Yale University
Department of Computer Science

Map Learning with Error Correction for Mobile Robots

Sean P. Engelson and Drew V. McDermott

YALEU/DCS/TR-874
September 1991

Submitted to the *IEEE Conference on Robotics and Automation 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.

Map Learning with Error Correction for Mobile Robots

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

For truly autonomous behavior, a robot must build representations of the places in the world by exploring the world and finding and remembering places of different sorts. The problem we address is how a robot can learn such maps automatically. Several methods for map-learning have been reported, but they all suffer from the problem of *error accumulation.* Since all sensors have noise, and sensor interpretation often depends on violable assumptions about the real world, any system which attempts to build a consistent representation of its environment will make errors. The primary emphasis on this problem in the literature has been on reducing errors entering the map to begin with. We suggest that this methodology must reach a point of diminishing returns, and explicit attention must be given to the problem of robustness in the face of learning errors. We focus on explicit error detection and correction. By identifying the possible types of mapping errors, we can exploit structural constraints to detect and diagnose mapping errors. Such robust mapping requires little overhead beyond that needed for non-robust mapping. It can also be integrated into existing mapping systems to make them more robust. We have implemented a mapping system based on our ideas. Extensive testing in simulation demonstrates the effectiveness of the proposed error-correction strategies.

1 Introduction

The problem of mobile robot navigation is getting the robot to a place it's been before. Some robots never need to revisit a place, but we will focus on those that do. Furthermore, we will assume that it is necessary to revisit particular, individual places, not just places with some interesting property. If a robot needs to go to a place that is likely to have bananas, it does so by going to the nearest place it knows about where bananas have been spotted, as opposed to being guided by some kind of 'banana tropism'. While tropisms based on local information are useful, they are inefficient at best for large-scale navigation. We will also assume that the robot builds up such lists by exploring the world and finding and remembering places of different sorts. It might be given such lists by a person, but it's more interesting to study the problem of building the lists up automatically.

However, there are several issues that must be resolved in systems which build such representations. We must pin down what we mean by 'place,' and do so in a way that supports efficient recording and recognition of places. Several methods for automated map construction have been reported (see below), but they all suffer from the problem of *error accumulation*. Since all sensors have noise, and sensor interpretation often depends on violable assumptions about the real world, any system which attempts to build a consistent representation of its environment will make errors. In particular, the robot's decision that it has been somewhere before (more generally, that two places are the same) can never be perfectly justified and always involves some margin of error. If a mistaken identification is allowed to persist, then attempts to make the rest of the map consistent with it will eventually turn the whole map into garbage. Hence, some mechanism must be provided by which these errors can be detected and corrected. Interestingly, this issue has been largely ignored in the literature, with the primary emphasis being on reducing errors entering the map to begin with. We suggest that this methodology must reach a point of diminishing returns, and explicit attention must be given to the problem of robustness in the face of learning errors.

As we've said, the robot will always make mistakes when mapping. So we bite the bullet and focus on correcting errors when they occur. If the robot can diagnose mapping errors when it discovers problems with its map, then it should be able to fix them as well. We have developed heuristic methods for doing just this. We identify certain classes of mapping errors, and exploit topological and geometric constraints to discover, diagnose, and correct mapping errors during robot execution. Every time a mapping decision is made, the system stores information about the decision in the map. Then, when an inconsistency is discovered in the map, the type of error responsible is diagnosed and the map adjusted appropriately. The methods we use are heuristic in nature, since the problem is not yet well-understood. Our architecture for robust mapping requires little overhead beyond that needed for non-robust mapping. We have implemented a system based on these ideas, and have extensively tested it in simulation. The remainder

of this paper discusses our system in more detail; it is organized as follows. After we review the related work in robot mapping, we describe, in Section 3, our robot model and the topological map structure it implies. Section 4 describes the mapping system architecture and the details of our error correction methods. We then present and analyze the results of extensive testing of an implementation of our system in simulation. Finally, we discuss future directions and the conclusions we draw from this work.

2 Related Work

Much research on navigational mapping deals with problems of local metric representation (eg., [3, 22, 2]), which is generally unsuitable in large-scale spaces. Kuipers and Byun first develop the notion of a topological place graph based on 'distinctive' locations [15]. However, while they go to some length to avoid error creeping into the map by using active experimentation, there is no provision for error correction. Thus, they must rely upon unambiguous, error-free actions for correctness; if this assumption is violated, errors accumulate. Levitt et al. also utilize a topological map which avoids accumulation of navigation error, by using local reference frames based on landmarks [16]. However, their system depends heavily on reliable landmark acquisition; it is unclear how their system would deal with mapping errors. Miller and Slack use information generated for reactive local navigation to build rough geometrical maps of rocky terrain [21]. Their maps are notable in that they can directly be used for reactive navigation.

Basye et al. develop a probabilistic theoretical framework for map learning [4] using Rivest's *reliable and probably almost always useful* learning criterion [23]. They probabilistically eliminate errors in the learned map by using active exploration, assuming limited directional certainty and globally recognizable places. However, their methods use very simple models of perception and action and do not use the rich geometrical and perceptual structure available. This structure can help provide a way to deal with errors explicitly.

The method for dealing with mapping errors we develop can also be incorporated into existing mapping systems with minimal modification. Virtually any system that uses a place graph representation can be reformulated in our terms. Specifically, Sarachik's system for visual navigation [24] is particularly apposite. Her system visually recognizes room shapes and finds doors, linking them together in a place graph. A room's perceived shape can be used as its perceptual description; its position can be described in a locally determined reference frame. Our error correction machinery could then be applied virtually as is.

Mataric [17] uses constraints derived from knowledge of the robot's underlying behavior to derive a topological map based on linear graph segments. Yeap [27] describes a hierarchical topological map, with place nodes described by 2D geometric models. Braunegg [6] develops a similar style of map, where rooms are

characterised by the geometric arrangement of vertical edges, measured by stereo vision. Kriegman [13] describes a method for visually instantiating generic models of the robot's surroundings, such as hallways, bringing top-down constraints to bear on geometric interpretation. Atiya and Hager [2] use an interval-based method to perform accurate model-based navigation using stereo vision.

3 Robot Modelling and Representation

The map-learning problem has been studied for some time, from a number of viewpoints. There are two basic types of approach—metric and topological (see, eg., [7, 9, 27, 26, 15]). The metric approach attempts to build up a detailed geometric description of the environment from perceptual data. This has the intuitive advantage of having a more-or-less well-defined relation to the real world. However, no one has found a satisfactory representation of uncertain geometry; and it is not clear that the volumes of information that one could potentially gather about the shape of the world are really useful.

The topological approach, pioneered by Kuipers [14] and gaining wide currency of late, represents the world as a graph with nodes representing places and arcs encoding robot actions that take the robot from one place to another. Such representations have two primary technical advantages. First, the action labels on the arcs are just what the robot needs for navigational planning (which becomes a kind of graph search). Second, because topological representations focus on the structure of the paths rather than the structure of the surroundings, they appear to be built out of a smaller volume of information.

3.1 Places and actions

We adopt the basic idea of Kuipers and Byun (described in [15]), with one twist: instead of focusing solely on the connectivity of the path graph, we have the robot attempt to learn its shape — the relative locations of the places — as it maneuvers through the world. There are two fairly obvious reasons for this move: the metric information can help in distingishing between perceptually similar places; and the metric information is useful in deciding where to go and how to get there when the map is used.

Following Kuipers and Byun, 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, using a local tropism that heads for shapes that look like doors. If there is no such entity visible, the action reports that outcome, and does nothing. If there is more than one door, the robot winds up at one of them (*without* noticing the choice). This action is called the 'door approacher.' We will also assume that there is a 'door recognizer' that can tell

if the robot is already located in a doorway. We will use the term *place type* to refer to a class of place for which the robot has an approacher and a recognizer. The approacher and recognizer are allowed to make occasional mistakes. Please note that not all actions are place-type approachers.

Thus, a map includes a graph with nodes representing 'places', i.e., connected regions of a particular type, and arcs labelled with sequences of actions, generally concluding with an approacher. (Presently, we deal only with 'point-like' places, small regions which can be treated as single points; the complexities of shape representation will be investigated in future work.) However, there is more to the graph. Each node has a record of what the place looks like, and what its position is with respect to other nodes. These are our topics in the next two sections.

3.2 Perception

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. In principle, it could store the view from every place, and identify two places only if their views are the same; but of course the views will never be exactly the same. There are many reasons for this variability, but the main one is that slight changes in the robot's orientation can make big differences in what it sees. Hence we must store a set of views for each place, and allow a new view to match an old view only approximately. We must now pin down what we mean by a view.

We consider two basic types of perceptual processes: measurers and classifiers. The distinction is primarily in the type of results they produce; measurements are continuous while classifications are discrete. A simple example of a measurer is a stereo-based depth estimator. Place type recognizers are boolean-valued classifiers. Measurements have the advantage that noise can be quantified; indeed, most reliable measurers can usefully be thought of as having bounded error (allowing a small chance of outliers). Classifiers, on the other hand, are either right or wrong, with no in-between. Thus, with the exception of place type recognition, we focus on measurements when dealing with perceptual information.

A view of the world from a particular place may be thought of as a vector of measurements, derived from processing sensory data in some fashion. These measurements may denote physical properties of the environment such as its shape, or purely visual properties like segmentation. In [12] we describe a measurement scheme based on image signatures, arrays of approximate perceptual invariants. If each element of a view has an error bound associated with it, we can match two views if all corresponding elements match within the given error bounds. Thus the robot's current view could be that of a stored place if the view matches an element of the place's set of views. However, since outliers are possible, this may still fail. The simplest fix is to allow a small number of mismatched elements for a matched view; if the outlier probability is small enough (as it should be) allowing one mismatch is sufficient. If a robot view is matched to a stored place

with an element mismatch, the new view is also added to the place's view set, as there is no way to tell which view contains the outlier. The main problem with this approach is that perceptual ambiguity can be increased; however, this doesn't appear to be a serious problem in practice.

3.3 Geometry

As in McDermott and Davis [20], the shape of the path graph is given by places' relative positions. For the time being, we ignore the robot's orientation at a place by treating it as unknowable. We assume odometry can provide, after each move, a set of points guaranteed to contain the robot's actual relative motion. Thus, place position estimates are represented by sets of possible positions. Positions are represented relative to local reference frames to avoid unnecessary accumulation of relative error. Geometric relations between frames are also explicitly represented, giving rise to a *reference graph*—note that this graph only represents those relations that are independently known. The use of local reference frames ensures that relative uncertainty remains locally bounded. For efficiency, we approximate uncertainty sets as intervals in $\mathbb{R}^2$ (see [1]). Our approach is also related to work on statistical representation of uncertain spatial relationships [25]; however, the interval representation simplifies computation significantly. Furthermore, in the absence of good statistical models of robot motion, calculations based on sets of feasible positions will be more reliable [2].

Matching and updating positional estimates is easily done using interval arithmetic [1]. Two position estimates are consistent if they intersect one another. If they are known in different reference frames, then the inter-frame transformation is applied first. If the transform is not immediately known, we search the reference graph and compose the intermediate transformations. Combining two different position estimates for the same place is done by intersection. If they are given with respect to different frames, the transformation betwen the frames can be updated as well. Thus the robot can easily merge odometric position estimates with previous place position estimates and consistently update the reference graph.

New reference frames are created whenever the robot's positional uncertainty grows too high; frames are merged when the uncertainty between them falls low enough. This policy ensures locally bounded uncertainty. Also, if frames are viewed as a packaging mechanism for places, related places will tend to be packaged together under this policy, particularly if local sensor feedback is used to improve odometry.

4 The Mapping System

The mapping system monitors execution in addition to maintaining the map; the system must know where it is in its map to modify it appropriately. 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. Each track maintains a position estimate with respect to a particular reference frame, a current view, and place type, as well as remembering the last place node matched in the map. Mapping proceeds as follows (refer to Figure 1). After each sequence of actions bringing the robot to a place, all current tracks are updated to reflect odometric and sensory readings. Then, there are a number of different operations that can be performed on the tracks, each corresponding to a decision about the state of the robot, the track, and the world. (A summary of the operations our system uses is given in Table 1.) For example, one operation is Continuation, where the robot decides it is at a place it could have predicted based on its last position and the action just performed. For example, if there is an action link labelled 'Go to door' from `place-1` to `place-2`, the robot performs 'Go to door', and the results are consonant with going from `place-1` to `place-2`, then a track which was at `place-1` would be Continued to `place-2`. Some operations correspond to decisions about the correctness of the map and adjust it accordingly. Various possible operations are proposed, and then the system decides which operations to perform. The operations chosen are then used to update the map by adjusting place descriptions and adding new place nodes and action links. Place nodes and action links examined may also be changed based on their long-term usage statistics. Each track is then updated to reflect the robot's new state of knowledge. The tracks are filtered to choose the best n as robot state estimates for the next step (keeping ambiguity under control). These components of the system are described below; a more detailed discussion of its workings can be found in [11].

4.1 Basic operations

What operations should the system include, and how should it decide which to actually perform? There are several basic operations needed for mapping even before considering error correction. Note that 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 first track operation is *Continuation*, which confirms an expected transition along an action link from one place to another. If a matching place exists, but there is no corresponding action link, the *Linkage* operation is applied, confirming the destination and adding an action link to the map. This operation performs a depth-limited search through the reference graph for a place match. In both of these cases, the operation suggests a new position estimate for the track and matched place. When there are multiple such updates to be performed on a place (due to multiple tracks matching it) the union of the various possibilities is the proper new estimate. The interval approximation to this is the least bounding interval (LBI) of the different estimates. The position estimates

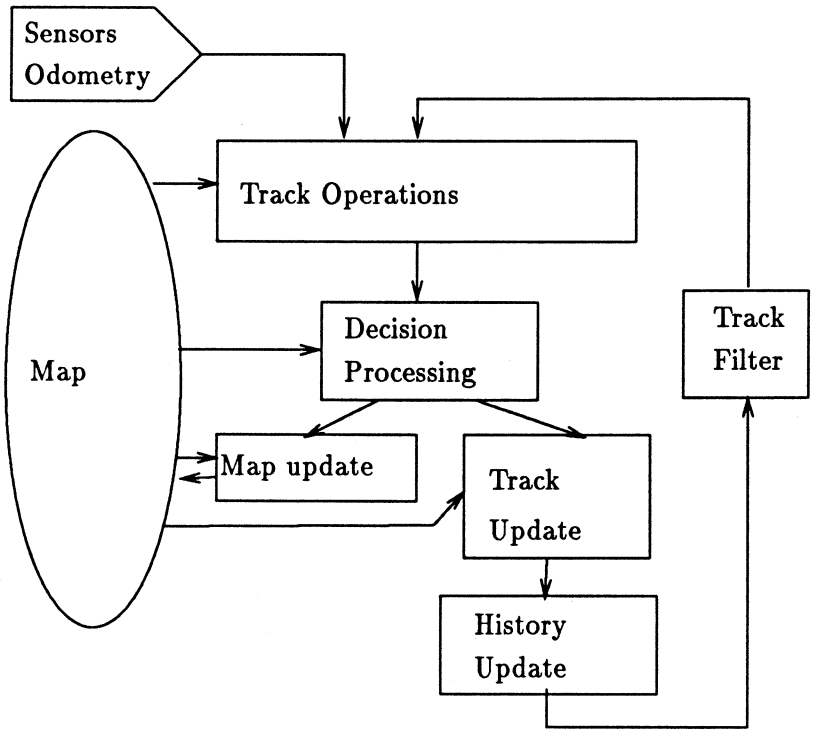


Figure 1: Mapping system architecture

of the tracks are then given by intersection of their projected positions with the new place position estimate. Finally, if there is no matching place node, a place node (and action link) is created by the *Creation* operation. Other operations relating to error detection and correction are discussed below, as is the process of deciding among operations.

4.2 Error correction

Our research addresses passive mapping, where the mapping process is independent of the control of the robot. That is, no experimentation or directed exploration is done; mapping is done while the robot carries out its normal activities, by 'eavesdropping' on the controller. To detect and correct errors, we must identify the causes and effects of different mapping errors. One approach that might be used is to record all decisions made by the system in a truth-maintenance system (TMS) [18]. Then when an inconsistency is detected, the TMS would be searched for a decision to fault to explain the problem. The faulty decision would then be removed, and (part of) the map rebuilt, removing the inconsistency. Even ignoring the great difficulty of credit assignment in this domain, this approach is infeasible due to the enormous space required and the continually increasing time

required to correct errors. The approach we take, therefore, 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.* Our results suggest that this sort of learning is feasible in the domain of navigational mapping. The types of errors possible and mechanisms to correct them are described below.

Position adjustment

The first type of error is *place inconsistency*, when a place's position estimate fails to contain its true position. This is caused by updating the place's position estimate by an incorrect match. To detect this situation, we define a place's *nominal envelope* as the LBI of the midpoints of all the position estimates ever used to update the place's position estimate. Interval midpoints are used instead of whole intervals to reduce the size of the nominal envelope without reducing its utility. The nominal envelope is guaranteed to eventually contain the place's true position, under unrestrictive statistical assumptions. The *α-Match* operation allows a position match with a place when the projected position of a track intersects the place's nominal envelope. If the track is correct, and the match is perceptually consistent, this suggests a place inconsistency. Hence, an α-Match suggests a new position estimate as the LBI of the place's old position estimate and its nominal envelope. This new estimate will almost certainly contain the true position. Thus the place's position estimate is reset to a consistent (though uncertain) state.

Another kind of positional error is *track inconsistency*, 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. For example, uncertainty in a track may become so great that it matches a known place where it should have discovered a new one. This causes the robot's state estimate to be in error, and if left uncorrected can lead to erroneous map construction. Track inconsistency is dealt with via a *β-Match*, which allows a track to match a place near its projected position. The nearness threshhold is a parameter of the system. This method assumes the track hasn't wandered too far from true. This assumption is reasonable if places are distributed sparsely in the environment, since bogus matches will be rare. Thus, if a track gets diverted from true, it is unlikely that a long sequence of incorrect matches will occur. Therefore, a β-Match with the correct place will be confirmed before track positioning error wanders too far.

Transient elision

Another category of errors are those whose detection depends on analyzing long-term developments in the map. The simplest to deal with are transients. 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 for by maintaining statistics on expected vs. actual encounters of place nodes, action links, and views. When one is encountered far

less often than it is expected, it is assumed to be transient and is removed from the map. This is safe, since removing a non-transient this way results in little change in navigational performance, since the removed map component would be encountered very seldom. Also, even if a map component does not represent anything real, but supports reliable navigation anyway, then as far as the robot is concerned, it may be treated as real. It is also easy to see that transient elision is effective in adjusting the map to slowly changing environments—outdated features will be removed, while new features will be added through normal mapping.

Merging

If a place can appear different on different visits, the map will eventually contain two place nodes representing a single place. If this is the case, then when the map matures, the nodes will have equivalent action link sets and the same estimated position. We do not assume that perceptual consistency can be used to correct this problem in the absence of an accurate model of perception. So, if the system can reasonably decide that two nodes have the same position and links to the rest of the world, then the nodes should be merged. This can also be done for larger subsets of the map by checking for isomorphisms.

This process can be approximated by maintaining *merge histories*, sequences of potentially matching place pairs. Each such pair must have matching position estimates. Futhermore, successive pairs of nodes must have consistent relative positions. Creating and maintaining merge histories is done incrementally. Each track is associated with a set of histories, which are extended, merged out, or killed at each step, based on the track's new place.

When there are two tracks at different places whose position estimates overlap, a merge history is created between them. If two tracks with a merge history between them move to a pair of commensurable places, the history is extended, otherwise it is killed. This only happens, of course, if the places look similar enough to match a single robot view. Commensurable place pairs which appear different are found during the spatial search for Linkages, and single-track histories are created for them. They are extended analogously to dual-track histories. When a history achieves a 'critical mass' demonstrating a sufficient level of neighborhood consistency, all its place node pairs are merged. The notion of criticality admits various definitions, the simplest of which (and the one which is implemented) is that the number of distinct elements in the history must exceed a given threshhold. From this it follows that a dual-track history whose tracks converge to the same place has 'gone critical' regardless of its length (as it could be extended indefinitely). A stricter notion of positional consistency is ensured for merging, either:

- The two places cannot be farther than δ_{sep} from each other, where δ_{sep} is a known lower bound on the distance between any two places, or
- The relative size of the intersection of the places' position estimates is above

a given threshhold.

If the first condition holds, the two places nodes must represent the same place. The second condition heuristically encodes a confidence in the nodes' equivalence. When two place nodes are merged, their view sets and act link sets are unioned and their position estimates are combined. If the LBI of the position intervals is smaller in width than δ_{sep} then the positional estimate of the new place is the LBI of the two constituents, since uncertainty up to δ_{sep} is acceptable and slight inconsistencies can be corrected. Otherwise the new position estimate is the intersection of the two estimates, by the usual rule of information combining.

Splitting

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 be merged erroneously. In any case, this situation will cause inconsistency to creep into the place node. Without further analysis, this can engender corruption elsewhere in the map as well. Moreover, unecessary ambiguity is introduced to the map, making navigation harder and less reliable. However, if we know that places are always at least some distance δ_{sep} from each other, this problem can be corrected. The system detects when this error has occurred, then splits the offending node into two.

Each place node maintains a set of *estims*, each consisting of a point position estimate and an integer weight. An estim's weight is a measure of its data support. Every time a node's position is updated, the midpoint of the constraining interval is treated as a point estimate of the robot position and an estim at that point is added to the node's estim set with weight 1. The estim set is then reduced by merging all estims whose position estimates are closer than δ_{sep}, by computing the weighted average of their positions. Weights of composite estims are computed as the sum of the weights of their primitive components.

To decide when to split, pairs of estims are tested to see if they indicate a split (are *splittable*). Intuitively speaking, if a single place is being represented, than a single estim will eventually accumulate enough weight to dominate all the others. If multiple places are represented by a node, on the other hand, several estims will dominate, each corresponding to a different place. If the different places are encountered with similar frequency, then splittable estims will have similar weights. If one of the places is much less likely than the other, however, the statistics will be skewed and a split will not be indicated. This is reasonable, since if the probabilities *are* that skewed, the place description will converge on that of the more probable place; the less probable one should then be created separately. And if it isn't, the cost of retaining the ambiguity is low anyway, since the low-probability place occurs infrequently.

Therefore, the following conditions determine if two estims are splittable. First, they must be more than δ_{sep} distant from one another, so that they represent positions that can be different places. Second, both estims must have more weight

than a pre-determined threshhold, ensuring a minimal standard of evidence to avoid flukes. Finally, the ratio between the estims' weights must be close to one, showing that they have a similar amount of evidence.

An issue requiring deeper inquiry is the fact that since a place may be approached from different directions and we rely on relative odometry, the position sample space can become biased, defeating the weight similarity condition. This problem appears to require being able to quantify and compensate for the bias by correlating directions the robot came from with corresponding estims. This is an area for further research.

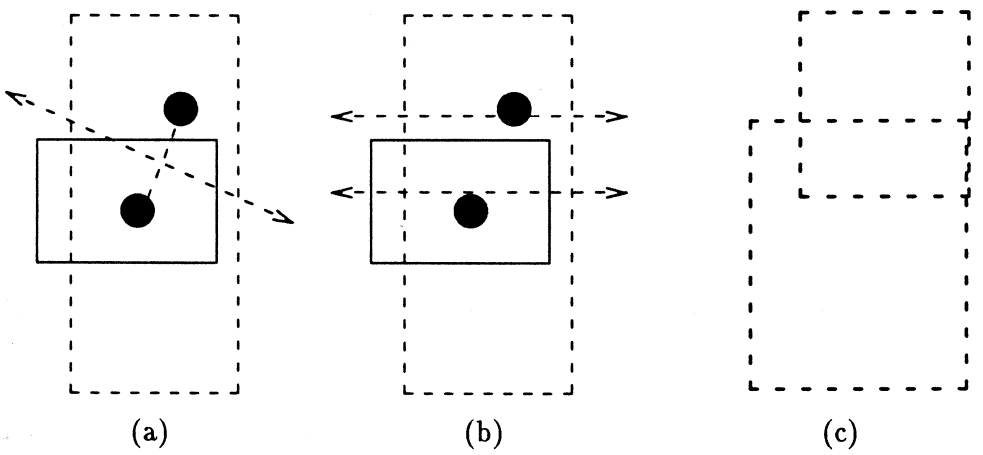


Figure 2: Splitting a place node; the solid rectangle is the place's position estimate, the dashed rectangle its nominal envelope; the circles are splittable estims. (a) Midline between the estims. (b) Applying the interval approximation. (c) Final place position estimates derived.

Splitting nodes is simplified by transient elision. Both of the new nodes can be given the same action links and view sets of their source, since incorrect parts of the description will be elided as transients. However, the positions of the new nodes should each contain the 'true' position of the place they correspond to. This is approximated by splitting the original node's nominal envelope along the line equidistant between the two splittable estims (see Figure 2a). This assumes that the true position of the place corresponding to an estim is closer to it than to the other estim. The full position of each new place is the LBI of its part of the nominal envelope and the original position estimate if the estim fall in the original position estimate, and just the estim's part of the nominal envelope otherwise (see Figure 2). As noted, transient elision allows the system to just copy the action links and view set of the original place. This problem of action links and percepts can more properly be dealt with by keeping track of associations between estims and links traversed and views recognized. This is complicated, however, by the

fact that tracking errors can cause these statistics to improperly reflect the actual situation.

Operation	When applicable	Action performed	Error corrected
Continuation	Matched expected place	Update position estimate	—
Linkage	Matched unexpected place	Update position estimate and add action link	Incomplete map
Creation	No matches	Add new place node and action link	Incomplete map
α-Match	View and type match, and position match with nominal envelope	Expand place nodes position interval	Inconsistent place position
β-Match	View and type match with nearby position	Set track to be at the place	Incorrect track position
Elision	Node, link, or view encountered far less often than expected	Object deleted from the map	Transient creation
Merging	Extended consistent merge histories	Merge place nodes	Multiple nodes for one place
Splitting	Strong competing estims in a place	Split place node	A single node representing multiple places

Table 1: Summary of mapping operations

4.3 Decision processing

Since different track operations reflect mutually conflicting decisions about the state of the map and they can interfere with one another, it is necessary to decide which track operations, out of those possible at each stage, to perform. Other operations, for example Splitting, are done as soon as they are seen to apply. A general framework in which to approach this problem is to assign each potential operation a priority, and then perform only those with the highest priority.

The simplest such scheme is to assign each type of operation a fixed priority, based on general preference criteria. Clearly, Continuation and Linkage are the highest priority operations. When there is only one current track, the assumption is that it is correct, and hence α-Match is to be preferred over β-Match; the reverse holds when there are multiple current tracks. Both are preferred over

Creation, as they will rarely interfere with a legitimate Creation, and they have no catastrophic effects (only increasing local uncertainty). Our results show that this priority scheme works quite well. However, a more principled approach would be a dynamic priority scheme, taking into account some notion of which operations are more likely to be 'correct' in the current context.

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, and 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. The simulator is described in detail in [10]; a brief overview is given below.

5.1 Simulated environment

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 'substance', representing a measurement of some intrinsic property of the material, eg. texturedness. Empty cells have an optional place type; the two currently used are `door` and `corner`, assigned to the configurations shown in Figure 3a. This categorization is used to implement place type approachers and recognizers.

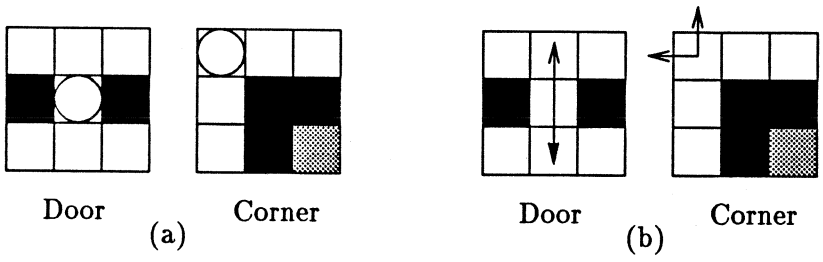


Figure 3: (a) Place type classification patterns (the circle indicates the classified cell). (b) Stable view directions for place types.

Place type actions are built upon the notion of *fixations*, which are visual markers for places of particular types (a kind of effective designator, see [19]). The robot can obtain a fixation on a place of a particular type if one is in its field of vision.

It can then use the resulting fixation to go to the place. Fixations only remain valid as long as the robot doesn't move; an invalid fixation is useless. 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. The corner approacher has a chance of performing according to two additional error modes; a corner can be missed and the next corner ahead (if any) is approached, and the robot could mistakenly turn a corner and find the next one ahead of it. The place type recognizer has a chance of misclassifying the robot's current place. Other actions available include turning in place by fixed amounts and aligning with an adjacent wall or with the door the robot is in. In addition to a success flag, each action returns an odometric estimate of the robot's relative motion as a random interval containing the true motion, of size proportional to the distance travelled.

The perceptual primitive is the view, which is simply a list of numbers, each representing a visual measurement of the substance of some object. The robot's field of view (typically $\pi/4$ radians) is sampled at evenly-spaced angles, and the nearest filled cell in each direction is seen. Such measurements have unbiased uniform bounded error added to them; there is also a chance that the measurement will be contaminated and chosen uniformly from the universe of posible measurement values. Since we use no statistical assumptions in perceptual processing, the only important property of the uncontaminated noise distribution is its boundedness. To improve perceptual stability, we assume that the robot can approximately orient its camera stably relative to its current place type before taking an image; stable camera directions for doors and corners are shown in Figure 3b. The camera's actual direction is randomly chosen from an interval about the true angle.

5.2 Experiments

To test the system's ability to correct for mapping errors, we ran experiments primarily in two simulated worlds. The CONFUSION world (Figure 4) has two identical door-corner pairs (on the right), and all places look alike. This exercises the system's ability to use splitting intelligently to tease out the true structure of the world. The LOOKTWICE world (Figure 5) has a simple structure, but the range of views possible from any one place is large. This tests the use of merging to achieve parsimonious representation. Also, the distance between places is large, leading to more geometric uncertainty. Both worlds, of course, also test transient elision and α/β-Matching.

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 hidden information. A map that looks good at one point, may easily become garbage in the near future. There are, however, some possible methods for measuring mapper success. The first we examine is

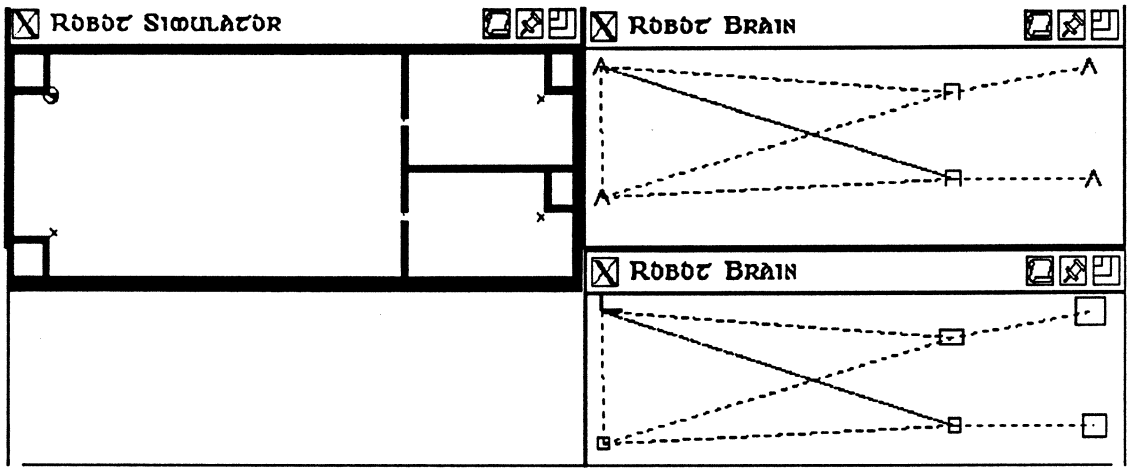


Figure 4: The CONFUSION world and a typical map learned. The upper map picture shows the topological structure with symbols denoting place types. The lower picture shows the reference frame and place position estimates as intervals. Note that links only mean that a sequence of actions is known to get from one place to another—no geometric interpretation is implied.

based on the concept of expectation error—the error inherent in allowing the robot to rely on the map to predict its expected position after each move. We can 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. For a track estimate given by a set of possible motions S and a true motion x, we have the SSD as $\int_S \| x - y \|^2 \, dy$. If the system is effective at mapping, we would expect the average SSD per move (SSD/M) to asymptotically converge on a small constant. In Figure 6 we plot cumulative SSD/M against time (robot moves). Figure 6a shows SSD/M for several runs in CONFUSION superimposed, demonstrating the basic properties of the SSD/M curves. When the system starts mapping, it doesn't yet have strong expectations, so they cannot be violated. As the robot moves and uncertainty increases, the SSD/M quickly grows. Then the SSD/M begins to plummet and approaches a stable level, as map error get corrected and the system settles down. Note that stable convergence does not always occur; instability is possible, though very infrequent. Figure 6b and c show average SSD/M over 10 runs for CONFUSION and LOOKTWICE; both curves indicate proper average convergence.

Another measurement of mapping error is the average number of error corrections per move. This can be examined by looking at the average number of map adjusting operations per move (MAO/M). The map adjusting operations are Create, α/β-Match, Splitting, and Merging. Plots of MAO/M against time are given in Figures 7 and 8. They have the same qualitative shape as the SSD/M plots, indicating convergence. In Figure 9 we see some of the effect of varying noise levels on the algorithm. Figure 9a shows curves for increasing odometric

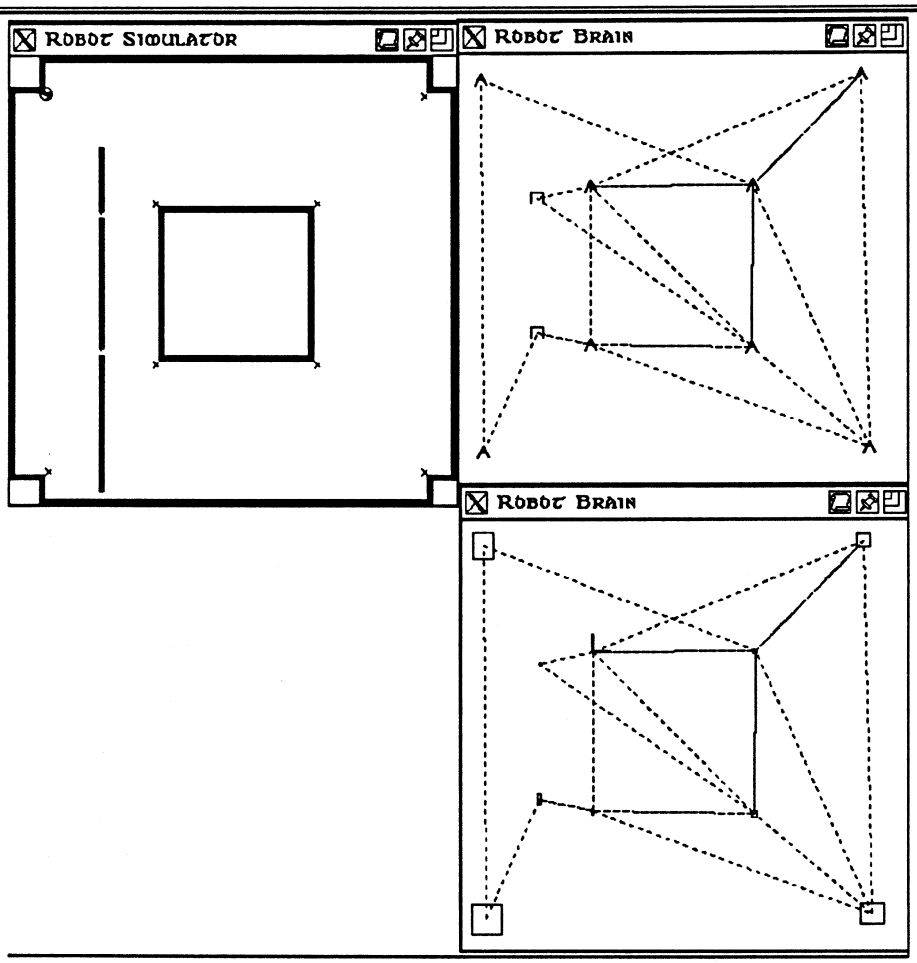


Figure 5: The LookTwice world and a typical map learned. The upper map picture shows the topological structure with symbols denoting place types. The lower picture shows the reference frame and place position estimates as intervals. Note that links only mean that a sequence of actions is known to get from one place to another—no geometric interpretation is implied.

noise levels, of MAO/M averaged over 10 runs in Confusion. Figure 9b shows curves for increasing camera angle noise levels, of MAO/M averaged over 10 runs in LookTwice. It is clear that the convergence is delayed somewhat, but not overly so, by increasing sensor error.

A third way to evaluate the performance of our system is to define a criterion for deciding that a mapping run has converged, and look at the fraction of mapping runs that converge in a certain number of moves. We can reasonably say that a map has stabilized if only the expected stable number of map-adjusting operations are performed on it for 100 moves. Radical map-adjusting operations are those that change the structure of the place graph: Creation, Splitting, Merging, and Place Elision. The expected number of Splittings per move in a stable map is 0, since all places should be known. The expected number of Creations or

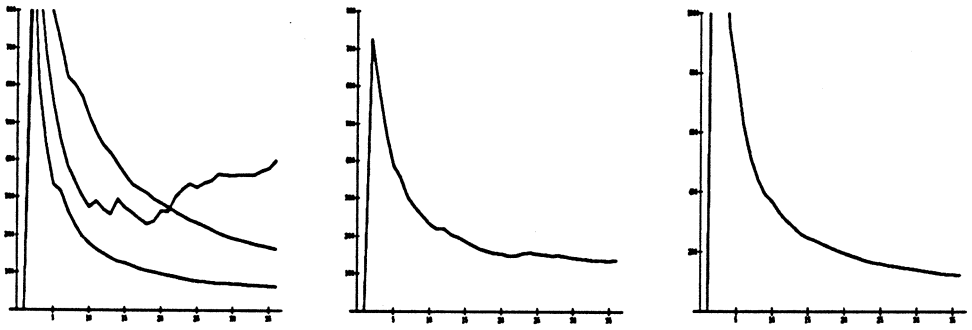


Figure 6: SSD/M vs. time for 700 moves, sampling each 20 moves. Note that the important thing here is the shape of the curves, not the absolute scale. (a) Several CONFUSION runs. (b) Average of 10 CONFUSION runs. (c) Average of 10 LOOKTWICE runs.

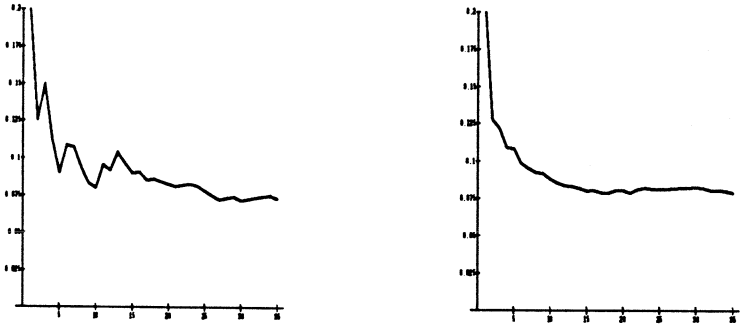
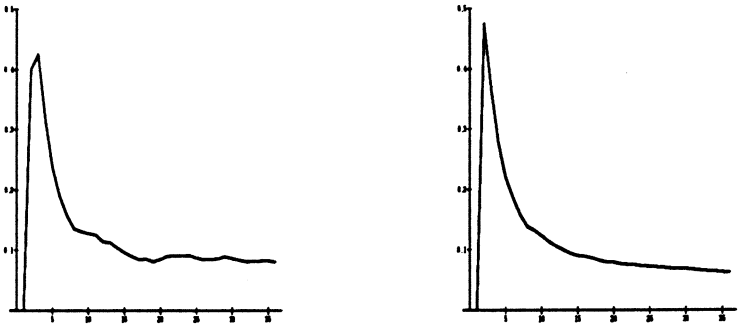
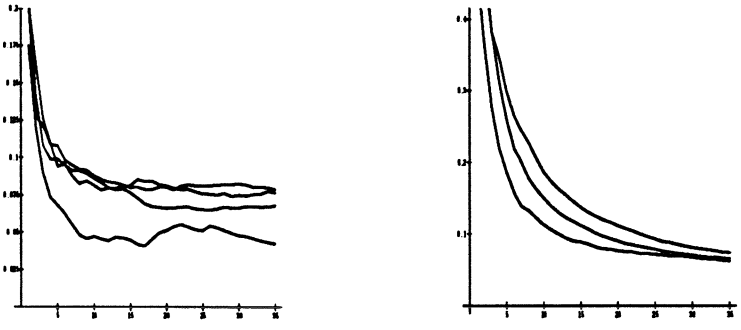


Figure 7: Plots of average MAO/M against time in CONFUSION. (a) A single run. (b) Average over 10 runs.



(a) (b)

Figure 8: Plots of average MAO/M against time in LOOKTWICE. (a) A single run. (b) Average over 10 runs.



(a) (b)

Figure 9: MAO/M vs. time vs. noise. (a) Average MAO/M over 10 runs in CONFUSION, with odometric error = {0.1,0.15,0.2,0.25}×distance travelled. (b) Average MAO/M over 10 runs in LOOKTWICE, with maximum view angle error = {0.1,0.2,0.3} radians.

World	Motion Noise	Angular Error	# stable by 200 moves	# stable by 300 moves	# stable by 400 moves
CONFUSION	0.1	0.1	6	9	10
CONFUSION	0.15	0.1	6	8	10
CONFUSION	0.2	0.1	7	9	10
CONFUSION	0.25	0.1	8	10	10
LOOKTWICE	0.1	0.1	10	10	10
LOOKTWICE	0.1	0.2	9	10	10
LOOKTWICE	0.1	0.3	8	10	10

Table 2: Mapper convergence statistics. 10 simulation runs were done for each world/parameter setting.

Mergings/Place Elisions per move in a proper map depends on the chances of measurement outliers appearing—when more than one occurs in a view, then a Creation will almost certainly be performed. Eventually, the transient will be Merged or killed. If the probability of a measurement outlier is p and the number of elements in a view is n, then the expected stable number of Creations (or Mergings + Elisions) per move is given by:

$$C_{\text{stable}} = \sum_{i=2}^{n} \binom{n}{i} p^i (1-p)^{n-i}$$

In our simulations, $p = 0.01$ and $n = 8$, thus $C_{\text{stable}} = 0.027$—so if no more than 3 Creations or Mergings/Place Elisions occur in 100 moves, we declare than a map to be stable. Table 2 shows the fraction of 20 test runs that converged under this definition within certain numbers of moves. Note that convergence time appears to be unaffected by increasing noise (the seeming trend of improvement with increasing noise is likely due to our small sample size).

6 Future Directions

Our primary short-term goal is to implement our system on a mobile robot. We have already done some work on developing primitives to support perceptual matching in our model [12]. Place type approachers can be implemented using known behavior-based techniques [17, 8]. In addition to having a more realistic testbed, more extensive analysis of the system's performance can be done, using methods of data analysis to determine the convergence and stability properties of the system. Furthermore, although parts of the system can be formally motivated, it is quite difficult to formally characterize our problem as a whole; this also forms an important thrust of our work for the near future.

One specific area that needs to be explored further is the principled design of dynamic preference criteria for track operation. One approach would be to take into account the 'explored-ness' of regions of the world. Continuation is preferred

to Linkage only if all possible actions have been well-explored in the current area, otherwise prefer neither. Prefer Creation to α/β-Match in unexplored areas, α/β-Match to Creation in well-explored areas, and neither in in-between areas. Of course, formalizing these notions is a difficult question in and of itself. Priority for individual operations (rather than just operation types) could use these notions along with a confidence in a track being correct—an operation of a lower priority type may take precedence if the system is particularly confident in its track. These ideas should be formalizable in the framework of decision theory [5] by formulating expressions for the utility and cost of performing each operation in given circumstances.

Though we partially utilize the structure of the environment to correct errors, there are also useful sources of information that are not currently taken into account. For example, the fact that a particular approacher succeeds or fails from a particular place provides a strong *local* functional constraint on the robot's location; the only functional constraint currently used is long-term trajectory consistency, whose application depends on the robot's behavior (which the mapper does not control). Thus, it would be desirable to incorporate active methods into our mapping system. Currently, it often takes quite some time for the robot to converge on a map; one cause of this is the undirected nature of the robot's exploration—essentially a random walk. However, since we use a passive mapping strategy, exploration is easily incorporated. Furthermore, unlike mappers that rely on active strategies, our system can easily accomodate interleaving exploration with other goal-directed behavior.

7 Conclusions

Though the problem of robot map learning has been much investigated, little attention has been paid to discovering and correcting the errors that must inevitably creep into any map constructed. We have described a novel approach which performs reliable mapping by explicitly detecting and correcting mapping errors. We show that such error correction can be done without keeping track of all mapping decisions ever made. The methodology makes use of environmental structure to determine the essential information needed to correct mapping errors. We also show that it is not necessary for the decision that caused an error to be specifically identified for the error to be corrected. It is enough that the type of error can be identified. Furthermore, since our method performs passive mapping and we make no demands on the behavior of the robot, the ideas described here can easily be integrated into other robot mapping systems. The system described has been implemented; extensive testing in a realistic simulation environment demonstrates its effectiveness.

Acknowledgements

Many fascinating discussions with Greg Hager, Dave Kriegman, and P. Anandan helped these ideas come to fruition. The report was much improved with the help of careful reading by Michael Black.

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. *IEEE Robotics and Automation*, (submitted), 1992.

[3] Nicholas Ayache and Olivier D. Faugeras. Maintaining representations of the environment of a mobile robot. *IEEE Trans. on Robotics and Automation*, 5(6), 1989.

[4] 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.

[5] James O. Berger. *Statistical Decision Theory and Bayesian Analysis.* Springer-Verlag, New York, 1985.

[6] David J. Braunegg. *MARVEL: A System for Recognizing World Locations with Stereo Vision.* PhD thesis, MIT, 1990.

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

[8] Jonathan Connell. *A Colony Architecture for an Artificial Creature.* PhD thesis, MIT, 1989. Technical Report 1151.

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

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

[11] Sean P. Engelson and Drew McDermott. Robotic map learning with explicit error correction. Technical Report (forthcoming), Yale University Department of Computer Science, 1992.

[12] Sean P. Engelson and Drew V. McDermott. Image signatures for place recognition and map construction. In *Proceedings of SPIE Symposium on Intelligent Robotic Systems*, 1991. (to appear).

[13] David J. Kriegman. *Object Classes and Image Contours in Model-Based Vision.* PhD thesis, Stanford University, 1989.

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

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

[16] T. S. Levitt, D. T. Lawton, D. M. Chelberg, and P. C. Nelson. Qualitative landmark-based path planning and following. In *AAAI-87 National Conference on Artificial Intelligence*, Seattle, Washington, 1987.

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

[18] Drew McDermott. A general framework for reason maintenance. Technical Report 691, Yale University Department of Computer Science, 1989.

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

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

[21] David P. Miller and Marc G. Slack. Global symbolic maps from local navigation. In *Proceedings of IJCAI-91*, pages 750–755, 1991.

[22] Hans P. Moravec and Alberto Elfes. High resolution maps from wide angle sonar. In *IEEE International Conference on Robotics and Automation*, pages 138–145, 1985.

[23] Ronald L. Rivest and Robert Sloan. Learning complicated concepts reliably and usefully. In *Proceedings of AAAI-88*, pages 635–640, 1988.

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

[25] Randall Smith, Matthew Self, and Peter Cheeseman. Estimating uncertain spatial relationships in robotics. In *Proceedings of the Second Workshop on Uncertainty in Artificial Intelligence*, Philadelphia, PA, 1986.

[26] 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.

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