Case-based reactive navigation:
A case-based method for on-line selection and adaptation of reactive control parameters in autonomous robotic systems*

Ashwin Ram

Ronald C. Arkin

Kenneth Moorman

Russell J. Clark

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
{ashwin,arkin,kennethm,rjc}@cc.gatech.edu

Abstract

This article presents a new line of research investigating on-line learning 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.

*Technical Report GIT-CC-92/57, College of Computing, Georgia Institute of Technology, Atlanta, Georgia, 1992.

Contents

1 Motivation

Autonomous robotics has received a great deal of attention in the recent past. However, most current robotics systems are severely limited when performing in complex and dynamic environments in the real world. It requires a great deal of 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, and learn from experiences, is crucial to adequate performance and survivability in the real world.

Machine learning and autonomous robotics have traditionally been two separate subfields in artificial intelligence, with little cross-fertilization over the past several years. Although several techniques for learning of control strategies have been developed in the machine learning community, much of this work deals with simple, simulated, "blocks world" environments that have little to do with the real world.[1] However, the two fields are now sufficiently mature to permit research which investigates theoretical constraints imposed by the real world on the nature of the learning task. Such an effort would contribute to both theories of learning as well as to theories of autonomous robotics and control, and would also be a crucial prerequisite in the development of actual autonomous agents for a variety of real-world application areas.

This article presents a case-based 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. The method is applied to a robot navigation task in cluttered environments, and evaluated empirically through extensive simulation.

[1]A notable exception is recent work on reinforcement learning (see discussion below). In contrast, the research presented in this paper is based on work in case-based reasoning.

2 Overview

Reactive robotic control systems [Arkin, 1989; Brooks, 1986; Kaelbling, 1986; Payton, 1986] have produced impressive results in the area of generating intelligent robotic action. Unlike traditional approaches to robot control, these systems typically decompose actions into simple behaviors in order to produce rapid real-time response to the environment. Generally, however, the behaviors available to such 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 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 sets of behaviors, called *behavior assemblages*, to represent appropriate collections of cooperating behaviors for complex environments, and that *behavior adaptation* be used 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 behavior based on immediate past experience. While useful, this is only a local response to the problem. A more global solution is to have the system select completely new assemblages of behaviors based on the current environment in which it finds itself. A robust system should be able to learn about and adapt to its environment dynamically in both these ways.

Our research incorporates both *behavior adaptation* and *behavior switching* into a reactive control framework. At the local level, this is accomplished by allowing the system to adapt its current behavior in order to build *momentum*. If something is working well, the system continues doing it and tries doing it a little bit harder; conversely, if things are not proceeding well, the system attempts something a little different. This technique allows the system to fine tune its current behavior patterns to the exact environment in which it finds itself [Clark *et al.*, 1992]. For example, if the robot has been in an open area for a period of time and has not encountered any obstacles, it picks up speed and does not worry as much about obstacles. If, on the other hand, it is in a cluttered area, it lowers its speed and treats obstacles more seriously. For behavior-based reactive systems, this readily translates into altering the schema gains and parameters continuously, provided the system has a method for determining the appropriate modifications. Our system uses a case-based reasoning method to retrieve behavior modification rules. These rules are then used incrementally to alter the gain and parameter values based on current environmental conditions and past successes.

The other method for behavior modification in our system is at a more global level. If the system is currently acting under the control of an assemblage of behaviors which are no longer suited to the current environment, it selects a new assemblage based on what the environment is now like. Applying this to the above example, assume that the robot is in a very cluttered environment and is employing a conservative assemblage of motor behaviors. It then "breaks out" of the obstacles and enters a large open field (analogous to moving from a forested area into a meadow). If only local changes were allowed, the robot would eventually adjust to the new environment. However, by allowing a global change to take place, the system needs only to realize that it is in a radically new environment and to select a new assemblage of motor behaviors, one better suited to the new surroundings [Moorman and Ram, 1992]. Interestingly, case-based reasoning is used to realize this type of modification as well.

Assemblages of behaviors represent *cases*, or standard scenarios known to the system, that can be used to guide performance in novel situations. As in a traditional case-based reasoning system [Hammond, 1989; Kolodner, 1990; Kolodner, 1992; Riesbeck and Schank, 1989], a case is used to propose a plan or a solution (here, a behavior assemblage) to the problem (here, the current environmental configuration). However, our method differs from the traditional use of case-based reasoning in an important respect. A case in our system is also used to propose a set of *behavior adaptations*, rather than merely the behaviors themselves. This allows the system to use different "strategies" in different situations. For example, the system might use a "cautious" strategy in a crowded environment by gradually slowing down and allowing itself to get closer to the surrounding obstacles. In order to permit this, strategies suggest boundaries on behavioral parameters rather than precise values for these parameters. Cases are used both to suggest behavior assemblages as well as to perform dynamic (on-line) adaptation of the parameters of behavior assemblages within the suggested boundaries.

The research presented here contributes to reactive control for autonomous robots in the following ways. One, we propose a method for the use of assemblages of behaviors tailored to particular environmental demands, rather than of single or multiple independent behaviors.

Two, our system can select and adapt these behaviors dynamically without relying on the user to manually program the correct behavioral parameters for each navigation problem. Finally, our system exhibits considerable flexibility over multiple domains. For example, it performs well in uncluttered worlds, highly cluttered worlds, worlds with box canyons, and so on, without any reconfiguration.

The research also contributes to machine learning, and to case-based reasoning in particular, 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 a solution to fit the current problem. Two, the method represents a novel variation on case-based reasoning for a new kind of task, one that requires continuous, on-line performance. 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 and adaptation strategies in many case-based reasoning systems. Finally, unlike traditional case-based reasoning systems which rely on deep reasoning and analysis (e.g., [Hammond, 1989]), and unlike other machine learning "augmentations" to reactive control systems which fall back on non-reactive reasoning (e.g., [Chien *et al.*, 1991]), 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 and both qualitative and quantitative results from several simulations are presented. ACBARR has been found to be robust, performing well in novel environments. Additionally, it is able to navigate through several "hard" environments, such as box canyons, in which traditional reactive systems would perform poorly.

3 Background and related research

Before presenting the technical details of our system, we discuss previous research in reactive control and in machine learning that forms the basis for our work.

3.1 Perception and reactive control

Reactive control [Arkin, 1989; Brooks, 1986; Brooks, 1989; Kaelbling, 1986; Payton, 1986] 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. A key facet of this approach lies in the massive reduction in computational load afforded by the distributed mechanisms of schema-based reactive control. This enables real-time operation of an autonomous robot.

"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 [Skinner, 1974]. Mentalistic (representational) structures are denounced and the robot reacts to the immediacy of sensory information in a very low-level non-cognitive manner. Complex behaviors emerge as a combination of simple low-level responses to the rich variety of stimuli the world affords. Typically this involves decomposition of tasks into a collection of distributed parallel subtasks. Further, sensor data is normally channeled directly to the individual subtasks, reducing significantly the computational demand typically found in navigational regimes requiring world model building.

There are many representative examples of this form of navigation, a few of which will be described here. Brooks' subsumption architecture [Brooks, 1986] has demonstrated robust navigation for mobile vehicles in dynamically changing domains. It is a layered architecture, well-adapted for hardware implementation [Brooks and Connell, 1986]. It has been used in a wide range of robots, including legged ones [Brooks, 1989]. There is a deliberate avoidance of world modeling which is captured by the statement that *the world is its own best model* [Brooks, 1988].

Payton has described a collection of motor responses that are termed "reflexive behaviors" [Payton, 1986]. These behaviors react directly to sensory information yielding intelligent emergent behavior. Payton, Brooks, 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. Only one behavior dominates the vehicle at any time, although the dominant behavior can change frequently in rapid response to environmental sensing. Earlier work by Kadonoff also employs an arbitration scheme [Kadanoff *et al.*, 1986]. Kaelbling has developed a reactive architecture [Kaelbling, 1986] 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. Firby has developed a different form of reactive control by utilizing modules called RAPs (Reactive Action Packages) which encapsulate tasks for a robot [Firby, 1989]. Situation-driven execution via goal satisfaction is the predominant mode of operation. Agre and Chapman in their PENGI system [Agre and Chapman, 1987] 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 [Arkin, 1989] 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; 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 [Amarel, 1968] 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 remainder of this section answers the question as to how to represent this knowledge.

3.2 Motor schemas

Our system is based on the Autonomous Robot Architecture (AuRA) that is implemented on the Georgia Institute of Technology autonomous robots, George, Ren and Stimpy, three Denning Mobile vehicles. 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. A partial listing of some of the available motor schemas for our robot include:

- MOVE-AHEAD: Directs the robot to move in a general compass direction.
- MOVE-TO-GOAL: Directs movement towards a discernible goal.
- AVOID-STATIC-OBSTACLE: A repulsion directed away from an observed barrier to motion.
- STAY-ON-PATH: The robot moves toward the center of a path.
- DOCKING: Combines aspects of ballistic and controlled motion to achieve a safe trajectory for mating with a workstation.
- NOISE: A random process used for wander behavior and handling problems with local minima.
- MOVE-UP: Move uphill on undulating terrain.
- MOVE-DOWN: Move downhill on undulating terrain.
- MAINTAIN-ALTITUDE: Follow isocontours on undulating terrain.
- ESCAPE and DODGE: To avoid threatening objects.
- AVOID-PAST and PROBE: To explore new areas.

Each of these schemas is instantiated as separate asynchronous computing agents with parameters reflecting current world knowledge. The computations for each schema are very simple, usually involving a couple of additions or subtractions and at most one or two multiplications or divisions (with the exception of docking which includes a transcendental function). It should be noted that the entire potential field is never computed by the robot;

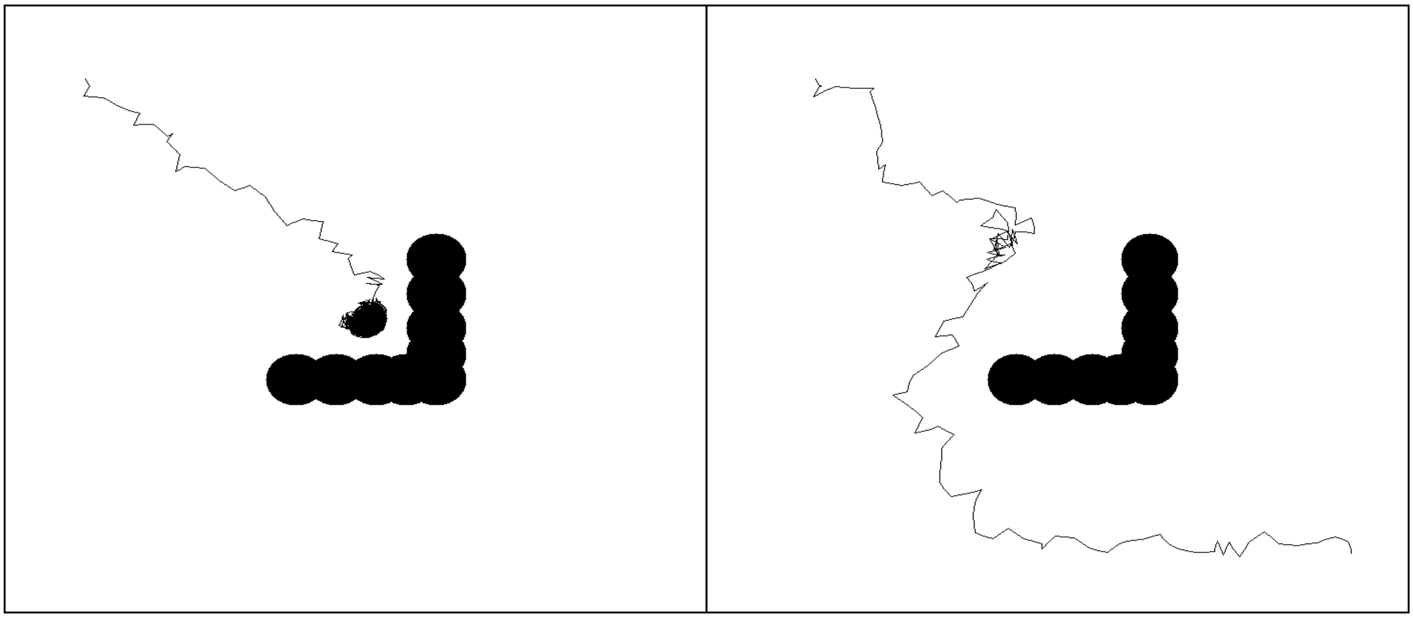

Figure 1: Box Canyon Runs a) Sphere of influence = 10 b) Sphere of influence = 35

only the point where the robot is currently located needs to be computed. Thus each process is performing relatively simple mathematical operations, outputting a single vector expressing the robot's desired motion for that behavior. The output of each primitive motor schema is combined using vector summation and normalization (keeping the resultant vector within the constraints of the actual robot's capabilities). This simple process can result in quite complex trajectories and behaviors as illustrated in the simulations and experiments reported in [Arkin, 1989].

To optimize system performance it is necessary to determine what gain 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 [Arkin, 1989]. The task of determining appropriate *a priori* gain values is non-trivial 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 [Arkin, 1990], but for purely reactive systems with no knowledge of the world, highly complex environments can produce difficulty in reaching near optimal solutions.

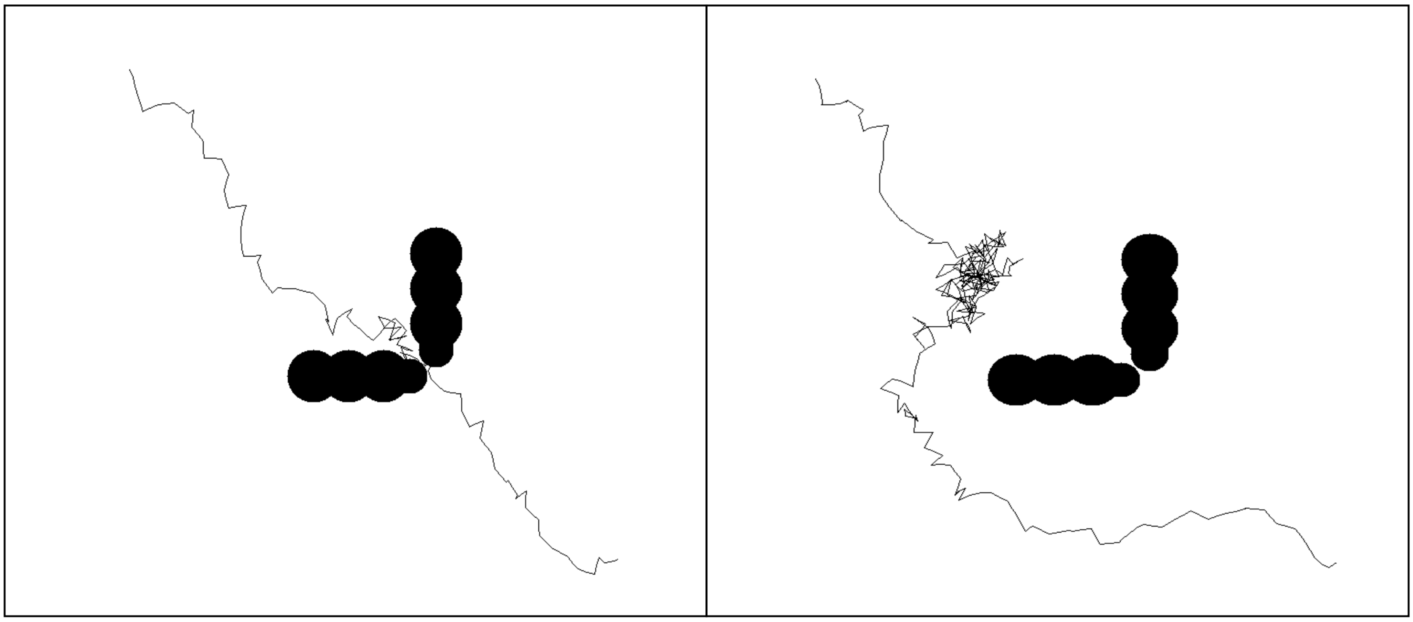

Figure 2: No Box Canyon Runs a) Sphere of influence = 10 b) Sphere of influence = 35

Furthermore, once this "best set" of gain values is established for a given world, it will likely be less efficient for navigation in a different environment. Figures 1 and 2 illustrate a scenario where this is true. Figure 1 shows a box canyon where a high AVOID-OBSTACLE sphere of influence is necessary to produce a successful path. Figure 2 shows a slightly different picture where a high sphere of influence results in a less than optimal 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 paper, it will be seen that our on-line learning system smooths out the paths by lowering the influence of noise when it is not required.

Finally, in a complex environment, it may be difficult to find one "right" set of gain values; different values may be appropriate for different areas of the environment. Similar arguments hold for other behavioral parameters as well. For all these reasons, it is 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 might call an *adaptive reactive control* method to automatically select and adjust the appropriate behavioral parameters to match the current task and environment.

3.3 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 of the domain, learning new control strategies, learning or improving domain theories and world models, and combinations of these. There are two broad classes of learning algorithms, inductive (or data-driven), and analytical (or theory-driven) [Michalski, 1983]. Inductive learning involves building generalizations based on instances encountered by the system. For example, a system could learn what a cup is by analyzing several examples and non-examples of cups to determine the similarities and differences between them. This type of learning is also known as similarity-based learning, and is data-driven in the sense that it relies on the data encountered by the system. A special type of inductive learning, called conceptual clustering, requires all the data to be present at once, rather than as one example at a time.

In contrast to inductive learning, analytical (or explanation-based) learning is theory-driven and relies heavily on the knowledge already in the system. In explanation-based learning, a system learns about a concept by attempting to build a causal explanation that justifies why a proposed example is indeed an example of the concept to be learned. Such a system might learn about a cup, not by noticing similarities between several examples of cups (e.g., that they all had handles), but by building a functional explanation of why a given object satisfied the requirements for "cupness" (e.g., cups have to be graspable, and a handle serves that function) [Winston *et al.*, 1983].

In some systems, examples are selected and presented by a teacher, as in Winston's program which relied on a teacher to present an appropriate sequence of examples and non-examples of the concept that the program was being taught [Winston, 1975]. Other systems encounter examples during problem solving or understanding, such as Lebowitz's IPP system [Lebowitz, 1983] or Ram's AQUA system [Ram, 1991; Ram, 1993] which learned by reading newspaper stories. Some systems, such as Rajamoney's program [Rajamoney, 1989] and Mitchell's LEX and LEX2 systems [Mitchell *et al.*, 1986], create and explore their own examples.

We view an intelligent system as one which cannot only perform the task it is programmed with in real world situations that the designer anticipated, but one which can perform this task, or pursue its goals, even in situations that are unusual, unexpected, and novel. For our navigation task, this means that the system must be able to learn during actual performance on a wide range of problems. The ability to learn continuously in a changing environment has been called "anytime learning" [Grefenstette and Ramsey, 1992]. Furthermore, we cannot assume that the system will have all the knowledge needed to deal with the situation, even if the situation itself was anticipated by the designer. Instead, the system will need to perform its task to the best of its abilities based on its experiences with these types of situations. In a reactive control system, this translates to a fundamental type of learning which we call *on-line adaptation*. This is a fast, 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. While such a system does not necessarily "learn" new control strategies, it does "learn" about the environment (or, equivalently, about behavioral parameters such as momentum appropriate for the environment) and is able to adapt to it dynamically during the performance task.

Since adaptation relies more on the features of the situation rather than on a deep causal analysis, one might expect to use similarity-based learning techniques as the underlying basis for a theory of adaptation. For example, *reinforcement learning* is an exploratory, trial-and-error method which allows a robot to improve its control strategies using feedback from the environment in the form of a simple scalar reward (e.g., [Sutton, 1992]). However, as with most forms of inductive learning, this approach requires either the presence of a benevolent teacher who presents the "right" set of problems to the system to train on, or large numbers of sufficiently varied training examples.

Analytical learning methods rely on a different assumption: the presence of a *domain theory* that provides a complete and correct causal model of the domain. While most of the work in induction has been in concept formation, explanation-based learning has also been applied to the learning of control strategies [Minton, 1988; Mitchell *et al.*, 1986]. Bennett described a robotic arm controller based on an explanation-based learning system which uses approximations to deal with inconsistencies between its internal theory of actions such as grasping and real-world experience [Bennett, 1990]. Mitchell's Theo-Agent architecture is an explanation-based learning system which learns to be more reactive by generating rules from explanations of failure [Mitchell, 1990]. The minimal deliberation approach proposed by Chien combines reaction-based control with a classical planning system to create plans for dealing with situations where reaction rules fail [Chien *et al.*, 1991].

The problem with using explanation-based learning methods for reactive control is twofold. First, these methods involve deep reasoning and are typically too slow for the fast, reflexive behavior required in reactive control systems. Second, these methods rely on a detailed theory of the domain, and possibly a detailed knowledge of the features of input situations; neither is available in a reactive control system. Instead, an on-line method for improving the performance of a reactive system must not slow down the system too much from pure reactive speeds, nor must it rely on perceptual input or knowledge of features or inference rules other than those that the input perceptual system in such a system can provide. Unlike the above systems, which require extensive domain knowledge in the non-reactive planning systems in order to deal with limitations in reactive control, our approach does not fall back on slow non-reactive techniques for improving reactive control.

Much of the other work in applying machine learning to robotic control is not based on reactive control systems, but is aimed at systems which improve navigational ability by learning environmental features [Zelinsky, 1988] or developing models of actions in the environment [Christiansen *et al.*, 1990]. The behavior based learning system proposed by Maes is a distributed algorithm in which the behaviors learn when to become active based on feedback during execution [Maes and Brooks, 1990]. One difference between this and our work is that our system learns varying levels of behavior "activation" rather than merely active versus inactive; also, our method is not based on statistical information.

A recent approach in learning that cuts across the traditional inductive/analytical learning dichotomy derives from Schank's work on dynamic memory [Schank, 1982]. *Case-based reasoning and learning* programs deal with the issue of using past cases to understand, plan for, or learn from novel situations [Hammond, 1989; Kolodner, 1990]. The intent behind case-based reasoning is to avoid the effort involved in re-deriving these lessons, explanations or plans by simply reusing the results from previous cases. Case retrieval and case modification rely both on similarity between situations, as in inductive learning, and causal theories, as in analytical learning. Both types of methods are also used for learning in case-based reasoning systems (e.g., [Hammond, 1989; Ram, 1993; Veloso and Carbonell, 1993]).

Hammond's CHEF program is an example of early work in the application of CBR to non-real-time and non-reactive planning problems [Hammond, 1989]. However, learning and adaptation in CHEF is done through explanation-based reasoning, which requires a detailed model of the domain. This is exactly what reactive planning systems are trying to avoid. Kopeikina, Brandau and Lemmon describe an application of CBR to real-time control [Kopeikina *et al.*, 1988]. 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. They suggest a system that performs the learning task in batch mode during off peak hours. In contrast, our approach combines the learning capabilities of case-based reasoning with the on-line, real-time aspects of reactive control.

3.4 Case-based reasoning for real-time reactive control

When a robot encounters difficulties during performance, it typically does not have the luxury of stopping and deliberating. It must adapt to the situation, go on with its task as well as it can, and learn from one or more experiences to improve its ability to deal with a similar situation in the future. A critical aspect of a learning system for robotic control is that the learning process must not interfere with the system's reactive response. A robot operating in a potentially hostile environment cannot stop to update a model when new information is found. This aspect of robotic systems is a central motivation for the current work, but it also provides an important constraint on machine learning methods that are developed in this domain.

In our ACBARR system, case-based reasoning is used on-line during the performance of the reactive control task to select and adapt navigational behaviors represented as motor schemas. Unlike traditional case-based reasoning systems which rely on cases to suggest solutions (here, behaviors) but on rules to adapt them, our method uses cases for both purposes. The reasoning process is shallow and fast, yet considerable power is derived from the strategies packaged in the system's cases.

The hope in case-based reasoning is that past experiences 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 case-based reasoning program will generate is dependent on the following factors:

- the actual experiences (*cases*) which the system has had
- the *representation* of the cases

- the ability to recognize that a new situation is like a previous one (*case retrieval*)
- the methods for *application* of an existing case to a new situation
- the ability to *evaluate* the efficacy of the case, and to decide when a new or different case might be needed

Each of these issues must be addressed in any case-based reasoning system. A suitable representation for cases, based on the features available for the task domain, must be developed. One of the key issues in case-based reasoning is that of retrieval. This process depends, in part, on how well the system's cases are stored in memory. Ideally, similar things should be grouped together. However, the selection of the proper set of features to index the cases in memory is a difficult problem and an ongoing research issue. This *indexing problem* has been approached in a variety of ways (e.g, [Domeshek, 1992; Kolodner, 1992]), but each indexing method is chosen to be efficient for a given problem or situation. There are other problems inherent to the case-based paradigm which must be handled for ACBARR to be effective. There is the question of knowing when to adapt and when to switch cases, and evaluating the efficacy of the case currently being used. There is also the problem of partial matching: what to do if there are no complete matches in the case library when attempting to perform a retrieval. In order for global behavior switching to be implemented, all of these problems with case-based reasoning must be handled in some way. These issues are addressed in the next section.

4 Technical details: The ACBARR system

ACBARR is a combination of a schema-based reactive control system with a case-based reasoning module. Figure 3 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 ADAPTOR adjusts the gain 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,

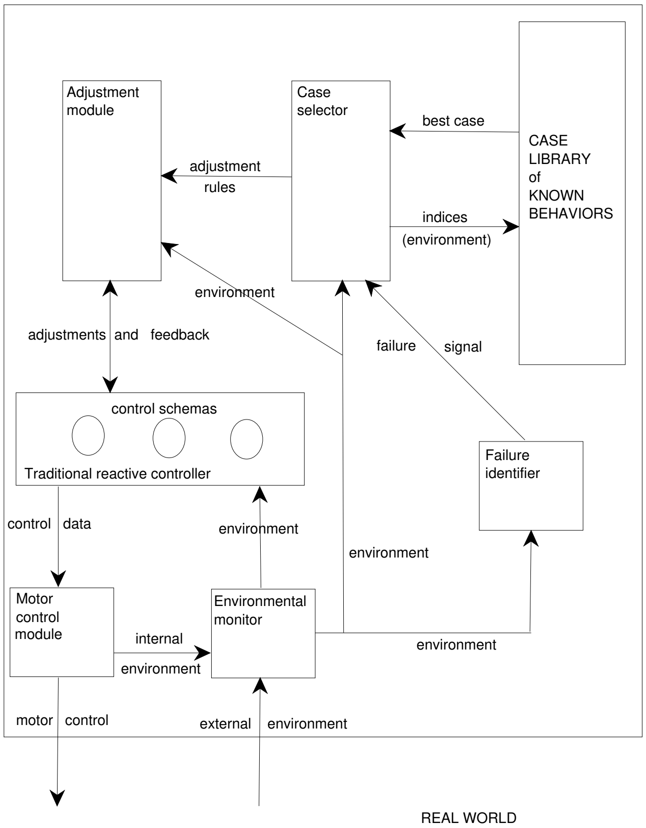

Figure 3: System architecture

the CASE SELECTOR chooses a better case from the case library, which suggests a new parameter sets and associated ranges for parameter adaptation.

A case within ACBARR's case library represents an assemblage of behaviors 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.

In order to understand the process more fully, we now present the details of how each of the components of the ACBARR system operates.

4.1 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.

$$O_{magnitude} = \begin{cases} 0 & \text{for } d > S \\ \frac{S-d}{S-R} * G & \text{for } R < d \leq S \\ \infty & \text{for } d \leq R \end{cases} \tag{1}$$
$$O_{direction} = \text{From center of obstacle towards robot}$$

  where:

$$\begin{aligned} S &= \text{Adjustable sphere of influence (radial extent of force from} \\ &\quad\ \text{center of obstacle)} \\ R &= \text{Radius of obstacle} \\ G &= \text{Adjustable gain} \\ d &= \text{Distance of robot to center of obstacle} \end{aligned}$$

- Move-to-Goal: Attract to goal with variable gain. Set high when heading for a particular goal.

$$\begin{aligned} V_{magnitude} &= \text{Adjustable gain value} \\ V_{direction} &= \text{Direction towards perceived goal} \end{aligned} \tag{2}$$

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

$$
\begin{aligned}
N_{magnitude} &= \text{Adjustable gain value} \\
N_{direction} &= \text{Random direction, persists for } N_{persistence} \text{ steps} \\
N_{persistence} &= \text{Adjustable persistence value}
\end{aligned}
\tag{3}
$$

MOVE-TO-GOAL creates an attraction between the robot and the desired goal location, AVOID-STATIC-OBSTACLE creates a repulsive force between the robot and an obstacle, and NOISE provides for random fluctuations on the robot's movement. The Noise schema is particularly important for escaping from equilibrium points in the force field. If caught at such a point, the random nature of the noise should free the robot. The vectors from the active sources are added, forming a resultant movement vector for the robot to use. We assume that there is one goal and multiple obstacles.

Each motor schema has one or more behavioral parameters associated with it. Gain values determine the impact level that a particular schema has on the overall system behavior. For example, a high goal gain will cause the MOVE-TO-GOAL schema to have a more pronounced effect on the system movement. In addition to the three gain values, there are two other numerical values associated with these schemas. NOISE has a persistence as well as a gain; since noise is a random effect, the amount of time which a given level of noise affects the system is variable. Finally, the sphere of influence parameter limits the area which the robot is able to receive the effects from the various obstacles around it.

If one has the proper set of gain values, optimal behavior can be achieved for the system in question for a given environment and task. The process of selecting the proper set of values, unfortunately, is not a simple one. The traditional method has been to select a set of starting values for the gains. Then, the system is run on a task in a particular environment. The actual performance is then compared with the desired performance and the set of gain values is adjusted accordingly. When no further increase in performance can be achieved between two successive runs, the training is declared complete. In ACBARR, the system can be set up with any set of values for a run. Its cases allow it to select appropriate values and adapt these values dynamically based on its perception of the environment and its evaluation of its progress.

4.2 Environmental information

One of the goals of reactive control is to produce "intelligent" behavior without resorting to higher-level reasoning processes, such as completely mapping a new environment and then planning a near-perfect path through the obstacles. There are two problems with high-level reasoning. First, it involves a large amount of overhead in the system, both in terms of processing resources and time. This means that the robot cannot immediately function in a new environment; it must first map and analyze the world around it. Second, if the environment is dynamic, the robot is faced with a changing world in which its world model may be become partially incorrect or obsolete. Reactive systems handle both of these problems. Although case-based reasoning requires some higher-level reasoning, we did not want to have to add so much processing requirements that these benefits were outweighed. Furthermore, the 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 about the world:

- CLUTTER is a metric of how many obstacles there are within the sphere of influence of the robot. To aid in partial matching for case selection, the exact number is converted into a class number representing a range of similar clutters. For example, for the purposes of selecting a strategy to use, seven obstacles and eight obstacles would be treated the same.

- WANDER is a measure of how efficient the robot's current path is. This is determined by examining the ratio of the robot's current path length over the actual distance between the initial position and the goal.

- MAXXED is a binary value representing whether the noise persistence has stayed at its upper bound for a significant period of time.

- CLEAR-TO-GOAL is a binary value which, if true, indicates that 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 sphere of influence with nothing between the goal and the robot.

- 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.

- CIRCLES is a rough metric of how much the robot has recently traveled over the same piece of ground (e.g., how much it is running around in circles).

- GRANNY is a binary variable which determines when particularly careful behavior should be employed. It will result from long-term poor performance by the system, 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 straight-forward mathematical calculation of sensor data. No higher level processing is required to produce this information.

4.3 Movement information

In addition to the information which ACBARR maintains concerning its *external environment*, it also needs information involving its *internal environment*. This is knowledge of how the current case behavior is performing, in terms of the robot's movement towards the goal position. In particular, there are four 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.

$$M < T_{movement} \tag{4}$$

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 threshold.

$$\begin{aligned} M &\geq T_{movement} \\ P &= \frac{H_{distance}}{H_{goal}} \geq T_{progress} \end{aligned} \tag{5}$$

where:

$$\begin{aligned} H_{distance} &= \text{Distance travelled over } H_{steps} \\ H_{goal} &= \text{Decrease in distance to goal over } H_{steps} \\ T_{progress} &= \text{Progress threshold} \end{aligned}$$

- NO-PROGRESS-WITH-OBSTACLES: The robot is moving but not toward the goal and there are several obstacles within the its sphere of influence.

$$\begin{aligned} M &\geq T_{movement} \\ P &= \frac{H_{distance}}{H_{goal}} < T_{progress} \\ O_{count} &\geq T_{obstacles} \end{aligned} \tag{6}$$

where:

$$\begin{aligned} O_{count} &= \text{Average number of obstacles within sphere of influence} \\ &\quad\ \text{over } H_{steps} \text{ steps} \\ T_{obstacles} &= \text{Obstacle count threshold} \end{aligned}$$

- NO-PROGRESS-NO-OBSTACLES: The robot is moving but not toward the goal and there are no obstacles within its sphere of influence.

$$\begin{aligned} M &\geq T_{movement} \\ P &= \frac{H_{distance}}{H_{goal}} < T_{progress} \\ O_{count} &< T_{obstacles} \end{aligned} \tag{7}$$

4.4 Short-term memory

ACBARR employs a primitive short-term 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, since to maintain an in-depth model of a dynamic environment while running in real-time would require considerable space and time resources. As a compromise, ACBARR's short-term memory only keeps track of the robot's travels within a small *window* of the entire world. The window represents a small geographic area. As long as the robot remains within the same window, its steps is kept tracked of. If ACBARR determines that the robot has been moving an excessive amount within the window, it declares that the robot is running in circles. 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. Short-term memory has also been utilized in other systems based on the AuRA architecture to provide information about spatial occupancy of recently visited areas [Arkin, 1990; Balch and Arkin, 1993].

4.5 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. This may be thought of as the "index" to the case. 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. Finally, the third part contains bookkeeping information about the case, including information about past experiences with the case.

A sample case is shown in Figure 4. 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 binary values, an entry of -1 indicates that the system does not care about this parameter. The sample case is best suited for a non-cluttered ("open field") environment. In such an environment,

Environmental and Movement Information	
Parameter	Value
clutter	0.0
wander	0.0
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	1
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

Figure 4: Sample ACBARR case

the robot should be making progress towards the goal and not wandering or running in circles. 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 and use the behavioral parameters recommended by the third 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 gain 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.

The third and final table contains bookkeeping information about the case. In the current example, this tells us that this is case 1 (CLEAR-FIELD; see section 4.6) 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. In the current ACBARR implementation, most of this information is unused. However, it is included in the case description for future extensions of the system.

4.6 The case library

A reactive robotic simulator[2] was first run several hundred times in order to gain insight into the types of behavior different environments would call for. This initial attempt to conceptualize the various schemas resulted in ten strategies being identified, each being represented as a case in the format discussed earlier:[3]

1. CLEAR-FIELD: In an open environment, the system should pay no attention to obstacles (since there won't be any), 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 hug 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 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 and if the incoming vectors from the obstacles are localized in front of it. In this situation, it 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; however, it has proven effective in the tested environments.

[2]The simulator was written in standard C with an X-Windows interface. All test runs described were performed on Sun SparcStation 1s. For more information concerning the simulator, see [Clark *et al.*, 1992].

[3]The representations of all ten cases are presented in the appendix.

7. RANDOM: The system raises the noise gain and goal gain, leaves the obstacle gain at a medium level, and wanders for a period of time. This is useful for exploration.

8. GRANNY: After H_{steps}, the system reconsiders the environment by actually attempting to build a limited model of it. It concentrates on the location and sizes of the obstacles within its sphere of influence and attempts to choose the 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.

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.

4.7 Indexing and case selection

Since there is a limited number of cases in the library, there is little need at this stage for a sophisticated indexing scheme. As a result, ACBARR employs a flat memory model of the case library [Kolodner, 1992]. 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 would need to be implemented.

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 in a uniform manner.

4.7.1 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 $H_{steps} = 1$, this method ensures that the best case will always be in place, and the system will always use the optimal navigational strategy. However, the strategy of switching every H_{steps} steps is pessimistic. 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 select 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 short-term 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.

Unless specified otherwise, ACBARR uses the failure-driven strategy for case switching. This design decision was based on extensive empirical evaluation of both strategies. As shown in section 5.4, 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.

4.7.2 Case matching

Another important issue in ACBARR, and indeed in any case-based reasoning system, is that of selecting the best case from the case library. 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 is due to the limited number of cases. Due to the numerical nature of the matching procedure, partial matches are taken care of as well as total matches. If no total match is found which has a high Goodness-Of-Match rating, a partial match can be chosen.

Occasionally, the best matching case may turn out to be the case already in use. In such a situation, the system could either choose the best alternative case from the library or continue to use the current case. Suppose the system is currently making use of case 1. At some point, the system determines that a new case is needed and selects the best one from the library, which turns out to be case 1 again. Obviously, case 1 may not actually be the best case for the given environment; otherwise, why did it fail? On the other hand, the matching criteria may not be in error; this environment may be a fluke. ACBARR uses both of these approaches. Generally, when confronted with a failure, ACBARR chooses the best case. This may turn out to be the current one, which is fine. If the cause of the failure was excessive wandering, ACBARR will elect to choose the best case not equal to the current one. The philosophy behind this design decision was that in such a situation, the same case is not likely to be helpful since the system has already exceeded the wandering limit for this case.

4.8 Case application

Once the best case is retrieved, ACBARR makes use of it in step-by-step movement. Each case contains a set of modifications which can be performed on the gains in the system, as well as the minimum and maximum values of each. ACBARR uses this information, along with information about its own movement in order to make the local adjustments to its behavior.

An overall algorithm depicting this sequence of case retrieval and use is presented in Figure 5.

```
MAIN-LOOP {
   environment = Examine-Environment ();
   movement = Examine-Movement ();
   current-case = Select-Case (environment, movement);
   DO {
      IF (granny = TRUE) {
         new-vector = Determine-Best-Path-Vector (environment); /* lengthy */
         robot.movement = Add-Vector (robot.movement, new-vector);
      }

      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-Short-Term-Memory ();

      environment = Examine-Environment ();
      movement = Examine-Movement ();
      if (New-Case-Needed (environment, movement))
         current-case = Select-Case (environment, movement);

      Adjust-Gain-Values (current-case)
   }  UNTIL (goal-reached OR number-steps > maximum-steps-allowed)
}  /* end MAIN-LOOP */


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

   FOR (each-case in CaseLibrary) DO {
      goodness = 0.0;

      FOR (each-factor in environment) DO {
         if (each-factor = each-case.factor)
            goodness = goodness + [0.5 to 3.0] ; /* add from 0.5 to 3.0 */
                                                 /* depending on factor */
      }

      FOR (each-factor in movement) DO {
         if (each-factor = each-case.factor)
            goodness = goodness + 1.0;
      }

      if (goodness > best-case-match) {
         best-case = each-case;
         best-case-match = goodness;
      }
   }
}

New-Case-Needed {
   if (environment.wander > WANDER_LIMIT) OR
      (environment.senses-goal = TRUE) OR
      (environment.clear-to-goal = TRUE) OR
      (running-in-circles) OR  /* determined by examining short-term memory */
        return NEED-NEW-CASE;
   else
        return KEEP-OLD-CASE:
}
```

Figure 5: Case retrieval and use algorithm

5 Evaluation

In order to evaluate the proposed methods, we performed extensive simulations with the ACBARR system to evaluate its performance both qualitatively and quantitatively. The performance was compared with the unenhanced reactive control system. 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.

5.1 Simulation environment

In order to facilitate our work in adaptive control systems for autonomous navigation, we have developed a simulation environment that includes both a graphical interface and batch mode. This system allows us to visually evaluate the progress of a simulated robot while it runs through a predefined world. It also displays a history of values as they are adjusted by the system to facilitate the identification of successful adjustment rules. 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 window displays the current obstacles as circles, each with varying radius. 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 ADAPTOR is called. 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.

5.2 Qualitative evaluation

Reactive control systems have difficulty with local maxima. 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, [Arkin, 1989]. Usually, a high level of random noise is used to try to kick

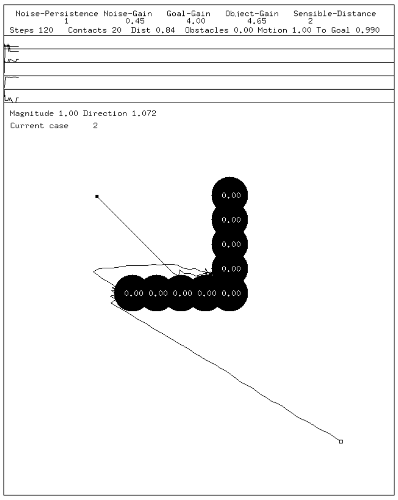

Figure 6: Box canyon — ACBARR

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. ACBARR learns about its current environment and adapts to it dynamically, using an appropriate level of noise, obstacle avoidance, and so on. The same method can also handle other difficult environments, such as "walls".

Performance evaluation: Figures 6 through 8 illustrate sample runs that demonstrate the qualitative improvement in ACBARR, as compared with the unenhanced reactive control system shown in figures 1 and 2. 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 (figure 6), the quasi-box canyon problem (a box canyon with an exit, figure 7), and the wall environment (figure 8) autonomously.

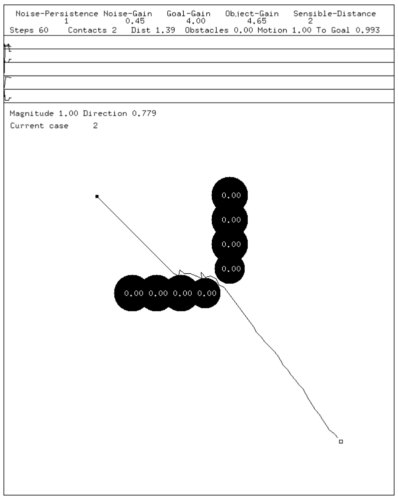

Figure 7: Quasi-box canyon — ACBARR

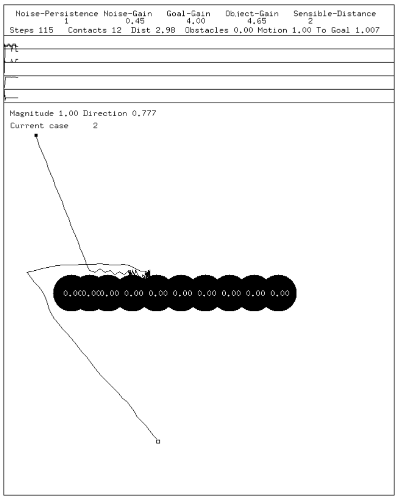

Figure 8: Wall environment problem — ACBARR

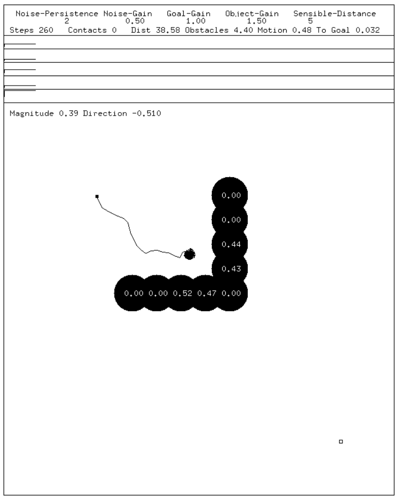

Figure 9: Box canyon — No cases, no adjustments allowed

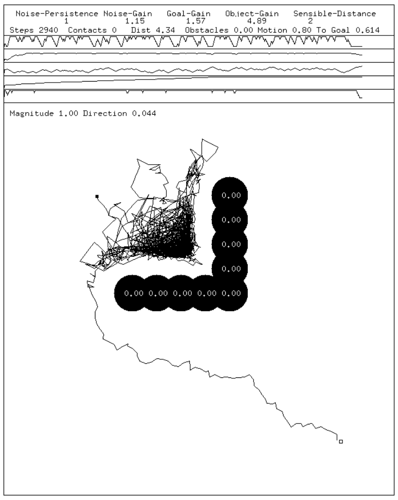

Figure 10: Box canyon — No cases, but adjustments are permitted

Method evaluation: In addition to evaluating the performance of the ACBARR system as a whole, several *ablation studies* [Cohen and Howe, 1988] 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 works. Figure 9 shows the system's performance on the box canyon world with neither local adjustments nor global cases available to the system. As can be seen, the robot achieves an *equilibrium point* and stays there. Figure 10 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. Comparing these results with those of figure 6, 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.

5.3 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 100 random worlds, each with 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. We collected data on total time needed to complete the navigational problem, the ratio of distance actually travelled to the straight-line distance from start to goal, the number of steps taken by the robot, and time per step of robot action. These results are depicted in figure 14 through figure 13. In the graphs, ACBARR-F is the ACBARR system utilizing a switch-on-failure case switching strategy, ACBARR-S is the same system with the switch-per-step strategy, and REACTIVE-10 and REACTIVE-50 are systems utilizing "pure" reactive control.[4] The reactive-only 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 and did not perform as well in environments for which they were not explicitly intended. These simulations demonstrate a substantial improvement with the ACBARR systems over reactive control alone; they performed better with respect to several metrics and over a wide range of environments.

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 figure 11. 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 1100 steps on average as compared with the approximately 100-step paths found by the ACBARR 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. 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 have to be hand-coded in a purely reactive system) for different situations.

Distance: Another useful metric for evaluating the performance of the system is the ratio of the actual distance travelled 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

[4]The schema parameters used to configure the REACTIVE systems, as well as the case representations used for the ACBARR systems, are presented in the appendix.

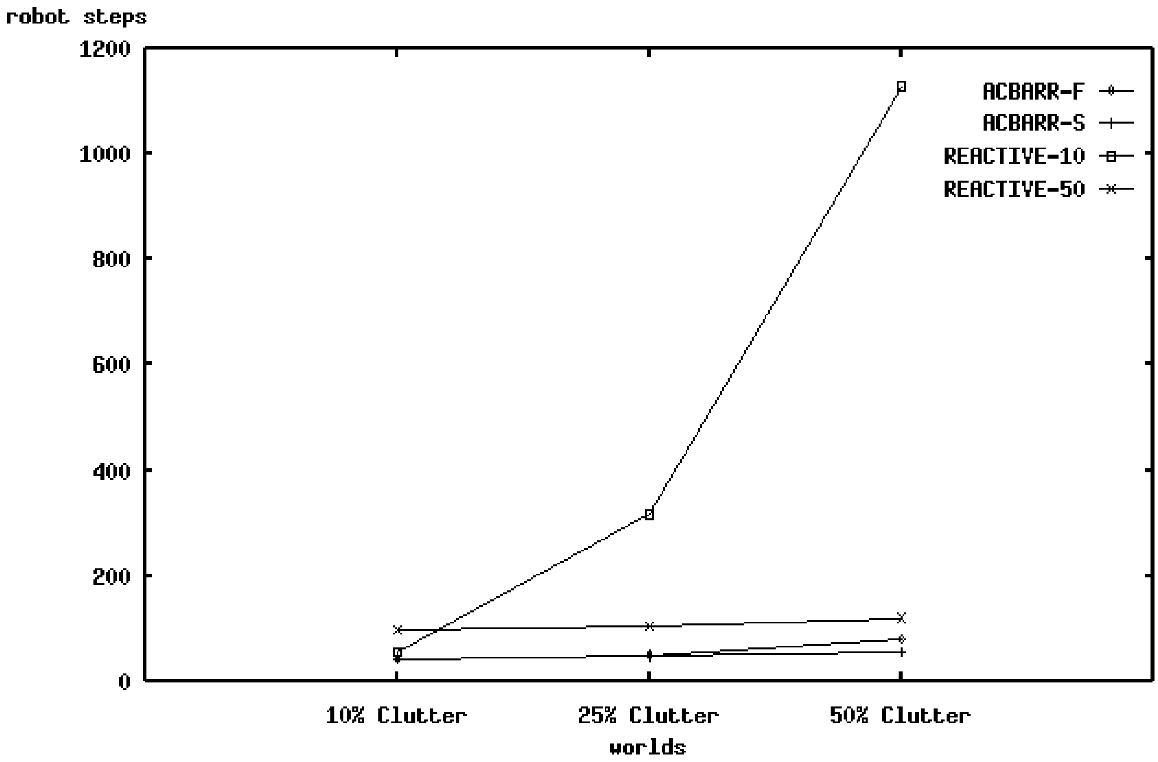

Figure 11: Robot steps

be greater than 1.0, no matter how good the navigational system is. Values closer to 1.0 are better than those farther away.

As shown in figure 12, 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. Finally, the most convincing portion of the figure is in the section for 50% cluttered worlds. REACTIVE-10 reached the goal along paths which were almost ten times as long as they needed to be. REACTIVE-50, which was hand-coded for such environments, performed better, but still not as well as the ACBARR systems. Notice also that at this level of obstacle density, ACBARR-F is worse than ACBARR-S. Case switching on every step allows ACBARR-S to find shorter paths than ACBARR-F can.

Since ACBARR is able to switch strategies and select the appropriate parameter values for different situations, one would intuitively expect its performance graph to lie at or below

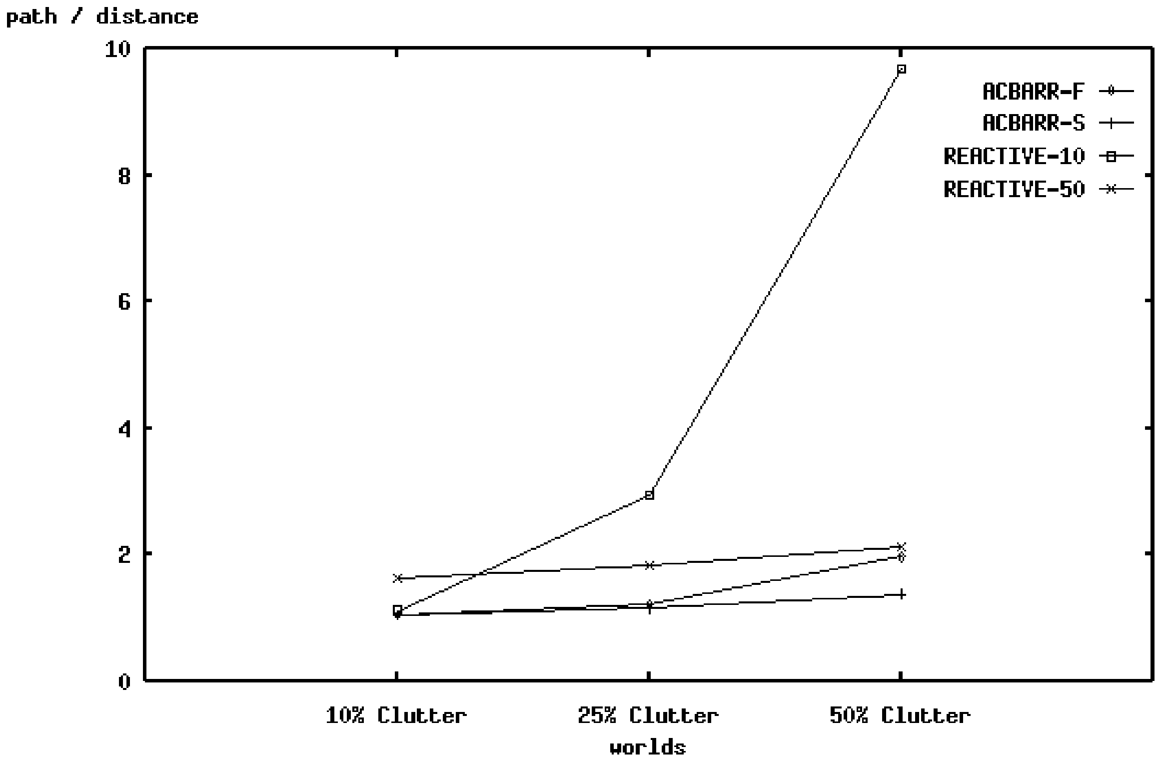

Figure 12: Path length over actual distance

the best non-adaptive system that was individually tuned for each type of environment. This is empirically demonstrated by the results in figure 12.

Time per step: The time per step metric is another interesting evaluation 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, figure 13 shows that ACBARR-S took the longest amount of time per step. This is due to the 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 differences could begin to have an impact on overall system time in a simulated system. This effect is evaluated in the following experiment; however, as argued below, we expect this computation time not to be the dominant factor in an actual robot.

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 (figure 14). In the 10% worlds,

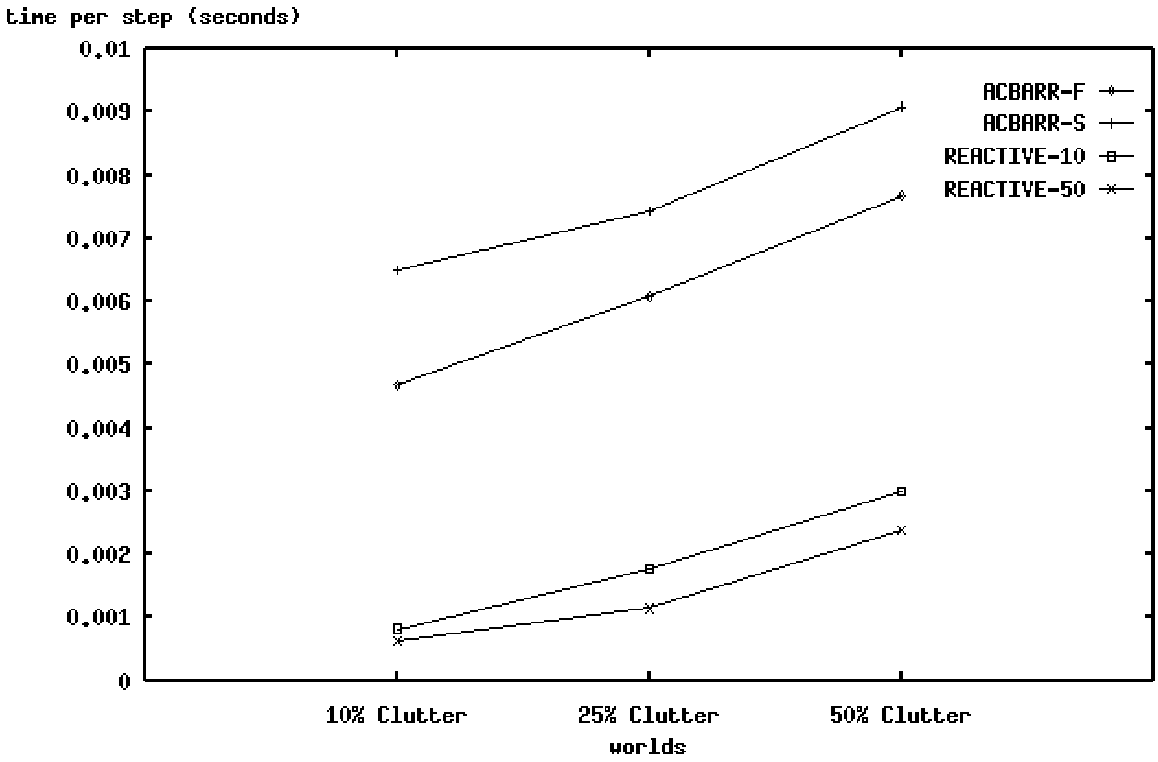

Figure 13: Time per step

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 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 5–10 ms per cycle, as shown in figure 13). Physical movement of the robot is also relatively slow. Since ACBARR's paths are considerably shorter (figure 12) and require less robot steps (figure 11), the processing overhead in ACBARR is negligible compared to the improved performance that results from the better navigational paths that are created.

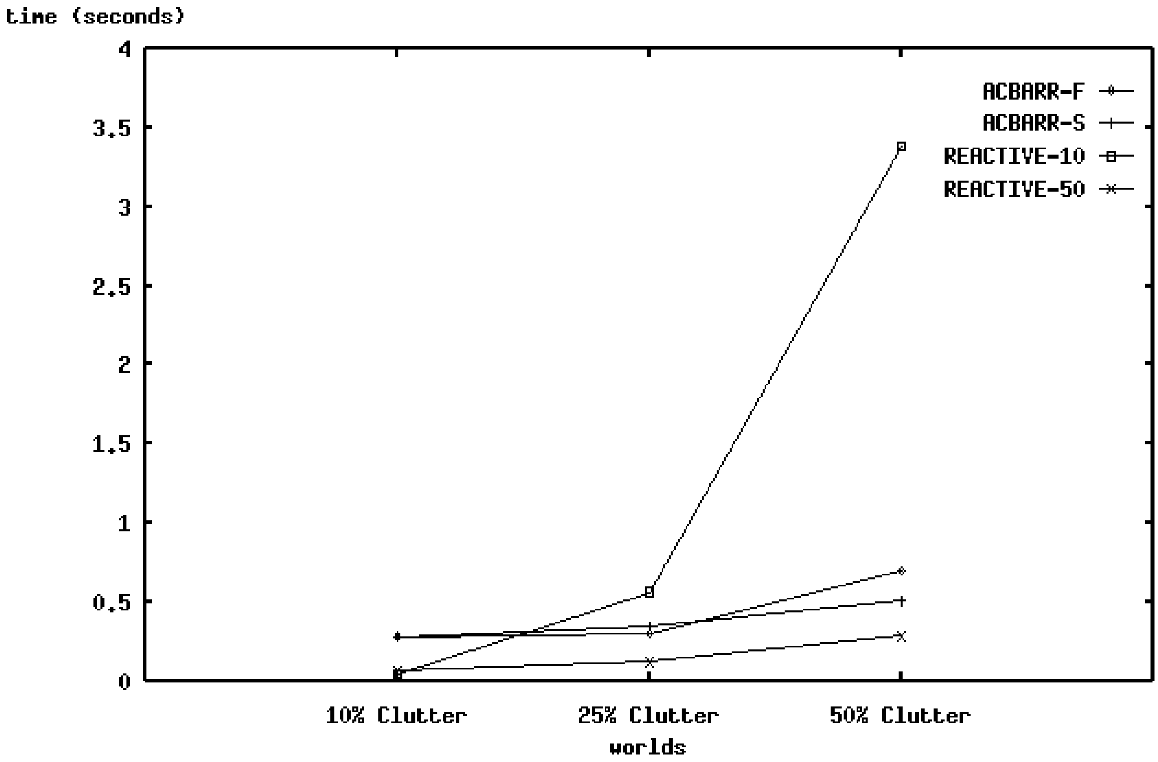

Figure 14: Total time to completion

5.4 Discussion

Considering all four data 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 more time to "ponder" each move they are going to make. This means that for worlds with 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, but overall its performance is fairly respectable. 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.

The benefits of the extra time per step taken by the ACBARR systems begin to be revealed in more cluttered environments. The 25% cluttered worlds show this beginning. The paths created by the ACBARR systems are much shorter than the purely reactive systems, and they require fewer robot 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 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. While the two methods are fairly close today, future versions of the system with more complex cases will cause the switch-on-failure strategy to become the clear method of choice.

It should be noted that while ACBARR does improve performance over the non-adaptive 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 major benefits of our approach is flexibility. 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.

The results also show support for another of our claims; namely, that ACBARR's case based additions do not significantly slow the system from pure reactive speeds. Due to the better quality of produced paths, the ACBARR systems generally completed a world in less time than the REACTIVE-10 system, and was not much slower than the REACTIVE-50 system. While the time per step value was slightly higher for the ACBARR systems, they produced significantly more efficient paths. If the actual number of steps taken is of importance, for example, on an actual robot where the physical time to carry out a step and the time to sense and process perceptual input far exceed the computation time to compute the direction and size of the step, the ACBARR systems are a significant improvement over traditional non-adaptive reactive control.

6 Limitations and future work

The results show that case-based reasoning can provide substantial benefits for a reactive control system in terms of its ability to navigate difficult environments as well as in terms of quantitative measures of its performance. The benefits arise from the higher-level strategies that are implemented, however, without in-depth or high-level reasoning. Case-based reasoning provides a method for fast retrieval of these strategies based on simple cues from the environment. The strategies, in turn, allow the system to "learn" about its environment and adapt to it.

While our experiments showed that the methods were in fact a substantial improvement over non-adaptive approaches, they also pointed out the need for further research. Let us discuss some of the problems that we encountered.

6.1 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 switching method, the system would generally use only a subset of its stored possible strategies during test runs. We attempted to define this behavior further by running more extensive simulations on various random worlds with ACBARR-S. The simulations revealed exactly how many cases were utilized and what percentage of the time each was being used. We found that out of our ten cases, only five were ever used. 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 used earlier. As it turned out, WALL-CRAWLING can handle most situations that MAXXED can.

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 optimal strategy is not always in place. If the current case is "good enough," the system will not bother to switch to the optimal 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 a new case was selected 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.

6.2 Single strategy or multiple cases?

In our research, we assumed that a set of strategies would be needed to deal with the wide 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 if such a strategy could be discovered? There are several reasons. First, it is highly unlikely that an all-encompassing strategy such as this could be discovered. It would require considerably 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 switch-on-failure strategy were employed, then this powerful strategy would stay in place until it failed (if it failed). While ACBARR does have some increase in time per step for a switch-on-failure strategy, this effect should be small enough to ignore in this scenario. 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 that was outside the scope of the initial strategy. Finally, ACBARR can provide a framework to test the effectiveness of any such strategy. A fairly robust strategy could be developed to the point where the researcher is happy with it. It could then be 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 s/he wished to further improve the strategy.

Thus our claim, which is supported by the evidence we have discussed, is that ACBARR adds robustness to any pure reactive control system, regardless of how robust the existing behavior already is. A perfect set of cases is not necessary to realize the benefits of the approach. ACBARR also provides a test-bed for new strategies to be researched as they are developed.

6.3 Initial case selection

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 and applicable than HUGGING, for instance 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.

6.4 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, George. 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.

7 Conclusions

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 learning. The methods presented in this article were developed as part of this on-going effort. 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 learn about and adapt to its environment dynamically, drawing on the power of analytical learning without sacrificing the speed of similarity-based comparisons. Due to these properties, 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 unlikely. This results in ACBARR being a highly efficient, adaptive control system.

Acknowledgements

This work was supported in part by a Fannie and John Hertz Foundation fellowship and by the Georgia Institute of Technology. The Mobile Robot Laboratory is supported by the National Science Foundation under grants IRI-9100149 and IRI-9113747, and the Material Handling Research Center of Georgia Tech.

References

[Agre and Chapman, 1987] P. Agre and D. Chapman. Pengi: An Implementation of a Theory of Activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 268–272. AAAI, 1987.

[Amarel, 1968] S. Amarel. On Representations of Problems of Reasoning about Actions. *Machine Intelligence*, 3, 1968. Reprinted in B. L. Webber and N. J. Nilsson (eds.), Readings in Artificial Intelligence, pp. 2-22, Tioga, Palo Alto, CA, 1981.

[Arkin, 1989] R. C. Arkin. Motor schema-based mobile robot navigation. *The International Journal of Robotics Research*, 8(4):92–112, August 1989.

[Arkin, 1990] R. C. Arkin. Integrating Behavioral, Perceptual, and World Knowledge in Reactive Navigation. *Robotics and Autonomous Systems*, 6:105–122, 1990.

[Balch and Arkin, 1993] T. Balch and R. C. Arkin. Avoiding the Past: A Simple but Effective Strategy for Reactive Navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Atlanta, GA, May 1993. To appear.

[Bennett, 1990] S. W. Bennett. Reducing Real-world Failures of Approximate Explanation-Based Rules. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 226–234, June 1990.

[Brooks and Connell, 1986] R. Brooks and J. Connell. Asynchronous Distributed Control System for a Mobile Robot. In W. Wolfe and N. Marquina, editors, *Proceedings of the SPIE, Volume 727: Mobile Robots*, pages 77–84, Bellingham, WA, 1986.

[Brooks, 1986] R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, August 1986.

[Brooks, 1988] R. Brooks. Intelligence without representation. Research paper, Massachussetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge, MA, 1988.

[Brooks, 1989] R. Brooks. A robot that walks: Emergent behaviors from a carefully evolved network. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 692–694, Scottsdale, AZ, May 1989.

[Chien *et al.*, 1991] S. A. Chien, M. T. Gervasio, and G. F. DeJong. On Becoming Decreasingly Reactive: Learning to Deliberate Minimally. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 288–292, June 1991.

[Christiansen *et al.*, 1990] A. D. Christiansen, M. T. Mason, and T. M. Mitchell. Learning Reliable Strategies without Initial Physical Models. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 1224–1230, 1990.

[Clark *et al.*, 1992] R. J. Clark, R. C. Arkin, and A. Ram. Learning Momentum: On-line Performance Enhancement for Reactive Systems. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 111–116, Nice, France, May 1992.

[Cohen and Howe, 1988] P. R. Cohen and A. E. Howe. How evaluation guides AI research. *AI Magazine*, 9(4):35–43, Winter 1988.

[Domeshek, 1992] E. A. Domeshek. *Do the right thing: A component theory for indexing stories as social advice*. Ph.D. thesis, Yale University, Department of Computer Science, New Haven, 1992. Available as Technical Report #26, Northwestern University, Institute for the Learning Sciences, Evanston, IL, May 1992.

[Firby, 1989] R. J. Firby. *Adaptive Execution in Complex Dynamic Worlds*. Ph.D. thesis, Yale University, Department of Computer Science, New Haven, CT, January 1989. Research Report YALEU/CSD/RR #673.

[Grefenstette and Ramsey, 1992] J. J. Grefenstette and C. L. Ramsey. An Approach to Anytime Learning. In D. Sleeman and P. Edwards, editors, *Machine Learning: Proceedings of the Ninth International Conference*, pages 189–195, Aberdeen, Scotland, 1992. Morgan Kaufman.

[Hammond, 1989] K. J. Hammond. *Case-Based Planning: Viewing Planning as a Memory Task.* Perspectives in Artificial Intelligence. Academic Press, Boston, MA, 1989.

[Kadanoff *et al.*, 1986] 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 W. Wolfe and N. Marquina, editors, *Proceedings of the SPIE, Volume 727: Mobile Robots*, pages 77–84, Bellingham, WA, 1986.

[Kaelbling, 1986] L. Kaelbling. An Architecture for Intelligent Reactive Systems. Technical Note 400, SRI International, October 1986.

[Kolodner, 1990] J. L. Kolodner. An introduction to case-based reasoning. Technical Report GIT-ICS-90/19, Georgia Institute of Technology, School of Information and Computer Science, Atlanta, GA, 1990.

[Kolodner, 1992] J. L. Kolodner. *Case-based reasoning.* Morgan Kaufmann, San Mateo, CA, 1992. In press.

[Kopeikina *et al.*, 1988] L. Kopeikina, R. Brandau, and A. Lemmon. Case-Based Reasoning for Continuous Control. In *Proceedings of a Workshop on Case-Based Reasoning*, pages 250–259, Clearwater Beach, FL, May 1988. Morgan Kaufmann.

[Lebowitz, 1983] M. Lebowitz. Generalization from Natural Language Text. *Cognitive Science*, 7(1):1–40, 1983.

[Maes and Brooks, 1990] P. Maes and R. A. Brooks. Learning to Coordinate Behaviors. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 796–802, Boston, MA, August 1990.

[Michalski, 1983] R. S. Michalski. A Theory and Methodology of Inductive Learning. *Artificial Intelligence*, 20:111–161, 1983.

[Minton, 1988] S. Minton. *Learning effective search control knowledge: An explanation-based approach.* Ph.D. thesis, Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA, 1988. Technical Report CMU-CS-88-133.

[Mitchell *et al.*, 1986] T. M. Mitchell, R. Keller, and S. Kedar-Cabelli. Explanation-Based Generalization: A Unifying View. *Machine Learning*, 1(1):47–80, 1986.

[Mitchell, 1990] T. M. Mitchell. Becoming Increasingly Reactive. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 1051–1058, Boston, MA, August 1990.

[Moorman and Ram, 1992] K. Moorman and A. Ram. A Case-Based Approach to Reactive Control for Autonomous Robots. In *Proceedings of the AAAI Fall Symposium on AI for Real-World Autonomous Mobile Robots*, Cambridge, MA, October 1992.

[Payton, 1986] D. Payton. An Architecture for Reflexive Autonomous Vehicle Control. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 1838–1845, 1986.

[Rajamoney, 1989] S. Rajamoney. *Explanation-Based Theory Revision: An Approach to the Problems of Incomplete and Incorrect Theories.* Ph.D. thesis, University of Illinois, Department of Computer Science, Urbana, IL, 1989.

[Ram, 1991] A. Ram. A Theory of Questions and Question Asking. *The Journal of the Learning Sciences*, 1(3&4):273–318, 1991.

[Ram, 1993] A. Ram. Indexing, Elaboration and Refinement: Incremental Learning of Explanatory Cases. *Machine Learning*, 10:201–248, 1993. In press. Also available as Technical Report GIT-CC-92/03, College of Computing, Georgia Institute of Technology, Atlanta, GA, 1992.

[Riesbeck and Schank, 1989] C. K. Riesbeck and R. C. Schank. *Inside Case-Based Reasoning.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1989.

[Schank, 1982] R. C. Schank. *Dynamic Memory: A Theory of Learning in Computers and People.* Cambridge University Press, New York, NY, 1982.

[Skinner, 1974] B. F. Skinner. *About Behaviorism.* Knopf (Random House), 1974.

[Sutton, 1992] R. S. Sutton. Introduction: The Challenge of Reinforcement Learning. *Machine Learning*, 8:225–227, 1992.

[Veloso and Carbonell, 1993] M. Veloso and J. Carbonell. Derivational Analogy: Automating Case Acquisition, Storage and Utilization. *Machine Learning*, 1993. To appear.

[Winston *et al.*, 1983] P. H. Winston, T. O. Binford, B. Katz, and M. Lowry. Learning Physical Descriptions from Function Definitions, Examples, and Precedents. In *Proceedings of the National Conference on Artificial Intelligence*, Washington, DC, 1983.

[Winston, 1975] P. H. Winston. Learning Structural Descriptions from Examples. In P. H. Winston, editor, *The Psychology of Computer Vision*. McGraw-Hill, New York, 1975.

[Zelinsky, 1988] A. Zelinsky. A Mobile Robot Environmental Learning Algorithm. Technical report, University of Wollongong, Department of Computing Science, April 1988.

Appendix

The following schema parameter values were used to configure the non-learning reactive systems, REACTIVE-10 and REACTIVE-50. REACTIVE-10 was hand-tuned to perform well in lightly cluttered worlds (here, 10% clutter), and REACTIVE-50 was hand-tuned to perform well in highly cluttered worlds (here, 25% and 50% clutter).

Parameter	REACTIVE-10	REACTIVE-50
goal gain	1.0	1.0
noise gain	0.3	1.0
noise persistence	2.0	8.0
object gain	1.1	1.1
sensible distance	5.0	5.0

The ACBARR system was configured with the following 10 cases, as discussed in section 4.6. In the current implementation, two of the cases contain procedural information which cannot be readily captured in our case representation format (see below).

1. Clear-Field: In an open environment, the system pays no attention to obstacles (since there won't be any), increases the goal gain, and lowers the noise gain and noise persistence.

Environmental and Movement Information	
Parameter	Value
clutter	0.0
wander	−1
clear-to-goal flag	1
goal-nearby flag	1
circles	0
granny	−1
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	[0.5 0.5]	[4.0 4.0]
noise gain	[−0.05 −0.05]	[0.0 1.0]
noise persistence	[−1.0 −1.0]	[1.0 1.0]
object gain	[0.0 1.0]	[0.0 5.0]
sensible distance	[−0.5 −0.5]	[0.0 4.0]

Bookkeeping Information	
Parameter	Value
case number	2
case goodness	0.9
average step size	0.5
dynamic obstacles	0
initial distance to goal	36.5
obstacle danger	0.5
goal importance	1.0

2. Ballooning: When there are relatively few obstacles, the system attempts to swing around them in a wide way by increasing obstacle gain.

Environmental and Movement Information	
Parameter	Value
clutter	2.0
wander	-1
clear-to-goal flag	-1
goal-nearby flag	-1
circles	1
granny	-1
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	$[-0.05\ -0.05]$	[0.05 2.0]
noise gain	[0.0 0.05]	[0.01 1.5]
noise persistence	[1.0 1.0]	[1.0 5.0]
object gain	[0.01 0.01]	$[-1.0\ -1.0]$
sensible distance	[0.0 1.0]	[2.0 5.0]

Bookkeeping Information	
Parameter	Value
case number	0
case goodness	0.2
average step size	0.5
dynamic obstacles	0
initial distance to goal	37.56
obstacle danger	0.5
goal importance	1.0

3. Squeezing: When there are many obstacles, the system attempts to find a path by squeezing between obstacles by lowering obstacle gain and increasing goal gain.

Environmental and Movement Information	
Parameter	Value
clutter	2.0
wander	-1
clear-to-goal flag	-1
goal-nearby flag	-1
circles	-1
granny	-1
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	[0.05 0.05]	[0.05 2.0]
noise gain	[0.0 0.0]	[0.01 1.5]
noise persistence	[-1.0 0.0]	[1.0 5.0]
object gain	[-0.01 -0.01]	[-1.0 -1.0]
sensible distance	[-1.0 0.0]	[2.0 5.0]

Bookkeeping Information	
Parameter	Value
case number	1
case goodness	0.1
average step size	0.5
dynamic obstacles	0
initial distance to goal	37.56
obstacle danger	0.5
goal importance	1.0

4. HUGGING: When there are many obstacles and the system is currently faced with an obstacle directly in its path, it attempts to hug the side of the obstacle as it makes its way around it.

Environmental and Movement Information	
Parameter	Value
clutter	2.0
wander	-1
clear-to-goal flag	0
goal-nearby flag	1
circles	-1
granny	-1
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	[5.0 5.0]	[5.0 5.0]
noise gain	[0.0 0.0]	[0.0 0.0]
noise persistence	[1.0 1.0]	[1.0 1.0]
object gain	[0.0 5.0]	[0.0 5.0]
sensible distance	[0.0 0.0]	[0.0 2.0]

Bookkeeping Information	
Parameter	Value
case number	4
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

5. SHOOTING: If the system sees its goal and there are no obstacles in the direct path, it ignores any obstacles that might be around, adopts an extreme version of the CLEAR-FIELD strategy, and goes directly to the goal.

Environmental and Movement Information	
Parameter	Value
clutter	1.0
wander	−1
clear-to-goal flag	−1
goal-nearby flag	1
circles	0
granny	−1
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	[0.5 0.5]	[4.0 4.0]
noise gain	[−0.05 −0.05]	[0.0 1.0]
noise persistence	[−1.0 −1.0]	[1.0 1.0]
object gain	[0.0 1.0]	[0.0 5.0]
sensible distance	[−0.5 −0.5]	[0.0 4.0]

Bookkeeping Information	
Parameter	Value
case number	6
case goodness	0.9
average step size	0.5
dynamic obstacles	0
initial distance to goal	36.5
obstacle danger	0.5
goal importance	1.0

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 and if the incoming vectors from the obstacles are localized in front of it. In this situation, it heuristically determines in 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. This is done by creating a "pseudo-goal" which is located towards an estimated shorter side of the wall. When it reaches this pseudo-goal, the original goal is reinstated. Thus WALL-CRAWLING behavior is similar to HUGGING, but with a varying goal. Pseudo-goal creation is currently implemented in a procedural manner.

Environmental and Movement Information	
Parameter	Value
clutter	2.0
wander	-1
clear-to-goal flag	0
goal-nearby flag	1
circles	-1
granny	-1
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	[5.0 5.0]	[5.0 5.0]
noise gain	[0.0 0.0]	[0.0 0.0]
noise persistence	[1.0 1.0]	[1.0 1.0]
object gain	[0.0 5.0]	[0.0 5.0]
sensible distance	[0.0 0.0]	[0.0 2.0]

Bookkeeping Information	
Parameter	Value
case number	10
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

7. Random: The system raises the noise gain and goal gain, leaves the obstacle gain at a medium level, and wanders for a period of time. This is useful for exploration.

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

Behavioral Parameters		
Parameter	Delta Limits	Range
goal gain	[-0.5 0.5]	[0.0 6.0]
noise gain	[-0.1 0.1]	[1.0 4.0]
noise persistence	[-2.0 2.0]	[1.0 10.0]
object gain	[-0.5 0.5]	[1.0 5.0]
sensible distance	[-0.5 0.5]	[1.0 5.0]

Bookkeeping Information	
Parameter	Value
case number	5
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

8. GRANNY: After H_{steps}, the system reconsiders the environment by actually attempting to build a limited model of it. It concentrates on the location and sizes of the obstacles within its sphere of influence and attempts to choose the direction which offers the best success possibilities while deviating the least from the goal direction.

Environmental and Movement Information	
Parameter	Value
clutter	2.0
wander	−1
clear-to-goal flag	0
goal-nearby flag	0
circles	−1
granny	1
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	[0.0 0.0]	[0.5 0.5]
noise persistence	[0.0 0.0]	[1.0 1.0]
object gain	[−0.1 0.1]	[1.0 5.0]
sensible distance	[−.1 0.1]	[1.0 5.0]

Bookkeeping Information	
Parameter	Value
case number	11
case goodness	0.9
average step size	0.5
dynamic obstacles	0
initial distance to goal	36.5
obstacle danger	0.5
goal importance	1.0

9. MAXXED: This case also contains procedural information. As with other cases, it is triggered given a certain environmental condition. Triggering is also affected when ACBARR notices that a value has stayed at a maximum point for a period of time. At that point, ACBARR enacts a procedure which increases the allowable range of the parameter which has maxxed out.

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

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.

Environmental and Movement Information	
Parameter	Value
clutter	−1
wander	0
clear-to-goal flag	0
goal-nearby flag	1
circles	−1
granny	−1
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	[−1.0 −1.0]	[−1.0 −1.0]
noise gain	[0.0 0.0]	[0.0 0.0]
noise persistence	[1.0 1.0]	[1.0 1.0]
object gain	[0.0 5.0]	[0.0 5.0]
sensible distance	[0.0 5.0]	[0.0 5.0]

Bookkeeping Information	
Parameter	Value
case number	3
case goodness	0.9
average step size	0.5
dynamic obstacles	0
initial distance to goal	36.5
obstacle danger	0.5
goal importance	1.0