# Learning Robust Plans for Mobile Robots From A Single Trial

**Sean P. Engelson**
Dept. of Mathematics and Computer Science
Bar-Ilan University
52900 Ramat Gan
Israel
Email: engelson@bimacs.cs.biu.ac.il

## Abstract

*We address the problem of learning robust plans for robot navigation by observing particular robot behaviors. In this paper we present a method which can learn a robust reactive plan from a single example of a desired behavior. The system operates by translating a sequence of events arising from the effector system into a plan which represents the dependencies among such events. This method allows us to rely on the underlying stability properties of low-level behavior processes in order to produce robust plans. Since the resultant plan reproduces the original behavior of the robot at a high level, it generalizes over small environmental changes and is robust to sensor and effector noise.*

## Introduction

Recently, a number of sophisticated 'reactive' planning formalisms have been developed (Firby 1989; Gat 1991; McDermott 1991; Simmons 1994), which allow a great deal of flexibility in control flow and explicitly include a notion of an intelligent plan execution system. However, the complexity of these plan representations makes planning very difficult. Much of the effort in developing planners for these new planning languages, therefore, has focused on *case-based*, or *transformational*, planning approaches (Hammond 1986; McDermott 1992). In this paradigm, given a set of goals to achieve, the planner retrieves a set of plan fragments from a plan library which it believes will help achieve those goals. The planner combines the fragments to form a complete plan, and then adapts it to fit the particular task at hand. When the planner is satisfied, the plan gets executed. If execution is satisfactory, the plan may get stored back in the plan library for future reuse.

An initial case-base is usually constructed by hand, to contain plan fragments thought to be useful for a particular domain. As more planning problems are solved by the system, the plan library grows, containing plans that solved previous problems. However, under this learning strategy, the library will contain only complete cases derived for previously solved tasks, without the possibility of learning other types of cases. This method thus assumes implicitly that the problems that will arise in the future are similar to those arising now; it also precludes serendipitous learning, as the robot only learns plans relevant to its current task.

We propose here another method for augmenting the plan library, by storing plan fragments derived by breaking up the robot's behavior in ways different from those given by its controlling plan. Suppose, for example, that our robot is to go from room A to room B, via a hallway which contains a water cooler. In the usual case-based learning framework, a plan fragment that gets the robot from room A to the water cooler would not be learned; in fact, the original plan may not mention the water cooler at all. However, if we observe the robot's *behavior* between room A and the water cooler, we may be able to 'reverse engineer' a useful plan fragment to store in its case library. This would enable the system to learn from its incidental experience as well as its planned experience.

Several related problems are beyond our scope in this paper: (a) how to properly index a new plan in the case base (Hammond 1986; Kolodner 1993), (b) how to evaluate if a learned plan is actually useful (Minton 1988; Chaudhry & Holder 1996), and (c) how to recognize interesting world states (very much an open problem, cf. (Benson & Nilsson 1995)).

In this paper, we describe a method for automatically constructing usable plan fragments from records of executed robot behavior over a period of time. Specifically, given observations of the robot's behavior over a restricted period of time, our system constructs a reactive plan which reliably repeats the behavior when started in a similar situation. These plans are not sensitive to small changes in the environment, and are resistant to sensor and effector noise.

The main idea behind our system is to represent robot behavior as a sequence of *behavior events*, which represent qualitative changes in the state of an underlying behavior-based control system. This representation corresponds naturally to statements in a modern reactive plan language, such that each type of event may be translated into a plan fragment for creating or handling that type of event. These plan fragments are then linked together to form plan which reproduces the entire behavior sequence. Our algorithm also incorporates several techniques to ensure that the resulting plans are robust. We have applied the system in the domain of mobile robot navigation, where it produces plans which are quite robust.

## The Execution Architecture

Our system learns reactive plans within the context of the two-level control architecture depicted in Figure 1 (similar to those of (Gat 1991) and (Firby 1994)). This

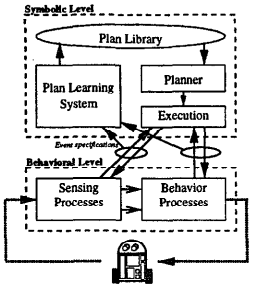


Figure 1: Our robot control architecture.

architecture divides robot control into two levels: 'symbolic' and 'behavioral'. The behavioral level consists of (a) a set of *behavior processes*, which implement continuous low-level controllers (eg, wall-following or grasping), and (b) a set of *sensing processes*, which implement object recognition and tracking. The symbolic level consists of a transformational (case-based) planner and reactive execution system (as in (McDermott 1992)), as well as the learning system described in this paper. In addition to control flow code, plans consist of activation and deactivation commands to behavior and sensing processes, as well as handlers for signals from those processes.

Sensing processes are connected to behavior processes via *designators* (McDermott 1991), which are data structures which refer to particular objects in the local environment. They thus form a sort of deictic representation (Agre 1988). When a behavioral process is activated, it may be provided with a designator upon which to act. It will then control the robot using the continuously updated state of the designator.

## Events and behavior traces

Both sensing and behavioral processes signal *effector events* to the symbolic level. One type of event is a termination event, signaling the termination of a process's execution, either as a success or a failure (with some error code). Another type of signal is used to return values from a sensing process, eg, a designator (which can then be given as a parameter to a behavior process). Many other kinds of events can also be accommodated; for example, a wall-following process may signal the presence of openings in the wall, which would enable a plan to count doorways. A sequence of effector events constitutes a *behavior trace*, which records the evolution of the behavioral level's state over some time interval. An event specification contains of the name of the process producing the event, the event type (activation, completion with error code, etc.), the robot resources required (such as a camera or an arm), and the values of any parameters or signaled values.

## The RPL Plan Language

Our goal is to translate a behavior trace into a plan which will reproduce the original behavior robustly, without being affected unduly by low-level failures or small changes in the environment. The plan notation we used for this work is a subset of McDermott's RPL, a reactive plan language developed for transformational planning (McDermott 1991). RPL is a full parallel programming language based on Lisp. The language contains facilities for sequential and parallel execution, including explicit process control. Most communication between processes is accomplished through fluents, including receiving signals from sensing and behavior processes. Plan failure is expressed explicitly, so that plans can fail *cognizantly* (Firby 1989).

## Behavior Traces to Reactive Plans

In this section, we describe our algorithm for translating a behavior trace into a RPL plan. Since behavior processes are nearly stateless, to a first approximation we can translate a behavior trace into a RPL plan by translating each event in sequence to a short sequence of RPL statements. Individual event specifications are translated by a set of translation rules, whose left-hand-sides are event specifications with variables to be instantiated. The right-hand-side of each rule is a plan fragment with variables to be substituted for.

The plans we construct are sequences consisting of three sorts of steps: process activation, receiving a return value, and testing required conditions. Activations provide parameters to sensor and behavior processes, while signals and completion events supply both values the plan can later use and conditions that must be tested in order for the plan to continue safely (such as error codes).

The way the translation rules is roughly as follows (more detail may be found in (Engelson 1996)):

- An activation event becomes a plan step which activates a sensor or behavior process and supplies it with its parameters. It also initializes any fluents needed for receiving signals from the new process.
- A condition handling plan step waits for a signal of the right type from a given process, tests any error code, and ensures that return values are as expected. If the value is needed later in the plan, it is bound to a local plan variable. The value-testing code ensures that if the values signaled at run-time do not match those in the original plan, the plan fails appropriately.

**Variabilization:** The use of plan variables allows plan steps to depend on one another by identifying values produced and used at different points in the trace. Value identification is impossible in general, without extensive knowledge, so our system simply assumes that equal values are meant to be logically identical. Quantities that are unequal are assumed to be unrelated; while this may be false, there is no way for us to resolve the ambiguity.

**Post facto parameterization:** In order to produce robust plans, we must also account for effector noise. For example, a command to turn the robot 90° may actually turn the robot 94.6°. If the command to turn the robot 90° is repeated some other time, it may end up turning the robot 85.2°, introducing a repetition error that is larger than strictly necessary. To avoid this problem, the original "turn 90°" should be translated as "turn 94.6°". This phenomenon also arises in perceptual routines, where such 'post facto parameterization' is needed. For example, when the robot looks for a doorway and two are in its field-of-view, the direction of the designator which is found and used should

be inserted into the plan step for finding the doorway so that the correct doorway is found. We address this problem by allowing translation rules to refer to other relevant events in the trace to instantiate needed parameters.

**Irrelevant events:** One further complication is that some trace steps reflect actions which do not affect the results of the original behavior in any way. This can cause problems if these actions are unreliable. In such a case, requiring the action to be performed (and its results to be consistent with the original trace) will cause the new plan to fail needlessly. The solution is to remove those steps from the behavior trace before translation, so that they can't affect the results. This is difficult in general, since nearly any action could be relevant. However, some perceptual events are clearly not needed and can be removed. For example, if the robot acquires a designator that is never used in the behavior trace, the acquisition event is removed.

**Dangling activation:** A more serious problem requiring trace preprocessing is *dangling activation*. Consider the case of an behavior trace meant to represent the robot's behavior between two corridor intersections. If the robot did not stop at the first intersection, but simply continued to follow the wall, activation of the wall-following behavior will not be reflected in the trace produced by observing behavior events only between the intersections. Simply translating the trace will result in a plan that does not move the robot at all. Dangling activation can be dealt with by adding a virtual activation event for the dangling process at the beginning of the trace. This works because behavior processes are essentially stateless, so appropriately activating the process at an intermediate point will produce similar behavior.

Prefixing requires taking a snapshot of the state of the behavior system when trace recording starts—which processes are active (and their parameters). Currently this is not implemented; a simpler form of prefixing is used which examines the trace for end events without matching activation events, and prefixes appropriate activation events to the trace. This works only if the dangling behavior process completes within the trace; in our tests, this was the case. A similar method is used by Torrance (1994) to deal with dangling activation, for a trajectory-based plan representation.

**The algorithm:** The full behavior trace translation algorithm for a trace $T$ is given as **TraceTranslate**. First, activation events are added for any dangling activations. Second, irrelevant trace steps are elided. Then, the steps in a behavior trace are translated sequentially by applicable translation rules. When a post facto parameterizable step is encountered, the rest of the trace is searched for the needed parameter, which is inserted in the translation. If an event signals a value, a new variable name is created, which is indexed under the value in a variable table. Then, when a value is used as a parameter, the value's variable name is inserted, if it exists, otherwise the value is assumed to be a hard-wired constant. After all the steps are translated, they are wrapped in a `let` form binding the local variables in the plan. This implicitly executes the plan steps in the same order that they appear in the trace.

**TraceTranslate**($T = (e_1, e_2, \ldots, e_n)$):

1. Let $P = ()$;
2. Let $H$ be a null hash-table, associating constants to variable names;
3. Find completion events in $T$ with no corresponding activation events, and prefix appropriate activations to $T$;
4. Elide irrelevant events from $T$;
5. For each event $e_i \in T$:
   (a) Find the translation rule $r$ matching $e_i$;
   (b) If $r$ requires values from another event in $T$, retrieve those values from $T$;
   (c) For each constant in $e_i$, substitute the associated variable name from $H$ (if one exists);
   (d) Let $S$ be the plan fragment resulting from translating $e_i$ according to $r$;
   (e) $P \leftarrow$ append($P$,$S$);
   (f) If $e_i$ is a signaling event, create a variable name for each new constant, and store them in $H$;
6. Bind all variable names in $H$ around $P$ and return the resulting plan.

## Results

We evaluated our plan-learning system using the ARS MAGNA mobile robot simulator. The simulator provides a robot moving in a 2-dimensional environment containing walls and objects of various types. The robot is equipped with a set of basic behaviors such as "follow wall" and "go to designator", as well as a full set of sensing processes such as "acquire designator on door". All of these processes incorporate noise—behavior processes may fail randomly and sensing processes may return noisy or erroneous data. Details of the simulator can be found in (Engelson & Bertani 1992).

In this section we will examine the performance of the trace translation technique described above on several navigational plans, using ARS MAGNA. Our tests demonstrate how our system produces robust repetition of robot behavior.

### Hand-generated behavior

We first test the stable repetition of a short behavior. The robot behavior was generated by hand; the experimenter manually activated sensor and effector processes in sequence. Figure 2(a) shows the original trajectory of the robot in a simple world with no obstacles. The behavior trace generated by this behavior contained 48 event specifications. The translated RPL plan was run ten times in each of two situations, where the initial location of the robot was the same as in the original run. These runs are summarized in Figure 2(b), which depicts the robot's trajectories on those ten runs. Note the variance in the precise trajectories the robot followed, due to noise. The plan failed in just one of these runs, when the robot lost track of the final corner on its approach due to perceptual noise. The plan was also tested with the addition of some obstacles (Figure 2(c)). As is clear from the fig-

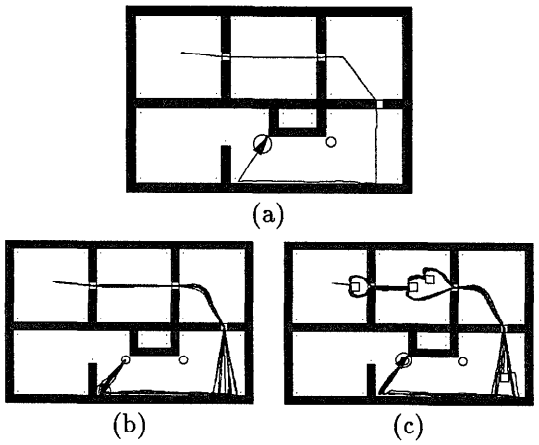


Figure 2: Robot trajectories and behavior repetition tests for a manually given behavior. (a) Original trajectory; the robot was started in the center of the first room. (b) Composite of trajectories of 10 executions of the translated plan. (c) Composite of trajectories of 10 tests with obstacles (squares) added.

ure, low-level obstacle avoidance enabled the robot to stably repeat its original behavior. Note that different trajectories were followed on different runs, depending on how the robot (locally) decided to go around the obstacles. One of these ten runs also failed, when the robot lost track of the third door in its trajectory due to perceptual noise. Note that both plan failures could not have been avoided without much more domain knowledge.

## Automatically generated behaviors

The next two experiments dealt with behaviors generated by a random wandering plan. Figure 3 shows the robot's trajectories during execution of the initial behavior (a) and the learned plan (b) for the first test case. The robot started in the rightmost doorway facing down, and ended up, as shown, in a corner of the rightmost upper room. The behavior trace contained 72 event specifications. Despite the variance in the robot's trajectory out of the door, 9 of the 10 tests succeeded. The one failure occurred when the robot failed to acquire a designator for the door on its return; it ended up in the upper-right corner of the lower room, as shown.

The second test case for automatically-generated behavior is depicted in Figure 4(a). The robot was started in the second doorway from the left, facing right. While describing a longer and more complex behavior than the last example, the behavior trace generated here was shorter, with 60 events in all. The behavior in this example was also less robust than those in the other tests, because it depends on the robot heading for the door it started from and then losing track of it due to occlusion. (This happens at the kink in the robot's trajectory, where it begins heading for the lower doorway.) Ten test runs were run on this example, with obstacles in different places than when the behavior was originally generated. Three of the test runs failed because the robot mistakenly headed

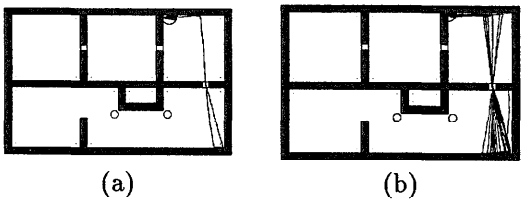


Figure 3: Robot trajectories for one test of plan generation from an automatically generated behavior trace. The robot was started in the rightmost doorway facing down. (a) Initial trajectory generated by random wandering. (b) Composite of trajectories of 10 test runs.

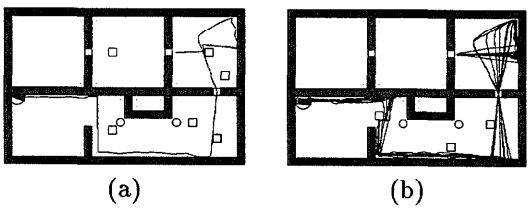


Figure 4: Robot trajectories for another example of plan generation from an automatically generated behavior trace. The robot was started in the center of the second room. (a) Initial trajectory generated by random wandering. (b) Composite of trajectories of 10 test runs.

for the lower doorway first. In all the other runs, the robot attained its goal, despite the different positions of the obstacles. Also note that the trajectories followed by the robot on different trials differed greatly, though the overall effect was the same.

# Limitations and Extensions

While the trace translation algorithm works well in many cases (as demonstrated above), it has some important limitations. Some can be remedied by simple extensions to the algorithm, while others are inherent in the current approach. In this section, we sketch some extensions to the approach which should correct for many of the limitations of the current approach.

## Basic extensions

**Timing:** One limitation of our technique is the fact that the plans it produces have no notion of time; they are purely reactive. If two events A and B are adjacent in a trace, the translated plan will execute B right after A, even though there may have been a period of time between A's completion and B's start in the original behavior. This can cause problems if, in the original behavior, the world state changed significantly after A completed, such as if the robot was moving. One solution would be to add a timing 'behavior process', so that when the execution system wishes to wait for a period of time, it activates the timing process with the desired wait time, which process then signals when the time is up and the plan should continue. These events go into the behavior trace like any other and become a part of the translated plan, causing the robot to wait appropriately upon repetition. If, however, the wait is

implicit, caused by waiting for a computation, another approach is required. In navigation, the main problem with getting the timing wrong comes about if the robot is moving between events A and B, so that it is at the wrong place to do B if it doesn't wait. This problem may be ameliorated by using odometry to measure the movement between adjacent trace events. If the distance moved is significant, the plan waits before performing B until the robot has moved the requisite amount.

**Effector failure:** When plans failed in our experiments, the main cause was effector failures, such as losing track of a designator or getting stuck following a wall. The effects of such runtime failures can be ameliorated heuristically. RPL contains a construct known as a *policy*, which serves to maintain a needed condition for a subplan. For example, if the robot needs to carry a box from A to B, after picking up the box, it will move from A to B with a policy to pick up the box if, for any reason, it finds itself not holding the box. This will (usually) ensure that the robot arrives at B with the box. A set of such heuristic policies could be designed for the different types of plan steps produced by the translation algorithm, to make these steps more robust. For example, if a trace contains a "go to designator" event, the translated plan may contain, in addition to an activation of "go to designator", a policy which attempts to recover the proper designator if it is lost before "go to designator" properly completes. Such policies can be developed for many common runtime failures, improving greatly the robustness of the resultant plans.

### Plans as a resource for learning

Fundamentally, the limitations of our approach are due to the fact that it assumes no knowledge aside from the translation rules which encode the relationships between different events. This means that the system does not understand the complete context of each event, and hence how the plan should be constructed in a *global* fashion. In this section, we sketch a proposed method for using the base plan, which originally generated the behavior to be modeled, as a resource for constructing more robust plans.

The idea is that a behavior trace that the learning system is given is a subsequence of that arising from the execution of some known plan. Let us conceive of that plan as a hierarchical decomposition of the robot's high-level goals into meaningful subplans, including partial ordering constraints among the subplans. In particular, execution of a RPL plan results in a *task network*, whose nodes are subplans and whose arcs are relationships between them. For example, the node for `(seq A B C)` will have three children, one for each subplan, and will contain the constraints that $A$ is before $B$ is before $C$.

Now, suppose first that we wanted to generalize a behavior trace generated by a complete run of a given plan. The best generalization (assuming our planner is reliable) would be the original plan itself. (This corresponds to case-based learning by storing newly-created plans in memory.) In our case, however, we are interested in only some part of a complete run which achieves some intermediate state of affairs. In this case, we wish, rather than translating behavior events individually, to abstract up the tree, to find the highest subplans whose execution was confined to the interval of interest. By doing so, and properly propagating parameter values, we can create a new plan which duplicates the intended behavior during the period of interest, inheriting the robustness properties of the original plan. What this means is that some of the effort used in making the original plan more robust can be saved when using the new plan as a building block in later case-based planning. At the same time, the flexibility of the system is enhanced by repackaging the behavior in new ways.

## Related Work

In the context of the development of a system for intelligent 'teleassistance', Pook (1995) describes a method for 'learning by watching' with similar goals to the current work. The system learns to perform an egg-flipping task from examples of master-slave teleoperation of a Utah/MIT hand. A coarse prior model of the task is given in the form of a Hidden Markov Model (HMM). For each example, the system segments signals from the hand's tendons into 'actions' by finding points of qualitative change. These sequences of actions are then matched to the HMM, and its parameters are estimated to form a complete model for the task. The primary difference between our work and Pook's is that her system relies on a prior model of the specific task, while ours makes use of predefined control primitives (sensing and behavior processes).

Our goal of learning plans to achieve particular goals from observing robot behavior is also related to recent work in learning action models for planning (Shen 1994; Wang 1994). Most of this work assumes discrete actions, unlike the present work. One significant exception is the TRAIL system described by Benson and Nilsson (1995). Their system learns models of 'teleo-operators', which are a kind of continuous version of STRIPS operator descriptions. These teleo-operators can then be used as the building blocks in constructing 'teleo-reactive' plans. TRAIL repeatedly attempts to achieve a given goal (possibly conjunctive), learning from both successful and failed attempts. The learning module uses inductive logic programming to induce from a set of execution records which achieve a given goal models of primitive operators that achieves that goal (Benson 1995).

Another related area of research is that applying case-based reasoning (CBR) to robotic control (Kopeikina, Brandau, & Lemmon 1988; Ram *et al.* 1992). In this work, the results of robot actions are observed, and formed into 'cases' which inform future behavior. In particular, Ram and Santamaria (1993), describe a system which learns how to adjust the parameters of a local behavior-based control system (using motor schemas (Arkin 1989)), based on sensor readings, in order to effectively navigate in crowded environments. By contrast, our work in this paper focuses on learning high-level reactive plans which combine and sequence multiple behaviors. The two approaches could probably be combined beneficially.

Finally, the problem of learning plans from observed behavior is particularly important in the context of topological mapping. Most previous work on topological mapping has assumed atomic action labels

(eg, (Dean *et al.* 1994; Kortenkamp, Baker, & Weymouth 1992; Kuipers & Byun 1988)). This approach, however, is not robust to even small environmental changes. Kuipers and Byun (1988) and Mataric (1990) both represent actions by activations of low-level behaviors, but only one behavior is represented per link, so they need not consider interactions between multiple behaviors.

## Conclusions

We have developed a system which learns reactive plans from traces of robot behavior. This problem arises both in the context of case-based planning systems (learning new cases) and in topological mapping (associating plans with links). In each case, we need to represent behavior over some period of time in a way that will enable it to be reliably repeated in the future. The idea is to store useful fragments of behavior in a way that will allow them to be reused in the future. For reliable repetition, the plans that are thus derived must be robust with respect to sensor and effector noise, as well as small changes in the environment.

Our system processes traces of the activity in a robot's behavioral control system, producing plans in a complex reactive plan language (RPL). Our results on learning navigation plans show the resulting plans to reliably repeat the original behavior, even in the presence of noise and non-trivial environmental modifications. The power of the approach comes from the choice of 'behavior events' as an action model. Rather than assume that a continuous action results from repetition of some discrete action, we take as given continuous control processes which signal the symbolic level regarding significant events. Our results show that this representation provides a natural level of abstraction for connecting symbolic (planning) and continuous (control) processes in intelligent robotic control.

**Acknowledgements** Thanks to Drew McDermott and Michael Beetz for many interesting and helpful discussions. The author is supported by a fellowship from the Fulbright Foundation. The bulk of this work was performed while the author was at Yale University, supported by a fellowship from the Fannie and John Hertz Foundation.

## References

Agre, P. E. 1988. *The Dynamic Structure of Everyday Life.* Ph.D. Dissertation, MIT Artificial Intelligence Laboratory.

Arkin, R. C. 1989. Motor schema-based mobile robot navigation. *International Journal of Robotics Research.*

Benson, S., and Nilsson, N. J. 1995. Reacting, planning, and learning in an autonomous agent. In Furukawa, K.; Michie, D.; and Muggleton, S., eds., *Machine Intelligence 14.* Oxford: Clarendon Press.

Benson, S. 1995. Inductive learning of reactive action models. In *Proc. Int'l Conf. on Machine Learning.*

Chaudhry, A., and Holder, L. B. 1996. An empirical approach to solving the general utility problem in speedup learning. In Anger, F. D., and Ali, M., eds., *Machine Reasoning.* Gordon and Breach Science Publishers.

Dean, T.; Angluin, D.; Basye, K.; Engelson, S.; Kaelbling, L.; Kokkevis, E.; and Maron, O. 1994. Inferring finite automata with stochastic output functions and an application to map learning. *Machine Learning.*

Engelson, S. P., and Bertani, N. 1992. ARS MAGNA: The abstract robot simulator manual. Technical Report YALEU/DCS/TR-928, Yale University Department of Computer Science.

Engelson, S. P. 1996. Single-shot learning of reactive navigation plans. Technical report, Department of Mathematics and Computer Science, Bar-Ilan University.

Firby, R. J. 1989. *Adaptive Execution in Complex Dynamic Worlds.* Ph.D. Dissertation, Yale University. Technical Report 672.

Firby, R. J. 1994. Architecture, representation and integration: An example from robot navigation. In *Proceedings of the 1994 AAAI Fall Symposium Series Workshop on the Control of the Physical World by Intelligent Agents.*

Gat, E. 1991. *Reliable Goal-Directed Reactive Control of Autonomous Mobile Robots.* Ph.D. Dissertation, Virginia Polytechnic Institute and State University.

Hammond, K. J. 1986. *Case-based Planning: An Integrated Theory of Planning, Learning and Memory.* Ph.D. Dissertation, Yale University Department of Computer Science.

Kolodner, J. 1993. *Case-Based Reasoning.* Morgan Kaufmann.

Kopeikina, L.; Brandau, R.; and Lemmon, A. 1988. Case-based reasoning for continuous control. In *Proc. Workshop on Case-Based Reasoning.*

Kortenkamp, D.; Baker, L. D.; and Weymouth, T. 1992. Using gateways to build a route map. In *Proc. IEEE/RSJ Int'l Workshop on Intelligent Robots and Systems.*

Kuipers, B., and Byun, Y.-T. 1988. A robust qualitative method for robot spatial reasoning. In *Proc. National Conference on Artificial Intelligence,* 774–779.

Mataric, M. J. 1990. A distributed model for mobile robot environment-learning and navigation. Technical Report 1228, MIT Artificial Intelligence Laboratory.

McDermott, D. 1991. A reactive plan language. Technical Report 864, Yale University Department of Computer Science.

McDermott, D. V. 1992. Transformational planning of reactive behavior. Technical Report YALEU/CSD/RR #941, Yale University Department of Computer Science.

Minton, S. 1988. Quantitative results concerning the utility of explanation-based learning. In *Proc. National Conference on Artificial Intelligence.*

Pook, P. 1995. *Teleassistance: Using Deictic Gestures to Control Robot Action.* Ph.D. Dissertation, University of Rochester.

Ram, A., and Santamaria, J. C. 1993. Multistrategy learning in reactive control systems for autonomous robotic navigation. *Informatica* 17(4).

Ram, A.; Arkin, R. C.; Moorman, K.; and Clark, R. J. 1992. Case-based reactive navigation. Technical Report GIT-CC-92/57, College of Computing, Georgia Institute of Technology.

Shen, W.-M. 1994. *Autonomous Learning from the Environment.* Computer Science Press, W. H. Freeman and Co.

Simmons, R. 1994. Structured control for autonomous robots. *Proc. Int'l Conf. on Robotics and Automation* 10(1).

Torrance, M. C. 1994. Natural communication with robots. Master's thesis, MIT Artificial Intelligence Laboratory.

Wang, X. 1994. Learning planning operators by observation and practice. In *Proc. 2nd Int'l Conf. on AI Planning Systems.*