From: AAAI-92 Proceedings. Copyright ©1992, AAAI (www.aaai.org). All rights reserved.

Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real-World Mobile Robots

Erann Gat
Jet Propulsion Lab, California Institute of Technology
4800 Oak Grove Drive
Pasadena, California 91109
gat@robotics.jpl.nasa.gov

ABSTRACT

This paper presents a heterogeneous, asynchronous architecture for controlling autonomous mobile robots which is capable of controlling a robot performing multiple tasks in real time in noisy, unpredictable environments. The architecture produces behavior which is reliable, task-directed (and taskable), and reactive to contingencies. Experiments on real and simulated real-world robots are described. The architecture smoothly integrates planning and reacting by performing these two functions asynchronously using heterogeneous architectural elements, and using the results of planning to guide the robot's actions but not to control them directly. The architecture can thus be viewed as a concrete implementation of Agre and Chapman's plans-as-communications theory. The central result of this work is to show that completely unmodified classical AI programming methodologies using centralized world models can be usefully incorporated into real-world embedded reactive systems.

1. Introduction

We have been investigating the problem of controlling autonomous mobile robots in real world environments in a way which is reliable, task-directed (and taskable), and reactive to contingencies. The result of our research is a control architecture called ATLANTIS[1] which combines a reactive control mechanism with a traditional planning system. In this paper we describe a series of experiments using the architecture to control real-world and simulated real-world robots. We demonstrate that the architecture is capable of pursuing multiple goals in real time in a noisy, partially unpredictable environment. The central result of the work is to show how a traditional symbolic planner can be smoothly integrated into an embedded system. We begin by reviewing the difficulties associated with embedding AI systems into real-world robots.

Controlling autonomous mobile robots is hard for three fundamental reasons. First, the time available to decide what to do is limited. A mobile robot must operate at the pace of its environment. (Elevator doors and oncoming trucks wait for no theorem prover.) Second, many aspects of the world are unpredictable, making it infeasible to plan a complete course of action in advance. Third, sensors cannot provide complete and accurate information about the environment. These are fundamental problems because they cannot ever be engineered away. No matter how powerful a computer we build, a finite amount of time will allow only a finite amount of computation. No matter how good a sensor we may build there is always information that it cannot deliver because the relevant situation is hidden behind a wall or across town. No matter how good our domain theory may be, many important aspects of the world simply cannot be predicted reliably.

Related Work: Classically the problem of mobile robot control has been addressed within a framework of functional decomposition into sensing, planning and acting components. There is a vast literature on the traditional sense-plan-act architecture and its variations. A good recent example of a sense-plan-act approach to mobile robot control appears in [Stentz90].

An alternative approach spearheaded by Brooks advocates decomposition of the robot control problem into special-purpose task-achieving modules (often called behaviors) rather than into general-purpose functional modules [Brooks86]. One of the interesting results of this work is that useful behaviors can be built out of very simple computations with very little internal state. (This result is often mistakenly believed to imply that reactive control is one of the tenets of the behavior-based approach. Behavior-based control makes reactive control possible, but it does not mandate it.)

A number of researchers have described systems which integrate these two approaches (e.g. [Connell91], [Kaelbling88], [Soldo90], [Arkin90], [Georgeff87]), or which start with one approach and try to push its capabilities towards that of the other (e.g. [Mataric90], [Simmons90]). Most of these systems are homogeneous, that is, they use basically the same computational structure throughout. ([Connell91] is a notable exception.)

Overview: This paper introduces ATLANTIS, a *heterogeneous, asynchronous* architecture for controlling mobile robots which combines a traditional AI planner with a reactive (not necessarily behavior-based) control mechanism. The next section describes the theory behind the approach. Section 3 describes the architecture. Section 4 describes experiments using the architecture to control real-world and simulated real-world mobile robots

[1] ATLANTIS is an acronym which stands for (among other things): A Three-Layer Architecture for Navigating Through Intricate Situations.

performing multiple tasks in real time in noisy, unpredictable environments. Section 5 summarizes, presents conclusions, and suggests directions for future research.

This paper is constrained by space limits to be somewhat superficial. For more complete technical details see [Gat91a], and forthcoming papers.

2. Activities and Decisions

The ATLANTIS architecture is based on an action model which is different from most traditional AI systems. This section presents a brief review of this model. For a more complete discussion see [Gat91a].

A majority of past work in AI on robot control architectures and planning systems is based fundamentally on a state-action model. This model is based on the idea that the temporal evolution of the configuration of a system (or an environment) can be described as a sequence of discrete *states*. One state is transformed into a subsequent state by the performance of an *action*. The word action is variously used to denote both the physical action as well as a computational structure which represents the physical action. The latter is sometimes called an *operator*, a term which we will adopt here to avoid ambiguity.

According to the classical paradigm, an action is produced by *executing* an operator. In order to distinguish between this technical notion of action as the physical activity produced by the execution of an operator, and the idea of physical activity in general, we will capitalize the former. Thus, executing an operator produces an Action, but an action might be produced in other ways.

There is a very close correspondence between Actions and operators, which is one reason the terms are sometimes used interchangeably. The process of execution is atomic, resulting in a strict one-to-one correspondence between operators an Actions. This structure facilitates analysis, but makes it difficult to model simultaneous, interleaving, or overlapping actions. It also makes it impossible to model a process where the execution of an atomic action is abandoned in the middle in response to a contingency.

ATLANTIS is based on a continuous action model similar to those described in [Miller84], [Hogge88], and [Dean88]. The ATLANTIS action model is based on operators whose execution consumes negligible time, and thus do not themselves bring about changes in the world but instead initiate processes which then cause change. These processes are called *activities*, and the operators which initiate (and terminate) them are called *decisions*.

A decision may initiate an activity which contains computational processes which initiate other activities. If we assume that there are no circularities in this network of initiations then we may classify activities into a hierarchy. High-level activities contain computational processes which initiate low-level activities. The hierarchy bottoms out in *primitive activities*, reactive sensorimotor processes which contain no decision-making computations.

Because there is no strict correspondence between decisions and changes in the world, an activity-based model of action is more difficult to analyze than a state-action model. Such an analysis is beyond the scope of this paper. Instead, we will use the activity model as an engineering tool to help us organize the computations required to produce robust behavior in our robots. The key observation is that activities consist of potentially overlapping sequences of primitive (physical) activities and computational activities. The results of the computations are used to guide the sequencing of primitives. Thus, to control a mobile robot we need three things: a control mechanism for controlling primitive activities, a computational mechanism for performing decision-making computations, and a sequencing system to control the interactions between the two. The next section describes such a system.

3. An Architecture for Navigation

This section briefly describes ATLANTIS, a *heterogeneous asynchronous* architecture for controlling mobile robots based on the activity model of action described in section 2. ATLANTIS consists of three components. The *controller* is a reactive control mechanism which controls primitive activities, i.e. activities with no decision-making computations. The *sequencer* is a special-purpose operating system which controls the initiation and termination of primitive activities, and of time-consuming deliberative computations like planning and world modelling which are performed in the *deliberator*.

3.1 The controller: This component is responsible for controlling primitive activities, that is, activities which are (mostly) reactive sensorimotor processes. It is possible to design the controller for a given application using nothing but classical control theory. However, in many cases control theory cannot be applied directly to the problem of controlling autonomous mobile robots because of the difficulty in constructing an adequate mathematical model of the environment. In such cases it is necessary to provide a framework wherein an appropriate control algorithm can be effectively engineered and empirically verified.

There are a great many issues which must be addressed in the design of such a system, not the least of which are control-theoretical issues. For now we shall lay these aside, concentrating instead on the computational organization of the system that allows a designer to conveniently engineer systems that do the right thing. The sorts of transfer functions required to control reactive robots tend to be highly nonlinear, of high dimension, and often discontinuous. The design of the system must be such that we can easily describe the functions we need

and, having defined them, actually implement them in a way that lets them be connected to actual hardware.

To support these requirements we have designed a new programming language called ALFA (A Language For Action) [Gat91b]. ALFA is similar in spirit to REX [Kaelbling87], but the sorts of abstractions the two languages provide are quite different. ALFA programs consist of computational *modules* which are connected to each other and to the outside world by means of communications *channels*. Both the computations performed and their interconnections are specified within module definitions, allowing modules to be inserted and removed without having to restructure the communications network. ALFA provides both dataflow and state-machine computational models. It has a clean syntax and a realistic uniform interface to external hardware. ALFA is currently compiled into uniprocessor code, but the semantics of the language are such that it could be compiled onto a parallel processor or even analog hardware.

3.2 The sequencer: This component is responsible for controlling sequences of primitive activities and deliberative computations. Controlling sequences is difficult primarily because the sequencer must be able to deal effectively with unexpected failures. This requires the careful maintenance of a great deal of internal state information because the sequencer must be able to remember what actions have been taken in the past in order to decide what action should be taken now.

The fundamental design principle underlying the sequencer is the notion of cognizant failure [Firby89]. A cognizant failure is a failure which the system can detect somehow. Rather than design algorithms which never fail, we instead use algorithms which (almost) never fail to detect a failure. There are two reasons for doing this. First, it is much easier to design navigation algorithms which fail cognizantly than ones which never fail. Second, if a failure is detected then corrective action can be taken to recover from that failure. Thus, algorithms with high failure rates can be combined into an algorithm whose overall failure rate is quite low provided that the failures are cognizant failures [Howe91].

The sequencer initiates and terminates primitive activities by activating and deactivating sets of modules in the controller. In addition, the sequencer can send parameters to the controller by means of channels. The progress of a primitive activity is monitored by examining the values of channels provided for this purpose.

The sequencer is modelled after Firby's Reactive Action Package (RAP) system. The system maintains a task queue, which is simply a list of tasks that the system must perform. Each task contains a list of methods for performing that task, together with annotations describing under what circumstances each method is applicable. A method is either a primitive action or a list of sub-tasks to be installed onto the task queue. The system works by successively expanding tasks on the queue until they either finish or fail. When a task fails, and alternate method is tried. (cf. [Simmons90], [Noriels90]).

The main difference between the original RAP system and the ATLANTIS sequencer is that the latter controls activities rather than atomic actions. This requires a few modifications to the original RAP system. First, the system must insure that two activities which interfere with each other are not enabled simultaneously. This is accomplished by attaching to each activity a list of resources that the activity requires and using a set of semaphores to prevent conflicts. Second, if a primitive activity must be interrupted (for example, to take care of some unexpected contingency) then the system must insure that the interrupted activity is properly terminated so that the modules which control that activity are disabled and any resources used by that activity are relinquished. The solution to this problem is to provide a mechanism similar to a LISP unwind-protect which allows an interrupted process to execute some clean-up procedures before relinquishing control.

3.3 The deliberator: This component is responsible for performing time-consuming computational tasks such as planning and maintaining world models. The deliberator performs computations under the control of the sequencer - all deliberative computations are initiated (and may be terminated before completion) by the sequencer. This allows the sequencer to direct scarce computational resources to the task at hand. Results of deliberative computations are placed in a database which can be accessed by the sequencer. The deliberator has no restrictions on its computational structure except that the sequencer must be able to initiate and terminate its computations. It typically consists of a set of LISP programs implementing traditional AI algorithms. This is a central feature of the system. The function of the sequencer and controller is to provide an interface which connects to physical sensors and actuators on one end, and to classical AI algorithms implemented in traditional ways on the other.

The following interesting question now arises: how does a planner based on a traditional AI state-action model interface with a control mechanism based on a continuous activity model? It turns out that this interface is quite straightforward. Because the output of the planner is used only as advice by the sequencer, it doesn't matter at all what the planner's internal representation is. The only requirement is that the output of the planner contains some information which the sequencer can effectively use. ATLANTIS can be considered an implementation of Agre and Chapman's theory of plans-as-communications (or plans-as-advice) [Agre90]. A concrete example is described in section 4.

3.4 Design methodology: ATLANTIS advocates a bottom-up design methodology (cf. [Simmons90]). Primitive activities are designed first,

keeping in mind that they must be designed to fail cognizantly. The primitives are then used as fundamental building blocks for the construction of task schemas for the sequencer task library. Finally, deliberative computations are designed to support the sequencer in making choices among multiple tasks, task methods, or in supplying task method parameters.

Although there are no restrictions on the computational structure of the deliberator, there is a caveat concerning the semantics of its computations. Deliberative computations by definition are time-consuming and maintain internal state information which contains implicit predictions about the world. Thus it is critical that the information content of the internal state pertain to predictable aspects of the environment. One way to do this is to insure that all deliberative computations are performed at a high level of abstraction where unpredictable aspects of the environment are abstracted away to be dealt with at runtime by the rest of the architecture.

4 Experiments

ALFA, and the ATLANTIS architecture and design methodology have been implemented on a variety of real robots operating in both indoor and outdoor environments. ALFA has been used to control Tooth, an extremely reliable indoor object-collecting robot [Gat91c]. The language has also been used to program the JPL outdoor microrover testbed Rocky III, the only example known to the author of an autonomous outdoor robot which collects and returns samples [Miller91]. A significant result of this work was that the control structures for these two robots was very similar, indicating that the abstractions used to program them may be more widely applicable. ALFA and a simple sequencer were also used to control an indoor robot performing a complex navigation task using very simple sensors [Gat91d]. (cf. [Mataric90], [Connell91]).

The ATLANTIS architecture has been used to control Robbie, the JPL Mars rover testbed [Wilcox87]. Robbie is a large outdoor testbed whose primary sensor is a pair of stereo cameras. A trace of a typical experiment is shown in figure 1. The robot's path is shown as a solid black line starting at the right. The light polygons are the areas scanned by the robot's stereo vision cameras. The shaded circles are obstacles detected by the robot during the traverse. The total length of the path is about forty meters. The robot moved at about two meters per minute. (The robot has now been retrofitted with a new drive mechanism which should significantly improve this performance.)

The sequencer in this experiment coordinated four active tasks running concurrently during the traverse: controlling the vehicle's direction, controlling the aiming of the stereo camera, and allocating processor time to the stereo processing and planning tasks running in the deliberator. The navigation task used an algorithm based on navigation templates (NaTs) [Slack90] to avoid obstacles. This algorithm used a symbolic map constructed by the vision system, together with the current strategic plan constructed by the deliberator, to quickly calculate a preferred direction of travel from the robot's current location about three times a minute.

The robot was given three goals and no advance knowledge of the environment whatsoever. The robot initially planned to achieve goal B first, then goal A, then goal C. On the way to the second goal, however, it acquired new obstacle data which indicated that goal C would be easier to achieve next. The deliberator, running asynchronously, advised the sequencer to temporarily abandon goal A in favor of goal C.

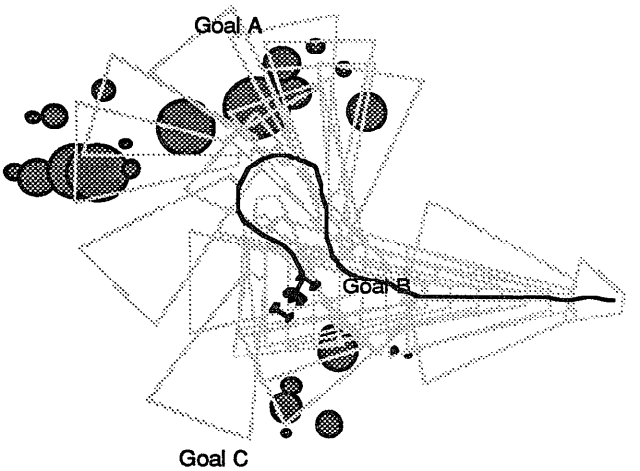

Figure 1: An outdoor robot performing a complex navigation task.

Simulation results: In order to facilitate experimentation, a sophisticated simulation of the Robbie robot was constructed. The simulation operates in real time, and includes an accurate kinematic simulation of the robot, as well as a sensor model with tunable noise parameters which can be adjusted to yield performance very close to the real robot. (In most of our simulator experiments we adjusted the noise parameters to give significantly *worse* performance than the real robot.)

The performance of the simulator was verified by reproducing the results obtained on the real robot. (See figure 2.) The code controlling the simulated robot was identical to the code which controlled the real robot. The world model built up by the real robot in the outdoor experiment was used as the simulator's internal world model (shown as shaded circles in the figure). The simulated robot had no direct access to this model, but could access it only through the simulated vision system. The obstacles actually detected by the robot are shown as pairs of hollow circles (representing uncertainty ranges on the size of the obstacle).

The results of the simulated run are qualitatively identical to the experiment on the real robot. The small quantitative differences are due to the random differences in the sensed world model due to sensor noise. While a single experiment does not warrant sweeping conclusions, these results do indicate that the performance of the simulator is not totally out of step with reality. (Extensive informal experience with the simulator also indicates that its performance is comparable to the real robot.)

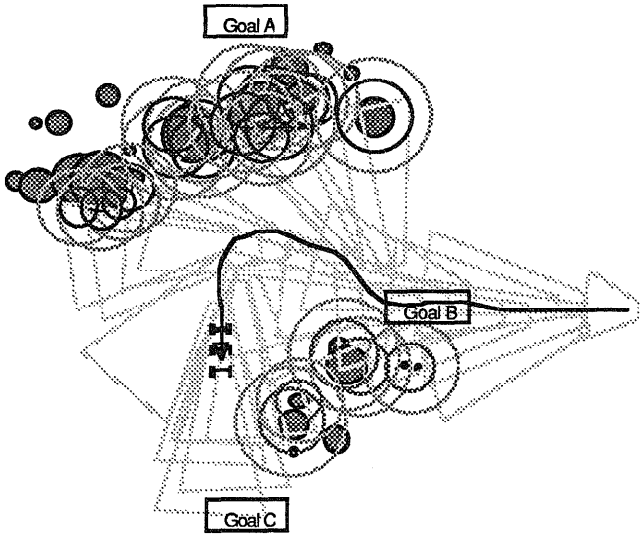

Figure 2: A simulated recreation of the outdoor navigation experiment.

Multiple tasks: The simulator was used to perform an extensive series of experiments in an augmented environment far more complex than that available in reality. (See figure 3.) First, a set of random obstacle fields were generated with obstacle densities far higher than those on our actual test course (shown as shaded rectangles in the figure). Second, the noise parameters of the simulated vision system were set to produce data which were sparser and noisier by an order of magnitude than on the real robot. Third, the environment was augmented with a set of simulated martians which roamed about in semi-random trajectories (shown as M-shaped objects). Fourth, a set of sample sites and a home base were added to the world model (labelled "REDSOURCE", "HOME-BASE", etc.).

The robot was given three tasks in this augmented environment: to photograph as many martians as possible, to collect and deliver samples from the sample sites to various destinations according to orders which were given to the robot at runtime (an example of a meta-goal), and to keep itself refueled by visiting home-base periodically.

To support this experiment, a task planner was written based on the work of Miller [Miller84]. The planner is a simple linear planner which performs a forward beam-search through a space of world states. The search is kept to manageable size through a set of powerful heuristics. The planner can deal with issues of limited resources, deadlines, and travel time. This planner was implemented entirely in Lisp.

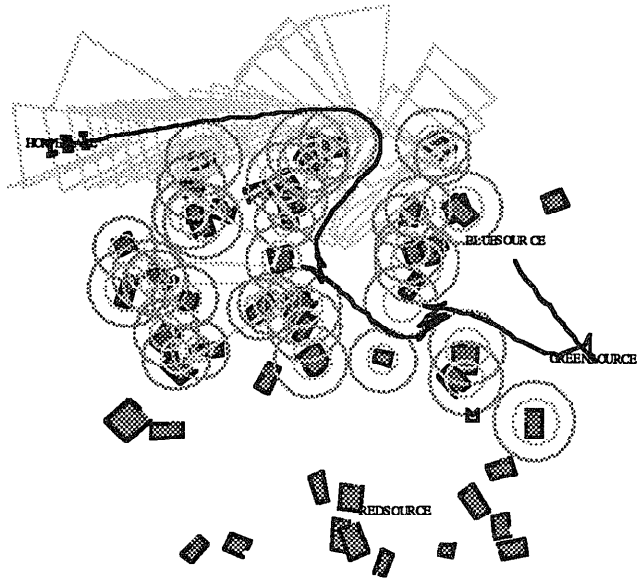

Figure 3: A complex delivery task.

To use the plan, the sequencer looked at the current first step which could be one of four things: collect or deliver a sample, refuel, photograph a martian, or go to a new destination. If the first step was to refuel or collect or deliver a sample, this step was simply executed as if it were a classical operator (since the simulator has no real-time model of manipulation or refueling). However, if the next step of the plan was to photograph a martian or to go to a new location, the sequencer extracted the target martian or the goal location and initiated an activity to aim the camera in the direction of the martian, or to go to that new location. Because the code to control navigation and camera aiming had already been developed and debugged, the information from the planner could be seamlessly incorporated as inputs to parameters in the code for controlling those activities. Furthermore, because the planner interfaced to the rest of the system only through previously designated inputs, there was a high degree of confidence that the combined system would work properly without modification to existing code. (This was confirmed by a subsequent experiment - see the last paragraph of this section.)

The rock-collecting martian-photographing system has logged over twenty hours of runtime, and over thirty kilometers of (simulated) traversed terrain. The longest

single run to date lasted eight hours, and we have no reason to believe that the system would not run indefinitely. Two snapshots of the system in action are shown in figure 3 and 4.

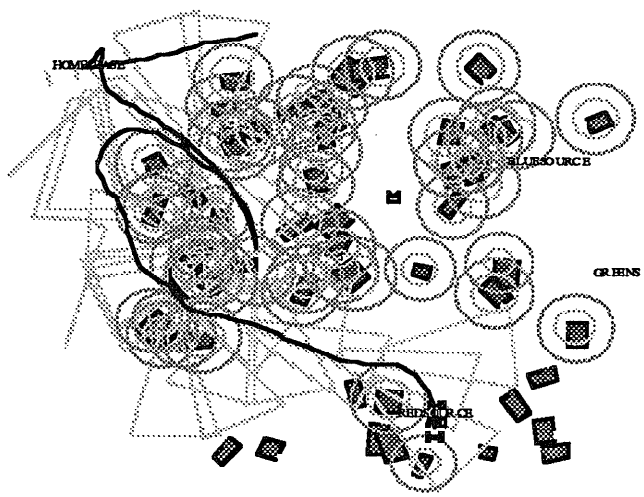

Figure 4: Recovering from failure.

The snapshot shown in figure 3 is particularly interesting because it shows an example of the planner interacting with the rest of the system. In this case the robot's goals were to collect a sample from each of the three sample sites and return them to the home base. The robot begins by collecting green and blue rocks. However, before collecting red rocks the robot returns to the home base. This is because the task planner, which had been running asynchronously, determined that there was not enough fuel to collect red rocks and safely return to the home base. Thus, the planner advised the sequencer to go back to refuel first. On the way, the robot encounters an obstacle blocking its intended route which was not detected by the vision system due to noise, forcing a detour. (A more dramatic example of this is shown in figure 4.) All of this is completely transparent to the task planner.

To demonstrate the ease with which different sorts of strategic planners could be incorporated into the architecture, a different planner (a topological path planner) was written and installed. The resulting system worked with no modifications to previously existing code. Details of this experiment can be found in [Gat91a].

5. Comparison to Other Work

In this section we contrast ATLANTIS with other current work. The purpose of these comparisons is to clarify the operation of ATLANTIS, and is not a comprehensive review of the literature. Only those architectures which are most similar to ATLANTIS are reviewed here.

One of the most similar architectures to ATLANTIS is Connell's recently introduced SSS architecture [Connell91]. SSS was developed independently of ATLANTIS at about the same time, and they share many of the same motivations and features. The primary differences between the two are: 1) ATLANTIS provides a more complete framework for engineering the controller, 2) The middle layer of SSS is based on Brooks' subsumption architecture whereas the sequencer in ATLANTIS is based on Firby's RAP system, and perhaps most important, 3) the symbolic layer of SSS is actually in the control loop, whereas the deliberator in ATLANTIS merely provides advice to the sequencer. Putting the symbolic layer in the control loop can adversely affect the real-time response of the system, requiring a special mechanism in SSS (the contingency table) to help circumvent the symbolic layer when speed is critical. In ATLANTIS, because the deliberator is not directly in the control loop its performance in no way affects the system's ability to respond to contingencies with dispatch. In fact, the deliberator can be completely removed and the resulting decapitated architecture is still quite capable of controlling a robot [Gat91d]. In this way, ATLANTIS achieves one the original aims of Brooks' subsumption architecture, namely, that the system should degrade gracefully with the failure of higher-level components.

ATLANTIS is the direct intellectual descendent of a complete control architecture described in the original work on RAPs [Firby89]. The main differences between ATLANTIS and the RAP architecture is that in the latter control flows top-down from the symbolic planner which installs tasks in the task queue (although the possibility of controlling symbolic computations by the sequencer is also suggested). The original RAP system also assumed a discrete action model and an optimistic sensor interface. It is interesting to note that most of the ideas in the RAP work turn out to extend with little or no modification to real-world sensors and continuous actuations. An architecture developed Bonasso is also very similar to ATLANTIS and shares many of its intellectual roots [Bonasso92].

6. Conclusions

We have shown that classical AI planning systems can be usefully embedded into a control mechanism for autonomous mobile robots operating in real world environments. Special compilation and implementation techniques are not required. Instead, a classical planner should be operated asynchronously in conjunction with a reactive control mechanism, and the planner's output should be used to guide the robot's actions but not to control them directly. This work can be viewed as an implementation of Agre and Chapman's plans-as-communications theory.

We have implemented a control architecture called ATLANTIS according to these principles on a variety of

real-world and simulated real-world robots operating in both indoor and outdoor environments. We have demonstrated that ATLANTIS can control a robot pursuing multiple goals in real time in a noisy, partially unpredictable environment. The robot's performance is reliable, task-directed and taskable, and reactive to contingencies.

We draw the following general conclusions:

1. Robot control architectures should be heterogeneous. Much effort has been expended trying to design architectures which perform strategic planning using the same computational structure which they use to do low-level motor control. There seems to be little to be gained by this. Using different computational mechanisms to perform different tasks is straightforward and it works.

2. Robot control architectures should be asynchronous. Slow computations should be performed in parallel with fast ones to allow fast reaction to contingencies. A continuous rather than a discrete action model should be used to allow actions to overlap or to be terminated before completing in response to unexpected situations.

3. Classical planning, abstraction, and centralized world models are at least useful, if not necessary, in real-world autonomous mobile robots. While it is certainly possible to implement planners and world models in non-classical or distributed ways, it is not clear that there are advantages in doing so over using established, classical techniques. Abstraction can be a powerful tool for dealing with unpredictable aspects of the environment, and need not introduce large control errors if the plans-as-advice model is followed.

4. Plans should be used to guide, not control, action. This is the view put forth by Agre and Chapman. We consider this work experimental evidence in support of their position.

Finally, we make one unsubstantiated conjecture:

5. Robot control systems should be designed bottom-up. This is a software engineering issue, and can probably be verified only with much more experience designing mobile robot control systems. (cf. [Simmons90])

Acknowledgements: This research was performed at the Jet Propulsion Laboratory, California Institute of Technology under a contract with the National Aeronautics and Space Administration. Many of the ideas in this paper grew out of discussions with Rodney Brooks, Jim Firby, Marc Slack, Paul Viola and David Miller. Portions of the control software for Robbie were written by Jim Firby, Marc Slack, Brian Cooper, Tam Nguyen and Larry Matthies. Portions of the simulator software were written by Jim Firby and Marc Slack.

References

[Agre90] Phil Agre, "What are Plans For?", *Robotics and Autonomous Systems*, vol. 6, pp. 17-34, 1990.

[Arkin90] Ronald C. Arkin, "Integrating Behavioral, Perceptual and World Knowledge in Reactive Navigation," *Robotics and Autonomous Systems*, vol. 6, pp. 105-122, 1990.

[Bonasso92] R. Peter Bonasso, "Using Parallel Program Specifications For Reactive Control of Underwater Vehicles," to appear, *Journal of Applied Intelligence*, Kluwer Academic Publishers, Norwell , MA, June 1992.

[Brooks86] Rodney A. Brooks, "A Robust Layered Control System for a Mobile Robot", *IEEE Journal on Robotics and Automation*, vol RA-2, no. 1, March 1986.

[Connell91] Jonathan Connell, "SSS: A Hybrid Architecture Applied to Robot Navigation," unpublished manuscript.

[Dean88] Tom Dean, R. James Firby and, David P. Miller, Hierarchical Planning with Deadlines and Resources, *Computational Intelligence* 4(4), 1988.

[Firby89] R. James Firby, Adaptive Execution in Dynamic Domains, Ph.D. thesis, Yale University, 1989.

[Gat91a] Erann Gat, "Reliable Goal-directed Reactive Control for Real-world Autonomous Mobile Robots", Ph.D. Thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia.

[Gat91b] Erann Gat, "ALFA: A Language for Programming Reactive Robotic Control Systems",*IEEE Conference on Robotics and Automation*, 1991.

[Gat91c] Erann Gat and David P. Miller, "Modular, Low-computation Robot Control for Object Acquisition and Retrieval," unpublished manuscript.

[Gat91d] Erann Gat, "Low-computation Sensor-driven Control for Task-directed Navigation," *IEEE Conference on Robotics and Automation*, 1991.

[Georgeff87] Michael Georgeff and Amy Lanskey, "Reactive Reasoning and Planning", *Proceedings of AAAI-87.*

[Hogge88] John Hogge, "Prevention Techniques for a Temporal Planner," *Proceedings of AAA88.*

[Howe91] Adele E. Howe, "Failure Recovery: A Model and Experiments," *Proceedings of AAAI91.*

[Kaelbling87] Leslie Pack Kaelbling, "REX: A Symbolic Language for the Design and Parallel Implementation of Embedded Systems," *Proceedings of the AIAA conference on Computers in Aerospace*, 1987.

[Kaelbling88] Leslie Pack Kaelbling, "Goals as Parallel Program Specifications", *Proceedings of AAAI-88.*

[Mataric90] Maja Mataric, "A Distributed Model for Mobile Robot Environment Learning and Navigation", Technical Report 1228, MIT AI Laboratory, 1990.

[Miller84] David P. Miller, "Planning by Search Through Simulations", Technical Report YALEU/CSD/RR423, Yale University, 1984.

[Miller91] David P. Miller, et al., "Reactive Navigation through Rough Terrain: Experimental Results," *Proceedings of AAAI92.*

[Noreils90] Fabrice Noreils, "Integrating Error Recovery in a Mobile Robot Control System," *IEEE International Conference on Robotics and Automation*, 1990.

[Simmons90] Reid Simmons, "An Architecture for Coordinating Planning, Sensing and Action," *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, 1990.

[Slack90] Marc G. Slack, "Situationally Driven Local Navigation for Mobile Robots", JPL Publication 90-17, California Institute of Technology Jet Propulsion Laboratory, April 1990.

[Soldo90] Monnett Soldo, "Reactive and Preplanned Control in a Mobile Robot," *IEEE International Conference on Robotics and Automation*, 1990.

[Stentz90] Anthony Stentz, "The Navlab System for Mobile Robot Navigation", Ph.D. Thesis, Carnegie Mellon University School of Computer Science, 1990.

[Wilcox92] Biran Wilcox, et al., "Robotic Vehicles for Planetary Exploration", *IEEE International Conference on Robotics and Automation*, 1992.