A Reactive Robot Architecture with Planning on Demand

Ananth Ranganathan Sven Koenig
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
{ananth,skoenig}@cc.gatech.edu

***Abstract*— In this paper, we describe a reactive robot architecture that uses fast replanning methods to avoid the shortcomings of reactive navigation, such as getting stuck in box canyons or in front of small openings. Our robot architecture differs from other robot architectures in that it gives planning progressively greater control of the robot if reactive navigation continues to fail, until planning controls the robot directly. Our first experiments on a Nomad robot and in simulation demonstrate that our robot architecture promises to simplify the programming of reactive robot architectures and results in robust navigation, smooth trajectories, and reasonably good navigation performance.**

I. Introduction

Reactive navigation is a popular robot-navigation approach since it is fast and relies only on the current sensor readings instead of a map, the use of which requires very accurate localization capabilities [Arkin98]. However, reactive navigation does not plan ahead and is therefore susceptible to local minima. For example, it can get stuck in box canyons or in front of small openings. These shortcomings are usually addressed by changing from one behavior to another in the reactive controller. The decision when to activate which behavior can be made either 1) before or 2) during execution.

1) In the first case, a programmer creates several behaviors, each of which is suited for a specific navigation scenario that the robot might get exposed to, for example, one behavior for navigation in corridors and another one for navigation outdoors. Then, the programmer encodes when to activate which behavior, for example, in the form of a finite state automaton (conditional off-line plan) whose states correspond to behaviors and whose transitions correspond to observations made during execution. An advantage of this scheme is that it results in good navigation performance if the programmer anticipated all navigation scenarios correctly. A disadvantage is that the terrain characteristics need to be approximately known. Also, the finite state automaton is terrain specific and can contain a large number of behaviors, which makes its programming time-consuming. The resulting navigation performance can be poor if the programmer did not anticipate all navigation scenarios correctly. Some schemes replace the programmer with an off-line learning method, resulting in similar advantages and disadvantages.
2) In the second case, the reactive controller uses only one behavior, but an on-line planner or learning method modifies the parameter values of the behavior during execution, for example, when the robot does not appear to make progress toward the goal. An advantage of this scheme is that it can solve navigation tasks even if the programmer failed to anticipate some (simple) navigation scenarios. A disadvantage is that it can result in poor navigation performance for some navigation scenarios since it takes time to detect when the parameters should be changed and experiment with how to change them.

In practice, one often uses a combination of both schemes, namely the first scheme for high-level terrain characteristics, that are often known in advance (for example, navigating through a forest), and the second scheme for low-level terrain characteristics, that are often not known in advance (for example, getting out of box canyons). The resulting navigation performance is good but programming is difficult since one has to choose behaviors, sequence them, and determine a large number of parameters in the process. We therefore explore an alternative scheme that utilizes on-line planning but whose reactive controller uses only one behavior without modifying the parameters of the behavior during execution. Our robot architecture requires only a small amount of programming (and testing) since one does not have to choose behaviors and sequence them. One only needs to determine a small number of parameters.

Combining planning and reactive navigation is not new. Many robot architectures use on-line planning to determine a nominal robot trajectory that reactive navigation has to follow. In this case, reactive navigation enables the robot to move around obstacles that planning did not know about or did not want to model. We, on the other hand, use on-line planning in a different way, namely to help reactive navigation when it is unable to make progress toward the goal. Our robot architecture differs from other robot architectures that use on-line planning in this way in that it gives the planner progressively greater control of the robot if reactive navigation continues to fail, until on-line planning controls the robot directly. The amount of planning and how closely the planner controls the robot therefore depend on the difficulty that reactive navigation has with the terrain.

The primary difficulty with implementing our robot architecture, and perhaps the reason why it is unusual to let on-line planning control robots directly, is that robot architectures need to plan on-line to be responsive to the current navigation scenario. Although computers are getting faster and faster, on-line planning is still slower than reactive navigation since it needs to repeatedly sense, update a map, and adapt its plans to changes in the map. Our robot architecture addresses this issue by determining the navigation mode in a principled way so that planning controls the robot no longer than necessary and by using fast replanning methods that do not plan from scratch but rather adapt the previous plan to the new situation.

II. Our Robot Architecture

Our robot architecture is a three-layered architecture with a reactive layer (that implements reactive navigation), sequencing layer (that determines the navigation mode), and deliberative layer (that implements the planner). The reactive and sequencing layers run continuously but the deliberative layer runs only in certain navigation modes.

A. Reactive Layer

The reactive layer uses motor schemata [Arkin89] to move the robot to given coordinates and implements a behavior that consists of two primitive behaviors, namely moving to the goal and avoiding obstacles. Each of the primitive behaviors generates a vector. The reactive layer calculates the weighted sum of the vectors for given weights that do not change during execution. It then moves the robot in the direction of the resulting vector with a speed that corresponds to its length.

B. Deliberative Layer

The deliberative layer obtains sensor readings from the on-board sensors, updates a short-term map (occupancy grid), and then uses D* Lite [Koenig02], a simplified and thus easy to understand version of D* [Stentz95a], to plan a path from the current location of the robot to the goal under the assumption that terrain is easily traversable unless the map says otherwise.

C. Sequencing Layer

The sequencing layer monitors the progress of the robot and determines the navigation mode. Our robot architecture uses reactive navigation as much as possible because of its speed. However, reactive navigation can get stuck in box canyons or in front of small openings. If the robot does not make progress toward the goal, the robot architecture activates the planner, which sets a way-point for reactive navigation to achieve, as has been done before [Wettergreen01][Urmson03]. Reactive navigation can still get stuck if it is unable to reach the way-point. For example, it can still get stuck in front of small openings. If the robot does not make progress toward the next way-point, our robot architecture bypasses reactive navigation completely and lets the planner control the robot directly, which is rather unusual in robotics. Our robot architecture thus operates in three different navigation modes. In mode 1, reactive navigation controls the robot and attempts to move it to the goal. In mode 2, reactive navigation controls the robot and attempts to move it to the way-point provided by the planner. In mode 3, the planner directly controls the robot and attempts to move it to the goal. Since planning is much slower than reactive navigation, our robot architecture always uses the smallest navigation mode that promises to allow the robot to make progress towards the goal.

We now describe how the sequencing layer determines the navigation mode with only two parameters, called PERSISTENCE and ANGLE DEVIATION.

- The mode switches from 1 to 2 when the robot travels less than a given distance during the time given by PERSISTENCE, and thus appears not to make progress. In mode 2, the planner plans a path and then returns as way-point the point on the path farthest away from the current location of the robot that is not occluded from it by known obstacles. This way, reactive navigation will likely be able to reach the way-point but still has control of the robot for a long time. The mode switches from 2 back to 1 when the difference in the movement direction recommended by mode 1 and the direction of the path generated by the planner is less than ANGLE DEVIATION for the amount of time given by PERSISTENCE. This condition guarantees that the robot continues to move in the same direction after the mode switch that it was moving in before the mode switch.
- The mode switches from 2 to 3 when the robot travels less than a given distance during the

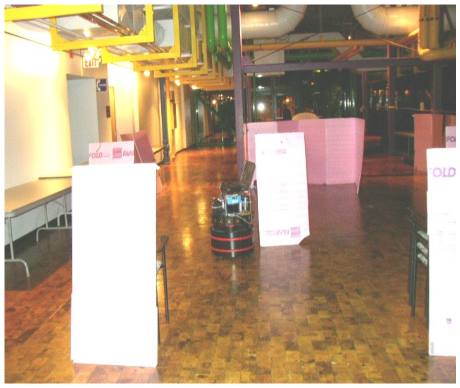


Fig. 1. Nomad Robot During an Experiment

time where the planner has returned the same way-point PERSISTENCE number of times or the difference in the movement direction recommended by mode 2 and the direction of the way-point set by the planner is greater than ANGLE DEVIATION for the amount of time given by PERSISTENCE. (A switch from mode 2 to 3 takes precedence over a switch from mode 2 to 1 in case both conditions are satisfied.) In mode 3, the planner controls the robot directly. It plans a path and then moves the robot along that path for a distance of two grid cells before it replans the path. This short distance ensures that the robot does not run into unknown obstacles. The mode switches from 3 back to 2 when the difference in the movement direction recommended by mode 2 (with a way-point set two grid cells away from the current cell of the robot on the planned path) and the direction of the path generated by the planner is less than ANGLE DEVIATION for the amount of time given by PERSISTENCE. This condition guarantees that the robot continues to move in the same direction after the mode switch that it was moving in before the mode switch.

III. Case Study: MissionLab

To demonstrate the advantages of our robot architecture, we integrated it into MissionLab [Mlab02], a robot programming environment that has a user-friendly graphical interface and implements the AuRA architecture [Arkin97]. All experiments were performed either in simulation or on a Nomad 150 with two SICK lasers that provide a 360 degree field of view, as shown in Figure 1. There was neither sensor nor dead-reckoning uncertainty in simulation but a large amount of both sensor and dead-reckoning uncertainty on the Nomad. The Nomad used no sensors other than the lasers and no localization technique

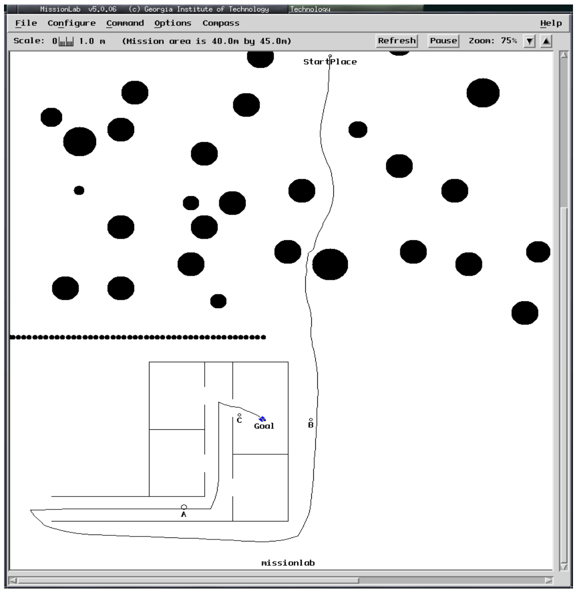

MissionLab v5.0.06 (c) Georgia Institute of Technology
File
Configure
Command
Options
Compass
Help
Scale: 0 1.0 m (Mission area is 40.0m by 45.0m)
Refresh
Pause
Zoom: 75%
StartPlace
C
Goal
B
A
missionlab

Fig. 2. Simulation Experiment 1 - MissionLab

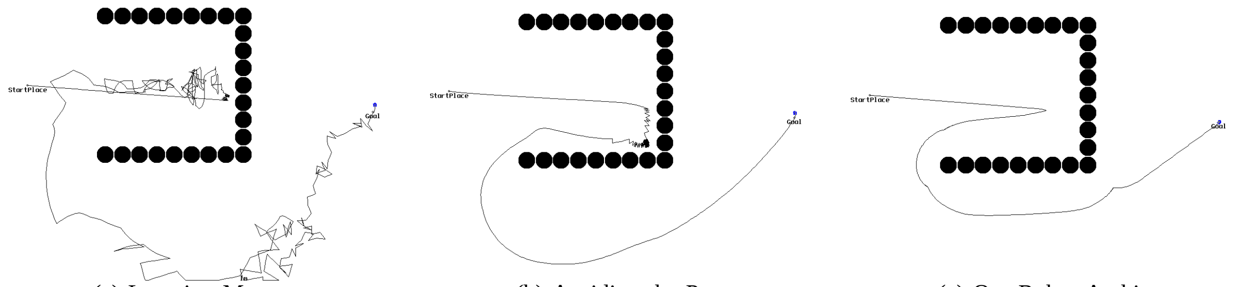
other than simple dead-reckoning (where walls were used to correct the orientation of the robot). We limited its speed to about 30 centimeters per second to reduce dead-reckoning errors due to slippage. MissionLab was run on a laptop that was mounted on top of the Nomad and connected to the lasers and the Nomad via serial ports.

A. Simulation Experiments

We first evaluated our robot architecture against the original version of MissionLab in simulation. MissionLab fits the first scheme mentioned in the introduction, namely where the decision when to activate which behavior is made before execution. Thus, we assume that a map of the terrain is available. The robot starts in a field sparsely populated with obstacles, has to traverse the field, enter a building through a door, travel down a corridor, enter a room and move to a location in the room, as shown in Figure 2.

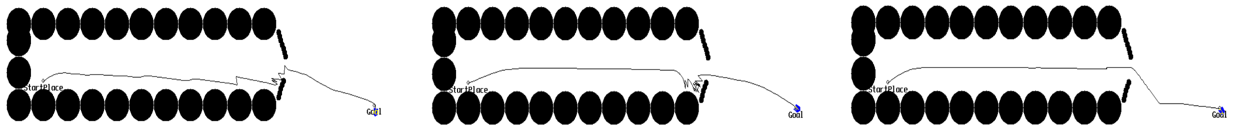
A programmer of MissionLab first creates several behaviors and then a finite state automaton that sequences them. Figure 3 shows a way of solving the navigation problem with MissionLab that needs eight different behaviors with a total of 32 parameters. For example, the behavior for moving in corridors uses a wall-following method with six parameters. We optimized the behaviors, their sequence, and the parameter values to yield a small travel time. Figure 2 shows the resulting trajectory of the robot. The travel time is 16.1 seconds. (All times include the start-up time for MissionLab.)

Our robot architecture uses only one behavior with



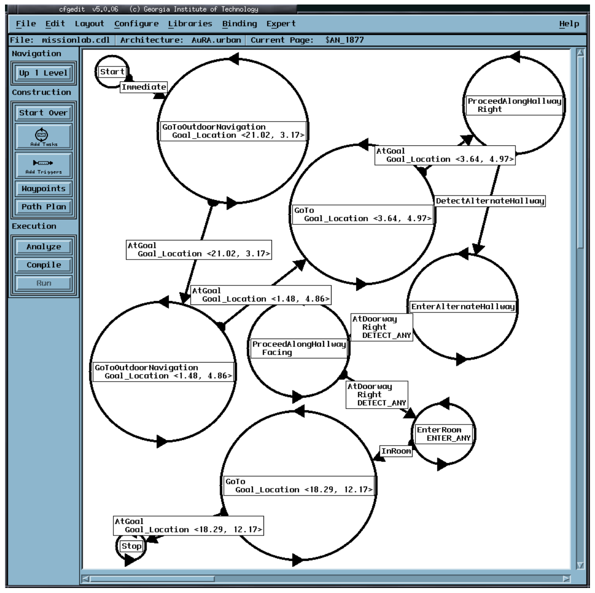
(a) Learning Momentum (b) Avoiding the Past (c) Our Robot Architecture

Fig. 6. Simulation Experiment 2



(a) Learning Momentum (b) Avoiding the Past (c) Our Robot Architecture

Fig. 7. Simulation Experiment 3


cfgedit v5.0.06 (c) Georgia Institute of Technology
File Edit Layout Configure Libraries Binding Expert Help
File: missionlab.cdl Architecture: AuRA.urban Current Page: $AN_1877
Navigation
Up 1 Level
Construction
Start Over
Add Tasks
Add Triggers
Waypoints
Path Plan
Execution
Analyze
Compile
Run
Start
Immediate
GoToOutdoorNavigation Goal_Location <21.02, 3.17>
ProceedAlongHallway Right
AtGoal Goal_Location <3.64, 4.97>
DetectAlternateHallway
GoTo Goal_Location <3.64, 4.97>
AtGoal Goal_Location <21.02, 3.17>
AtGoal Goal_Location <1.48, 4.86>
EnterAlternateHallway
AtDoorway Right DETECT_ANY
ProceedAlongHallway Facing
GoToOutdoorNavigation Goal_Location <1.48, 4.86>
AtDoorway Right DETECT_ANY
EnterRoom ENTER_ANY
InRoom
GoTo Goal_Location <18.29, 12.17>
AtGoal Goal_Location <18.29, 12.17>
Stop

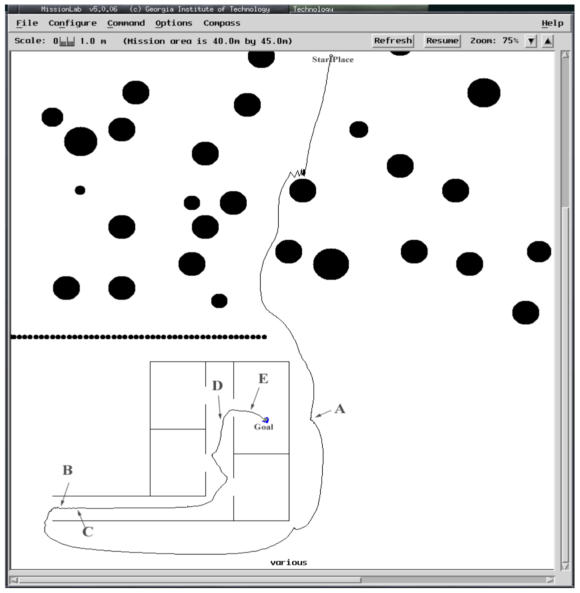

MissionLab v5.0.06 (c) Georgia Institute of Technology
File Configure Command Options Compass Help
Scale: 0 1.0 m (Mission area is 40.0m by 45.0m)
Refresh
Resume
Zoom: 75%
StartPlace
A
B
C
D
E
Goal
various

Fig. 3. Finite State Automaton for MissionLab

Fig. 4. Simulation Experiment 1 - Our Robot Architecture

four parameters plus two parameters to switch navigation modes. Consequently, it requires only a small amount of programming (and testing) since one only needs to set six parameter values but does not have to choose behaviors and sequence them. Figure 5 shows the time that the robot spent in mode 3 and the travel time of the robot for different values of PERSISTENCE and ANGLE DEVIATION. If ANGLE DEVIATION is too large, then the robot does not complete its mission and the times are infinity. Notice that the travel time first decreases and then increases again, as ANGLE DEVIATION increases for a given PERSISTENCE. This systematic variation can be exploited to find good values for the two parameters with a small number of experiments. The travel time is minimized if PERSISTENCE is 2 and ANGLE DEVIATION is 5. Figure 4 shows the trajectory of the robot for these parameter values. The robot started in mode 1, entered mode 2 at point A, mode 3 at point B, mode 2 at Point C, mode 3 at Point D, and mode 2 at point E. The travel time of the robot was 25.5 seconds, which is larger than the travel time of the robot under MissionLab (as expected, since we spent a long time tuning MissionLab) but still reasonable. Note that the

PERSISTENCE (cycles)	ANGLE DEVIATION (degrees)	Time in Mode 3 (seconds)	Travel Time (seconds)
1	5	13	35
	15	6	28
	25	5	31
	35	5	31
	45	10	71
	$\geq$ 55	∞	∞
2	5	3	17
	15	1	20
	25	1	20
	35	1	20
	75	1	32
	85	1	41
3	5	5	20
	15	2	19
	25	1	25
	35	1	25
	45	1	29
	75	1	51
	85	1	71
4	5	1	20
	15	1	19
	25	1	20
	35	1	20
	45	1	21
	75	1	44
	85	1	50
5	5	3	24
	15	1	21
	25	1	22
	35	1	23
	45	1	41
	$\geq$ 55	∞	∞
6	5	1	22
	15	3	28
	25	1	23
	35	1	27
	45	1	75
	$\geq$ 55	∞	∞

Fig. 5. Varying the Parameter Values

parameter values of the controller prevent it from entering the room that contains the goal. Our robot architecture therefore eventually switches to mode 3 and lets the planner control the robot, demonstrating that it is able to correct poor navigation performance caused by parameter values that are suboptimal for the current navigation situation.

We now evaluate our robot architecture in simulation against other techniques that can be used to overcome poor navigation performance but do not use on-line planning, namely biasing the robot away from recently visited locations (called "avoiding the past") [Balch93] and adjusting the parameter values of behaviors during execution (called "learning momentum") [Lee01]. These techniques fit the second scheme mentioned in the introduction, namely where the decision when to activate which behavior is made during execution. Thus, we assume that a map of the terrain is not available. Different from our robot architecture, these schemes are designed to be used only for simple navigation scenarios, such as box canyons and small openings. and not to relieve one from choosing behaviors, sequencing them, and determining their parameter values for complex navigation tasks such as the one discussed above. For each experiment, we used the same parameter values for the reactive controller (taken from the MissionLab demo files) and optimized the remaining parameter values of each technique to yield a small travel time. In fact, learning momentum was successful only after we tuned the parameter values very carefully.

- In the first experiment, the robot operated ten times in a terrain with a box canyon, as shown in Figure 6. Our robot architecture succeeded in all ten runs, invoked the planner only twice per run, and needed an average travel time of 13.9 seconds. Avoiding the past and the ballooning version of learning momentum also succeeded, with average travel times of 9.8 and 26.1 seconds, respectively.
- In the second experiment, the robot operated ten times in a terrain with a small opening, as shown in Figure 7. Our robot architecture succeeded in all ten runs and needed an average travel time of 4.7 seconds. Avoiding the past and the squeezing version of learning momentum also succeeded, with average travel times of 4.3 and 2.8 seconds, respectively.

Note that the trajectories of our robot architecture were very smooth in both experiments.

B. Robot Experiments

We now evaluate our robot architecture on the Nomad robot, using the same parameter values in both experiments.

- In the first experiment, the robot operated in a corridor environment, as shown in Figure 8 together with the resulting trajectory of the robot. (This map was not generated by the robot but was constructed from data obtained during the trial. Since the robot used only simple dead-reckoning, its short-term map deteriorated over time and was discarded whenever the goal became unreachable due to dead-reckoning errors.) The robot had to navigate about 20 meters from our lab via the corridor to the mail room. The robot started in mode 1, entered mode 2 at point A, mode 3 at point C, mode 2 at point D, mode 1 at point F, mode 2 at point G, and finally mode 1 at point H. The other points mark additional locations at which the planner was invoked in mode 2 to set way-points.

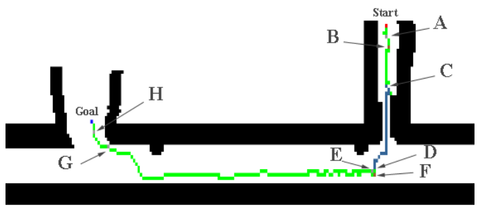


Fig. 8. Robot Experiment 1 (Grid Cell Size 10x10 cm)

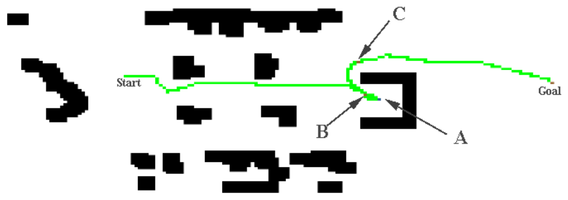


Fig. 9. Robot Experiment 2 (Grid Cell Size 15x15 cm)

- In the second experiment, the robot operated in an open space that was sparsely populated with obstacles, as shown in Figure 9 together with the trajectory of the robot. The robot had to navigate about 28 meters in the foyer of our building, through a sparse field of obstacles past a box canyon to the goal, as shown in Figure 1. The robot started in mode 1, and entered mode 2 at point A. Point B and C mark additional locations at which the planner was invoked in mode 2 to set a way-point.

These experiments demonstrate that the amount of planning performed by our robot architecture and how closely the planner controls the robot depend on the difficulty that reactive navigation has with the terrain. The planner is invoked only if necessary. For example, mode 1 is used throughout the easy-to-traverse corridor in the first experiment. Mode 3 is invoked only close to the narrow doorway but not the wider one in the first experiment and not at all in the second experiment.

IV. Related Work

Our robot architecture is a three-layered architecture with a powerful deliberative layer and a degenerated sequencing layer, whereas many three-tiered architectures fit the first scheme described in the introduction and have a degenerated deliberative layer but a powerful sequencing layer, for example, one based on RAPS [Firby87]. The planners of some of these robot architectures run asynchronously with the control loop [Gat91], whereas the planners of others run synchronously with the control loop [Bonasso97]. Similarly, the planners of some of these robot architectures run continuously [Stentz95] [Lyons95], whereas the planners of others run only from time to time [Bonasso97]. The planner of our robot architecture runs synchronously with the control loop and, depending on the navigation mode, either continuously (to control the robot in mode 3) or only from time to time (to plan the next way-point in mode 2). It differs from the planners of other robot architectures in that it can control the robot directly, when needed. This is a radical departure from the current thinking that this should be avoided [Gat98] and the suggestion to use plans only as advice but not commands [Agre90] which is based on experience with classical planning technology that was too slow for researchers to integrate it successfully into the control loop of robots [Fikes71]. Our robot architecture demonstrates that using plans sometimes as advice (mode 2) and sometimes as commands (mode 3), depending on the difficulty that reactive navigation has with the terrain, can result in robust navigation without the need for encoding world knowledge in the robot architecture.

V. Conclusions

We described a reactive robot architecture that uses fast replanning methods to avoid the shortcomings of reactive navigation, such as getting stuck in box canyons or in front of small openings. Our robot architecture differs from other robot architectures in that it gives planning progressively greater control of the robot if reactive navigation continues to fail to make progress toward the goal, until planning controls the robot directly. To the best of our knowledge, our robot architecture is the first one with this property. It also requires only a small amount of programming (and testing) since one does not have to choose behaviors and sequence them. One only needs to determine a small number of parameters. Our first experiments on a Nomad robot and in simulation demonstrated that it results in robust navigation, relatively smooth trajectories, and reasonably good navigation performance. It is therefore a first step towards integrating planning more tightly into the control-loop of mobile robots. In future work, we intend to increase the navigation performance of our robot architecture even further. We also intend to explore how to use on-line learning and, if available, an a-priori map to automatically determine the parameter values of our robot architecture to enable it to operate in any kind of terrain without a programmer having to modify them.

Acknowledgments

The authors would like to thank Ronald Arkin for valuable comments and suggestions during the course of this work. This research is supported under DARPA's Mobile Autonomous Robotic Software

Program under contract #DASG60-99-C-0081. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies, companies or the U.S. government.

VI. REFERENCES

[Agre90] P. Agre and D. Chapman, “What are plans for,” *Journal for Robotics and Autonomous Systems*, Vol. 6, 1990, pp. 17-34.

[Arkin89] R. Arkin, “Motor schema-based mobile robot navigation,” *International Journal of Robotic Research*, Vol. 8, No. 4, 1989, pp. 92-112.

[Arkin97] R. Arkin and T. Balch, “AuRA: Principles and practice in review,” *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 9, No. 2-3, 1997, pp. 175-188.

[Arkin98] R. Arkin, “Behavior-based robotics,” MIT Press, 1998.

[Balch93] T. Balch and R. Arkin, “Avoiding the past: A simple but effective strategy for reactive navigation,” *In: Proceedings of the IEEE International Conference on Robotics and Automation*, 1993, pp. 678-685.

[Bonasso97] R. Bonasso, R. Firby, E. Gat, D. Kortenkamp, D. Miller and M. Slack, “Experiences with an architecture for intelligent reactive agents,” *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 9, No. 2, 1997, pp. 237-256.

[Fikes71] R. Fikes and N. Nilsson, “Strips: A new approach to the application of theorem proving to problem solving,” *Artificial Intelligence*, Vol. 2, 1971, pp. 189-208.

[Firby87] R. Firby, “An investigation into reactive planning in complex domains,” *In: Proceedings of the National Conference on Artificial Intelligence*, 1987, pp. 809-815.

[Gat91] E. Gat, “Integrating planning and reacting in a heterogeneous asynchronous architecture for mobile robots,” *SIGART Bulletin*, Vol. 2, 1991, pp. 70-74.

[Gat98] E. Gat, “On three-layer architectures,” *In: Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, D.Kortenkamp, R. Bonasso and R. Murphy (Editors), MIT Press, 1998, pp. 195-210.

[Koenig02] S. Koenig and M. Likhachev, “Improved fast replanning for robot navigation in unknown terrain,” *In: Proceedings of the IEEE International Conference on Robotics and Automation*, 2002, pp. 968-975.

[Lee01] J. Lee, R. Arkin, “Learning momentum: integration and experimentation,” *In: Proceedings of the IEEE International Conference on Robotics and Automation*, 2001, pp. 1975-1980.

[Lyons95] D. Lyons and A. Hendriks, “Planning as incremental adaptation of a reactive system,” *Robotics and Autonomous Systems*, Vol. 14, No. 4, 1995, pp. 255-288.

[Mlab02] Georgia Tech Mobile Robot Laboratory, “MissionLab: User manual for MissionLab version 5.0,” Georgia Institute of Technology, 2002.

[Stentz95] A. Stentz and M. Hebert, “A complete navigation system for goal acquisition in unknown environments,” *Autonomous Robots*, Vol. 2, No. 2, 1995, pp. 127-145.

[Stentz95a] A. Stentz, “The focussed D* algorithm for realtime replanning,” *In: Proceedings of the International Joint Conference on Artificial Intelligence*, 1995, pp. 1652-1659.

[Urmson03] C. Urmson, R. Simmons and I. Nesnas, “A generic framework for robotic navigation,” *In: Proceedings of the IEEE Aerospace Conference*, 2003.

[Wettergreen01] D. Wettergreen, B. Shamah, P. Tompkins and W. Whittaker, “Robotic planetary exploration by sun-synchronous navigation,” *In: Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2001.