Yale University
Department of Computer Science

Passive Robot Map Building with Exploration Scripts

Sean P. Engelson and Drew V. McDermott

YALEU/DCS/TR-898
March 1992

This work was partially supported by the Defense Advanced Research Projects Agency, contract number DAAA15-87-K-0001, administered by the Ballistic Research Laboratory. The first author is supported by a graduate fellowship from the Fannie and John Hertz Foundation.

Passive Robot Map Building with Exploration Scripts

Sean P. Engelson and Drew V. McDermott

Yale University Department of Computer Science
P.O. Box 2158 Yale Station
New Haven, CT 06520
`engelson@cs.yale.edu, mcdermott@cs.yale.edu`

Abstract

Consider a mobile robot building representations of its environment. Unless it lives exclusively in a research lab, it will have goals other than exploration. Hence, such a robot must be able to learn about its environment as it goes about its business, interleaving small bits of exploration with specifically goal-directed behavior. Most automatic map building systems, however, require that the mapper be able to take over whenever it wants, disregarding the robot's other goals. This problem is solved by using a passive mapper, which builds a map by observing the robot's actions and their effects. We can achieve this by explicitly considering and correcting mapping errors. We investigate here the use of *exploration scripts*, simple exploration strategies which are easily interleaved with other behavior, to improve passive robotic mapping. We have developed several such scripts, with purposes ranging from disambiguating robot position, to reducing positional uncertainty, to exploring the world more thoroughly. We have empirically evaluated the use of exploration scripts via simulation.

1 Introduction

Consider a honey bee. To efficiently accomplish its goal of finding the right kind of flowers to feed the hive, it builds a map of its environment [9]. At times, exploration, rather that direct goal-seeking behavior, would help to elucidate the structure of the environment—for example, discovering the extent of a particular field may provide for better navigational robustness. However, the bee must balance the utility of exploration against the need for satisfying its higher goals, and so explore only intermittently, if at all. Consider now an office courier, just assigned to a new building. She hasn't the leisure to walk about the building for a few days to learn its layout—she has deliveries to do. Thus, she must learn on the job. But if her delivery schedule allows a few minutes here and there, she can explore a little, possibly discovering new and efficient ways to perform her job. Finally, consider a mobile robot building representations of its environment. Unless it lives exclusively in a research lab, it will have goals other than exploration. Furthermore, even given an initial 'exploration phase', learning will be required continuously in a changing environment. Hence, such a robot must be able to learn about its environment as it goes, interleaving small bits of exploration with specifically goal-directed behavior. We investigate here the use of simple exploration strategies, easily interleavable with other behavior, to improve robotic mapping.

1.1 Passive map-learning

This approach, of intermittent, opportunistic, exploration requires the overall map-learning strategy be a *passive* one. That is, the map-learning system cannot always require the ability to direct the robot to do its job. In systems where the map system must control the robot's behavior (active mappers), it is difficult, if not impossible, to accomplish goal-oriented tasks and map at the same time. Most useful map-learners to date have been active. For example, the system of Kuipers and Byun [11] needs to be able to get the robot to a recognizable landmark place every so often, to confirm its location. More extreme are the methods described by Basye et al.[3] which set out exploration algorithms that must be followed precisely for accurate mapping. One main difficulty of map-learning is that decisions about the structure of the world must continually be made based on uncertain information. Using active control of the robot can often ameliorate, or even in some circumstances eliminate, this problem. However, as noted, purely active mapping does not coexist well with the pursuit of other robot goals. Passive mapping helps here, but the problem of uncertain information remains. We have developed a passive map-learning framework (described in [7, 8]) which deals with this problem by (a) having a map representation which is usable even with errors, and (b) incrementally discovering inconsistencies introduced by bad mapping decisions and correcting the map.

Despite the fact that passive mapping with error-correction can be done, it can also be quite inefficient. There are two reasons for this. First, since the robot's movements are not determined by considerations of mapping, important parts of the environment may remain unexamined for long periods of time. More fundamentally, since mapping errors may persist for some time before they are noticed and corrected, construction of a correct map can be delayed. Matters can be improved by allowing the mapper control over the robot's behavior. If this control can be exercised intermittently when not interfering with the robot's other goals, the problems of purely active mapping can be avoided. Thus, we propose here to use *exploration scripts*, simple high-level robot programs, to improve mapping. Such scripts are relatively context independent, and so can be applied whenever the robot decides that its high-level goals can be put off for a short time. Since the underlying passive mapping system makes no assumptions about the actions the robot takes, even 'bad' scripts cannot cause much trouble.

1.2 Diktiometric representation

A few words are now in order regarding types of environmental representation. Previous approaches to environmental representation can be roughly divided into two sorts—geometric and topological. The geometric approach (eg., [4, 5, 20, 19, 13]) attempts to build a more-or-less detailed geometric description of the environment from perceptual data. This has the intuitive advantage of having a reasonably well-defined relation to the real world. However, there is, as yet, no truly satisfactory representation of uncertain geometry; and it is unclear whether the volumes of information that one could potentially gather about the shape of the world are really useful. The topological approach, pioneered by Kuipers [10], has gained wide currency of late. The world is represented as a graph of places, with arcs representing actions which take the robot from one place to another. Diktiometric[1] representation is a generalization of this, which also explicitly represents geometric relations between places. By thus phenomenologically representing the robot's potential interactions with the world, the representation directly supports navigational planning. Furthermore, learning diktiometric representations is eased by the ability of the mapper to just record the robot's actions and their results [8, 14].

[1]The term 'diktiometric' is from the Greek *δίκτυον*, meaning 'network' and *μέτρον* meaning 'measurement'. Diktiometric representations represent the world as a network of places with positions. The term 'topological' as used by Kuipers, though nearly accurate, has too many other unrelated meanings.

2 Modelling the robot and its world

Following Kuipers and Byun [11], we assume that the robot has a repertoire of actions that take it to the nearest 'distinctive' place, often passing through fairly large swatches of territory. For example, there might be an action 'Go to door' that takes the robot to a doorway in its vicinity (eg., see [18]). Such actions use *fixations*, visual markers for places of particular types (a kind of effective designator, see [15]). The robot will generally attempt to get a fixation by looking for a place of a particular type in its vicinity; if a fixation is gotten, it will then be tracked as the robot approaches the destination (as in, eg., [17]). If no fixation can be found, the action reports that outcome, and does nothing. If there is more than one fixation, we assume the robot winds up at one of them arbitrarily. Actions may occasionally arrive at the wrong kind of place as well, due to errors in visual interpretation.

For a map representation, we adopt a diktiometric representation that represents both the connectivity of the path graph and its shape — the relative locations of the places. Thus, a map includes a graph with nodes representing places and arcs labelled with actions, as well as geometric reltions between places. Each node also has a record of what the place looks like.

Similar to McDermott and Davis [16], the shape of the path graph is given by places' relative positions. We assume odometry can provide, after each move, a set of points guaranteed to contain the robot's actual relative motion. Thus, position estimates are represented by sets of possible positions, relative to local reference frames. We approximate uncertainty sets as intervals in $\mathbb{R}^2$ and use interval arithmetic for matching and updating (see [1, 2]). The use of local reference frames ensures that relative uncertainty remains locally bounded.

All doors look like doors, but the view is not the same from all doors, and this fact enables the robot to tell one door from another. A view of the world may be thought of as a vector of measurements. These may denote physical properties of the environment such as its shape, or purely visual properties like segmentation. A set of views is stored for each place, representing the possible variations in what the world looks like from that place.

3 The Passive Mapper

The mapper, first of all, must have an idea of where the robot is. Due to uncertainty in odometry and perception, the robot's place in the map can be ambiguous. Hence, the system maintains a set of *tracks*, alternative estimates of the robot's current state with respect to the map. Mapping proceeds as follows. After an action is performed, bringing the robot to a new place, all tracks are updated to reflect odometric and sensory readings. Then, some of a number of different operations are performed on each tracks, each one corresponding to a decision about the state of the robot, the track, and the world. Some operations correspond to decisions about the correctness of the map and adjust it accordingly. Possible operations are proposed, then the system decides which operations are best to perform. The operations chosen then update the map by adjusting place descriptions and adding new place nodes and action links. Each track is then updated to reflect the robot's new state of knowledge. These operations are described briefly below; a more detailed discussion can be found in [7]. The way in which the operators serve to achieve robust, error-correcting mapping is important when designing exploration scripts to help with mapping.

First off, there are several basic operations always needed for mapping. A track matches a place when their position estimates match and the the track's view matches the place's view set. If the map is correct and complete, the results of all actions taken will be expected. Hence, the system should then confirm the expected transition, and put the track at the expected place. If a matching place exists, but there is no corresponding action link, the destination is confirmed and an action link is added. In both of these cases, the operation suggests a new position estimate for the track and matched place. Finally, if there is no matching place node, a place node (and action link) is created. Other operations relate to error detection and correction, and are discussed below.

3.1 Error correction

Since ours is a passive mapper, with the mapping process independent of the control of the robot, to detect and correct errors we must identify the causes and effects of different mapping errors. Then it becomes possible to detect, diagnose, and correct these errors after the fact. The approach we have taken is to store a small amount of extra information in the map such that errors can be detected and corrected for, *without knowing the precise cause of the error.* The types of errors possible and mechanisms to correct them are described below. These operations are only applied when the usual operations above cannot be.

One type of geometric error is when a place's position estimate (an interval) fails to contain its true position. This is caused by updating the place's position estimate by an incorrect match. This is detected and corrected by allowing a track to match a place's position estimate approximately, and expanding the estimated position interval to be more likely consistent (though uncertain). Another kind of geometric error occurs when a track is diverted by an incorrect match. This can occur even when the map is correct, if an incorrect match appears more plausible than a correct one. The robot's position estimate is then erroneous, and if left uncorrected can lead to erroneous map construction. This is dealt with by allowing a track to match with a place nearby, resetting the track's position to the place's position.

Another category of errors are those caused by transient errors during mapping. The robot may incorrectly decide it is at a place of some type when it is not, or that it can get from one place to another via a particular action. This may be due to perceptual errors or transient environmental conditions (such as a person walking by). This can be corrected by noticing when a place or action link is encountered far less often than it is expected. It is then assumed to be transient and is removed from the map. Such transient elision is also effective in adjusting the map to changing environments.

Since a place can appear different at different visits, the map may eventually contain two place nodes both representing a single place. However, if the system can decide that the two nodes represent places with the same position and links to the rest of the world, then the nodes should be merged. This is done by approximately matching the local graph structure by observing the robot behavior in the vicinity of the two places. The new place is a composite of the two merged.

The dual of representing one place by multiple nodes is representing multiple places by one node. Excessive odometric error may cause two nearby places to appear as one; two place nodes may also be merged erroneously. In any case, this situation will cause inconsistency to creep into the place node. This can engender corruption elsewhere in the map as well. Moreover, unnecessary ambiguity is introduced to the map, making navigation harder and less reliable. This situation is diagnosed when it appears that position estimates for a place come from multiple sources (details are given in [7]). When this occurs, the system splits the offending place node into two, copying all views and action links. Transient elision, as above, prunes the view and action link sets to reflect the real structure of the new places.

4 Exploration Scripts for Mapping

The essential idea of using exploration scripts to improve mapping is that they can be performed whenever a high-level decision process determines that other goals can be put off for a short while. This implies, first of all, that these scripts must apply in general circumstances, as the mapper has no control over when they will be called upon. Secondly, the scripts must be fairly limited in duration, so that they can do their business of improving mapping and then let the robot get back to its high-level goals. Scripts have two components, an application test which determines if the script is relevant, and a (loop-less) procedure which is executed if the script is applied. Scripts use the mapper's data structures, in particular the set of current tracks and the map itself.

Recall the motivations for using scripts to improve passive mapping. One is that a robot will not naturally explore the world efficiently, since its actions are determined by other considerations. The other is that mapper-directed activity may improve the reliability of mapping decisions and reduce the introduction of errors into the map. We have developed and tested some heuristic exploration scripts to thus aid mapping; they are described below.

4.1 Exploration scripts

We have investigated a number of exploration scripts. Some seek to reduce positional uncertainty. Others attempt to find new places and action links. Scripts are also used to reduce ambiguity in the map or in the robot's position estimate. Some probe map places which seem likely to not really exist. Keep in mind that all these scripts do is direct the behavior of the robot; they direct the attention of mapping system, improving the robot's map.

RETRACE STEP:

One important source of error and ambiguity in a map is positional uncertainty. When the robot performs an action with a very uncertain estimate of relative position, and the robot's *a posteriori* position estimate (after matching to its map) is also particularly uncertain, a simple and useful heuristic for reducing uncertainty both in the map and in the current position estimate is to retrace the last step taken. That is, the robot tries to return to the place it just came from; if it manages that, it tries to get back to where it started.

If:

- Uncertainty of the last Δposition high
- Uncertainty of all track positions high
- Didn't just retrace step

Do:

- Try to get a fixation on the last place visited;
- If found, go there, otherwise exit;
- Try to return, if possible.

RETRACE STEP

HEAD FOR UNCERTAINTY:

A more generally applicable heuristic for reducing map uncertainty is to simply head for nearby places with large positional uncertainty, under the assumption that if they are reached, new constraints will improve the positional estimate. This can only be reasonably tried, of course, if the robot's current place is unambiguous.

If:	• There is a nearby place whose uncertainty is high • Only one track is current
Do:	• Try to get a fixation on the uncertain place based on its relative position; • If fixation found, go there, otherwise exit.
	HEAD FOR UNCERTAINTY

HEAD FOR CERTAINTY:

The converse of the last script is useful when the robot's positional uncertainty gets too high, and that is to head for a nearby place whose position is known very precisely. If the robot reaches and recognizes the place, the robot's position will then also be known more precisely, improving further mapping.

If:	• There is only one track, with high uncertainty • There is nearby place with low uncertainty
Do:	• Try to go to that place.
	HEAD FOR CERTAINTY

DISAMBIGUATE TRACKS:

It will often occur that the robot's estimate of its current position will be ambiguous. If there are thus multiple current tracks, a good way to distinguish between them is to try to perform an action with different results for the different possible places the robot is at. This will usually result in the incorrect track becoming inconsistent and thus dropped.

PROBE AMBIGUOUS ACTION:

One kind of map ambiguity is when a place has multiple action links coming from it labelled with the same action. While this ambiguity may be inherent, it may also be that one of the links is due to a transient; hence, further examination is warranted. This is achieved by attempting to perform such an ambiguous action—this will tend to speed up elision of any transients, and just maintain the real action links.

If:	• There are multiple current tracks.
Do:	• Choose an action link whose destination is known reachable from some, but not all, of the tracks' places; • Try to go to that link's destination place.
	DISAMBIGUATE TRACKS

If:	• There is a single current track, • There are multiple links from the current place labelled with the same action.
Do:	• Perform that action.
	PROBE AMBIGUOUS ACTION

<u>PROBE SPLITTAGE</u>:

The kind of map ambiguity we consider for now is where two identical links from different places end at the same place and there is reason to believe that only one is real. This happens when a place is split (see above). Since both places resulting from a split have copies of the same action links, many of those links will be invalid. Hence, when the robot is at a place which resulted from a recent split, the PROBE SPLITTAGE script tries to perform an action that has (in the map) identical consequences in the two split-off places. This will accelerate elision of the invalid action links.

If:	• There is a single current track, • The current place was recently split off from another place.
Do:	• Choose an action link which both places have in common, • Try to traverse it.
	PROBE SPLITTAGE

HEAD FOR UNEXPLORED AREA:

Most of the previous scripts have dealt with improving the system's knowledge of places already in its map. Finding new, unknown places in an efficient manner, however, would also be useful, so that the world may be more quickly explored. We thus try to head for an area of the world which has not likely been visited by the robot before. To decide that this holds of some area, each local reference frame has associated with it a *coverage grid*, which tesselates the area about the frame into a coarse grid, and keeps track of an estimated certainty that the robot has visited each grid cell. Then, an area is deemed to be unexplored if the likelihood of part of it having been visited in the past is sufficiently low. Thus, HEAD FOR UNEXPLORED AREA looks in the vicinity of the current place for a nearby area which looks unexplored, and if one is found, attempts to head in its direction.

HEAD FOR RARE PLACES:

Recall that transient environmental features are eventually elided by the mapping system by noting their frequency of apprehension. This process can be speeded up if the robot tries to reach places which look likely to be transients. Since transients, by their very nature, are not encountered often, it is likely that a place that has not been visited often is transient. If so, then repeatedly trying to reach the place will cause the system to notice that it is not encountered as expected, and so it will eventually be elided. If it is not a transient, then the mapper will just gain a bit more information about the place.

If:	• There is only one current track. • The current place has a neighbor which has been visited less than a threshhold number of times.
Do:	• Choose such a rarely visited neighbor, • Try to go to there.
	HEAD FOR RARE PLACES

5 Results

To test a robotic mapping system, a large number of experiments must be run in a variety of different environments, so that the generality and stability of the method can be properly evaluated. However, this is difficult to do with a real robot, as it is often hard to work in multiple controlled environments [12], and

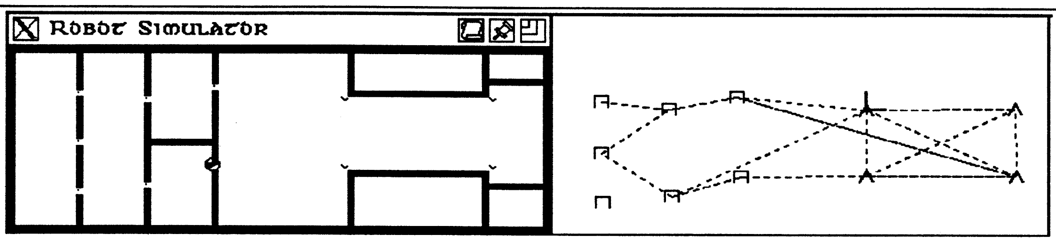

Figure 1: The HALL world and a learned map. The map picture shows the topological structure with symbols denoting place types. Note that links only mean that a sequence of actions is known to get directly from one place to another—no geometric interpretation is implied.

running experiments is very time-consuming. Therefore experiments were run on a simulated robot, designed with a realistic, though abstract, approach to sensing and control error; worst-case assumptions were made where necessary. There is, of course, no complete substitute for trials on a real robot, so we are currently developing a robot control system to support the mapping module. The simulator is described in detail in [6]; a brief overview is given below.

5.1 The simulator

Our simulator provides a point robot moving in $\mathbb{R}^2$. The structure of the environment is given by a 2D occupancy grid—individual cells are either full (walls) or empty (space). Filled cells have a single numerical 'color', representing some intrinsic property of the material, eg. texturedness. Empty cells have an optional place type; the two currently used are `door` and `corner`.

The perceptual primitive is the view, which is simply a list of numbers, each representing a visual measurement of the color of some object. The robot's field of view is evenly sampled, and the nearest filled cell in each direction is seen. Measurements have uniform bounded error added to them; there is also a chance that the measurement will be contaminated and chosen uniformly from the universe of possible measurement values.

The robot can obtain a fixation on a place of a particular type and then go to the place. Fixations are only valid until the robot moves. There is also error inherent in getting a fixation—there is a chance that a bogus fixation will be found, pointing in a random direction. When a fixation is approached, the robot moves to the position indicated—if the place type is recognized, the approacher succeeds, otherwise it fails. There is also a chance that the robot will go elsewhere than the original fixation. The odometric estimate of the robot's relative motion is given as a random interval containing the true motion, with size proportional to the distance travelled.

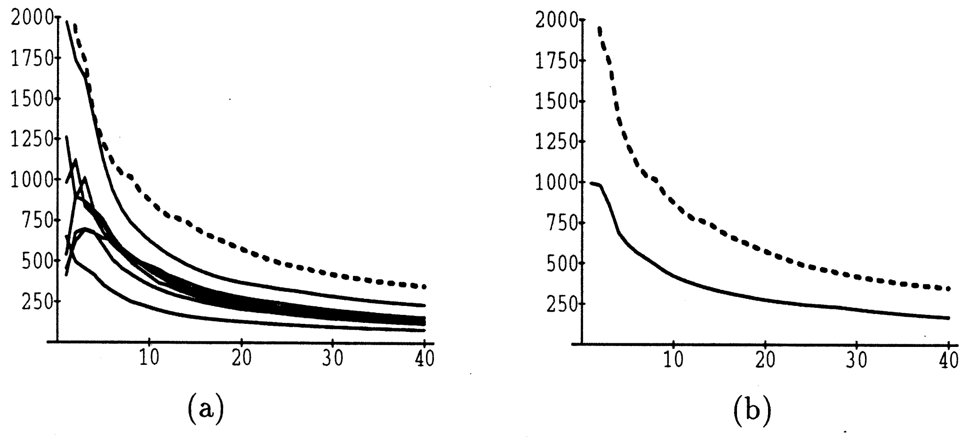

Figure 2: SSD error v. number of moves. (a) With no exploration scripts (dashed) and single exploration scripts (solid). Note that all scripts provide significantly reduced prediction error. (b) No scripts (dashed) v. all scripts (solid).

5.2 Evaluation

The question of evaluating mapper results to determine the utility of different scripts is not simple. The mere fact that a learned map 'looks right' is no guarantee that it is a good one, since there is quite a bit of invisible information. A map that 'looks' good may easily become garbage in the near future. There are, however, more objective methods for measuring mapper success. The method we use here is based on prediction error—the error inherent in allowing the robot to rely on the map to predict its expected position after each move. We measure this by calculating the sum-of-squared-distance (SSD) between the robots actual relative motion and predicted relative motion for each track after a move. If the system is effective at mapping, we expect the average SSD per move to asymptotically converge to a small constant. There are other objective ways of evaluating performance, as discussed in [8], but as seen there, the different methods give qualitatively similar results—space does not permit inclusion here.

5.3 Experimental results

To determine the advantage of using exploration scripts, we ran experiments in the simulated world shown in Figure 1. The robot moved on a random walk, when not controlled by exploration scripts. Experiments were run for different sets of exploration scripts active. Each experiment consisted of 10 mapping runs, with the robot making 400 moves (from place to place) in each run. Cumulative SSD was sampled each 10 moves. Active exploration scripts were applied occasionally (typically about half the time) when applicable. The results are summarized in

Figures 2. As can be seen from Figure 2(a), application of any one exploration script provides a noticable advantage over a random walk. The mapping improvement provided by any particular script depends on the structure of the world. Figure 2(b) shows the effect of using exploration scripts together and shows an averaging effect; this is due to the fact that when multiple scripts are applicable, one is chosen at random. We would expect scripts to act more synergistically given good preference criteria.

6 Conclusions

For autonomous mapping systems to be useful, mapping must be able to coexist with other goals. However, this implies that mapping may only control the robot's behavior when such control does not conflict with other goals. Hence, the paradigm of active mapping, where the mapper requires that the robot behave in a certain narrowly-defined manner, is inappropriate. The answer is passive mapping, which operates by simply observing the behavior of the robot. Semi-active methods can then be introduced, in the form of exploration scripts, which can be spliced into the robot's behavior when other goals permit. We have developed several such scripts for use with our mapping systems, and have tested them in simulation. Our results show that use of even very simple scripts noticably improves mapping effectiveness. Our future work includes studying more closely interactions between different mapping scripts, as well as the porting of the mapping system to a real robot.

Acknowledgements

This work was partially supported by the Defense Advanced Research Projects Agency, contract number DAAA15-87-K-0001, administered by the Ballistic Research Laboratory. The first author gratefully acknowledges the support of the Fannie and John Hertz Foundation. Greg Hager's thoughtful comments improved this paper greatly. Thanks also to Kostas Vassilakis for his help with the $o\varrho o\lambda o\gamma\iota\alpha$.

References

[1] G. Alefeld and J. Hertzberger. *Introduction to Interval Computation.* Academic Press, New York, NY, 1983.

[2] Sami Atiya and Greg Hager. Real-time vision-based robot localization. In *IEEE Robotics and Automation*, 1991.

[3] Kenneth Basye, Tom Dean, and Jeffrey S. Vitter. Coping with uncertainty in map learning. Technical Report CS-89-27, Brown University Department of Computer Science, June 1989.

[4] Raja Chatila and J. Laumond. Position referencing and consistent world modeling for mobile robots. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 138–170, Washington, D.C., 1985.

[5] Alberto Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, RA-3(3):249–265, 1987.

[6] Sean P. Engelson. The abstract robot simulator manual. Technical Report (forthcoming), Yale University Department of Computer Science, 1992.

[7] Sean P. Engelson and Drew McDermott. Map learning with explicit error correction for mobile robots. Technical Report YALEU/DCS/TR-874, Yale University Department of Computer Science, September 1991.

[8] Sean P. Engelson and Drew McDermott. Error correction in mobile robot map learning. In *Proc. Int'l Conf. on Robotics and Automation*, Nice, France, May 1992. (to appear).

[9] Charles R. Gallistel. *The Organization of Learning.* The MIT Press, Cambridge, MA, 1990.

[10] Benjamin Kuipers. Modeling spatial knowledge. *Cognitive Science*, 2:129–153, 1978.

[11] Benjamin Kuipers and Yung Tai Byun. A robust qualitative method for robot spatial reasoning. In *Proceedings of AAAI-88*, pages 774–779, 1988.

[12] Philippe Lemoine and Claude Le Pape. Simulating actions and perception of autonomous mobile robots in a multi-agent indoor environment. In *Proc. IEEE/RSJ Int'l Workshop on Intelligent Robots and Systems*, Osaka, Japan, 1991.

[13] John J. Leonard and Hugh F. Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Proc. IEEE/RSJ Int'l Workshop on Intelligent Robots and Systems*, Osaka, Japan, 1991.

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

[15] Drew McDermott. A reactive plan language. Technical Report 864, Yale University Department of Computer Science, 1991.

[16] Drew V. McDermott and Ernest Davis. Planning routes through uncertain territory. *Artificial Intelligence*, 22:107–156, 1984.

[17] Robin R. Murphy and Ronald C. Arkin. Adaptive tracking for a mobile robot. In *Proc. 5th IEEE Int'l Symp. on Intelligent Control*, Philadelphia, PA, 1990.

[18] Karen B. Sarachik. Visual navigation: Constructing and utilizing simple maps of an indoor environment. Technical Report 1113, MIT Artificial Intelligence Laboratory, 1989.

[19] Charles Thorpe, Martial H. Hebert, Takeo Kanade, and Steven Shafer. Vision and navigation for the Carnegie-Mellon Navlab. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 10(3):362–373, 1988.

[20] Wai K. Yeap. Towards a computational theory of cognitive maps. *Artificial Intelligence*, 34(3), April 1988.