AAAI Fall Symposium on "AI for Real-World Autonomous Mobile Robots", Cambridge, MA, October 1992.

A Case-Based Approach to Reactive Control for Autonomous Robots *

Kenneth Moorman and **Ashwin Ram**
College of Computing
Georgia Institute of Technology

Abstract

We propose a case-based method of selecting behavior sets as an addition to traditional reactive robotic control systems. The new system (ACBARR — A Case BAsed Reactive Robotic system) provides more flexible performance in novel environments, as well as overcoming a standard "hard" problem for reactive systems, the box canyon. Additionally, ACBARR is designed in a manner which is intended to remain as close to pure reactive control as possible. Higher level reasoning and memory functions are intentionally kept to a minimum. As a result, the new reasoning does not significantly slow the system down from pure reactive speeds.

Introduction

Reactive robotic control systems [Arkin, 1989a, Brooks, 1986] have produced impressive results in the area of generating intelligent robotic action. Unlike traditional approaches, these systems typically decompose actions into simple behaviors in order to produce rapid real-time response to the environment. Generally, however, the behaviors which such a system has access to 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 allowed to alter to become even more successful. A third problem is that most reactive control systems do not utilize sets of behaviors; instead, they rely on a single type of behavior to guide the system. In order to achieve better robotic control, we feel that behavior sets need to be used along with adaptations of existing behaviors to new environments. 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 sets of behaviors based on the current environment in which it finds itself.

Our work attempts to incorporate both of these solutions 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.

*This work was supported by a Fannie and John Hertz Foundation fellowship and by the Georgia Institute of Technology

The other method used in our system is a global level one. If the system is currently acting under the control of a set of behaviors which are no longer suited to the current environment, it should select a new set 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 set 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 set of motor behaviors, one better suited to the new surroundings.

This paper describes our work on ACBARR (A Case-BAsed Reactive Robotic system), a system which performs both types of behavior modifications described above. The work is a continuation of a line of research involving reactive systems in general [Arkin, 1989a, Clark *et al.*, 1992]. In addition, the system uses *case-based reasoning* [Kolodner, 1990] to implement the global level of modifications. The system presented is extremely robust, doing 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 significantly worse. As a result, the system is a step closer to the lofty goal of having a truly autonomous robot—one which can act and react in novel situations without the need for human aid.

Overview of Reactive Control using Schemas

Schema-based control is a type of reactive robotic control system currently being used in the Autonomous Robot Architecture (AuRA) that is implemented on the Georgia Institute of Technology autonomous robot, GEORGE [Arkin, 1990]. Under this system, each basic type of motor behavior is represented by a *schema*. By selecting the proper schema or combination of schemas to employ the robot is able to react to its environment. Various schemas have been proposed; the ones used in this research are the following:

- **Avoid-Obstacle**: Repel from object with variable gain and sphere of influence. Used for collision avoidance.

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

  where:
  - S = Adjustable sphere of influence (radial extent of force from the center of the obstacle)
  - R = Radius of obstacle
  - G = Adjustable gain
  - d = Distance of robot to center of obstacle

  $O_{direction}$ = Along a line from robot to center of obstacle moving away from obstacle

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

  $V_{magnitude}$ = Adjustable gain value
  $V_{direction}$ = Direction towards perceived goal

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

  $N_{magnitude}$ = Adjustable gain value
  $N_{direction}$ = Random direction that persists for $N_{persistence}$ steps ($N_{persistence}$ is adjustable)

`Move-to-Goal` creates an attraction between the robot and the desired goal location, `Avoid-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. For the purposes of this research, it is assumed that there is only one goal and multiple obstacles.

In order for the system to make use of these schemas, there are various *gains* associated with each. These gain values determine the impact level that a particular schema has on the overall system behavior. 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 bit of noise affects the system is variable. Finally, the concept of *sensible distance* 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.

Reactive control forms the base level of robotic control within our system. Additionally, the system permits the adaptation of values in order for a behavior set to become better suited for a given environment [Clark *et al.*, 1992]. In this way, the local adjustment technique discussed earlier is implemented within ACBARR. There is still the question of how to implement the more global behavior changes. To do this, case-based reasoning is used to select sets of behaviors appropriate for the current environment as perceived by the system.

Overview of Case-Based Reasoning

Case-based reasoning (CBR) is an artificial intelligence technique which relies on past experiences to guide the reasoning process [Kolodner, 1990]. The hope in CBR 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 CBR program will generate is dependent on the following factors:

- the actual experiences (*cases*) which the system has had
- the ability to recognize that a new situation is like a previous one (*retrieval*)
- *adaptation* of an old case to a new situation
- the ability to *evaluate* these retrieval and adaptation processes and to *repair* any deficiencies which are identified

One of the key issues in case-bases 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 (see, for example, [Kolodner, 1992]), but none are totally effective. 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 switch cases. There is also the problem of what to do if there are no complete matches in the case library when attempting to perform a retrieval, only partial matches. In order for the global control to be implemented, all of these problems with case-based reasoning must be handled in some way.

The ACBARR System

ACBARR is a combination of a traditional reactive control system with ideas from case-based reasoning research. Figure 1 illustrates the system's high-level architecture. Data about the world is gathered by the *environmental monitor* and combined with the robot's status, which involves how well the robot is moving in relation to the world. The system uses this environmental information for two things. It adjusts the current gain values of the active schema, and it monitors the feedback

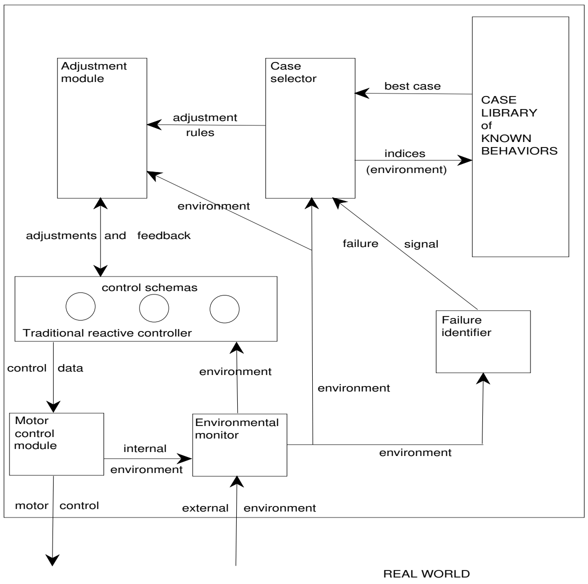

Figure 1: System architecture for ACBARR

for a failure situation. If the *failure identifier* determines that the current schema is failing, then the *case selector* chooses a better behavior schema from the case library.

A case within ACBARR's case library represents a set of behaviors which is well suited for a particular environment. It contains both the desired behavior and 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. The case selector has to determine which index environment in the stored cases best matches the current environment and then switch to that case.

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

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 this high level of 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. 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.

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 sensible distance of the robot. To aid in 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 indicates if the goal is within the robot's sensible distance with nothing between the goal and the robot.
- *Goal-Nearby* is a binary value which indicates if there the robot senses the goal but there

are obstacles between the two.

- *Goal-Direction* is a value, in radian angle measure, which indicates which direction 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 in circles.

- *Granny* is a binary variable which indicates if the ultra-careful behavior should be selected.

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:

- **No-Movement**: The robot's average step size has dropped below a certain threshold.
  $M < T_{movement}$
  where:
  M = Step size averaged over H_{steps}
  $T_{movement}$ = Movement threshold

- **Movement-Toward-Goal**: The robot's step size and rate of approach to the goal are both above a threshold.
  $M \geq T_{movement}$
  $P = \frac{H_{distance}}{H_{goal}} \geq T_{progress}$
  where:
  $H_{distance}$ = Distance travelled over H_{steps}
  H_{goal} = Decrease in distance to goal over H_{steps}
  $T_{progress}$ = Progress threshold

- **No-Progress-With-Obstacles**: The robot is moving but not toward the goal and there are several obstacles within a sensible distance.
  $M \geq T_{movement}$
  $P = \frac{H_{distance}}{H_{goal}} < T_{progress}$
  $O_{count} \geq T_{obstacles}$
  where:
  O_{count} = Average number of sensible obstacles over H_{steps}
  $T_{obstacles}$ = Obstacle count threshold

- **No-Progress-No-Obstacles**: The robot is moving but not toward the goal and there are no obstacles within a sensible distance.
  $M \geq T_{movement}$
  $P = \frac{H_{distance}}{H_{goal}} < T_{progress}$
  $O_{count} < T_{obstacles}$

Short term memory

ACBARR also employs a primitive short term memory (STM) mechanism in order to recognize whether or not the robot is running in circles. The system 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. There are not adequate resources to maintain an in-depth model of a dynamic environment while running in real-time. As a compromise, the STM system in ACBARR only keeps track of the robot's travels within a small *window* of the entire world. The window is divided into *quadrants* through which the robot travels. As long as the robot remains within the window, its progress in each quadrant is recorded. 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, then a new window is established and the old one is forgotten.

ACBARR case description

An ACBARR case entry consists of three sections. The first part represents bookkeeping information about the case. The second section details how the various gain values can change and what their limits are. The remainder of the case is a breakdown of the types of environments in which this case is applicable. This last section is used when determining which case to switch to.

A sample case is shown in Figure 2. Reading the data from the first table, we see that this is case 1 which has a goodness rating of 90 percent. Additionally, we know that the average step size was 0.5, there were no dynamic obstacles, and that the system was initially 37.56 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.

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.

Finally, 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 environment. The robot is making progress towards the goal and is not wandering or running in circles. The

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

Gain Value Adjustments		
	Delta limits	Range
Noise Persistence	[−1.0 0.0]	[1.0 5.0]
Noise Gain	[−0.05 0.0]	[0.01 1.5]
Goal Gain	[0.0 0.05]	[0.05 2.0]
Object Gain	[−0.01 −0.01]	[1.0 5.0]
Sensible Distance	[−1.0 0.0]	[2.0 5.0]

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

Figure 2: Sample ACBARR case

case doesn't care if the goal is sensed or not, nor does it care about the other three movement flags. If this case is selected, then the modifications of gains which it describes will take place.

Behaviors stored in the case library

A reactive robotic simulator[1] 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:

1. *Clear-Field*: In an open environment, the system should ignore the data about 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 the *goal gain*).
4. *Hugging*: When there are many obstacles and the system is currently faced with an obstacle directly in its path, it will attempt 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 it see its goal and there are no obstacles in the way, it will adopt the *clear-field* strategy and go directly to it.
6. *Wall-Crawling*: If there is an obstacle the system cannot seem to get around by *hugging*, it will check to see if it is actually in front of a wall. If so, it determines which way is the shorter side of it, and go for a distance in that direction.
7. *Random*: The system will raise the *noise gain* and *goal gain*, leave the *obstacle gain* at a medium level, and wander for a period of time.
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 *sensible distance* 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. This is accomplished by setting the *goal gain* to a negative amount.

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. The system can handle both of these scenarios with no problem.

When to select a new case There are two options available when using a case-based control system with regards to when a new case should be selected. The first one is to have the system look for a new case every H_{steps} steps. This ensures that the optimal behavior will always be in place. This is also a pessimistic strategy. It assumes that every H_{steps} steps the environment is going to be so changed that a new case is needed, even if the current one has

[1] 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].

not failed. The second option is to select a new case only if the current one is failing in some way. This is the optimistic control strategy. It assumes that a given case is "good enough" as long as it does not actually *fail*. This is the method currently used by ACBARR. It provides low system slowdown, while achieving respectable results.

How to determine failure ACBARR makes extensive use of its STM and environment knowledge to check for failing cases. There are two criteria for failing. The first is excessive wandering by the robot. This is kept track of by the *wander* entry in the environment data. If it rises above a given threshold, failure is assumed. The second 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, then failure can be assumed.

How cases are matched Finally, there is the question of actually selecting the best case from the case library. This is done in the ACBARR system through a *goodness-of-match* metric. The matching algorithm attempts to match the current environment to each known case. If corresponding internal values in the two cases are equal, then the goodness-of-match is incremented by 1. Near matches are also handled. If, for example, the current environment is very cluttered and the case being considered is just moderately cluttered, the goodness-of-match will have 0.5 added to it. After all cases have been examined, the one with the best match is chosen. 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 is good, a partial match can be chosen.

Some additional intelligence is also embedded in the ACBARR system. It has the option to choose either the best case from the library or to choose the best case without consideration of 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. By selecting to choose the best case without consideration of the current one, this problem is minimized.

Empirical Testing

To test the program, sample runs were performed with ACBARR. Three predefined environments were used, as well as several random ones. The predefined ones represented particularly difficult problems. These were a wall between the start position and the goal, a box canyon, and a quasi-box canyon.

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, the quasi-box canyon problem (a box canyon with an exit), and the wall environment (see Figure 5 through Figure 7) with no help from the human operator.

As a test of the ACBARR system, an *ablation study* was performed on it [Cohen and Howe, 1988], in which the system was tested without one or more of its components in order to evaluate their impact. Figure 3 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 4 shows the results of the system being run on the same world, this time with adjustments permitted. The goal is reached, although the path is wasteful. Finally, Figure 5 shows the complete ACBARR system's performance. With the benefits of both adjustments and case knowledge, the system achieves its goal with a fairly good path.

It was noticed in the empirical tests that the system would generally use only a subset of its stored possible strategies during test runs. There were two reasons hypothesized for this behavior. First, the system only alters a strategy if that strategy is failing in some way. This means that the optimal strategy is not always in place. If the current strategy is "good

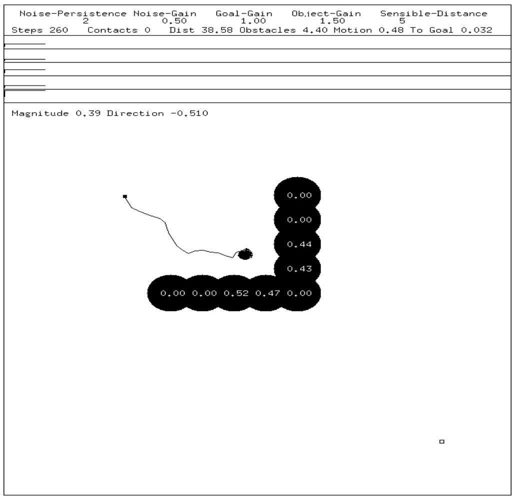

Figure 3: Box canyon — No cases, no adjustments allowed

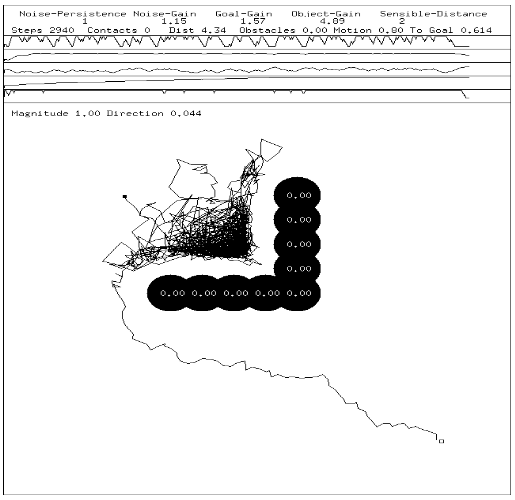

Figure 4: Box canyon — No case, but adjustments are permitted

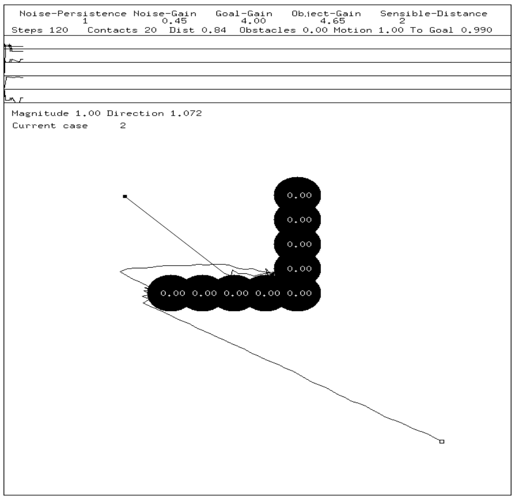

Figure 5: Box canyon — ACBARR

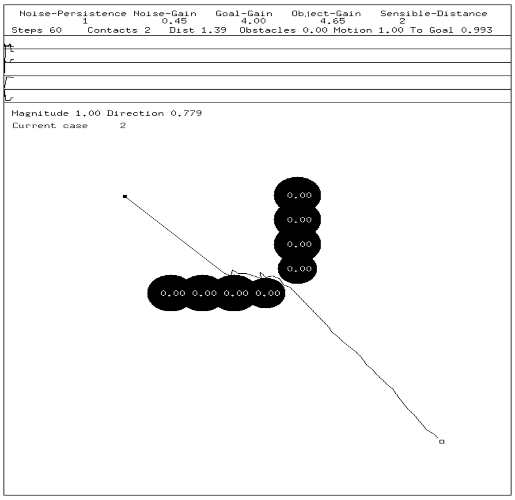

Figure 6: Quasi-box canyon — ACBARR

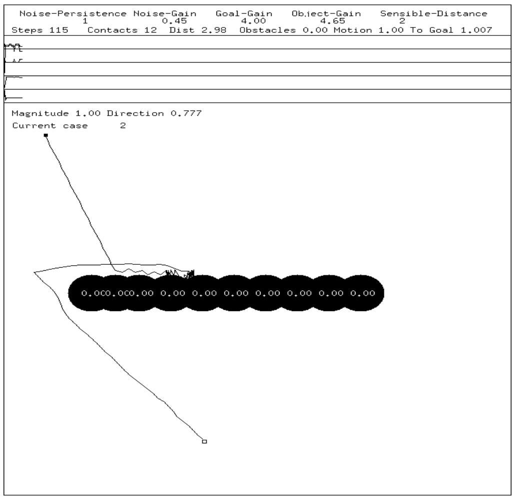

Figure 7: Wall environment problem — ACBARR

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* strategy, it will continue to do so until there is a clear failure. This reason was shown to be a correct explanation by altering ACBARR's control strategy so that new cases were selected every H_{steps} steps. Upon doing this, more movement strategies were utilized. The second reason for only a subset of strategies being used was the robustness of several of the strategies involved. In particular, *clear-field*, *hugging*, and *wall-crawling* are especially robust. These three alone can account for the majority of behaviors noted in the system.

Future work

An important issue at this point is where the set of cases in the library comes from initially. 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, these did not turn out to be the most used ones by the system. Luckily, there was enough variety within the hand-created cases to allow the system a relatively comprehensive selection. 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. If the system had the ability to learn its own cases, this problem could be alleviated. At the very least, the system needs to be able to add new cases to an already existing library. At the most, it would 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 this automatic case learning.

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, partially 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.

Conclusion

Intuitively, combining case selection based on environment with traditional reactive robotic control systems should lead to better performance. The empirical data supports this claim as well. By adding basic environmental data to the system, we have increased the level of its success. We have not sacrificed the inherent benefits of reactive control. Although the ACBARR system is not a pure reactive control system as traditionally defined, it combines the best features of that paradigm with the benefits of case-based reasoning. There is still the chance that ACBARR will fail in certain environments, although no such failures were identified in the extensive empirical testing. 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.

References

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

[Arkin, 1989b] Ronald C. Arkin. Towards the unification of navigational planning and reactive control. In the working note: AAAI Fall Symposium on Robot Navigation, May 1989.

[Arkin, 1990] Ronald C. Arkin. Integrating behavioral, perceptual, and world knowledge in reactive navigation. In Pattie Maes, editor, *Designing Autonomous Agents*. The MIT Press, Cambridge, Massachusetts, 1990.

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

[Clark *et al.*, 1992] Russell J. Clark, Ronald C. Arkin, and Ashwin Ram. Learning momentum: On-line performance enhancement for reactive systems. Submitted to 1992 IEEE International Conference on Robotics and Automation, Nice, France, May 1992.

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

[Kolodner, 1990] Janet L. Kolodner. An introduction to case-based reasoning. Technical Report GIT-ICS-90/19, Georgia Institute of Technology, 1990.

[Kolodner, 1992] Janet L. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann Publishers, San Mateo, CA, 1992. In press.