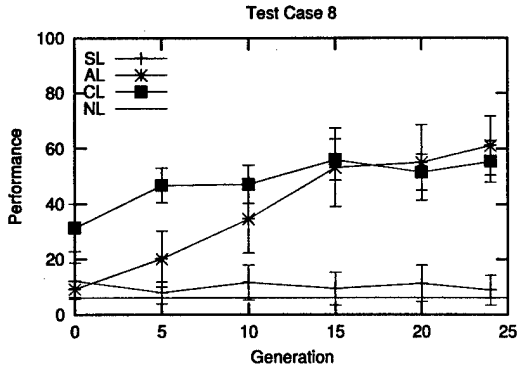
The following figures show the performance of the four regimes NL, SL, AL and CL on the 20 test cases comprising the simulated operational lifetime of the robot. Since the NL regime used the same rule set throughout the operational lifetime, its graph is a horizontal line. As expected, NL performs poorly on all test cases except the Case 0, the no fault scenario on which it was trained. Except for Cases 0 and 3, the SL regime performed similarly to NL, or worse. This result was expected, since SL continues to optimize its strategies for the no-fault case, and does not adapt as the capabilities of the robot change.

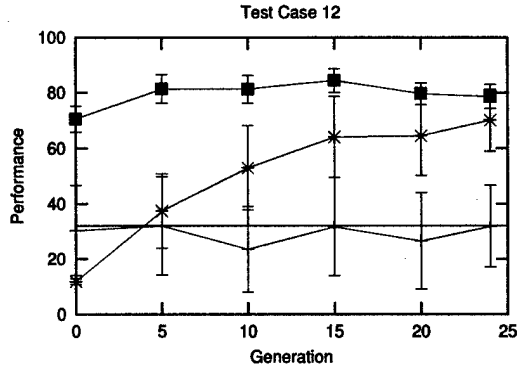
The AL and CL regimes show more interesting results. Recall that the AL regime reinitializes the population and revises the offline simulation model whenever the monitor indicates a change in the capabilities of the robot. As in the more simple training cases, AL is shown to adapt well to nearly every fault scenario in the test cases. By the end of the 25 generations, AL has evolved a rule set that significantly outperforms the baseline NL and SL regimes (except for the no-fault Case 0 and Case 3). These results show the effectiveness of adapting the offline learning simulation model over the operational lifetime of the robot. The CL regime, unlike the AL regime, generally shows significantly improved performance over the baseline throughout the operational lifetime. In most cases, CL shows a significant improvement over AL at the start of each case, and ends up with similar performance to AL after 25 generations. The difference between CL and AL at the beginning of each case shows the advantage gained through the use of the case base of previously stored strategies. The fact the CL shows a significant improvement over the 25 generation for most cases shows that the offline learning is synergistic with the improved starting position provided by the rules in the case base. Furthermore, the results show that CL benefits from previous learning experience even for cases that are much different from previously seen cases. Recall that Cases 4 though 19 represent new situations that did not appear in the training regime. In Case 4 for example, the CL regime reinitializes its population using strategies from three different training cases, which gives its an immediate advantage over AL (54% success vs. 8% success). The offline learning system was then able to adapt these previously learned strategies to the new fault scenario and thereby improve the performance from 54% to 75% within 25 generations.

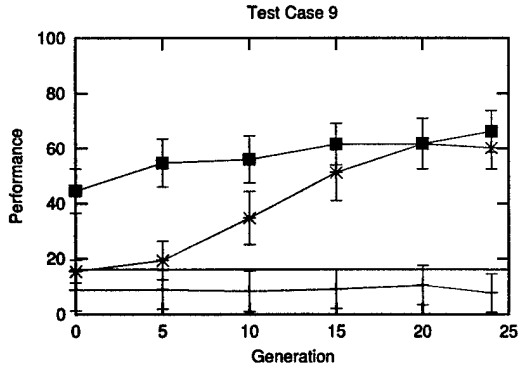
It is also worth noting that both AL and CL were able to adapt to 2 out of 3 of the difficult triple-fault cases (Cases 18 and 19). Once again, the CL regime shows a statistically significant advantage over the AL regime in the initial periods for these cases. Finally, as Case 17 illustrates, some combinations of system faults will lead to poor performance of the task regardless of learning or adaptation regime.

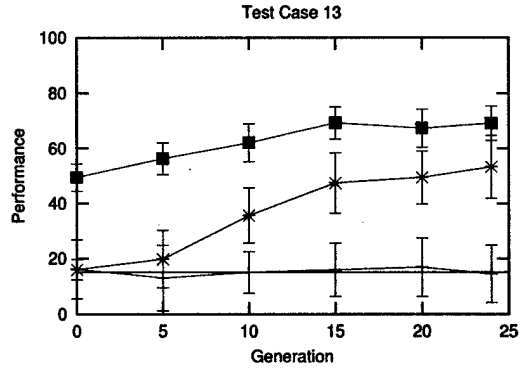
[1] That is, one generation of the offline algorithm was taken as the time unit for the operational lifetime. This assumption was made in the interest of repeatability of the experiment.

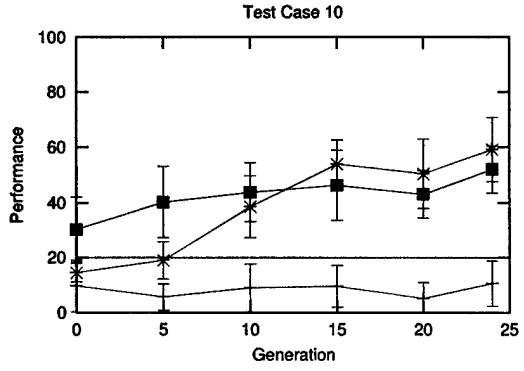

Test Case 0
Test Case 4
Test Case 1
Test Case 5
Test Case 2
Test Case 6
Test Case 3
Test Case 7
Performance
Generation
SL
AL
CL
NL

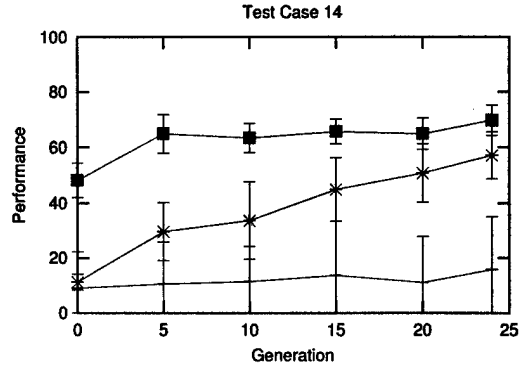

Test Case 8
SL
AL
CL
NL
Performance
Generation
0
20
40
60
80
100
0
5
10
15
20
25

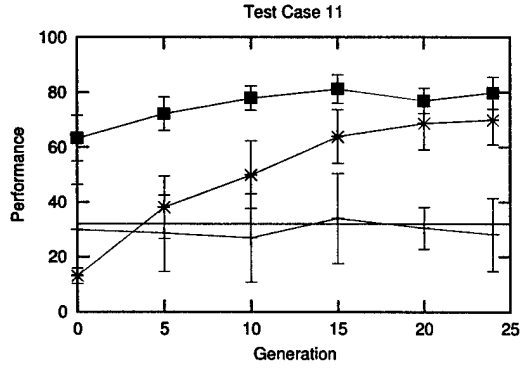

Test Case 12
Performance
Generation
0
20
40
60
80
100
0
5
10
15
20
25

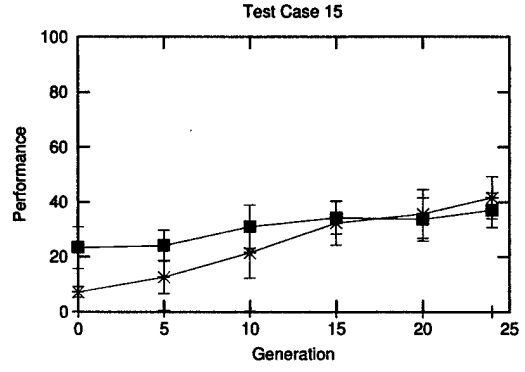

Test Case 9
Performance
Generation
0
20
40
60
80
100
0
5
10
15
20
25

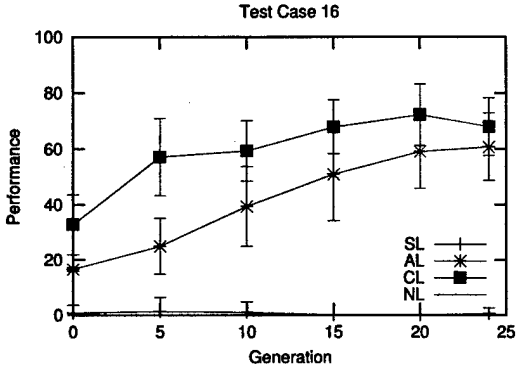

Test Case 13
Performance
Generation
0
20
40
60
80
100
0
5
10
15
20
25

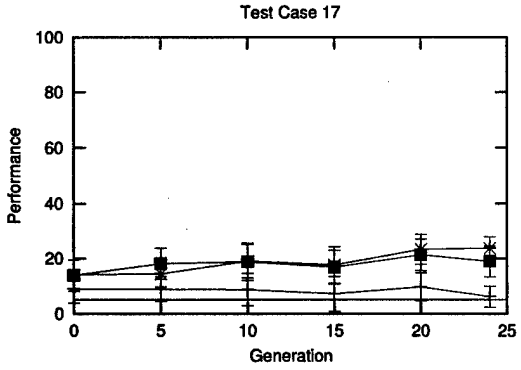

Test Case 10
Performance
Generation
0
20
40
60
80
100
0
5
10
15
20
25

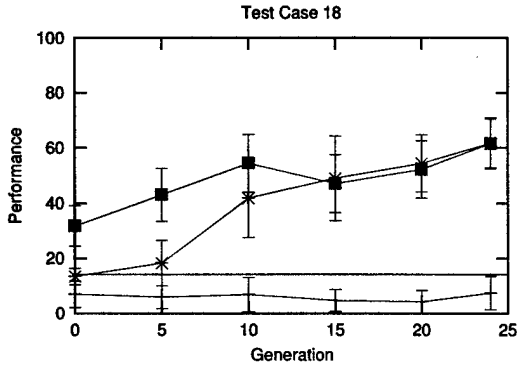

Test Case 14
Performance
Generation
0
20
40
60
80
100
0
5
10
15
20
25

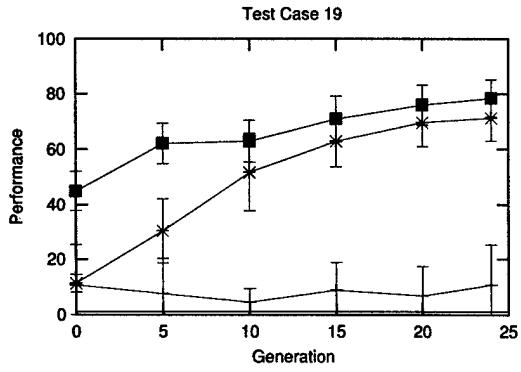

Test Case 11
Performance
Generation
0
20
40
60
80
100
0
5
10
15
20
25


Test Case 15
Performance
Generation
0
20
40
60
80
100
0
5
10
15
20
25


Test Case 16
Performance
Generation
SL
AL
CL
NL


Test Case 17
Performance
Generation


Test Case 18
Performance
Generation


Test Case 19
Performance
Generation

Test Case	NL (stdev)	SL (stdev)	AL (stdev)	CL (stdev)
0	0.79 (0.0)	0.77 (0.07)	0.48 (0.24)	**0.82** (0.06)
1	0.36 (0.0)	0.31 (0.18)	0.48 (0.24)	**0.74** (0.09)
2	0.10 (0.0)	0.15 (0.05)	0.37 (0.22)	**0.54** (0.08)
3	0.49 (0.0)	0.58 (0.08)	0.35 (0.17)	**0.69** (0.06)
4	0.07 (0.0)	0.06 (0.08)	0.39 (0.20)	**0.69** (0.09)
5	0.21 (0.0)	0.06 (0.06)	0.42 (0.24)	**0.70** (0.08)
6	0.04 (0.0)	0.04 (0.04)	0.39 (0.22)	**0.46** (0.17)
7	0.04 (0.0)	0.06 (0.07)	0.40 (0.21)	**0.60** (0.09)
8	0.06 (0.0)	0.10 (0.06)	0.38 (0.21)	**0.47** (0.11)
9	0.16 (0.0)	0.09 (0.07)	0.40 (0.20)	**0.58** (0.11)
10	0.20 (0.0)	0.08 (0.08)	0.40 (0.18)	**0.43** (0.12)
11	0.32 (0.0)	0.31 (0.14)	0.51 (0.21)	**0.77** (0.07)
12	0.32 (0.0)	0.29 (0.15)	0.51 (0.22)	**0.79** (0.06)
13	0.15 (0.0)	0.16 (0.11)	0.37 (0.16)	**0.64** (0.08)
14	0.00 (0.0)	0.11 (0.16)	0.38 (0.17)	**0.63** (0.08)
15	0.00 (0.0)	0.00 (0.00)	0.25 (0.12)	**0.30** (0.07)
16	0.00 (0.0)	0.00 (0.02)	0.40 (0.19)	**0.60** (0.15)
17	0.05 (0.0)	0.09 (0.05)	0.18 (0.07)	**0.18** (0.06)
18	0.14 (0.0)	0.06 (0.05)	0.41 (0.21)	**0.50** (0.12)
19	0.01 (0.0)	0.11 (0.14)	0.50 (0.23)	**0.68** (0.12)

Table 4: Global performance of alternative learning regimes on all test cases. In all cases, the CL regime outperform all others. The AL regime generally performs significantly better than either SL or NL.

On a global level, the most straightforward way to compare the alternative adaptation regimes is by the mean performance over the operational lifetime of the robots. Table 4 lists the mean and standard deviation for the four regimes, over the entire operational lifetime of the robot, and all 20 replications thereof (except for NL, which was executed only once). These results show that for fault scenarios (i.e., ignoring Case 0), the CL regime outperformed all others by a statistically significant margin, except for the very difficult triple fault case 17, in which there is no significant difference between CL and AL. The AL regime in turn provides significantly improved adaptation compared to the baseline methods SL and NL.

4.3 Experiments with Physical Robots

As previously reported [8], experiments on an actual mobile robot were performed at the Naval Research Lab showing that the CEL approach yields effective adaptation to a variety of partial sensor failures. The robot used in these experiments was a Nomadic Technologies Nomad 200 mobile robot, a three-wheeled, synchronized-steering vehicle used in experimental studies at NRL. Figure 3 shows shots of the physical robots. The first picture shows the robot finding the opening with its front sonar, and proceeding straight through the opening. The front sonar was then covered to simulate its failure. After a brief period of adaptation using previously learned plans in its case base, the robot was able to solve the task despite its sonar failure (third picture). After adaptation, the robot used a side sonar to find the opening and then turned towards the opening. Future experiments with these laboratory robots will simulate effector faults (restricted turn rates and

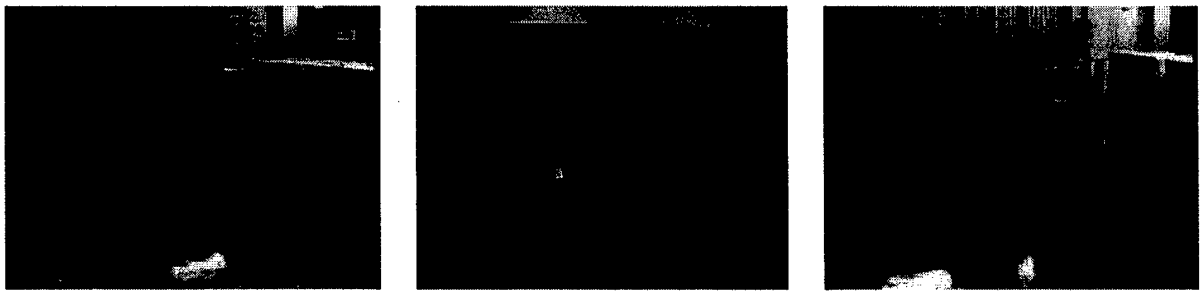


Figure 3: a) Robot with full sensors passing directly through doorway. b) Robot with front sonar covered. c) Robot after adapting to covered sonar. It uses side sonar to find opening, and then turns into the opening.

speeds) as well as multiple sensor fault modes.

5 Discussion

This work shows that the Continuous and Embedded Learning model is a promising approach to adapting to both sensor failures and effector faults. In CEL, when the monitor detects a sensor failure, it modifies the system's learning simulation. The learning system operates indefinitely, and the execution system uses the results of learning as they become available. Combined with our previous work showing adaptation to changing environments [1], actuator failures [4, 3], and sensor failures [8], this work indicates the generality of the CEL model for the design of robust autonomous robot systems.

The specific contributions of the project include:

- The Continuous Embedded Learning approach is able to adapt very quickly to fault scenarios that have been previously experienced, taking advantage of a case-base of learned strategies.
- The CEL approach can adapt its learned strategies to novel fault scenarios by exploiting previously learned strategies from a combination of previously learned cases.
- Using a case base for adaptation provides a significant advantage over merely learning new cases from scratch (as shown by the results of CL vs. AL).
- Modifying the offline simulation model provides significant advantages over continuous learning using a static simulation model (as shown by the results of AL vs. SL).

5.1 Directions for Further Research

This study focused on the adaptation ability of the SAMUEL learning system, augmented by the additional components shown in Figure 1. It would be of interest to investigate other modifications

of SAMUEL that might make it even more robust in the face of changing environments, especially within the domain of autonomous robots. One direction suggested by the results of the training regime in this study would be to investigate changes to the rule-matching procedures in SAMUEL, so that conditions relating to faulty sensors could be ignored at the level of the matching algorithm. Such a change would permit rules learned in one fault scenario to be used in another fault scenario without the level of performance disruption indicated by Table 3. For example, if a set of rules are learned with one blocked sonar, instead of learning rules that assume the blocked sonar returns a fixed value of 0, it might be better to learn rules that simply ignore the value of the blocked sonar. Such rules might work well in other fault scenarios, so as when the sonar was operating properly.

A serious challenge is to develop more of a theory of adaptation that quantifies the utility of previously learned knowledge in new situations. One can imagine several variations of the CEL approach that might try to measure the adaptive utility of the case base. For example, it may be possible for the online system to assess the difficulty of adapting previously learned strategies to the current case. When applying the CL approach to a novel case, if little progress is made after a certain period of time (e.g. 25 generations), the offline system might decide to try reinitializing the population from scratch, thus switching to the AL regime.

5.2 Software deliverables

All software developed under this project has been delivered to the Adaptive System Section (Code 5514) of the Navy Center for Applied Research in Artificial Intelligence, at the Naval Research Laboratory.

Acknowledgments

The work reported here was supported by the Naval Research Laboratory under Grant N00173-00-1-G013.

References

[1] J. J. Grefenstette and C. L. Ramsey. An approach to anytime learning. In *Proc. Ninth International Conference on Machine Learning*, pages 189–195, San Mateo, CA, 1992. Morgan Kaufmann.

[2] J. J. Grefenstette, C. L. Ramsey, and A. C. Schultz. Learning sequential decision rules using simulation models and competition. *Machine Learning*, 5(4):355–381, 1990.

[3] John J. Grefenstette. Genetic learning for adaptation in autonomous robots. In *Robotics and Manufacturing: Recent Trends in Research and Applications, Volume 6*, pages 265–270, New York, 1996. ASME Press.

[4] C. L. Ramsey and J. J. Grefenstette. Case-based anytime learning. In D. W. Aha, editor, *Case Based Reasoning: Papers from the 1994 Workshop*, Menlo Park, CA, 1994. Technical Report WS-94-07, AAAI Press.

[5] Alan C. Schultz. Learning robot behaviors using genetic algorithms. In *Intelligent Automation and Soft Computing: Trends in Research, Development, and Applications*, pages 607–612, Albuquerque, 1994. TSI Press.

[6] Alan C. Schultz and John J. Grefenstette. Using a genetic algorithm to learn behaviors for autonomous vehicles. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Hilton Head, SC, 1992.

[7] Alan C. Schultz and John J. Grefenstette. Robo-shepherd: Learning complex robotic behaviors. In *Robotics and Manufacturing: Recent Trends in Research and Applications, Volume 6*, pages 763–768, New York, 1996. ASME Press.

[8] Alan C. Schultz and John J. Grefenstette. Continuous and embedded learning in autonomous vehicles: Adapting to sensor failures. In Grant R. Gerhart, Robert W. Gunderson, and Chuck M. Shoemaker, editors, *Unmanned Ground Vehicle Technology II, Proc. of SPIE Vol 4024*, pages 55–62, 2000.

Appendix A: file training.cases

```
# no faults
000: -30  30  -4  10   0
# single sensor failures
001: -30  30  -4  10   1
002: -30  30  -4  10   2
003: -30  30  -4  10   4
004: -30  30  -4  10   8
005: -30  30  -4  10   16
006: -30  30  -4  10   32
007: -30  30  -4  10   64
#
# turn faults
#
# limited right turn
008: -30 10  -4  10   0
# limited left turn
009:  -10  30  -4  10   0
# limited turn
010: -10  10  -4  10   0
#
# translation faults
#
# no backing up
011: -30  30  0  10   0
# slow speed
012: -30  30  -4  4   0
# fixed slow speed
013: -30  30  4  4   0
# fixed fast speed
014:  -30  30  10  10   0
```

Appendix B: file test.cases

```
# no faults
000: -30  30  -4  10   0
# single sensor failures
001: -30  30  -4  10   8
# limited turn
002: -10  10  -4  10   0
# fixed fast speed
003:  -30  30  10  10   0

# limited right turn with sensor failures
004: -30 10  -4  10   8
005: -30 10  -4  10   64

# limited left turn with sensor failures
006: -10 30  -4  10   1
007: -10 30  -4  10   8

# limited turn with sensor failures
008: -10 10  -4  10   2
009: -10 10  -4  10   8
010: -10 10  -4  10   32

# fixed slow speed with sensor failures
011: -30  30   4  4   4
012: -30  30   4  4   16

# fixed fast speed with sensor failures
013: -30  30   10 10  1
014: -30  30   10 10  64

# limited right turn, fixed fast speed
015: -30 10  10  10   0

# limited left turn, fixed slow speed
016: -10 30   4  4   0

# limited turn, fixed fast speed
017: -10 10  10  10   0

# limited turn with two failures
018: -10 10  -4  10   65

# triple sensor failures
019: -30  30  -4  10   73
```