Case-Based Reactive Navigation: A Method for On-Line Selection and Adaptation of Reactive Robotic Control Parameters

Ashwin Ram, *Associate Member, IEEE*, Ronald C. Arkin, *Senior Member, IEEE*, Kenneth Moorman, and Russell J. Clark, *Member, IEEE*

***Abstract*— This article presents a new line of research investigating on-line adaptive reactive control mechanisms for autonomous intelligent agents. We discuss a case-based method for dynamic selection and modification of behavior assemblages for a navigational system. The case-based reasoning module is designed as an addition to a traditional reactive control system, and provides more flexible performance in novel environments without extensive high-level reasoning that would otherwise slow the system down. The method is implemented in the ACBARR (A Case-BAsed Reactive Robotic) system and evaluated through empirical simulation of the system on several different environments, including "box canyon" environments known to be problematic for reactive control systems in general.**

***Index Terms*— Case-based reasoning, reactive control, robot navigation.**

I. Motivation

SEVERAL methods for autonomous robotic control have been proposed, ranging from high-level planning methods—in which extensive world knowledge is used to formulate a detailed plan of action—to the more recent methods for "reactive control"—in which a simple representation of the robot's immediate environment is used to select the next action to be performed. However, most current robotics systems are severely limited when performing in complex and dynamic environments in the real world. It requires careful design and tuning on the part of the human designer to develop the control systems that drive such robots, and even then these systems run into serious difficulties when faced with environments which are different from those that the designer anticipated. Furthermore, even if the designer could anticipate and model all the relevant aspects of the operating environment of the robot, the dynamic nature of the real world would render parts of this model obsolete. Clearly, the ability to adapt to changes in the environment is crucial to adequate performance and survivability in the real world.

Manuscript received August 23, 1993; revised August 21, 1994, April 26, 1995, and February 2, 1996. This work was supported in part by a Fannie and John Hertz Foundation Fellowship, the Georgia Institute of Technology, the National Science Foundation under Grants IRI-9100149 and IRI-9113747, and by the Material Handling Research Center of the Georgia Institute of Technology.

A. Ram, R. C. Arkin, and K. Moorman are with the College of Computing, Georgia Institute of Technology, Atlanta, GA 30332-0280 USA (e-mail: ashwin.ram@cc.gatech.edu).

R. J. Clark is with the Department of Computer Science, University of Dayton, Dayton, OH 45420-2160 USA.

Publisher Item Identifier S 1083-4419(97)02915-4.

This article presents a case-based reasoning method for on-line selection of robotic control behaviors, and on-line adaptation of selected behaviors to the immediate demands of the environment, within a reactive approach to autonomous robotic control. The method allows an autonomous robot to automatically select appropriate behaviors and to adapt these behaviors dynamically based on the demands of the environment, and results in a substantial improvement in performance. In particular, the system makes use of both global behavior changes when needed as well as local modifications which act to optimize its behavior. Consider, for example, the scenario shown in Fig. 1. This figure shows a simulated robot navigating through a world which consists of both relatively clear areas and relatively crowded ones. While it is possible for the system to perform under the control of one behavior set, this may not lead to high quality performance. In this case, two navigational strategies should be employed—one for navigating clear areas and one for navigating cluttered areas. The system begins with a strategy for navigating in clear areas (toward the left side of the figure). This provides excellent performance until the cluttered area is encountered. At this point, the system switches (based on its perception of the environment) to a strategy better suited for obstacle-filled regions. Once the obstacle region is successfully navigated, the system switches back to the navigational strategy suited to clear areas (toward the right side of the figure).

Such global changes in navigational strategies allow the system to adapt to various regions within a single environment, thereby allowing more robust behavior. In addition, within the confines of a particular strategy, local adjustments allow the system to "fine-tune" itself to a specific environment. For example, while the robotic system may have a particular strategy which is useful in general cluttered environments, local adjustments may be necessary to adapt the strategy to the specific cluttered environment the robot is attempting to navigate. Thus, the global changes suggest general behavior patterns which are then tuned via step-by-step adjustments.

While traditional reactive robotic control systems [3], [10], [19], [30] have produced impressive results in the area of generating intelligent robotic action, the behaviors available to such a system are hard-wired and immutable. This approach has some significant shortcomings. Hard-wired behaviors are unable to handle environments which the initial programmer did not foresee. They are also incapable of taking advantage of navigational successes; even if a behavior has proven

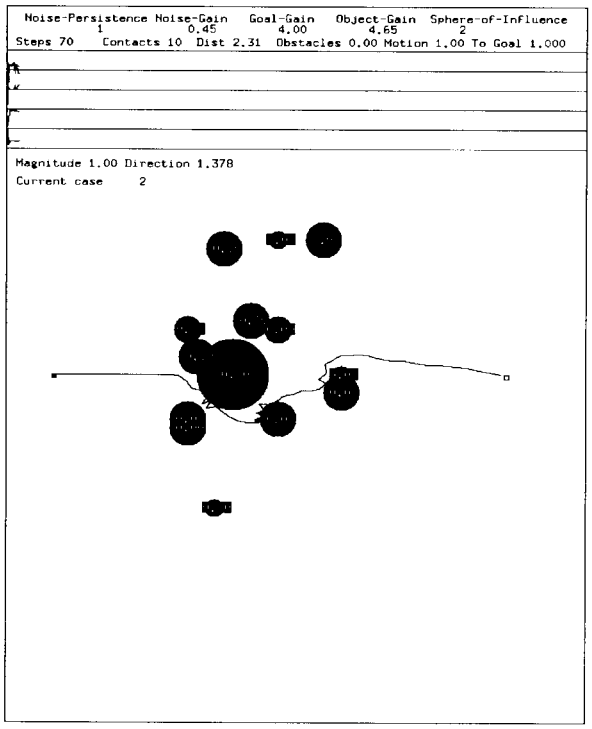


Fig. 1. Example of adaptive reactive control in a simulated robot.

extremely successful, it is not used more than any other behavior in the system, nor is it altered to become even more successful. In order to achieve more robust robotic control, we advocate the use of *behavior assemblages*, to represent appropriate collections of cooperating behaviors for complex environments; *behavior switching* to dynamically select behaviors appropriate for a given environment; and *behavior adaptation* to adapt and fine-tune existing behaviors dynamically in novel environments. There are two types of behavior adaptations that might be considered. One option is to have the system modify its current behavioral parameters based on immediate past experience (behavior adaptation). This is a local response to the problem. A more global solution is to have the system select completely new sets of behavioral parameters based on the current environment in which it finds itself (behavior switching). A robust system should be able to adapt to its environment dynamically in both these ways. Our research incorporates both behavior adaptation and behavior switching into a reactive control framework.

The research presented here contributes to reactive control for autonomous robots in the following ways. First, a method is presented for the use of behavioral parameters tailored to particular environmental demands, rather than of single or multiple independent behaviors. Second, the system can select and adapt these behaviors dynamically without relying on the user to select the "correct" behavioral parameters for each navigation problem. Finally, the system exhibits considerable flexibility over multiple environments. For example, as shown below, it performs well in uncluttered worlds, highly cluttered worlds, worlds with box canyons, and so on, without any reconfiguration.

The research also contributes to case-base reasoning and machine learning in several ways. One, our system uses case-based reasoning to suggest global modifications (behavior selection) as well as to suggest more local modifications (behavior adaptation). The knowledge required for both kinds of suggestions are stored in a case, in contrast with traditional case-based reasoning systems in which cases are used only to suggest solutions, and a separate library of adaptation rules is used to adapt solutions. Two, the method represents a novel variation on case-based reasoning for a new kind of task, one that requires continuous, on-line performance (see also [34] and [36]). The system must continuously evaluate its performance, and continue to adapt the solution or seek a new one based on the current environment. Furthermore, this evaluation must be done using the simple perceptual features available to a reactive control system, unlike the complex thematic features or abstract world models used to retrieve cases in many case-based reasoning systems. Finally, unlike traditional case-based reasoning systems which rely on deep reasoning and analysis (e.g., [17]), and unlike other machine learning "augmentations" to reactive control systems which fall back on nonreactive reasoning (e.g., [13]), our method does not require the system to "stop and think;" the system continues to perform reactively with very little performance overhead as compared to a "pure" reactive control system.

The methods are fully implemented in ACBARR, A Case-BAsed Reactive Robotic system. The system has been evaluated extensively; both qualitative and quantitative results from several simulations are presented below. ACBARR has been found to be robust in simulation, performing well in novel environments. Additionally, it is able to navigate through several "hard" environments, such as box canyons, in which purely reactive systems would perform poorly. Such environments pose a chronic problem for all memory-less methods that rely entirely upon direct sensory information, not only for the potential-field-based methods used in schema-based navigation.

II. Background and Related Research

A. Perception and Reactive Control

Reactive control [3], [10], [12], [19], [30] is concerned with how to coordinate multiple motor behaviors. It is characterized by a tight coupling between perception and action with little or no intervening representation. This results in systems which do not perform detailed planning but are able to function in dynamic and complicated environments. "Pure" reactive control is characterized by a stimulus-response type of relationship with the world, not unlike the viewpoint held by the behaviorist psychologists, epitomized by Skinner [38]. Mentalistic (representational) structures are denounced and the robot reacts to the immediacy of sensory information in a very low-level noncognitive manner. Complex behaviors emerge as a combination of simple low-level responses to the rich variety of stimuli the world affords.

There are many representative examples of this form of navigation. Brooks' subsumption architecture has demonstrated robust navigation for mobile vehicles in dynamically chang-

ing domains [10]. There is a deliberate avoidance of world modeling which is captured by the statement that *the world is its own best model* [11]. Recently learning methods have been introduced into this type of architecture to further increase its flexibility (e.g., [27]). Payton has described a collection of motor responses that are termed "reflexive behaviors" [30]. These behaviors react directly to sensory information yielding intelligent emergent behavior. Payton, Brooks, Kadanoff [18], and several other proponents of reactive control incorporate the concept of *arbitration*. Multiple behaviors compete for control of the vehicle with a winner-take-all mechanism deciding the result. Kaelbling has developed a reactive architecture [19] that is an extension of Brooks' work. The emphasis is on embedded systems for real-time control. A hierarchical competency level for behaviors is established which is mediated by a high-level controller. She has additionally conducted research in learning in autonomous agents [20]. Firby has developed a different form of reactive control by utilizing modules called Reactive Action Packages which encapsulate tasks for a robot [16]. Situation-driven execution via goal satisfaction is the predominant mode of operation. Agre and Chapman [1] have used reactive control in the domain of game playing. Several behaviors are active at any time, controlling the strategies used by a video game penguin and its relationship with other objects and entities in the world.

Reactive navigation in our system [3] addresses reactive control in a manner that is significantly different than the approaches described above. Arbitration is not used for coordinating the multiple active agents; instead, potential field formulations are employed to describe the reactions of the robot to the world, and explicit representational knowledge is used to select and configure both the motor and perceptual strategies used for reactive control.

Despite the assumptions of early work in reactive control, *representational knowledge is important* for robot navigation. The fundamental problem lies in representing what is appropriate for the task. Amarel's classic paper [2] shows the importance of appropriate knowledge representation for problem solving using artificial intelligence. The question is, first, what needs to be represented for successful general-purpose mobile robot navigation, and, second, how it is to be represented. Our answer to the first question is threefold: *motor behaviors* that are used to describe the set of interactions the robot can have with the world, *perceptual strategies* that provide the required sensory information to the motor behaviors, and *world knowledge* (both *a priori* and acquired) that is used to select (and reconfigure when necessary) the motor behaviors and perceptual strategies that are needed to accomplish the robot's goals. The following section answers the question as to how to represent this knowledge.

B. Motor Schemas

Our system is based on the Autonomous Robot Architecture (AuRA) [3]. In this system, each basic type of motor behavior is represented by a *schema*. Motor schemas comprise a collection of individual motor behaviors each of which reacts to sensory information gleaned from the environment. The output of each individual motor schema is a velocity vector representing the direction and speed at which the robot is to move given current environmental conditions. Some of the available motor schemas for our robot include MOVE-AHEAD (move in a general direction), MOVE-TO-GOAL (move toward a discernible goal), AVOID-STATIC-OBSTACLE (move away from an observed obstacle), and NOISE (move randomly in a wandering fashion).

Each of these schemas is realized as a separate asynchronous computing agent with parameters reflecting current world knowledge. The output of each primitive motor schema is combined using vector summation and normalization. This simple process can result in quite complex trajectories and behaviors as illustrated in the simulations and experiments reported in other work (e.g., [3]).

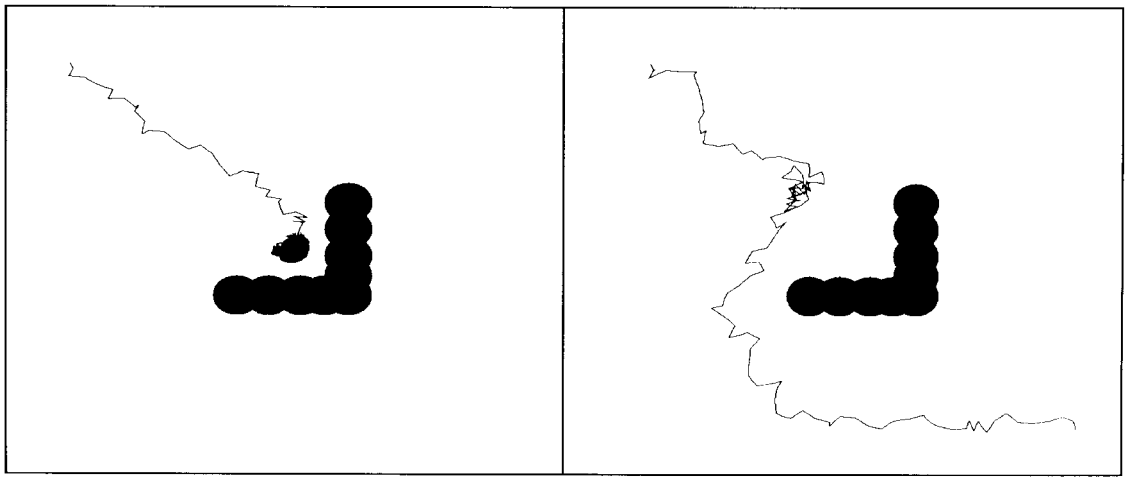
To optimize system performance it is necessary to determine what parameter values should be used to accomplish a specific task in a given environment. For instance, an exploration behavior can be observed by providing a relatively high gain and persistence to the NOISE schema with an accompanying AVOID-STATIC-OBSTACLE schema [3]. The task of determining appropriate *a priori* parameter values is nontrivial in highly cluttered environments. For a given environment, this gain determination process involves empirical evaluation of the system's performance. The process is repeated until further changes result in no visible improvement. When structural environmental knowledge is available, this task becomes simpler [4], but for purely reactive systems with no knowledge of the world, highly complex environments can produce difficulty in reaching near optimal solutions.

Furthermore, once this "best set" of parameter values is established for a given world, it will likely be less efficient for navigation in a different environment. Figs. 2 and 3 illustrate a scenario where this is true. Fig. 2 shows a trap, known as a "box canyon," where a high AVOID-STATIC-OBSTACLE sphere of influence is necessary to produce a successful path. Fig. 3 shows a slightly different trap, a "quasi" box canyon, where a high sphere of influence results in an inferior path. The relatively jagged paths in both of these sample runs were produced by high levels of NOISE being used. This was necessary in order to free the robot from local minima it may encounter. Later in this article, it will be seen that our on-line adaptive reactive control system smooths out the paths autonomously by lowering the influence of noise when it is not required.

Finally, in a complex environment, it may be difficult to find one "best" set of parameter values; different values may be appropriate for different areas of the environment. It is, therefore, important to develop methods by which the reactive control system can select and adapt gain and other parametric values automatically. The system presented here uses what we call an *adaptive reactive control* method to select and adjust the appropriate behavioral parameters to match the current task and environment.

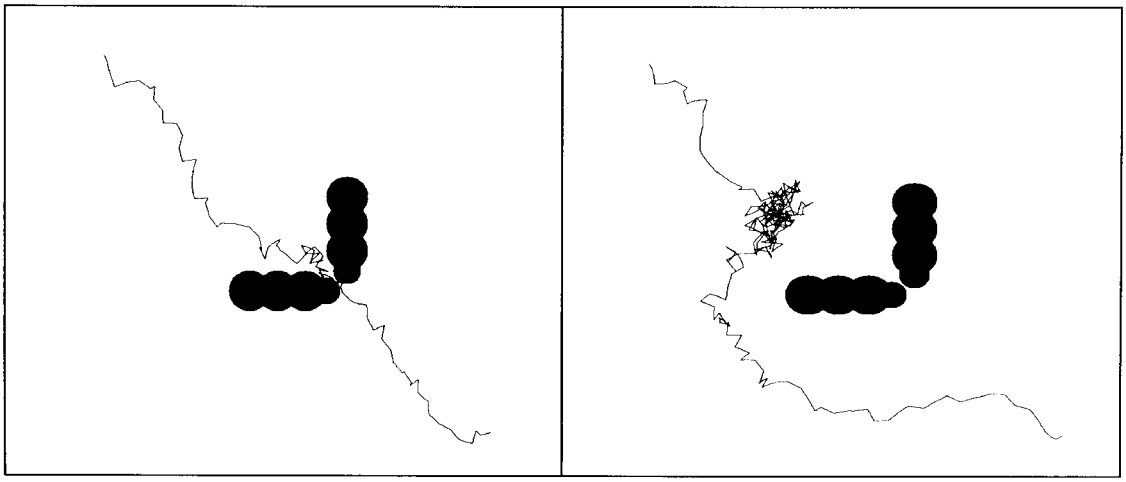
C. Case-Based Reasoning and Machine Learning

Machine learning is concerned with developing methods with which an intelligent system can improve its own performance by learning new conceptual characterizations



(a) (b)

Fig. 2. Box canyon runs: (a) sphere of influence = 10 and (b) sphere of influence = 35.



(a) (b)

Fig. 3. Quasibox canyon runs: (a) sphere of influence = 10 and (b) sphere of influence = 35.

of the domain, by learning new control strategies, by learning or improving domain theories and world models, and combinations of these. In robotic systems, these methods provide an agent with, for example, a means to learn a map of its environment (e.g., [24]) or to learn control strategies at a high level (e.g., [37]) or at the reactive level (e.g., [20], [35], and [36]). Much of this work focuses on learning control behaviors; less emphasis has been placed, however, on flexible adaptation of learned (or pre-programmed) control behaviors in different task environments.

Even in situations anticipated by the designer, we cannot assume that the system will have an "optimal" control strategy (whether designed or learned) that is tailored to that situation. Instead, the system will need to perform its task to the best of its ability, based on control strategies that are flexible enough to provide reasonable performance in a wide range of situations. For our navigation task, this means that the system must be able to fine-tune its behaviors and adapt to its environment during actual performance on a wide range of problems in dynamic task environments. In principle, the system could analyze the new situation (or its failure to perform in that situation) to determine how to proceed [9], [13], [29]; however, such an analysis is typically too slow and requires extensive high-level knowledge about the environment. A reactive control system requires a fast, on-line, adaptive process which allows the system to deal with the novel situation without having to perform a deep and extensive analysis of the situation. Such a process pays more attention to the easily available (but perhaps superficial) aspects of the situation, and uses them to make quick decisions about performance without falling back on slower, nonreactive techniques. While such a system does not necessarily "learn" new control strategies, it does adapt to the environment dynamically during the performance task by selecting behavioral parameters, such as momentum, that are appropriate for the environment. The issue, then, becomes the kind of knowledge that the system needs to adapt its behaviors appropriately.

While others have worked on systems which integrate learning and reactive control in order to learn the proper behaviors (e.g., [13], [25], [28], and [29]), our approach is

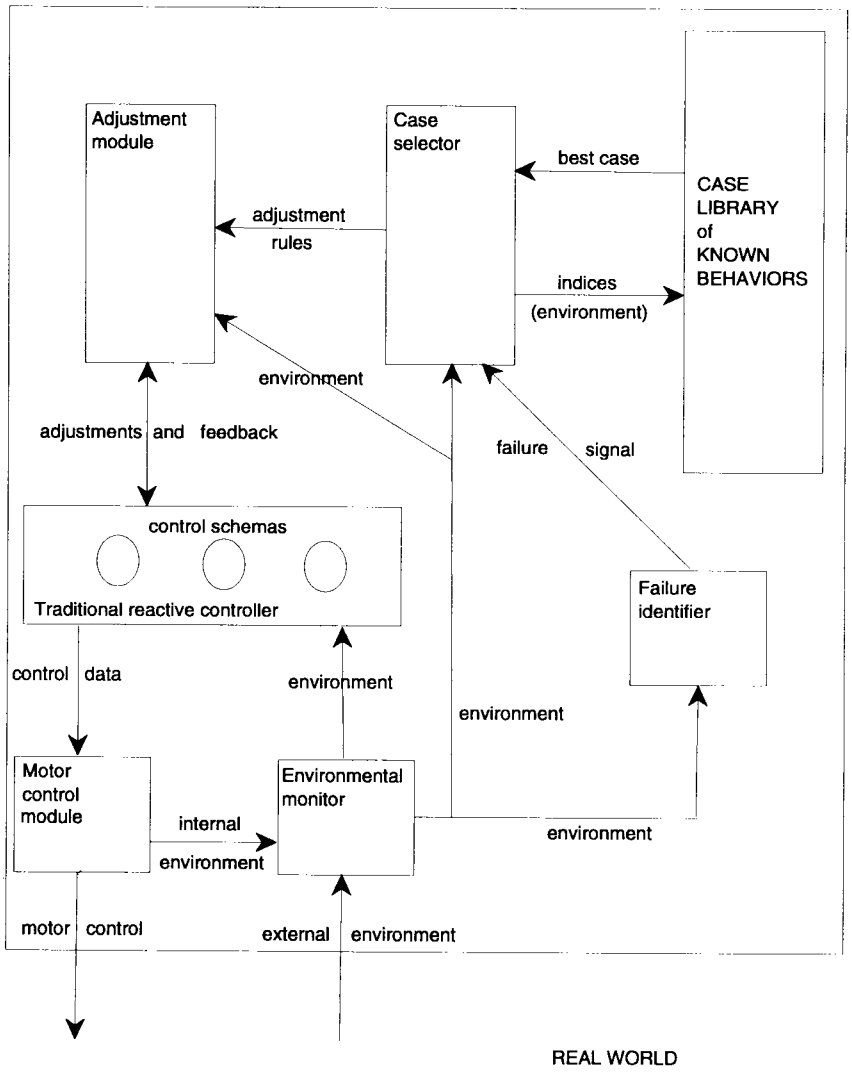


Fig. 4. System architecture.

different in the sense that the system is *not* learning behavior sets for future use. Instead, our system is using a set of global and local adaptations on existing behavior sets in order to tune (at both a gross and a fine level) the system's performance. The underlying philosophy is different—rather than attempt to learn a near-optimal behavior for a given environment (which then has to be generalized in some fashion if it is to be useful), our approach is to have the system be flexible in the on-line adapting of existing behavior sets to a wide variety of situations. A given behavior may not be optimal (or even near-optimal) for any particular environment; instead, the adaptation techniques we describe allow it to be a "jack-of-all-trades" and adapt to the needs of each environment it faces.

Our approach to on-line adaptation is based on case-based reasoning (CBR), which integrates aspects of inductive and analytical reasoning and provides a method for adaptation to novel situations [17], [21], [31], [39]. The intent behind CBR is to use a library of *cases* to provide suggestions for action in new situations. Thus, cases in a CBR can, in principle, be used to represent knowledge that is applicable to a broad class of problems. Hammond's CHEF program [17] is an example of early work in the application of CBR to nonreal-time and nonreactive planning problems. However, learning and adaptation in CHEF is carried out through higher-level reasoning, which requires a detailed model of the domain. This is exactly what reactive systems are trying to avoid. Another difference between traditional CBR systems and ACBARR is that many (although not all) CBR systems perform "case adaptation," a process in which cases are modified before being applied to a new problem. ACBARR, on the other hand, does not modify its cases; instead, it uses cases to perform "adaptive navigation" by adapting the reactive control parameters in use by the robotic controller.

Furthermore, CHEF and similar systems have focussed on planning and not on execution-time control. Likewise, Ramsey and Grefenstette use case-based reasoning to seed a genetic algorithm [36]; in their system, CBR is used to initialize a machine learning algorithm and not for real-time adaptive control of the navigational agent itself. Kopeikina, Brandau, and Lemmon describe an application of CBR to real-time control [23]. Their system, though not intended for robotics, is designed to handle the special issues of time-constrained

processing and the need to represent cases that evolve over time. Our approach, called *continuous case-based reasoning* [34], combines the adaptive aspects of case-based reasoning with the on-line, real-time aspects of reactive control; in particular, it uses cases to guide on-line adaptation of reactive control parameters in a robotic system.

In general, the underlying assumption in case-based reasoning is that past cases will be applicable to new situations; what worked in the past on a certain problem will more than likely work on a new problem which is similar to the previous one. The quality of the solution which a CBR program will generate is dependent upon the following factors:

- set of *cases* in the system's case library;
- *representation* of the cases;
- ability to recognize that a new situation is like a previous one (*case retrieval*);
- methods for *application* of an existing case to a new situation;
- ability to *evaluate* the efficacy of the case, and to decide when a new or different case might be needed.

Our research, then, must be concerned with how cases are stored in memory, how they are indexed for retrieval, how to handle the issue of partial case matches, and knowing when to adapt versus when to select a new case. In addition, a critical aspect of an adaptive system for robotic control is that the adaptive process must not interfere with the system's reactive response. Finally, unlike traditional CBR systems which rely on cases to suggest solutions (here, behaviors) but then rely on rules to adapt those cases, the ACBARR method uses cases for both purposes.

III. Technical Details: The ACBARR System

ACBARR is a combination of a schema-based reactive control system with a case-based reasoning module. Fig. 4 illustrates the system's high-level architecture. Data about the world is gathered by the ENVIRONMENTAL MONITOR, and combined with the robot's status gathered by the INTERNAL MONITOR which represents how well the robot is moving in relation to the world. The system uses this environmental information in several ways. The ADJUSTMENT MODULE adjusts the parameter values of the active motor schema based on the recommendations of the current case. The FAILURE IDENTIFIER monitors the feedback for a failure situation. If it determines that the current schema assemblage is inadequate, the CASE SELECTOR chooses a better case from the case library, which suggests a new parameter set and associated ranges for parameter adaptation.

A case within ACBARR's case library represents a parameter set for the behavior assemblage which is well suited for a particular environment. It contains information both about the desired behaviors and about the environment in which it should be applied, which acts as the index for that case. The behavior information contains the new limits for each schema gain value and how each is allowed to change while this case is active. The environmental section represents the environment and movement status of the robot under which to apply this set of gain behaviors. To switch to a case, the case selector has to determine which index environment in the stored cases best matches the current environment and the evaluator has to determine whether the case switch is appropriate.

```
ACBARR-MAIN-LOOP {

   /* Determine the current state of world */
   /* and select a starting case */
   environment = Examine-Environment ();
   movement = Examine-Movement ();
   current-case = Select-Case (environment, movement);

   /* Main processing loop */
   DO {

      IF (granny = TRUE) { /* If in "careful" mode */
         /* Analyze detected obstacle position and */
         /* calculate a movement vector intended to */
         /* "aim" the robot towards a less filled area. */
         /* The system is still influenced by the subsequent */
         /* reactive code. */
         new-vector =
            Determine-Best-Path-Vector (environment);
         robot.movement =
            Add-Vector (robot.movement, new-vector);
      }

      /* The next lines represent "pure" reactive control */
      obstacle-vectors =
         Determine-Obstacle-Influence (environment);
      robot.movement =
         Add-Vector (robot.movement, obstacle-vectors);
      robot.movement =
         Add-Vector (robot.movement, random-noise-vector);

      Update-Robot-Position ();
      Update-Spatial-Memory ();
      /* This ends the reactive control portion */

      /* Update environment and movement knowledge */
      environment = Examine-Environment ();
      movement = Examine-Movement ();

      /* Test to see if a new case is needed */
      IF (New-Case-Needed (environment, movement)) {
         current-case =
            Select-Case (environment, movement);

         /* If a new case is selected, set the parameter values */
         /* to a random value between the defined limits */
         FOR (each parameter-value in current-case) DO {
            current-case.parameter-value =
               RANDOM(current-case.parameter-value.low-limit,
                      current-case.parameter-value.hi-limit);
         }
      }

      /* Locally adjust parameter values based on current case */
      Adjust-Parameter-Values (current-parameters,
                               current-case)

   } UNTIL (goal-reached OR
            number-steps > maximum-steps-allowed)
} /* end ACBARR-MAIN-LOOP */
```

Fig. 5. Top-level case retrieval and use algorithm.

In order to understand the process more fully, we now present the details of how each of the components of the ACBARR system operates. To frame the following discussion, the top-level algorithm used by the system is presented in Fig. 5.

A. Reactive Control

The following motor schemas were used in this research:

- AVOID-STATIC-OBSTACLE: Repel from object with variable gain and sphere of influence. Used for collision avoidance

$$O_{\text{magnitude}} = \begin{cases} 0 & \text{for } d > S \\ \frac{S-d}{S-R} * G & \text{for } R < d \leq S \\ \infty & \text{for } d \leq R \end{cases};$$

$$O_{\text{direction}} = \text{From center of obstacle toward robot;}$$

where

S Adjustable sphere of influence (radial extent of force from center of obstacle).
R Radius of obstacle.
G Adjustable gain.
d Distance of robot to center of obstacle.

- MOVE-TO-GOAL: Attract to goal with variable gain. Set high when heading for a particular goal.

$V_{\text{magnitude}}$ Adjustable gain value.
$V_{\text{direction}}$ Direction toward perceived goal.

- NOISE: Random wander with variable gain and persistence.[1] Used to overcome local maxima, minima, cycles, and for exploration.

$N_{\text{magnitude}}$ Adjustable gain value.
$N_{\text{direction}}$ Random direction, persists for $N_{\text{persistence}}$ steps.
$N_{\text{persistence}}$ Adjustable persistence value.

The reactive control system has been successfully demonstrated in both simulation and real robot systems [3], [8]. In the simulation, the radius and center of each obstacle as well as the goal location are available from the simulated environment model. In a real robot, the obstacle and goal location information is determined from the input of several possible sensors including ultrasound, laser scanners, video cameras, and odometry sensors. Previous research shows that the schema-based reactive control mechanism utilized in this work is particularly robust in the presence of uncertainty from changing environments and imperfect sensor input [3].

B. Environmental Information

Since one overall goal of this research was keep the modifications to a "pure" reactive control system as simple as possible, any environmental information represented in the system must be obtainable through perceptual input during the normal reactive control process. With this in mind, the following environmental information is maintained:

- CLUTTER is a metric of how many obstacles are affecting the robot, defined as the ratio of sensed obstacles to a preset clutter value.[2] In our tests, this value was set to 5.0. Thus, five sensed obstacles would result in a clutter metric of 1.0. Ten obstacles would equate to a clutter of 2.0. To aid in partial matching for case selection, a clutter value can match to range of similar clutter values, defined to be 0.4, i.e., any two clutter values within 0.4 of each other are considered indistinguishable. For example, a case that specifies a clutter value of 1.4 would exactly match an environment with seven obstacles (CLUTTER $=$ 1.4), partially match one with eight obstacles (CLUTTER $=$ 1.6), but not match one with ten obstacles (CLUTTER $=$ 2.0) along this particular match dimension. These parameters were determined empirically; while they may not be optimal, they have proven effective.
- MAXXED is a binary value representing whether a parameter value has stayed at its upper bound for a significant period of time. Notice that there is no equivalent MINNED value. Empirical tests of the system indicated that parameters were more likely to need to be expanded upwards rather than below a preset minimum.
- CLEAR-TO-GOAL is a binary value which indicates if the robot is facing the goal and there are no obstacles between it and the goal along a straight line path.
- SENSES-GOAL is a binary value which represents whether the goal is within the robot's perceptual range with nothing between the goal and the robot. Unlike CLEAR-TO-GOAL, the robot may or may not be physically facing the proper direction to the goal.
- GOAL-NEARBY is a binary value which indicates that the robot senses the goal but there are obstacles between the two.
- GOAL-DIRECTION is a value, in radian angle measure, which represents the direction of the goal relative to the current position and heading of the robot.
- GRANNY[3] is a binary variable which determines when particularly careful behavior should be employed. It results from long-term poor performance, which will indicate that no known case is working.

Each of the items on the above list is information which is readily available, either directly from the robot's sensors or from a straightforward mathematical calculation of sensor data. No higher level processing is required to produce this information.

C. Movement Information

ACBARR also needs information involving its performance. This is knowledge of how well the current case behaviors are performing in terms of the robot's movement toward the goal position. There are six situations to which ACBARR needs to pay attention in order to evaluate its current progress:

- NO-MOVEMENT: The robot's average step size has dropped below a certain threshold

$$M < T_{\text{movement}}$$

where

M Step size averaged over H_{steps} steps.
T_{movement} Movement threshold.

- MOVEMENT-TOWARD-GOAL: The robot's step size and rate of approach to the goal are both above some

[1] Since noise is a random effect, a persistence value is needed to indicate the amount of time a particular noise direction will be in effect.

[2] In a simulated environment, the exact number of obstacles can be known by a robotic system. In an actual robot, the sensors may only sense one obstacle if a number of obstacles are touching. Thus, several small touching or even nearby obstacles may be considered one large obstacle. As a result, in an implemented robotic system, the CLUTTER metric will have an estimated value.

[3] GRANNY is a "careful" mode which adds an additional step to the standard reactive control process—in addition to calculating obstacle influences, the system, if in granny mode, also determines a direction in which lie fewer obstacles. This helps the system avoid densely populated regions of obstacles, much like the BALLOONING behavior; unlike BALLOONING, however, the granny computation allows the robot to treat obstacles in less dense regions in whatever fashion is dictated by the current control behavior. The term "granny" was selected in a light-hearted jest of the stereotypical grandmother who is ultraconservative, careful, and worried about safety. When in this mode, the system will avoid densely populated regions of obstacles, giving the appearance of being extremely careful about traveling in those areas.

threshold

$$M \geq T_{\text{movement}}$$
$$P = \frac{H_{\text{distance}}}{H_{\text{goal}}} \geq T_{\text{progress}}$$

where

H_{distance} Distance traveled over H_{steps}.
H_{goal} Decrease in distance to goal over H_{steps}.
T_{progress} Progress threshold.

- NO-PROGRESS-WITH-OBSTACLES: The robot is moving but not toward the goal and the robot is within the sphere of influence of one or more obstacles

$$M \geq T_{\text{movement}}$$
$$P = \frac{H_{\text{distance}}}{H_{\text{goal}}} < T_{\text{progress}}$$
$$O_{\text{count}} \geq T_{\text{obstacles}}$$

where

O_{count} Average number of obstacles over H_{steps} steps.
$T_{\text{obstacles}}$ Obstacle count threshold.

- NO-PROGRESS-NO-OBSTACLES: The robot is moving but not toward the goal and the robot is not within the sphere of influence of any obstacle.

$$M \geq T_{\text{movement}}$$
$$P = \frac{H_{\text{distance}}}{H_{\text{goal}}} < T_{\text{progress}}$$
$$O_{\text{count}} < T_{\text{obstacles}}.$$

- CIRCLES is a metric of how much the robot has recently traveled over the same piece of ground. It is calculated by consulting an internal, short-term, spatial memory (Section III-D). The metric used in ACBARR is the number of steps taken by the robot in a specific locale (a five-by-five region of space centered at the robot) divided by a constant value (10.0). These values were determined empirically.
- WANDER is a measure of how efficient the robot's current path is, indicated by the ratio of the robot's current path length over the actual distance between the initial position and the goal.

D. Spatial Memory

ACBARR employs a primitive short-term spatial memory mechanism in order to recognize whether or not the robot is running in circles. In our use here, the memory is primitive as a matter of choice. It was designed to keep the overhead of ACBARR's additions to pure reactive control as small as possible. As a compromise, ACBARR's spatial memory only keeps track of the robot's travels within a small *window* of the entire world, representing a small geographic area. As long as the robot remains within the same window, its steps are tracked. If the robot moves out of the window, a new window is established and the old one is forgotten. A window, while fixed in size, is not fixed in relation to location in the world. When a new window is established, it is always done with the center located at the robot's current position. If the windows were static in location, it would be possible for the robot to wander in circles between two windows and never have this detected. This type of memory has also been utilized in other systems based on the AuRA architecture to provide information about spatial occupancy of recently visited areas [4], [8].

Environmental and Movement Information	
Parameter	Value
clutter	0.0
wander	0.0
maxxed	0
clear-to-goal flag	−1
goal-nearby flag	−1
circles	0
granny	0
no-movement flag	−1
movement-to-goal flag	1
no-progress-with-obstacles flag	−1
no-progress-with-no-obstacles flag	−1

Behavioral Parameters		
Parameter	Delta Limits	Range
goal gain	[0.0 0.05]	[0.05 2.0]
noise gain	[−0.05 0.0]	[0.01 1.5]
noise persistence	[−1.0 0.0]	[1.0 5.0]
object gain	[−0.01 −0.01]	[1.0 5.0]
sensible distance	[−1.0 0.0]	[2.0 5.0]

Bookkeeping Information	
Parameter	Value
case number	20
case goodness	0.9
average step size	0.5
dynamic obstacles	0
initial distance to goal	37.56
obstacle danger	0.5
goal importance	1.0

Fig. 6. Sample ACBARR case.

E. Case Representation

A case entry in ACBARR consists of three parts. The first part represents the types of environments in which the case is applicable, and is used to determine which case to switch to. The second part details how the various parameter values can change and what their limits are, and is used to apply new parameter values and to adapt them. The third part contains bookkeeping information about the case, including information about past experiences with the case.

A sample case is shown in Fig. 6. The first section of the case contains information about the types of environment that

this behavior is well suited for. This is a combination of the environment knowledge along with the movement information. For the values, an entry of -1 indicates that the system does not care about this parameter; 0 indicates FALSE, and 1 represents TRUE. This case is best suited for a noncluttered environment. The robot should be making progress toward the goal and not wandering or running in circles. The case should not be used if the system is in "granny" mode and attempting to find the best path through a set of obstacles (by appealing to an additional level of processing above the pure reactive and pure sensor data in order to detect regions of clear space). The case should also not be used if the system has maxxed on a parameter value. The WANDER parameter of 0.0 will match with any situation where WANDER has a value of 0.0 through 1.0 (WANDER always matches to a range of values from the floor of WANDER to the floor plus one), i.e., the current performance is of sufficiently high quality. Finally, the case doesn't care if the goal can be sensed directly or not, nor does it care about the other three movement flags. If these conditions are satisfied, the system will switch to this case (unless the evaluate-per-step strategy is in place and the new case is the current case), and use the behavioral parameters recommended by the second section of the case. Additionally, the system will continue to adapt these parameters as long as the case is in use.

The second section of a case contains the information describing the bounds on each parameter value in the system. Each value is associated with two numeric ranges. The first describes the limits for the changes which the system is allowed to make on each value. In the sample case, noise persistence has a limit on changes of [-1.0 0.0]. The system is, therefore, allowed to alter the noise persistence value by a number from -1.0 to 0.0. If a constant change is desired, then the endpoints of the range simply need to be equal. The rest of the data for each value is the range which that value is allowed to change. For noise persistence, this range is [1.0 5.0]. The system is allowed to change the value of noise persistence by a real number from -1.0 to 0.0, as long as the value remains between 1.0 and 5.0. These two ranges, then, represent behavior which will successively decrease the value until it reaches its lower bound; behavior such as this would be acceptable in an open field. In addition to noise persistence, this section of a case contains the adjustment range and delta values for the other important values contained in the included motor schemas (see Section III-A).

The third section contains bookkeeping information about the case. In the example, this indicates that this is case 20 which has a goodness rating of 90%. The average step size was 0.5, there were no dynamic obstacles, and that the system was initially 37.56 distance units from the goal. Finally, the obstacle danger was 0.5, and the goal importance is 1.0. These last two values are intended to allow the system to navigate in environments where, for example, some obstacles are more dangerous than others and need to be treated in a special manner. While the current ACBARR implementation ignores this, it is included in the case description for future extensions. In particular, the information would be useful if ACBARR was able to learn new cases, since the values listed above would be those which existed in the simulation run which resulted in a specific case being formed.

F. The Case Library

Prior to creating the case library, the pure reactive robotic simulator was first run several hundred times. By abstracting from the successful test runs, we gained insight into the various types of behavior different environments would call for. This initial attempt to conceptualize the various schemas resulted in ten navigational strategies being identified, each being represented as a case in the format discussed earlier:[4]

1) CLEAR-FIELD: In an open environment, the system should pay little attention to obstacles, increase the goal gain, and lower the noise gain and noise persistence.
2) BALLOONING: When there are relatively few obstacles, the system attempts to swing around them in a wide way (increase obstacle gain).
3) SQUEEZING: When there are many obstacles, the system attempts to find a path by squeezing between obstacles (lower obstacle gain, increase goal gain).
4) HUGGING: When there are many obstacles and the system is currently faced with an obstacle directly in its path, it attempts to stay close to the side of the obstacle as it makes its way around it.
5) SHOOTING: Regardless of the number and size of the obstacles surrounding the robot, if the system sees its goal and there are no sensed obstacles in the way, it adopts an extreme version of the CLEAR-FIELD strategy and goes directly to it.
6) WALL-CRAWLING: If there is an obstacle the system cannot seem to get around by HUGGING, it checks to see if it is actually in front of a wall. The system considers to be trapped by a wall if HUGGING has failed[5] and if the incoming vectors from the obstacles are localized in front of it. In this situation, the system determines which direction the shorter side of the wall lies by looking at the vectors coming at it from each side of a centerline straight ahead, and travels for a distance in that direction. Since the system is limited to sensory data which would be available to a reactive system, this heuristic is not foolproof.
7) RANDOM: The system raises the noise gain and goal gain, leaves the obstacle gain at a medium level, and wanders in an exploratory fashion.
8) GRANNY: After H_{steps}, the system reconsiders the environment by appealing to an additional level of processing above the pure reactive control level. It concentrates on the location of the obstacles currently influencing it and attempts to discover a direction which offers the best success possibilities while deviating the least from the goal direction.
9) MAXXED: If a value has remained at its maximum level for a period of time, the system increases the maximum by some δ value.

[4] The representations of all ten cases are available in [33].

[5] This is implemented through a built-in bias of the case-selection algorithm—it does not allow re-selection of the case which produced the need for a case switch.

10) REPULSION: In certain situations, the system considers moving away from the goal for a period of time. If, for example, the system senses the goal and there is a large obstacle between the two, it may decide to "back away" for a distance before attempting to get to the goal. This is accomplished by setting the goal gain to a negative amount.

G. Indexing and Case Selection

Due to the number of existing cases, there is little need for a sophisticated indexing scheme. As a result, ACBARR employs a *flat memory model* of the case library [22]. Cases are located by a sequential search of the memory for index matches. Although generally an inefficient method, this suffices for the current system without significant slow-down. If ACBARR was extended to include several dozen or possibly hundreds of cases, a better indexing scheme (such as the use of a redundant discrimination network) would need to be implemented. With the present number of cases, however, the flat memory model provides advantages which a more complex indexing scheme would lose. Most importantly, it ensures that for a case retrieval request, the entire memory space is searched. This enables the best match to be found each time. Secondly, the addition of new cases is more straightforward with this form of indexing than with any other scheme.

There are no restrictions placed on the relationship between cases and known environments. Multiple cases may be stored for the same environment, and a case may apply to multiple environments. Both of these scenarios are handled by the system.

1) When to Select a New Case: A important issue in case-based reactive control is determining when to switch cases. The simplest method is to look for a new case every H_{steps} steps, and to switch cases if the current case does not match the environmental conditions as well as one of the other cases in the case library. In the extreme, with $H_{\text{steps}} = 1$, this method ensures that the best case will always be in place, and the system will always use the best available navigational strategy. However, the strategy of reevaluating the system's performance every H_{steps} steps is pessimistic (if H_{steps} is small). It assumes that the environment will vary in fewer than H_{steps} steps, so that the cost incurred in searching for a new case will be justified even if the current case does not obviously appear to be failing.

A second method is to evaluate the need for a new case only if the current case is failing in some manner. This is an optimistic strategy. It assumes that a given case is "good enough" as long as it does not actually fail. In order for this method to be effective, the system needs a good heuristic to determine when the current case is not leading to good performance. ACBARR makes use of its STM and environmental knowledge to check for failing cases. There are two criteria for failing. The first is excessive wandering by the robot, as determined by the WANDER parameter in the environment data. If the value of this parameter rises above a given threshold, the system tries to find a new case. The second failure criterion is the condition of running around in circles. This is kept track of by the spatial memory which tracks the robot's movement over a limited area of terrain. If the robot is not making progress out of an area, failure can be assumed.

```
New-Case-Needed {
   IF (environment.wander > WANDER_LIMIT) OR
      (environment.senses-goal = TRUE) OR
      (environment.clear-to-goal = TRUE) OR
      (running-in-circles) OR
        return NEED-NEW-CASE;
   ELSE
        return KEEP-OLD-CASE;
} /* end New-Case-Needed */
```

Fig. 7. New case decision algorithm.

Unless specified otherwise, ACBARR uses the failure-driven strategy for case switching; the algorithm used for this is shown in Fig. 7. This design decision was based on extensive empirical evaluation of both strategies. As shown in Section IV-D, the failure-driven strategy achieves respectable results with little overhead. However, both strategies are available in the system to allow the user to specify the strategy of choice.

2) Case Matching: An important issue in ACBARR, and indeed in any case-based reasoning system, is that of selecting the best case from the case library. The algorithm used by the system for this selection is shown in Fig. 8. Once ACBARR has decided to look for a new case, it uses a GOODNESS-OF-MATCH metric to find the case that best matches the current environment. In the flat indexing scheme, this is done using a matching algorithm that compares the current environment, as perceived by the robot, to each known case. If corresponding internal values in the two are equal, the GOODNESS-OF-MATCH value is incremented by 1.0. Near matches are handled through partial credit. If, for example, the current environment is very cluttered and the case being considered is just moderately cluttered, the GOODNESS-OF-MATCH value is incremented by 0.5. Finally, a special case occurs when the current environment has the CLEAR-TO-GOAL flag set to 1 and so does the case being compared. In this situation, 3.0 is added to the match value. The system, therefore, "looks for" the situation of having a clear path to the goal and attempts to exploit this. After all cases have been examined, the one with the best match is chosen. Although this is a relatively simple matching algorithm, it works well in ACBARR due to the limited number of cases. The case representation also contains sufficient information if the indexing algorithm ever needs to be extended in complexity.[6] Due to the numerical nature of the matching procedure, the GOODNESS-OF-MATCH metric can capture both partial and total matches. If no total match is found which has a high GOODNESS-OF-MATCH rating, a partial match will be chosen.

It is possible that the "best" case found for retrieval is the same case being used currently. If this is the situation, there are two possibilities. One, the retrieval mechanism may not be capturing some nuance of the environment which is causing the current case to fail. If this is true, the proper course of action is to not allow the system to select a new case which is the same as the current case. The other possibility is that the

[6] For example, the current indexing match algorithm does not make use of the DONT-CARE value in a parameter—a mismatch is considered equal in quality to a parameter match with a DONT-CARE. An extended indexing algorithm might make use of this distinction.

```
Select-Case {
   best-case = current-case;
   best-case-match = -1.0;

   /* Do a flat memory scan of all the cases in the library */
   /* looking for a case better than the current best case */
   FOR (each-case in CaseLibrary) DO {
      goodness = 0.0;

      /* Examine each environmental factor for */
      /* a match: add to match goodness value */
      FOR (each-factor in environment) DO {
         /* MATCH is a function which tests */
         /* the quality of match between */
         /* each-factor and each-case.factor.      */
         /* It returns a value from 0.5 to 3.0 */
         /* depending on specific factor and quality of match. */
         goodness = goodness +
                        MATCH(each-factor, each-case.factor)
      }

      /*  Examine each movement factor for a match: */
      /*  add to match goodness value */
      FOR (each-factor in movement) DO {
         /* Since movement factors have a limited value set. */
         /* no MATCH function is needed.                     */
         IF (each-factor = each-case.factor)
            goodness = goodness + 1.0;
      }

      /* If the case being examined is better */
      /* than the best case found so far and the is */
      /* not the current case, make it the */
      /* new best case. */
      IF ((goodness > best-case-match AND
          (each-case != current-case)) {
         best-case = each-case;
         best-case-match = goodness;
      }
   }
}  /* end Select-Case */
```

Fig. 8. Case selection algorithm.

reason that the case is currently failing is that not enough time has been given to allow the case to succeed. If this is true, the proper course of action would be to allow the system to select the same case it is dealing with as the "new" case and continue processing. After considering both of these alternatives, in theory and with implementation, we decided on the first course of action. Failures in the ACBARR system are typically serious; if a case leads to a failure situation, the chances are good that the case needs to be replaced. Thus, if the same case is retrieved as the best match, the case selector will choose the second best match as the new case.[7]

H. Case Application

Upon successful case retrieval, each current parameter value (which includes all the gain values, the sphere of influence, and the noise persistence value) in use by the system are set to a random value within the range defined by the case for that particular parameter value.[8] Once this global alteration

[7] If the system is running under the control strategy of evaluating the need to switch cases after a certain number of steps (see Section III-G-1), this restriction is lifted.

[8] Several other possibilities exist instead of random setting of these initial values. For example, an average of the allowed range may be used, either limit of the range could be the initial value, or the value in the range closest to the current value might have been chosen. A random selection was decided on to minimize the effect of the different types of ranges and allowable delta ranges which may be used. Consider a range with an associated delta range which allows only upward movement of the parameter's value. Setting the initial value to an average or to the high point of the range would have the effect that a wide portion of the range is never considered. The random nature of the initial parameter setting provides the system with some insurance that such anomalies do not occur consistently. If a specific parameter value is desired in a given application, a case may be encoded with that value at both endpoints of the allowable range.

```
Adjust-Parameter-Values (parameters, case) {

   /* Alter each parameter in the case */
   FOR (each parameter in case) DO {

      /* Alter the parameters by a random amount */
      /* as determined by the delta limits */
      parameters.parameter-value =
         parameters.parameter-value +
         RANDOM(case.parameter-value.delta-low,
                case.parameter-value.delta-hi);

      /* Ensure that the modification leaves */
      /* the gain within the define limits */
      IF parameters.parameter-value >
            case.parameter-value.hi-limit
         parameters.parameter-value =
            case.parameter-value.hi-limit
      ELSE IF parameters.parameter-value <
            case.parameter-value.low-limit
         parameters.parameter-value =
            case.parameter-value.low-limit;
   }
}  /* end Adjust-Parameters-Values */
```

Fig. 9. Parameter adaptation algorithm.

in parameter values occurs, ACBARR makes use of the case for step-by-step modifications. Each case contains a set of modifications which can be performed on the parameters in the system, as well as the minimum and maximum values of each. During each system step, ACBARR will adjust the current parameter values by a value within the delta range for that parameter, as defined in the case. If this adjustment produces a new parameter value which is outside the limits defined by the case, the parameter value is then set to the limit. The algorithm which guides these step-by-step adjustments is shown in Fig. 9.

IV. Evaluation

In order to evaluate the proposed methods, we performed extensive simulations with the ACBARR system to evaluate its performance both qualitatively and quantitatively. Qualitative results were obtained using predefined environments that represented problems that are known to be difficult for reactive control systems. Quantitative results were obtained using several randomly generated environments with different densities and configurations of obstacles.

A. Simulation Environment

The test environment for this research is written in C using the X Windows graphics package. The simulator has been a useful tool for other research in the Mobile Robot Lab at Georgia Tech, including [5], [8], [14], [26], and [32], among others. Results generated in this simulation environment have routinely been demonstrated on actual mobile robots (e.g., [3] and [6]–[8]). Except for minor changes, the present simulator is the same one used in these projects.

In order to facilitate our research, the simulation environment was extended to include both a graphical interface and batch mode. The graphical mode allows us to visually evaluate the progress of a simulated robot while it runs through a

predefined world.[9] The batch mode facility allows us to run several simulations to gather statistics on the system's average performance over a range of environments. The simulation system also includes a world generator that creates random worlds with a starting point, a goal, and a set of obstacles. The number of obstacles generated is controlled by the *obstacle density* requested. For instance, a world with obstacle density 50% will have one-half of the total world area covered by obstacles. To simplify the area calculation, overlap of obstacles is not allowed in the random worlds.

B. Qualitative Evaluation

Reactive control systems have difficulty with local minima. For example, in a box canyon situation, the system, without a global picture of the entire world, does not have the knowledge to "back out" of the canyon and go around it. This has been referred to as the "fly-at-a-window" problem [3]. Usually, a high level of random noise is used to try to kick the system out of the box canyon. However, apart from being inefficient and unpredictable, this method suffers from the problem that such a high level of noise deteriorates performance in other environments, such as "quasi" box canyons where the system could squeeze through the obstacles if obstacle avoidance and noise were suitably low. The adaptive strategies encoded in ACBARR's cases can handle both types of situations without any reconfiguration of the system. ACBARR adapts to its current environment dynamically, using an appropriate level of noise, obstacle avoidance, and so on. The same method can also handle other difficult environments, such as "walls."

1) Performance Evaluation: Figs. 10–12 illustrate sample runs that demonstrate the qualitative improvement in ACBARR, as compared with the unenhanced reactive control system shown in Figs. 2 and 3. The ACBARR system did not fail to find the goal in any test run. Paths chosen in no clutter to low cluttered environments were particularly efficient. ACBARR was able to successfully navigate the standard box canyon problem (Fig. 10), the quasibox canyon problem (a box canyon with an exit, Fig. 11), and the wall environment (Fig. 12).

2) Method Evaluation: In addition to evaluating the performance of the ACBARR system as a whole, several *ablation studies* [15] were also performed in which the system was tested without one or more of its components in order to evaluate their impact. These studies lend insight into why the method of global and local changes employed by ACBARR produces the overall behavior which it exhibits. The system consists of two additions to "pure" reactive control systems—the local adjustments and the global changes. By removing either of these, both, or neither, we create four scenarios. We can then test each of the variations in the same environment to judge performance and thereby judge the contribution of each addition. We chose the box canyon environment for this portion of the evaluation since it represents a fairly difficult, yet realistic world.

Fig. 13 shows the system's performance on the box canyon world with neither local adjustments nor global cases avail-

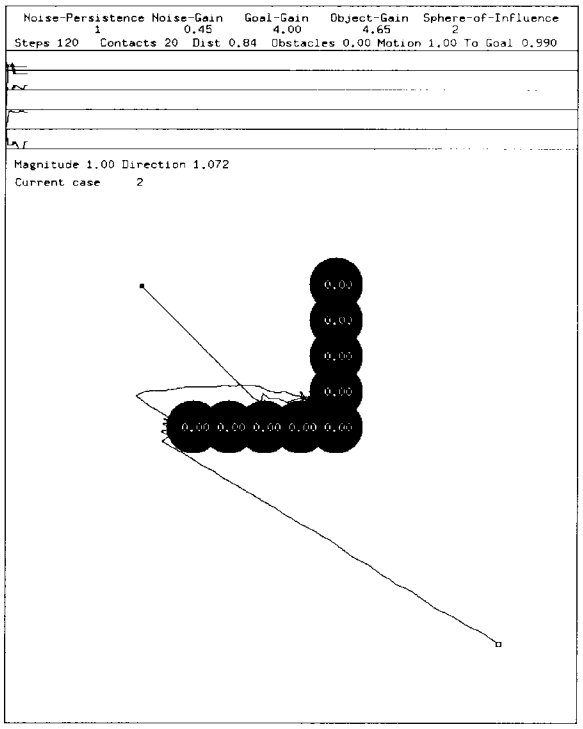


Fig. 10. Box canyon—ACBARR performance.

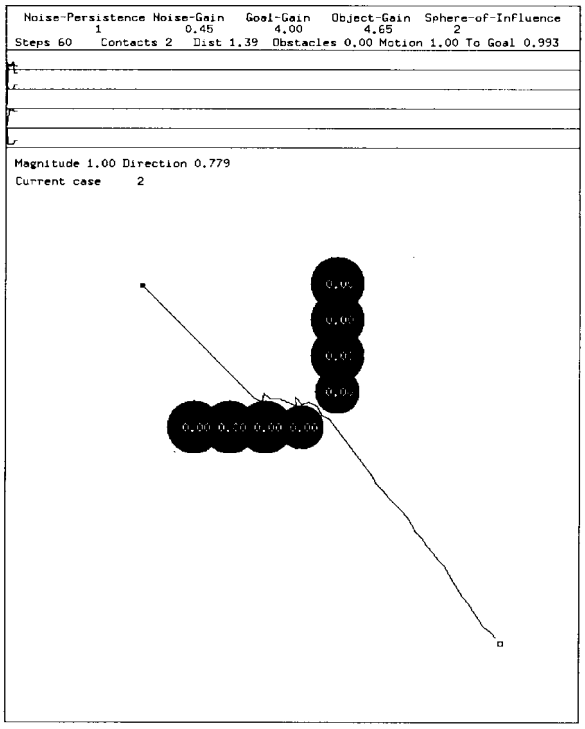


Fig. 11. Quasibox canyon—ACBARR performance.

[9] The simulation window displays the current obstacles as circles, each with varying radius (an example was shown in Fig. 1). As the robot navigates this world, a line is drawn indicating the robot's progress from start to goal. At the top of the window is a set of numbers displaying the current control values. These values are updated each time the ADJUSTMENT MODULE is called (see Fig. 4). This display also indicates the number of steps, total distance traveled, distance to goal, and number of obstacle contacts. Below the numerical display is a set of five line graphs that provide a history of the control values as they are adjusted throughout the run.

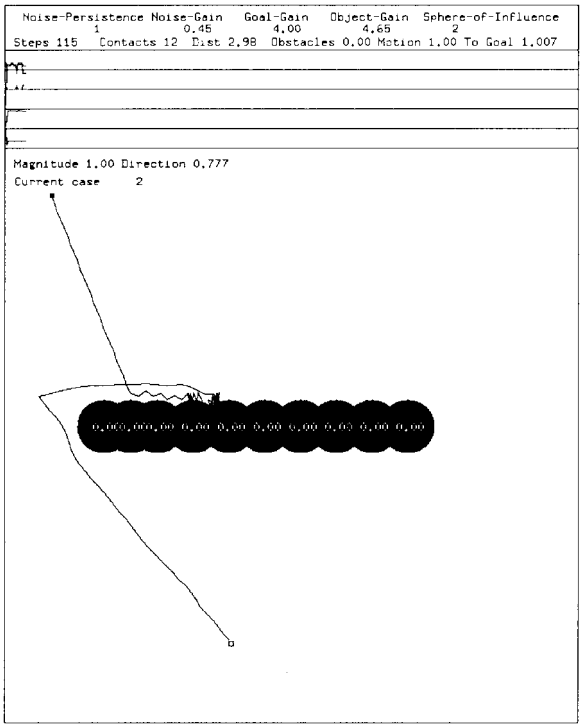


Fig. 12. Wall environment—ACBARR performance.

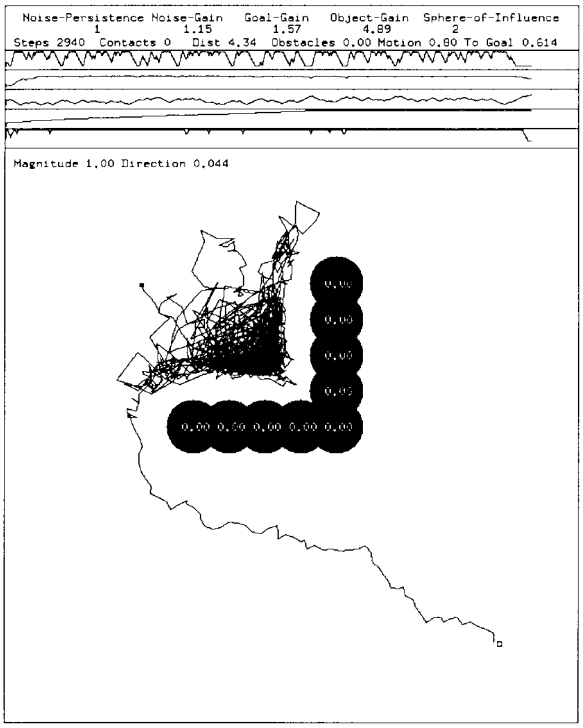


Fig. 14. Box canyon—No cases, but adjustments are permitted.

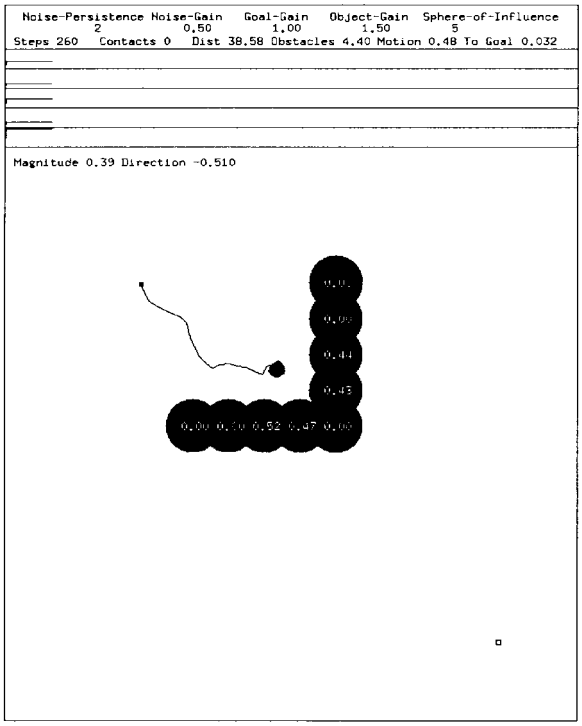


Fig. 13. Box canyon—No cases, no adjustments allowed.

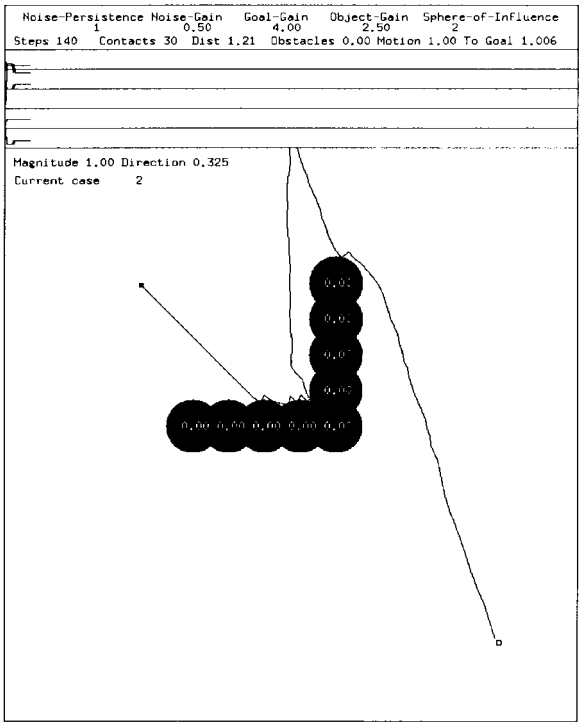


Fig. 15. Box canyon—Cases, but no adjustments are permitted.

able to the system. As can be seen, the robot achieves an *equilibrium point* and stays there. Fig. 14 shows the results of the system being run on the same world, this time with adjustments permitted. The goal is reached, although the path is wasteful. Finally, Fig. 15 presents ACBARR's handling of the box canyon with the case library available but with no local adjustments allowed. The goal is achieved but with a rather nonoptimal path. In particular, the system is unable to adjust its step-by-step performance to avoid running off the visible screen. Thus, a lot of wasted motion is generated. Notice that the robot is unable to adjust its performance as it reapproaches the wall and comes too close to the obstacles.

Also, the overall path is more jagged than that which is generated by the complete system. Comparing these results with those of Fig. 10, which shows the complete ACBARR system's performance, we can see that the system achieves its goal with a fairly good path with the benefits of both adjustments and case knowledge. The cases provide global performance benefits (which explains why adjustments alone produce such an inefficient path in Fig. 14), while the step-by-step adjustments provide "fine-tuning" at a local level, resulting in much smoother paths (explaining the nature of the path seen in Fig. 15).

C. Quantitative Evaluation

Several simulations were performed in batch mode to evaluate the improvement in performance yielded by our case-based method for on-line adaptation. The simulator has the potential for an almost unlimited number of environments with various sizes, numbers and configurations of obstacles. The clutteredness of an environment can be characterized by its *obstacle density*, which is the fraction of available space that is occupied by obstacles. We created 300 random worlds, 100 worlds for each of the three obstacle density levels. These density levels reflected an easy world (10%), a medium cluttered world (25%), and a difficult, fairly cluttered world (50%). Across the 100 worlds were ones where there were many small obstacles making up the density percentage as well as worlds where the indicated density was the result of a few large obstacles. These 300 different environments were an attempt to subject ACBARR to a wide range of possible worlds. Finally, since each run of the system varies from the others due to the randomness of the noise parameters, we ran each simulation a total of five times and averaged the results.

The results are depicted in Figs. 16–19. In the graphs, ACBARR-F is the ACBARR system utilizing a evaluate-on-failure case switching strategy, ACBARR-S is the same system with the evaluate-per-step strategy, and REACTIVE-10 and REACTIVE-50 are systems utilizing "pure" reactive control. The nonadaptive reactive systems were hand-configured to be efficient at navigating in environments with 10% clutter (the REACTIVE-10 system) and 50% clutter (the REACTIVE-50 system) respectively; however, as the results demonstrate, they were less flexible than the ACBARR systems and did not perform as well in environments for which they were not explicitly intended. The improvements were analyzed using statistical methods and shown to be statistically significant.

1) Number of Robot Steps: The steps metric illustrates the speedup in the actual number of robot steps required to reach the goal. The number of steps taken by the various systems is depicted in Fig. 16. The two ACBARR variants are almost identical, with ACBARR-S edging out ACBARR-F in the 50% cluttered world. The two REACTIVE systems had the worst performance, as shown in the figure. REACTIVE-10 performed well in worlds with 10% clutter, for which it was designed, but deteriorated significantly on highly cluttered words. For 50% cluttered worlds, this system follows paths of over 1125 steps on average as compared with the approximately 40- to 95-step paths found by the ACBARR

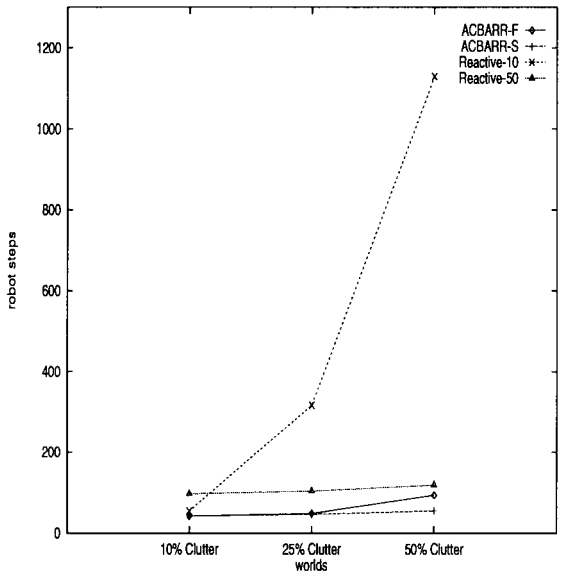


Fig. 16. Number of robot steps required to reach goal.

systems. REACTIVE-50 performed reasonably well on 50% cluttered worlds as well as less cluttered worlds, but the ACBARR systems were about twice as good (specifically, the REACTIVE-50 paths were on an average 2.15 times as long as the ACBARR-S paths in 50% cluttered worlds). A t-test analysis showed that the improvement of ACBARR-S over REACTIVE-50 is statistically significant $(t(499) = 29.69, p \ll 0.01)$.

The results show that ACBARR's methods allow it to perform well on less cluttered worlds, and are flexible enough to carry over to highly cluttered worlds without significant degradation in performance. It remains as good as or better than a purely reactive system tailored to each type of environment. Intuitively, this is to be expected since ACBARR is able to switch strategies and select the appropriate parameter values (which are typically fixed in a purely reactive system) for different situations.

2) Distance: Another useful metric for evaluating the performance of the system is the ratio of the actual distance traveled during the navigational task (the "path length") to the straight-line distance between the start position and the goal. This metric gives us an idea of how much "wasted motion" the system performed. Ideally, this value should be 1.0 in a world with no obstacles. Obviously, if the environment is cluttered, the ratio will in general be greater than 1.0, no matter how good the navigational system is.

As shown in Fig. 17, in the 10% cluttered worlds, all four variants performed the task with little wasted motion, although both ACBARR systems were slightly better than the better of the reactive control systems. When we consider the 25% cluttered worlds, however, we see that REACTIVE-10, the reactive system that was configured for 10% worlds, is beginning to lose in this area, navigating along paths which were over three times the length they needed to be. REACTIVE-50 performs better but still not as well as the ACBARR systems. In the 50% cluttered worlds, REACTIVE-10 reached the goal along paths which were almost ten times as long as

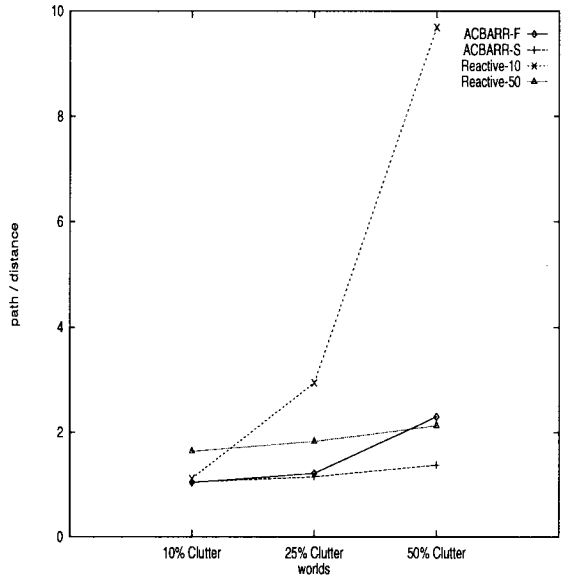


Fig. 17. Robot path length to goal over actual distance to goal.

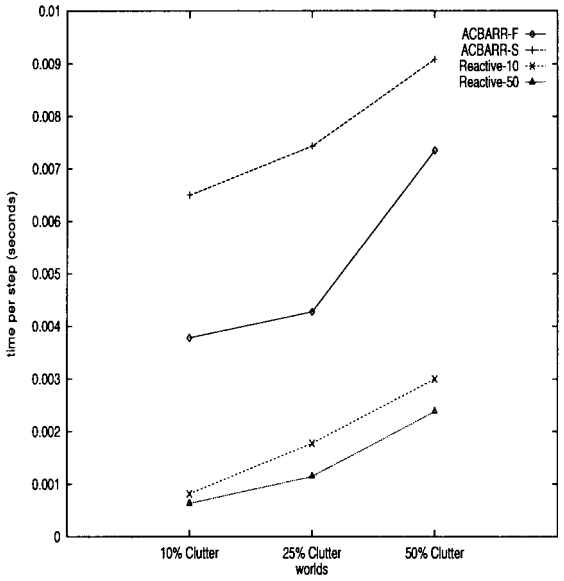


Fig. 18. Time per robot step.

they needed to be. REACTIVE-50, which was hand-coded for such environments, performed better, but still does not match the performance of the ACBARR-S system (specifically, the REACTIVE-50 paths were on an average 1.55 times as long as the ACBARR-S paths). A t-test analysis showed that this improvement is statistically significant $(t(499) = 20.58, p \ll 0.01)$. Notice also that at this level of obstacle density, ACBARR-F is worse than ACBARR-S. Potential case switching on every step allows ACBARR-S to find shorter paths than ACBARR-F can.

3) Time per Step: The time per step metric is another interesting metric since it allows us to evaluate the overhead of the extra processing in the ACBARR systems. We measured the average time the systems took in order to make each step of the journey. As intuition would predict, Fig. 18 shows that ACBARR-S took the longest amount of time per step. This is due to the performance evaluation and potential case switching taking place at each step. ACBARR-F was the second slowest, with the two REACTIVE systems being the fastest. While the differences seem minute, if a navigational task were to take hundreds to thousands of steps, the cumulative effect of case retrieval, evaluation, and switching could begin to have an impact on overall system time in a simulated system. This effect is evaluated in the following experiment.

4) Time to Completion: The time metric illustrates the speedup in the total navigational time, as measured by the actual time taken to reach the goal (Fig. 19). In the 10% worlds, although all four systems performed well, the REACTIVE-10 system took the least time to get to the goal. This advantage of the REACTIVE-10 system breaks down in more cluttered worlds, where the additional processing performed by the ACBARR systems begins to pay off. The REACTIVE-50 system was designed for highly cluttered worlds and performs somewhat better than the ACBARR systems in these worlds as well as less cluttered worlds.

It should be noted, however, that the simulated time metric is not a realistic indicator of performance of a physical robot

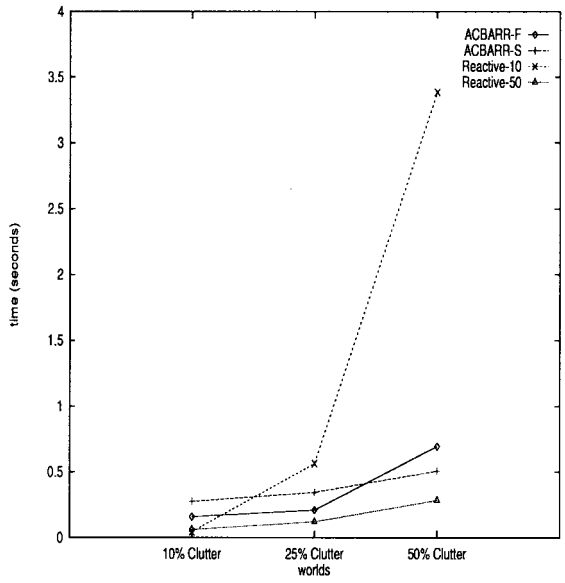


Fig. 19. Total time for robot to move to goal.

in the real world. In a simulated environment, perception and movement are instantaneous. However, in the real world, sensing is at least an order of magnitude slower than the ACBARR computations (which are on the order of 5 ms to 10 ms per cycle, as shown in Fig. 18). Physical movement of the robot is also relatively slow. The results show that the computation overhead, in contrast, is much less than an order of magnitude (specifically, in simulation, ACBARR-S takes on an average 1.78 times as long as REACTIVE-50 in 50% cluttered worlds to navigate from the starting point to the goal; $t(499) = 22.30, p \ll 0.01$). Since ACBARR's paths are considerably shorter (Fig. 17) and require less robot steps (Fig. 16), the processing overhead in ACBARR is negligible compared to the improved performance that results from the better navigational paths that are created.

D. Discussion

Considering the graphs together allows us to draw conclusions about the overall performance of ACBARR as seen against the reactive control system which it augments. Both ACBARR versions take some time to "ponder" each move they are going to make. If perception and movement are instantaneous and there are few obstacles, a purely reactive system may be able to navigate to the goal in less overall time than the ACBARR systems. However, its path will be a little more jagged than the ACBARR systems, and it will use more steps to reach the goal. If the path quality or the actual number of steps taken is of importance, the ACBARR systems are better even at lower obstacle densities. This is the case with physical robots in which perception and movement time far outweigh the computation time required to adapt schema parameters (although this article presents only simulation results). Furthermore, the computations necessary to derive the next step can be performed while the robot is executing the motions for the previous step.

The benefits of the extra time per step taken by the ACBARR systems are revealed in more cluttered environments. The paths created by the ACBARR systems in cluttered worlds are much shorter than the purely reactive systems, and they require fewer time steps to complete the navigation task. Notice that, in these worlds, ACBARR-F is slightly faster than ACBARR-S; both systems are faster than REACTIVE-10, though not quite as fast as REACTIVE-50.

The most convincing evidence for the case-based ACBARR systems can be seen in the highly cluttered worlds. Compared to ACBARR-F and ACBARR-S, the REACTIVE-10 system performs extremely badly in 50% cluttered worlds, being beaten in length of path, time to completion, and number of steps required. Also, it is in this class of worlds that ACBARR-S's higher time per step value begins to benefit the system even in a simulated environment. While it is possible to design a purely reactive system (REACTIVE-50) that will perform better along the simulated time metric, ACBARR outperforms that system along the distance and steps metrics. To compare the two ACBARR case-switching strategies, note that while ACBARR-F continues to perform respectably in highly cluttered worlds, ACBARR-S manages to complete the navigational task in less time, creating a better path, while using fewer steps.

The above results are statistically significant and consistent with the commonly held belief that more analysis will lead to a better result. If the result is better enough so that it counteracts the additional overhead, the extra analysis is worth it. This brings up an interesting point with regards to future enhancements of the system. As the case library grows and becomes more complex, the amount of time needed to perform a case switch will also increase. This means that the disparity between the two case switching strategies in terms of time needed per step will increase. We predict that future versions of the system with more complex cases will cause the evaluate-on-failure strategy to become the clear method of choice.

It should be noted that while ACBARR does improve performance over the nonadaptive reactive systems that it was compared with, the actual quantitative improvement shown in the graphs depends on the particular parameter settings used for the reactive systems. However, one of the benefits of our approach is flexibility, in the sense that the system's parameters do not need to be reconfigured in different environments. As is evident from the graphs, ACBARR performs extremely well across a wide range of environments, from relatively uncluttered to highly cluttered, with a wide range of obstacle configurations, from random to box canyons to walls. For example, over the entire set of experiments summarized in Fig. 16, REACTIVE-50 took an average of 223% as many steps as ACBARR-S to navigate the same environment; this improvement is realized over the entire range of environments tested. Similarly, from Fig. 17, the paths found by REACTIVE-50 were 55% to 60% longer than those found by ACBARR-S.

V. Limitations and Future Work

A. Distribution of Cases Used by ACBARR

In our empirical tests, we noticed an interesting result that is worth mentioning. When using the failure-driven case evaluation and switching method, the system would generally use only a subset of its stored possible strategies during test runs. We studied this behavior further by running more extensive simulations on various random worlds with ACBARR-S. We found that out of our ten cases, only five were ever used: WALL-CRAWLING, HUGGING, CLEAR-FIELD, SHOOTING, and BALLOONING. Part of the reason for this is methodological; the cases were added incrementally as new situations were discovered which warranted new behavior patterns. We began with only two cases, BALLOONING and SQUEEZING, and built the library up from that point. The problem, then, is that some of our later cases made earlier cases superfluous. For example, the MAXXED cases was added to ACBARR to handle a specific environmental problem. Later, when WALL-CRAWLING was added, MAXXED ceased to be used as often as it was earlier.

The solution to the methodological problem is to go back and revise the case library to exclude redundancies. However, it is still likely that the system would use some of its cases more often than others. There are two reasons for this. First, ACBARR-F only switches cases if the current case is failing in some way. This means that the best available strategy is not always in place. If the current case is "good enough," the system will not bother to switch to a better one. For example, if the system initially chooses to make use of the CLEAR-FIELD case, it will continue to do so until there is a clear failure. In order to determine whether this was indeed the explanation for this behavior, we compared this with the alternative case switching method in which the need for a new case is evaluated every H_{steps} steps. Many more cases are utilized if this method is used. The second explanation for why only a subset of strategies were being used is the robustness of several of the strategies involved. In particular, CLEAR-FIELD, HUGGING, and WALL-CRAWLING are especially robust and can account for the majority of behaviors noted in the system. The only way to force ACBARR to reconsider

all its cases is to set H_{steps} to 1, in other words, to use the ACBARR-S method. This results in higher overhead which, as discussed earlier, only pays off in very highly cluttered environments.

B. Single Strategy or Multiple Cases?

In our research, we assumed that a set of strategies would be needed to deal with the range of problem situations that are possible in the ACBARR world. However, it is possible that a single reactive control strategy could be developed which would be able to handle the same range of situations which ACBARR can handle. The issue, then, is: why add additional processing to the system? There are several reasons. First, it would require considerable effort to develop such a strategy, and no such strategy has been proposed thus far. ACBARR enables a system to produce respectable performances with less than optimal cases. Second, if such a strategy became available, it could easily be added to ACBARR's case library. If the evaluate-on-failure strategy were employed, then this powerful strategy would stay in place until it failed (if it failed). Third, since worlds are dynamic, even if a good strategy that covered a wide range of situations was found, the system might need to switch to a different strategy in an unexpected situation. Finally, ACBARR can provide a framework to test the effectiveness of any such strategy. A fairly robust strategy could be developed and then added to the ACBARR system, which would then be allowed to operate in hundreds of simulated worlds. These simulations could then be examined to see if there were any environmental situations which caused the strategy being tested to be switched out. This information would then guide the researcher, if he or she wished, to further improve the strategy. Thus our claim, which is supported by the evidence we have discussed, is that ACBARR adds robustness and methodological power to any pure reactive control system, regardless of its level of existing behavior.

C. The Case Library

An important research issue at this point is where the set of cases in the library comes from. For now, these cases are coded by hand. This is not the optimal solution for two reasons. One, it allows human biases to enter the process. To illustrate this point, consider our own experiences. At first, we believed that BALLOONING and SQUEEZING were relatively robust, general-purpose strategies. As pointed out earlier, however, these did not turn out to be the strategies used most often by the system. Luckily, there is enough variety within the hand-created cases to allow the system a relatively comprehensive selection, and the empirical evaluations demonstrate that the set of cases we have identified is indeed a good one. Yet, the nagging question remains: Is there a behavior even more robust which we have overlooked? A second potential problem is that a completely novel situation unseen by the human teacher may not be handled in the best way. There is still the possibility that ACBARR will fail in certain environments, although no such failures were identified in the extensive empirical testing. If the system had the ability to learn its own cases, this potential problem could be alleviated. At the very least, the system needs to be able to add new cases to an already existing library; for some applications, it may be desirable to produce a system which could learn all of its cases from scratch. We are currently developing a system which is capable of automatic case learning through navigational experiences (see [35]); related work by other researchers was discussed in Section II-C.

D. Implementation on a Physical System

Another area of future work involves the actual implementation of the ACBARR system on a real robot. The work to date has been restricted to the simulator. The transfer to a physical robot should not be difficult, in part because AuRA is already implemented on a physical system and previous results from the simulated environment have been shown to transfer to physical robots. Every effort was made in the system so that it performed in a way suitable for both a simulated world and the real world. Part of the remaining challenge is to find sufficiently varied domains to test these ideas effectively using the robot.

VI. Conclusion

The objective of our research effort is to develop mechanisms for learning and adaptation that can be used by an intelligent agent to learn from its experiences in a complex and dynamic environment, and to develop corresponding mechanisms for planning and action in such environments that support such learning and adaptation. The methods presented in this article were developed as part of this ongoing effort, and focus specifically on the issue of adaptive reactive control. Case-based reasoning allows a reactive system derive the benefits of higher-level reasoning without sacrificing the real-time response and fast performance. It allows the system to adapt to its environment dynamically, resulting in flexibility in performance across a wide range of environmental conditions. Thus, combining case selection and behavioral adaptation based on environmental demands with traditional reactive robotic control systems should theoretically lead to better performance, and the empirical data supports this claim as well.

The methods presented in this article are fully implemented in the ACBARR system. By adding basic environmental data to the system, we have realized substantial improvements in its performance without sacrificing the inherent benefits of reactive control. Although the ACBARR system is not a pure reactive control system as normally defined, it combines the best features of that paradigm with the benefits of case-based reasoning. The performance of the system is tightly coupled with the adequacy of the cases in its library. As pointed out, the cases currently in use have proven to be extremely robust, making failure in new environments less likely. This results in ACBARR being a highly efficient, adaptive control system.

References

[1] P. Agre and D. Chapman, "Pengi: An implementation of a theory of activity," in *Proc. Sixth Nat. Conf. Artificial Intelligence*, 1987, pp. 268–272.

[2] S. Amarel, "On representations of problems of reasoning about actions," *Mach. Intell.*, vol. 3, 1968. Reprinted in *Readings in Artificial Intelli-*

gence, B. L. Webber and N. J. Nilsson, Eds. Palo Alto, CA: Tioga, 1981, pp. 2–22.
[3] R. C. Arkin, "Motor schema-based mobile robot navigation," *Int. J. Robot. Res.*, vol. 8, pp. 92–112, Aug. 1989.
[4] ——, "Integrating behavioral, perceptual, and world knowledge in reactive navigation," *Robot. Auton. Syst.*, vol. 6, pp. 105–122, 1990.
[5] R. C. Arkin, T. Balch, and E. Nitz, "Communication of behavioral state in multi-agent retrieval tasks," in *Proc. IEEE Int. Conf. Robotics Automation*, May 1993, vol. 1.
[6] R. C. Arkin *et al.*, "Buzz: An instantiation of a schema-based reactive robotic system," in *Proc. Int. Conf. Intelligent Autonomous Systems*, 1993.
[7] R. C. Arkin, R. R. Murphy, M. Pearson, and D. Vaughn, "Mobile robot docking operations in a manufacturing environment: Progress in visual perceptual strategies," in *Proc. IEEE Int. Workshop Intelligent Robots Systems*, 1989.
[8] T. Balch and R. C. Arkin, "Avoiding the past: A simple but effective strategy for reactive navigation," in *Proc. IEEE Int. Conf. Robotics Automation*, Atlanta, GA, May 1993.
[9] S. W. Bennett, "Reducing real-world failures of approximate explanation-based rules," in *Proc. 7th Int. Conf. Machine Learning*, June 1990, pp. 226–234.
[10] R. Brooks, "A robust layered control system for a mobile robot," *IEEE J. Robot. Automat.*, vol. RA-2, pp. 14–23, Aug. 1986.
[11] ——, "Intelligence without representation," Res. Paper, Mass. Inst. Technol., Artif. Intell. Lab., Cambridge, 1988.
[12] ——, "A robot that walks: Emergent behaviors from a carefully evolved network," in *Proc. IEEE Conf. Robotics Automation*, Scottsdale, AZ, May 1989, pp. 692–694.
[13] S. A. Chien, M. T. Gervasio, and G. F. DeJong, "On becoming decreasingly reactive: Learning to deliberate minimally," in *Proc. 8th Int. Workshop Machine Learning*, Chicago, IL, June 1991, pp. 288–292.
[14] R. J. Clark, R. C. Arkin, and A. Ram, "Learning momentum: On-line performance enhancement for reactive systems," in *Proc. IEEE Int. Conf. Robotics Automation*, Nice, France, May 1992, pp. 111–116.
[15] P. R. Cohen and A. E. Howe, "How evaluation guides AI research," *AI Mag.*, vol. 9, pp. 35–43, Winter 1988.
[16] R. J. Firby, "Adaptive execution in complex dynamic worlds," Ph.D. dissertation, Dept. Comput. Sci., Yale Univ., New Haven, CT, Jan. 1989, Res. Rep. YALEU/CSD/RR 673.
[17] K. J. Hammond, *Case-Based Planning: Viewing Planning as a Memory Task, Perspectives in Artificial Intelligence*. Boston, MA: Academic, 1989.
[18] M. Kadanoff, F. Benayad-Cherif, A. Franklin, J. Maddox, L. Muller, B. Sert, and H. Moravec, "Arbitration of multiple control strategies for mobile robots," in *Proc. SPIE, Vol. 727: Mobile Robots*, W. Wolfe and N. Marquina, Eds., Bellingham, WA, 1986, pp. 77–84.
[19] L. P. Kaelbling, "An architecture for intelligent reactive systems," Tech. Note 400, SRI Int., Oct. 1986.
[20] L. P. Kaelbling, "Foundations of learning in autonomous agents," *Robot. Auton. Agents*, vol. 8, no. 1–2, pp. 131–144, 1991.
[21] J. L. Kolodner, "An introduction to case-based reasoning," Tech. Rep. GIT-ICS-90/19, School Inform. Comput. Sci., Georgia Inst. Technol., Atlanta, GA, 1990.
[22] ——, *Case-Based Reasoning*. San Mateo, CA: Morgan Kaufmann, 1993.
[23] L. Kopeikina, R. Brandau, and A. Lemmon, "Case-based reasoning for continuous control," in *Proc. Workshop Case-Based Reasoning*, Clearwater Beach, FL, May 1988, Defense Adv. Res. Projects Agency, pp. 250–259.
[24] B. J. Kuipers and Y.-T. Byun, "A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations," *Robot. Auton. Syst.*, vol. 8, no. 1–2, pp. 47–63, 1991.
[25] L.-J. Lin, "Scaling up reinforcement learning for robot control," in *Proc. 10th Int. Conf. Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1993, pp. 182–189.
[26] D. C. MacKenzie and T. R. Balch, "Making a clean sweep: Behavior based vacuuming," in *Working Notes AAAI Fall Symp. Instantiating Real-World Agents*, 1993.
[27] P. Maes and R. A. Brooks, "Learning to coordinate behaviors," in *Proc. 8th Nat. Conf. Artificial Intelligence*, Boston, MA, Aug. 1990, pp. 796–802.
[28] S. Mahadevan, "Enhancing transfer in reinforcement learning by building stochastic models of robot actions," in *Proc. 9th Int. Workshop Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1992, pp. 290–299.
[29] T. M. Mitchell, "Becoming increasingly reactive," in *Proc. Nat. Conf. Artificial Intelligence*, Boston, MA, Aug. 1990, pp. 1051–1058.
[30] D. Payton, "An architecture for reflexive autonomous vehicle control," in *Proc. IEEE Conf. Robotics Automation*, 1986, pp. 1838–1845.
[31] A. Ram, "Indexing, elaboration and refinement: Incremental learning of explanatory cases," *Mach. Learn.*, vol. 10, pp. 201–248, 1993.
[32] A. Ram, R. C. Arkin, G. Boone, and M. Pearce, "Using genetic algorithms to learn reactive control parameters for autonomous robotic navigation," *Adaptive Behav.*, vol. 2, no. 3, pp. 277–305, 1994.
[33] A. Ram, R. C. Arkin, K. Moorman, and R. J. Clark, "Case-based reactive navigation: A case-based method for on-line selection and adaptation of reactive control parameters in autonomous robotic systems," Tech. Rep. GIT-CC-92/57, College Comput., Georgia Inst. Technol., Atlanta, GA, 1992.
[34] A. Ram and J. C. Santamaria, "Continuous case-based reasoning," in *Proc. AAAI Workshop Case-Based Reasoning*, Washington, DC: AAAI Press, D. B. Leake, Ed., Tech. Rep. WS-93-01, July 1993, pp. 86–93.
[35] ——, "Multistrategy learning in reactive control systems for autonomous robotic navigation," *Informatica*, vol. 17, no. 4, pp. 347–369, 1993.
[36] C. L. Ramsey and J. J. Grefenstette, "Case-based anytime learning," in *Proc. AAAI Workshop Case-Based Reasoning*, 1994.
[37] A. M. Segre, *Machine Learning of Robot Assembly Plans*. Norwell, MA: Kluwer, 1988.
[38] B. F. Skinner, *About Behaviorism*. New York: Knopf/Random House, 1974.
[39] M. Veloso and J. Carbonell, "Derivational analogy in prodigy: Automating case acquisition, storage and utilization," *Mach. Learn.*, vol. 10, pp. 249–278, 1993.

Ashwin Ram (S'94–A'95) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, New Delhi, in 1982, the M.S. degree in computer science from the University of Illinois, Urbana-Champaign, in 1984, and the Ph.D. degree from Yale University, New Haven, CT, for his dissertation on "Question-Driven Understanding: An Integrated Theory of Story Understanding, Memory, and Learning" in 1989.

He is an Associate Professor with the College of Computing, an Associate Professor of Cognitive Science, and an Adjunct Professor, School of Psychology, Georgia Institute of Technology, Atlanta. His research interests lie in the areas of machine learning, case-based reasoning, natural language understanding, and cognitive science, and he has several research publications in these areas. He is co-editor of the book *Goal-Driven Learning* (MIT Press/Bradford Books).

Dr. Ram is a member of the editorial boards of the *International Journal of Applied Intelligence* and the *Journal of the Learning Sciences*, and an Associate of *Behavioral and Brain Sciences*. He recently co-chaired the Sixteenth Annual Conference of the Cognitive Science Society.

Ronald C. Arkin (M'90–SM'91) received the B.S. degree from the University of Michigan, Ann Arbor, the M.S. degree from the Stevens Institute of Technology, Hoboken, NJ, and the Ph.D. degree in computer science from the University of Massachusetts, Amherst, in 1987.

In 1987, he joined the College of Computing, Georgia Institute of Technology, Atlanta, where he is Associate Professor and Director of the Mobile Robot Laboratory. His research interests include reactive control and action-oriented perception for the navigation of mobile robots and unmanned aerial vehicles, robot survivability, multi-agent robotic systems, and learning in autonomous systems. He has published over 80 technical publications in these areas. His funding sources have included the National Science Foundation, ARPA, U.S. Army, Savannah River Technology Center, and the Office of Naval Research. He is an Associate Editor of *IEEE Expert* and a member of the Editorial Board of *Autonomous Robotics*.

Dr. Arkin is a member of AAAI and ACM.

Kenneth Moorman received the B.A. degree in computer science and mathematics from Transylvania University, Lexington, KY, in 1991. He is currently pursuing the Ph.D. degree at the College of Computing, Georgia Institute of Technology, Atlanta. He holds a fellowship from the Fannie and John Hertz Foundation.

His research interests include case-based reasoning, creativity, and natural language comprehension, all within real-world contexts.

Mr. Moorman is a member of the International Reading Association, the Cognitive Science Society, and AAAI.

Russell J. Clark (S'80–M'81) received the B.S. degree in mathematics and computer science in 1987 from Vanderbilt University, Nashville, TN, and the M.S. and Ph.D. degrees in information and computer science from the Georgia Institute of Technology, Atlanta, in 1992 amd 1995, respectively.

He is an Assistant Professor at the University of Dayton, Dayton, Ohio. His research interests include interoperable systems using multiple protocols, scalable network services using multicast distribution, and automated network management systems.