AI Magazine Volume 14 Number 1 (1993) (© AAAI)

On the Role of Stored Internal State in the Control of Autonomous Mobile Robots

Erann Gat

■ This article informally examines the role of stored internal state (that is, memory) in the control of autonomous mobile robots. The difficulties associated with using stored internal state are reviewed. It is argued that the underlying cause of these problems is the implicit predictions contained within the state, and, therefore, many of the problems can be solved by taking care that the internal state contains information only about predictable aspects of the environment. One way of accomplishing this is to maintain internal state only at a high level of abstraction. The resulting information can be used to guide the actions of a robot but should not be used to control these actions directly; local sensor information is still necessary for immediate control. A mechanism to detect and recover from failures is also required. A control architecture embodying these design principles is briefly described. This architecture was successfully used to control real-world and simulated real-world autonomous mobile robots performing complex navigation tasks. The architecture is able to incorporate standard AI planning and world-modeling algorithms into a real-time situated framework.

Controlling autonomous mobile robots consists of generating actuator commands in response to local sensor information and stored internal state (that is, memory). Researchers differ about the relative importance of these two factors. Classical approaches, in which robots construct complete plans prior to taking any action, rely on state information to the near-total exclusion of local sensor data (for example, Nilsson [1980]). Reactive approaches, brandishing slogans such as "the world is its own best model," prefer local sensor data and tend to avoid the use of internal state whenever possible (for example, Connell [1989]).

Reactive approaches arose as a sort of "popular revolt" backlash against the shortcomings of traditional sense-plan-act architectures. When the slow, plodding operation of traditional approaches proved ineffective at dealing with the realities of reality, Rodney Brooks turned the problem sideways: Instead of general-purpose functional modules, Brooks proposed an architecture based on special-purpose, task-oriented modules that coupled sensors directly to actuators. The dramatic result of Brooks's work was that many tasks that had previously been thought to be difficult (most notably, the task of collision-free navigation) turned out to be achievable using simple control mechanisms (Brooks 1986).

In the subsequent euphoria, internal state was somehow transformed into a greater villain than Brooks ever intended.1 No reactive architecture advocates the wholesale elimination of internal state. Internal state is necessary, reactivists readily concede, but a few odd bytes scattered here and there should suffice. It is classical planning, along with its associated centralized world models, abstractions, and large linked data structures, that the reactivists would do away with, to be replaced by distributed networks of small finite-state machines or something similar.

Unfortunately, the line between where a few odd bytes of internal state leave off and large, centralized linked data structures begin is fuzzy. Because it is possible to embed arbitrary computations in (potentially infinite) networks of finite-state machines, it is possible for the evils of planning to arise in reactive architectures if one is not careful. Furthermore, even small amounts of isolated internal state can cause problems. If an erroneous sensor datum, say, is stored in memory and is used to control the actions of the robot, then this error persists for longer than it would have if the robot's actions had been based on updated information. (This argument was one of the original ones in favor of reactive architectures.)

In this article, I argue that the ongoing debate about planning versus reacting is really an argument about the proper use of internal state information, which is, in turn, an argument about making predictions about the world. I further argue that although the indiscriminate use of internal state can lead to problems, the solution is not to eschew internal state but, rather, to manage it more carefully. In particular, I argue that internal state should be maintained at a high level of abstraction and that it should be used to guide a robot's actions but not to control these actions directly. I also argue that planning can usefully be viewed as merely another form of sensor processing, and therefore, the problem of deciding when to plan is the same as deciding when to sense. (This observation does not simplify either of these difficult problems, but it does allow them to be consolidated.)

To illustrate these issues, consider a person driving a car to an unfamiliar destination. This task is impossible using local sensor information alone because in most places, no information is available locally to indicate where the destination is. Some state information (that is, a world model) is required: a map, prior knowledge about the layout of the streets, and so on. However, local sensors are necessary because there are many details that an a priori world model cannot supply—the locations of other cars, for example. In general, people use state information to construct rough plans at high levels of abstraction and local sensor information to fill in the details at run time. I might plan to, say, turn left at Main Street, but I don't need to know in advance precisely how far it is to Main Street; I just count intersections or drive until I see the sign.

The informal arguments and observations presented in this article were used to guide the design of a robot-control architecture that was demonstrated on a variety of real-world and simulated real-world robots performing complex tasks. The architecture is unique in that it incorporates unaltered classical planning techniques into a situated framework that operates in real time in an unpredictable environment. This work is briefly described at the end of the article.

Planning and Internal State

Let us begin by stipulating what is meant by the terms planning and plan. Classically, a *plan* is a sequence of operators, and *planning* is the process of constructing such sequences that will bring about certain desired effects given certain initial conditions. However, as Agre and Chapman (1990) pointed out, these definitions are not adequate to account for many of the phenomena that one might intuitively call planning; so, the meanings of the terms plan and planning must be broadened. McDermott (1991) recently defined a plan as any sort of construct that constrains or provides guidance for the actions of a robot. Thus, plans can be anything from a data structure to an analog control circuit, and planning can be anything from classical STRIPS planning to circuit design. I adopt this notion of planning in this article.

Under this broad view of planning, it is useful to distinguish between *offline planning*, which is done before the robot engages in the performance of any task, and *online planning*, which occurs during the performance of a task. Offline planning can be done by either the robot (or other computer) or a human designer. Offline planning is typically not subject to strict time constraints and is usually of greatest utility when it makes as few assumptions as possible about the task and the state of the world. (*Note*: Classical planning has almost always been tacitly considered offline; the quality or optimality of a classical plan is invariably judged by considering only the plan and not the time or effort required to produce the plan.)

By contrast, online planning is planning done by the robot during the performance of a task. This article is concerned mainly with online planning. Online planning can be *classical*; that is, the robot can plan a complete course of action and then execute the plan. However, it is now clear that this simple

In this article, I argue that the ongoing debate about planning versus reacting is really an argument about the proper use of internal state information

If the world is completely predictable, and the world model is always correct, then classical online planning is feasible

approach does not work well. A number of variations on the classical theme have emerged to support online planning, usually involving some sort of interleaving of classical planning and the execution of the resulting plans. Because online planning occurs during the execution of a task, it is usually subject to more stringent time constraints than offline planning and can often make more assumptions about the robot's task and current state than an offline planner can.

Online planning is problematic for three main reasons: First, planning is time consuming. It can cause the robot to miss deadlines or fail to respond in a timely manner to contingencies. (Oncoming trucks wait for no theorem prover.) Second, most planners require a world model at a level of completeness and fidelity that current sensor technology cannot provide. Third, while the robot is planning, the world can change in a way that invalidates the plan (for example, passing a critical exit while planning a route on the freeway).

The first problem can be solved by operating the planner in parallel with a process that deals with contingencies. This solution still leaves the second problem and exacerbates the third because the robot can now take actions while planning that the planner did not anticipate. Note, however, that both of these remaining problems now arise because of difficulties in maintaining internal state information. The second problem involves the explicit state contained in the world model, and the third problem involves the implicit state contained in the execution thread of the planner.

Note two important points: First, the problems manifest themselves only when the information contained in the internal state does not match the actual state of affairs in the environment. If the world is completely predictable, and the world model is always correct, then classical online planning is feasible. Problems arise when there is a mismatch, for any reason, between the semantics of some piece of stored internal state and the actual state of affairs in the world.

Second, these difficulties are not unique to planning but apply to all sequential computations. Any sequential computation generates internal state that persists for the duration of the computation (otherwise, it is not a sequential computation). Thus, any time-consuming computation, planning or otherwise, is a potential source of difficulty if something unexpected happens in the world while the computation is going on. A reactive obstacle-avoidance algorithm can fail, for example, if it relies on a time-consuming stereo-matching algorithm for input. If objects in the world move between the time a frame is grabbed and the time the stereo matching is complete, then a collision is possible even though no planning took place.

The reactivists' solution to these problems is to minimize the lifetime and the quantity of internally stored data by eliminating time-consuming sequential computations and stored world models. However, by examining more closely the underlying cause of the problems, we can do better. It is not necessary to abandon planning and world models, only to be a bit more careful about how they are used.

Internal State and Predictions

The root of the difficulties caused by internal state is not anything inherent in the state itself but, rather, in the implicit predictions made by the storage of internal state information. Every piece of internal state carries with it an implicit prediction that the information contained in this state will continue to be valid for some time to come. These predictions (whether implicit or explicit), coupled with the unpredictability of the world, are the root of the problem.

Let us consider two examples to clarify this point. Consider a simple forward-searching linear planner that expands a world-state node on its search tree. The prediction in this case is explicit: The planner is predicting that the performance of a certain action under a certain set of circumstances will cause certain changes in the state of the world. This is an explicit prediction about the outcome of performing an action. There is also an implicit prediction about the way the world will evolve until the time comes to perform the action.

As a second example, consider a single sonar range reading stored in a memory location. The prediction made by this internal state depends on how it is used. Suppose, for example, that the robot now moves in the direction of this reading for some distance less than the reading (with the intention that the robot not collide with obstacles). Under such circumstances, there are implicit predictions that the obstacle detected by the sonar will not move toward the robot and that no new obstacles will enter the robot's path. (There is also the assumption that the range reading was correct to begin with.) In most real-world environments, none of these predictions is likely to be correct, and the use of such internal state is likely to cause problems. Note that these problems arose entirely without any classical planning or other time-consuming computations and without any complex, centralized, linked world models; the mere use of a stored sensor datum can cause persistent errors in the robot's actions.

Viewed in this way, avoiding internal state is tantamount to assuming that nothing about the world can usefully be predicted, which is plainly not the case. Internal state, planning, world models, and the associated classical computational mechanisms might yet be salvageable if we take care to ensure that the predictions made by the internal state contained within them are reasonable predictions about the world. In the next section, I suggest one way of ensuring reasonable predictions.

Predictions and Abstraction

One way to make predictions that are likely to remain valid for the lifetime of the internal state of the system is to make predictions at a high level of abstraction. Abstraction, like internal state, has been vilified unfairly by reactivists. An *abstraction* is simply a description of something. The level of an abstraction is simply an inverse measure of the precision of the description. A higher-level abstraction is a less precise description. Abstraction is enormously useful for making predictions because the accuracy of an imprecise description is unaffected by small variations in the state of affairs. Thus, for example, a prediction such as "I will be in my office this afternoon" is more likely to be correct than "I will be seated at my computer at 2:49 PM typing the third line of the fourth section of my *AI Magazine* article."

Humans tend to remember such abstracted information about the world. In my own internal state, I store, for example, the location of my house. By remembering this fact, I make the implicit prediction that my house will remain (more or less) in the same place. However, I don't really know exactly where my house is; all I know is the topological layout of the streets surrounding it together with some rather rough metric information. For all I know, my house and the surrounding neighborhood might move several meters in random directions whenever I am at work. Items within the house actually do tend to move around mysteriously while I am away. Nevertheless, I am able to find my house and my belongings because I have a world model that describes the state of affairs at a high level of abstraction: My house is on Margaret Drive, which is the first left after turning onto El Nido from Colorado Boulevard; the mustard is in the refrigerator, which is in the kitchen.

This imprecise stored information is crucial to almost everything people do. I go to the refrigerator when I want mustard not because any local sensor information tells me that the mustard is there but because I remember that it is there. However, sensor data are crucial to fill in the details that the world model does not provide. Once I open the refrigerator, I usually have to hunt for the mustard because it tends to wander around between uses. I can't drive with my eyes closed because there are aspects of the world that cannot be included in my world model even in principle because they are impossible to predict (for example, the state of traffic lights).

Of course, world models are sometimes wrong even at high levels of abstraction. Sometimes, a street on the way to my house is closed for construction. Sometimes, my wife uses the last of the mustard and forgets to tell me. Thus, actions based on stored state sometimes fail. In humans, occasional errors of this sort are not a problem for two reasons: First, people use the information in their world models to guide their actions but not to control them directly. Second, people can tell when things go wrong and can take corrective action. This ability to fail cognizantly is the main reason that people manage to get along in the world. People make mistakes all the time. Usually, they are able to detect these mistakes and recover. We carefully engineer our environment to eliminate the opportunity to make the sort of mistakes that are difficult to recover from—falling off cliffs and the like.

Abstract descriptions of the world allow the unpredictable aspects of the world to be abstracted away. What remains is incomplete but is nevertheless useful information that can be used to guide a robot's actions. For the resulting system to be robust, it is important to ensure that the system is able to detect failures when they occur, as they inevitably will. In the next section, I describe an implemented robot control system that successfully integrates classical planning into a real-time situated robot-control system by following these guidelines.

A Robot-Control Architecture

This article proposes a number of principles to guide the design of autonomous robot-con-

One way to make predictions that are likely to remain valid for the lifetime of the internal state of the system is to make predictions at a high level of abstraction

Primitive activities are used as building blocks to construct highly conditional sequences of primitives that accomplish more complex goals under a larger variety of circumstances

trol architectures. To review briefly, these are to (1) maintain state information and construct plans at a high level of abstraction, (2) use the results to guide the robot's actions but not to control them, and (3) provide a mechanism to detect and recover from failures. These design principles were incorporated into the ATLANTIS control architecture (Gat 1991, 1992), which was successfully used to control a variety of autonomous mobile robots in both the real world and simulated real-world conditions.

The Architecture

The architecture consists of three components: the controller, the sequencer, and the deliberator.

The *controller* is responsible for moment-by-moment control of the robot's actuators in response to the current values of the robot's sensors. The controller is a purely reactive control system. Because of the data flow nature of this component, a special programming language called ALFA was developed to program it.

The *sequencer* is responsible for selecting which of several transfer functions the controller is to compute and parameterizing this transfer function as appropriate. The sequencer is also responsible for taking corrective actions in the event of failures.

The *deliberator* is responsible for maintaining world models and constructing plans. The deliberator performs all manner of time-consuming computations. These computations occur at a high level of abstraction (see Experiments for an example), and the results are not used to control the robot directly. Instead, results of deliberative computations are placed in a database, where they are used as an additional source of input data by the sequencer. (See Mataric [1990] for an example of a reactive architecture, where the results of planning are used to control the robot directly.)

The ATLANTIS sequencer is modeled after Firby's (1989) RAP system and is the prime mover in the architecture. All the activity in the controller, as well as the deliberator, is controlled by the sequencer. This control structure is a consequence of the observation that a structural parallel exists between planning and sensor processing: Both are time-consuming and resource-consuming processes that produce information to guide the robot's actions (Rosenschein and Kaelbling 1986).

Design Methodology

The design methodology associated with the architecture advocates bottom-up design. First, a set of transfer functions is designed for the controller that produces a set of useful primitive behaviors to be used as building blocks for more complex activities. Examples of primitives are following walls, avoiding obstacles, and going through doors. Each primitive is engineered to fail cognizantly, that is, to detect failures when they occur. For example, a wall-following primitive should be engineered to detect when the robot reaches the end of the wall or when the robot loses its alignment with the wall because of accumulated sensor errors.

Primitive activities are then used as building blocks to construct highly conditional sequences of primitives that accomplish more complex goals under a larger variety of circumstances. An example of such a higher-level activity is moving to a destination given a set of directions. This process might involve generating a lengthy sequence of following walls, going through doors, and registering landmarks. Many of the steps in the sequence might be contingent on things that happen at run time. For example, if a wall that the robot wanted to follow turned out to be blocked by an obstacle, the robot might have to invoke a contingency procedure to get around the obstacle.

Deliberative computations are performed only when some information is needed by the sequencer, which requires time-consuming computations. For example, if a robot is given multiple tasks, it might invoke a task planner to decide which task it should perform next to make efficient use of its resources. The deliberator might also be called on to perform time-consuming sensor processing and world modeling such as stereo vision processing.

Experiments

ATLANTIS was used to control a number of real-world robots (three to date) and a simulated real-world robot. Detailed accounts of these experiments can be found in Gat (1991, 1992). To give a flavor of the system's capabilities, two snapshots of a simulator experiment are shown in figures 1 and 2. The simulator is a fairly accurate simulation of ROBBIE, the Jet Propulsion Laboratory planetary rover test bed. In this experiment, the robot is given three tasks: collect and deliver rock samples of various colors, keep the robot fueled, and photograph martians (the M-shaped figures) that move about in semirandom ways. The world is full of obstacles (shaded rectangles) that the robot has no advance knowledge of. Its primary sensor is a

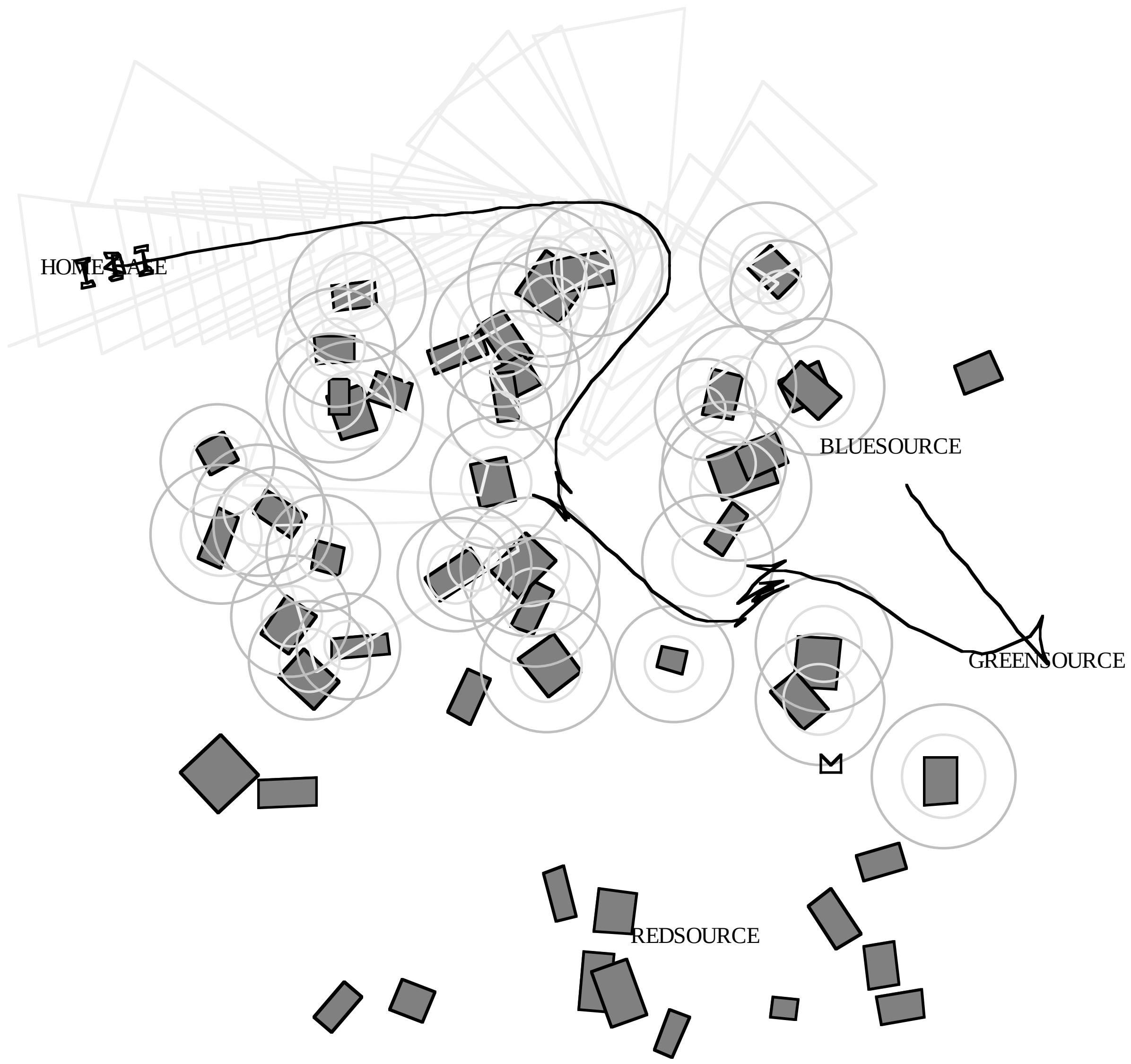

Figure 1. A Snapshot of a Typical Run of the ATLANTIS *System.*

After visiting two of the robot's three destinations, the deliberator determines that there is not enough fuel to visit the third destination and return home safely. The plan produced by the deliberator says simply, "Return home." All the details are filled in at run time by the reactive components, allowing the plan to remain valid even though unexpected obstacles (which were not detected because of sensor noise) are encountered along the way.

fairly slow and unreliable stereo vision system[2] that the robot must actively control to scan areas of interest. The circles are obstacles that the robot detected.

The sequencer in this experiment controlled two physical processes running in the controller—(1) controlling the robot's speed and direction and (2) aiming the camera—and five computational processes running in the deliberator—(1) local navigation (using an algorithm similar to Slack [1990]), (2) task planning (using a stripped-down version of the planner in Miller [1985]), (3) stereo vision processing, (4) global map maintenance and route planning, and (5) the computation of intercept trajectories for the martians. The system runs three times faster than real time, including the central processing unit time needed to run the simulation. The simulator gives a fairly high-fidelity simulation of a real robot and is able to reproduce results obtained on the real robot.

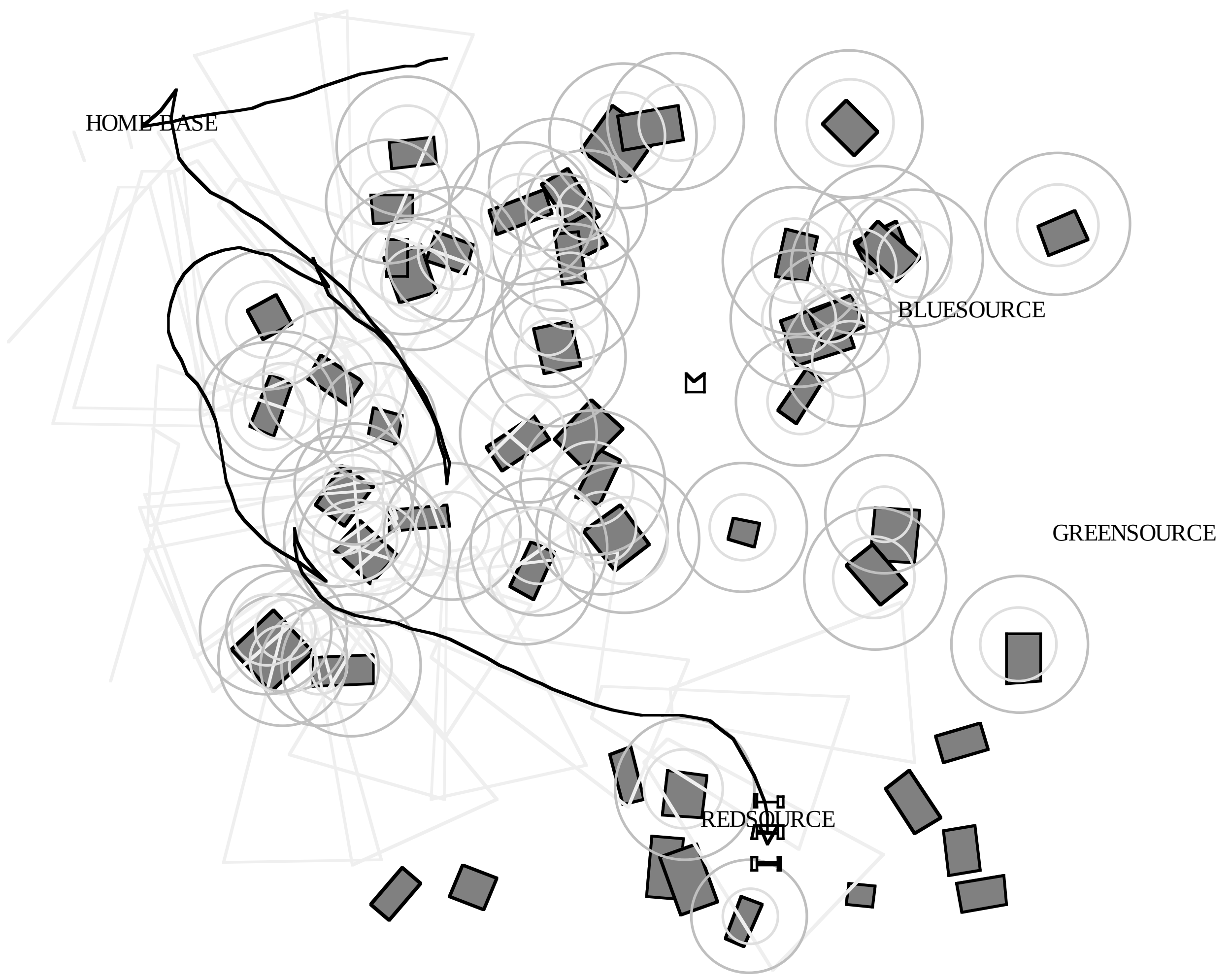

Figure 2. The Continuation of the Snapshot in Figure 1.

After refueling, the robot proceeds to its third destination. An unexpected obstacle blocks the robot's route, requiring a dramatic change in route but no change in the global plan. (Compare the locations of known obstacles—designated by circles—in this figure and figure 1.)

The course of events shown in figure 1 is particularly interesting because it demonstrates a behavior that is difficult to achieve without planning. In this figure, the robot has the task of collecting one blue rock, one green rock, and one red rock and returning them to the home base. The robot begins by picking up a blue and a green rock. However, before picking up a red rock, the robot returns to the home base because the task planner determined that the robot was in danger of running out of fuel if it tried to get the red rock first.

The task planner in this experiment chose the robot's next destination based on how much fuel it had available; how much rock storage space remained; what color rocks it had already collected; and, roughly, what the travel time was between locations. The global route planner simply assigned *spins* to obstacle groups; that is, it decided whether to traverse an obstacle to the right or the left. All

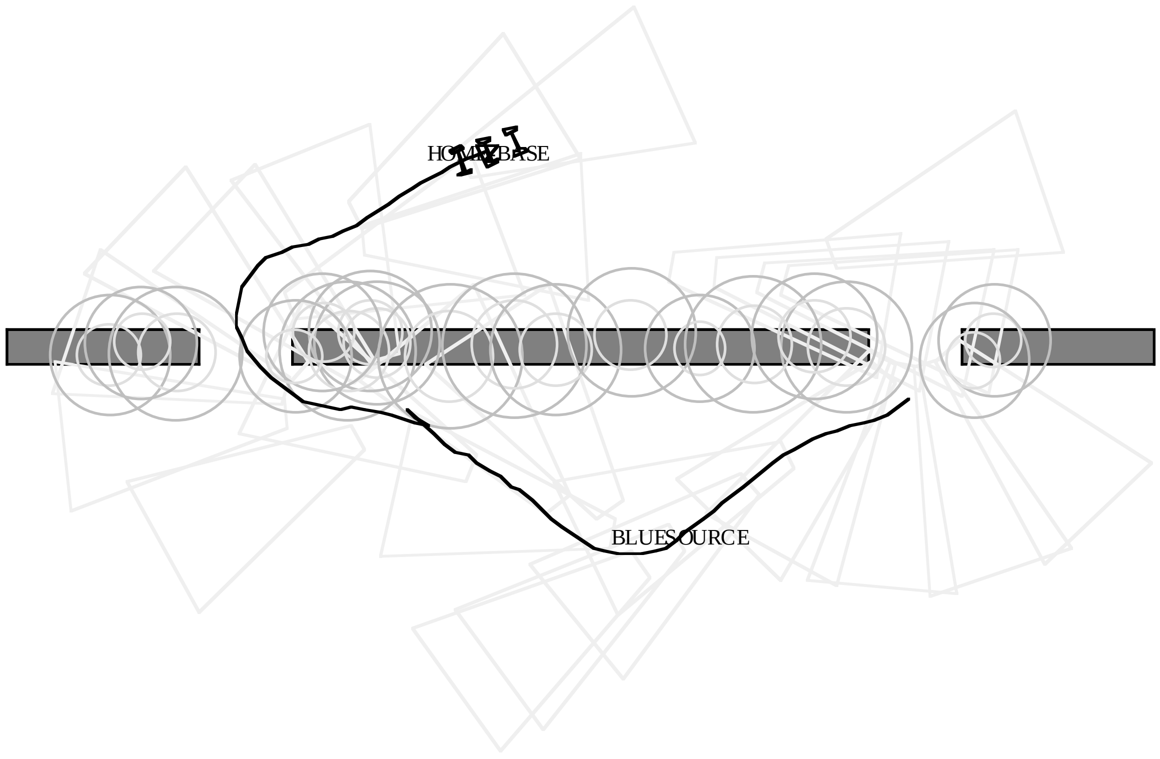

Figure 3. The Original Route Planner Used in the System Assumed That Unknown Areas Were Free of Obstacles.

This assumption can produce inefficient behavior. In this run, the robot went through the opening on the right to reach its destination. On the return trip, it travels toward the unknown left side of the figure. If there were not a second opening in the wall, this strategy would have resulted in an extremely inefficient path.

the details of controlling the robot's exact path were left to the controller, which continuously readjusted the robot's course. Sometimes, there were failures, such as when the robot tried to squeeze between two obstacle groups that led to a dead end (figure 2). In this case, the failure was detected, and the robot chose a new global route.

This experiment demonstrates that by maintaining internal state and planning at high levels of abstraction, classical planning algorithms can be integrated usefully into a situated control architecture operating in real time in realistic environments. There is no need to abandon classical planning to control mobile robots.

Modularity

One of the benefits of the ATLANTIS architecture is that it provides a clean separation between symbolic computations and control of actions. This separation makes it easy to make modifications to the symbolic computations to improve or modify the robot's performance under varying circumstances. For example, in the previous illustration, the robot maintains a world model that contains information about previously encountered obstacles. Thus, if the robot were again asked to go from its home base to the red rock quarry, it would not go into the dead end. However, avoiding the dead end is the right thing to do only if obstacles never move. If there is a possibility that something might move one of the obstacles out of the way, then it would be worthwhile to go back into the dead end to check it now and then.

Similarly, built into the robot's navigation strategy is the assumption that areas that have not yet been seen are free of obstacles, which can sometimes lead to inefficient behavior. For example, figure 3 shows the performance of the original navigation strategy on a simplified task. The robot starts at the home base and proceeds to the blue rock quarry through the opening on the right. At this point, only the right half of the wall has been seen, which makes it appear that the

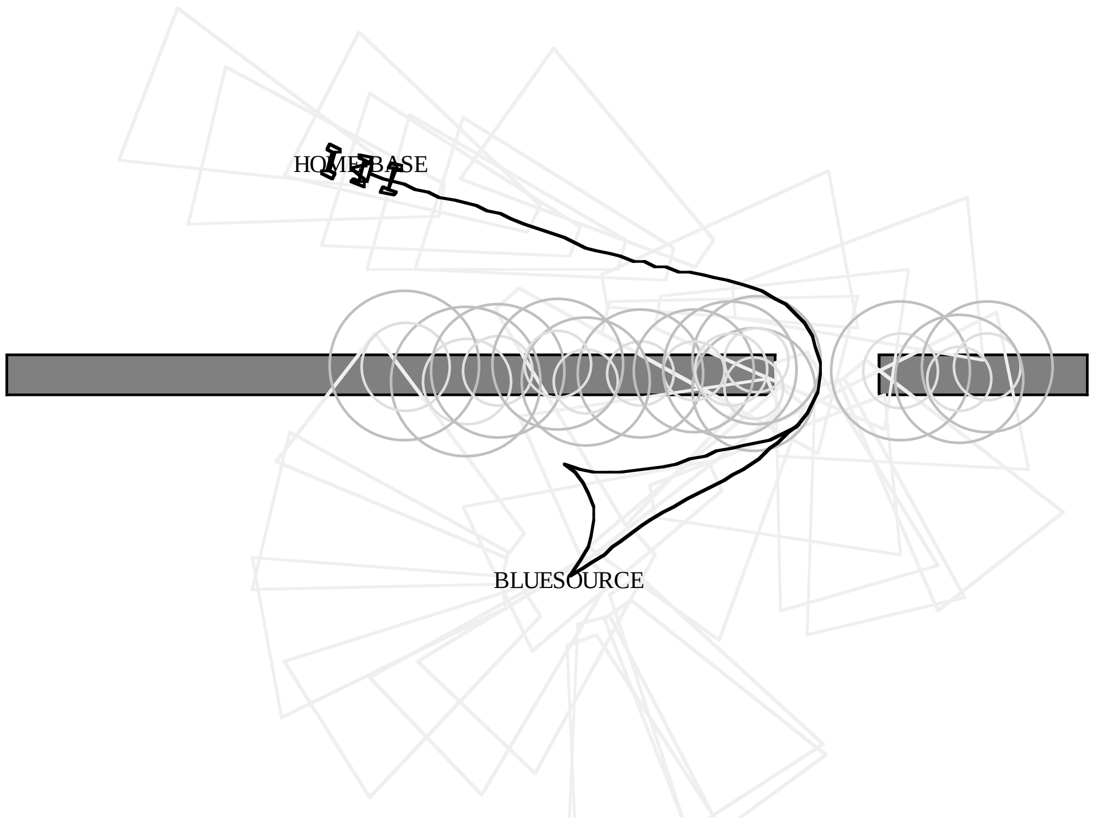

Figure 4. Because Plans in ATLANTIS *Are Used Only as Advice, It Is Easy to Interchange Different Planning and World-Modeling Strategies.*

In this run, a completely new planner is being used that builds a map of *corridors*, narrow openings between obstacles. This planner prefers paths through known corridors over paths through unknown regions. It took less than a day to integrate this new planner with the rest of the system, which was not changed.

fastest way home is to proceed to the left. However, this strategy is risky because the wall might extend for an arbitrary distance to the left.

In a world of long walls, a different navigation strategy is needed. To demonstrate the ease with which such a strategy can be integrated into ATLANTIS, a completely new high-level planner was written that not only implemented a different navigation strategy but a different world-modeling scheme as well. The new planner modeled the world as a collection of *corridors*, narrow openings between two nearby obstacles. A topological network of corridor sequences between adjacent destinations was maintained, and a simple best-first search planner operated on this model. The resulting behavior was that the robot tended to move between destinations using routes that it had previously traveled (figure 4).

The main point is not that this algorithm is a particularly effective navigation strategy but that it was incorporated into the existing ATLANTIS framework with no modifications whatsoever to existing code (other than removing the old high-level planner). The fast installation of a new planner was possible because the two planners produced the same sort of output—general advice to guide the rest of the system about where to go next.

Conclusions

I made the following argument in this article: Online planning (and control of mobile robots in general) requires the maintenance of internal state. Internal state contains implicit predictions about the world. It is important to ensure that these implicit predictions are at least usually correct. One way to make this assurance is to store internal state information (and, therefore, to plan) at a high level of abstraction. The resulting plans and world models can usefully be employed to guide the actions of an autonomous mobile robot but not to control its actions directly. Because the predictions implicit in the internal state can be wrong even when maintained at high levels of abstraction, the immediate control of the robot must ultimately rely on local sensor data. A robot can't drive with its eyes closed, but neither can it reach its destination knowing nothing about the surroundings beyond what is locally perceptible.

These observations were embodied in a robot-control architecture that was implemented on several real robots and a simulation of a real-world robot. The architecture is capable of controlling these robots in the performance of multiple, complex tasks in real time with high reliability using fairly simple algorithms. The system does classical planning at high levels of abstraction and uses the results to guide a reactive controller that uses local sensor information to directly control the robot's actuators. Robustness is achieved through the mechanism of cognizant failure, which allows the robot to detect and recover from unexpected problems.

Acknowledgments

Some of the ideas in this article grew out of discussions with Rodney Brooks, Paul Viola, David P. Miller, Marc Slack, and Jim Firby. Portions of the simulator and navigation software were written by Jim Firby and Marc Slack. This work was performed in part by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration and in part by the Massachusetts Institute of Technology Artificial Intelligence Laboratory under the sponsorship of Rodney Brooks.

Notes

1. Rodney Brooks, Massachusetts Institute of Technology, 1990, conversation with Erann Gatt.

2. The vision system on the real robot is actually quite good. Extra noise is introduced artificially into the simulated vision to ensure that the system will work with noisy data.

References

Agre, P., and Chapman, D. 1990. What Are Plans For? *Robotics and Autonomous Systems* 6:17–34.

Brooks, R. 1986. A Robust Layered Control System for a Mobile Robot. *IEEE Journal on Robotics and Automation* RA-2(1): 14–23.

Connell, J. 1989. A Colony Architecture for an Artificial Creature, Technical Report, 1151, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Firby, R. 1989. Adaptive Execution in Complex Dynamic Worlds, Technical Report, YALEU/CSD/RR672, Dept. of Computer Science, Yale Univ.

Gat, E. 1992. Integrating Reaction and Planning in a Heterogeneous Asynchronous Architecture for Controlling Real-World Mobile Robots. In Proceedings of the Tenth National Conference on Artificial Intelligence, 810–815. Menlo Park, Calif.: American Association for Artificial Intelligence.

Gat, E. 1991. Robust Task-Directed Reactive Control of Autonomous Mobile Robots. Ph.D. diss., Dept. of Computer Science, Virginia Polytechnic Institute and State University.

McDermott, D. 1991. Robot Planning. Invited presentation given at the Tenth National Conference on Artificial Intelligence, Anaheim, California, 14–19 July.

Mataric, M. 1990. A Distributed Model for Mobile Robot Environment Learning and Navigation, Technical Report, 1228, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Miller, D. 1985. Planning by Search through Simulations, Technical Report, YALEU/CSD/RR423, Dept. of Computer Science, Yale Univ.

Nilsson, N. 1980. *Principles of Artificial Intelligence*. San Mateo, Calif.: Morgan Kaufmann.

Rosenschein, S., and Kaelbling, L. 1986. The Synthesis of Digital Machines with Provable Epistemic Properties, Technical Note 412, Artificial Intelligence Center, SRI International.

Slack, M. 1990. Situationally Driven Local Navigation for Mobile Robots, JPL Publication 90-17, Jet Propulsion Laboratory, California Institute of Technology.

Erann Gat received a Ph.D. from the Virginia Polytechnic Institute and State University in 1991. He is currently a member of the technical staff at the Jet Propulsion Laboratory, California Institute of Technology, where he has been working on autonomous mobile robots since 1988.