AIAA-94-1293-CP

Passive Mapping and Intermittent Exploration for Mobile Robots

Sean P. Engelson
Yale University
Department of Computer Science
P.O. Box 208285 Yale Station
New Haven, CT 06520-8285
Email: engelson@cs.yale.edu

Abstract

An autonomous robot must be able to learn maps of its environment while accomplishing meaningful tasks. Such 'passive' map-learning is difficult, since it cannot rely on active exploration. However, it gives the robot more flexibility in how it learns about a complex novel environment. The primary difficulty is that incorrect maps may be inferred, since insufficient information may be available at any one time (since exploration is disallowed). We address this problem by allowing maps to contain errors, while correcting those errors when possible via a set of heuristic error-correction strategies. Results in a realistic simulation with a random walk (to simulate worst-case explorative action) demonstrates the efficacy of the basic technique.

Passive mapping may still be inefficient, so we incorporate intermittent exploration while maintaining the benefits of the passive approach. This is done by using predefined 'opportunity scripts' which direct the robot in ways that improve the map. Scripts are short plans which are applied depending on whether or not they interfere with other robot goals. The scripts we have implemented mostly use known techniques to improve mapper efficiency. Occasional use of these scripts over a base random walk strategy shows a speedup in map convergence, demonstrating the efficacy of intermittent exploration. Also, by using intermittent exploration, very complex worlds could be learned efficiently.

1 Motivation

While robotic manipulation technology has revolutionized manufacturing in recent years, the potentials of mobile robotics remain virtually unused. There are a number of reasons for this, some of them sociological, but the main reason is that the technology of mobile robotics is not yet sufficiently mature for most practical applications. One of the main research areas which requires development is *map-learning*. By this, we mean the automatic acquisition by a mobile robot of a model of its large scale environment (floor layout, for example). This is needed, even in known environments, because hard-wiring the structure of its environment into a robot can be very costly, and it is difficult (if not impossible) to keep such a representation up to date when the environment is changed. Naturally, there are also cases where the structure of the environment cannot be known accurate in advance even to the system designers; automated mapping is required in those cases.

There are many useful tasks that mobile robots can perform that require the use of some sort of environmental map. One example is that of an office or factory courier, whose function is to transfer materials between areas of a building. This task is one of the few for which mobile robots are currently being used; TRC has a robotic courier installed in hospitals, for delivering non-critical items to patients upon request. Their robot contains a detailed, preprogrammed map of the hospital building it works in; the hospital environment was also engineered somewhat to support reliable navigation (beacons were places, for example). Effective map-learning would cut the costs of installing such a system, by both easing the initial setup, and by making the robots more flexible (they could be transferred between different buildings, say).

Another class of tasks are those in which little or no a priori information about the environment exists. One such task, ripe for robot use, is search-and-rescue missions. Rescuing people often involves going into hazardous environments, so it would be desirable to use robots wherever possible. Such rescue missions, whether from burning buildings or from caves, often require learning the structure of the environment on the fly, so that searching is done efficiently (time is usually of the essence). Also, there is no time to explore for exploration's sake (we will return to this point below). A task which is similar, though in a completely different domain, is planetary exploration. Getting men to Mars to carry out scientific exploration and experimentation is, at present, a difficult proposition. A cost-effective solution that has been proposed is to send robots to gather scientific data. These robots will need to move about in large areas of the surface to gather that data, and hence will need some sort of internal mapping to support navigation.

1.1 Passive mapping

There are several features of the tasks described above that underscore important issues for robotic map learning. The first is that mapping does not occur in a vacuum. It is a part of a larger system, aiding navigational planning. A related point is that mapping should take place during normal goal-directed task execution. In fact, a 'mapping phase' wherein the agent maps out its entire environment may be wholly infeasible due to the environment's size or changeability. We therefore propose the view that mapping should be 'passive', and not require controlling the robot's actions. It thus functions as a static map, as far as planning is concerned, but the map's utility transparently

improves over time. A schematic diagram of the high-level cognitive architecture our view implies is shown in Figure 1. The mapper and deliberator (action selector) are functionally independent. Exploration enters deliberation through a sort of 'back door', via suggested exploration scripts (described below in Section 5).

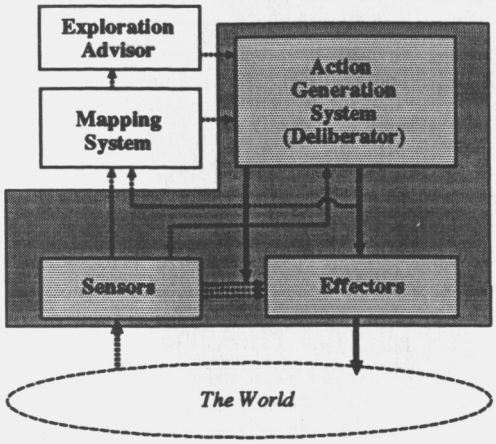

Figure 1: Cognitive modular divisions in the passive mapping paradigm. Solid arrows indicate control flow; dashed arrows indicate information flow.

Our approach may be contrasted with much previous research in map learning, where the mapper is viewed as a procedure which, after some amount of time, outputs a 'correct' map; this representation is then to be used for planning and reasoning. This paradigm, or variations thereof, is ubiquitous (eg, [3; 4]). These methods typically require the mapper to be 'active' and to take control of the robot when uncertain information must be verified. This is needed due to the desire to learn a 'complete' and 'correct' map in finite time; however, this active approach interferes with goal-achievement. This approach has other problems in the real world as well, since the real world is (practically) open-ended—the world to be learned cannot be bounded. Also, attention must be paid, in constructing a world representation, to the use to which it will be put. We address these issues below.

1.2 Overview

In the next section, we describe a general framework for developing adaptive models of a robot's environment, which we have used to develop our mapping system. We then describe the representation scheme we use for mapping, which incorporates both topological and metric information. In Section 4, we describe the algorithms used in the mapping system, including mapping-error diagnosis and correction methods. Given this passive mapper, we then develop methods for intermittent exploration, to improve mapping efficiency. We then describe our results in a robotic simulation. Section 7 reviews related previous work in map-learning. We close with a discussion of how our methods could be used in practice, and future directions for our research.

2 Adaptive Modeling

2.1 Adaptive State-Space Models

We view the mapping system as a sort of *adaptive state-space model*. There are two fundamental operations for which a state-space model is used: state estimation and state prediction. Planning uses these operations in conjunction with a domain theory describing the physics of the world. We may thus view such a model as a black box which outputs a state prediction given a previous state and a robot action, and outputs an improved state estimate given a state prediction and sensory input. This is essentially what is known to control theorists as an *observer*. It gives a very general conceptual framework, encompassing both continuous estimation methods such as the Kalman filter and discrete methods such as Markov chain models. The fundamental point we wish to stress is the use of the model as a black box for estimation and prediction, as far as the rest of the planning/control system is concerned. Internal issues of representation, and indeed whether or how the representation changes over time, should be largely irrelevant to the rest of the system. In what follows, we develop an architecture for such systems, and then show how we have applied it to the specific problem of mobile robot map-learning.

2.2 Discretizing the World

Robots typically live in continuous state-spaces (eg, the plane for a mobile robot); to represent these spaces effectively, some sort of discretization must be done. Our work focuses on robots designed to operate in indoors environments (office buildings, factories, etc.). We adopt a technique based on Kuipers' topological mappers [13; 14], which use control laws with good stability properties (actions) to pick out distinguished 'places' in a continuous space. This allows a distinguished set of points (actually, small regions) to constitute 'waypoints'. Waypoints may be corridor intersections, or areas near particular pieces of equipment; the main requirement is that they be recognizable. Waypoints can be used to represent the structure of the entire space, relative to the capabilities of the robot. As noted by Kuipers, this approach allows a representation to directly support navigational planning. Abstractly, the state space is represented as a finite state machine with non-deterministic transitions (which can often be assigned transition probabilities).

2.3 The Architecture

We now present an architecture for adaptive discrete state-space models. We first divide the system at a high level into an *estimator* and an *adapter* (refer to Figure 2). The core of the estimator is the loop between projection and matching. The projector predicts new states given old state estimates and control inputs. There may be multiple state estimates in the system, which we call *tracks* (each tracks a possible true state). The matcher decides which states are consistent with each predicted estimate, given the current perceptual input. Thus far, we have the standard recursive estimation framework. Now, the spaces we are interested in are both very large (thousands of states) and relatively unstructured (without even ap-

proximate closed form estimators). Hence we need indexing, to find likely candidate states to feed to the matcher. Further, in different situations, different matching methods will be appropriate (for error recovery, for example). Therefore, the matcher is divided into a general matching engine, dependent only on the representation language, and a task-dependent set of matching methods (*matchers*). The matching engine also provides a method of conflict resolution to determine which matchers are applicable in given circumstances.

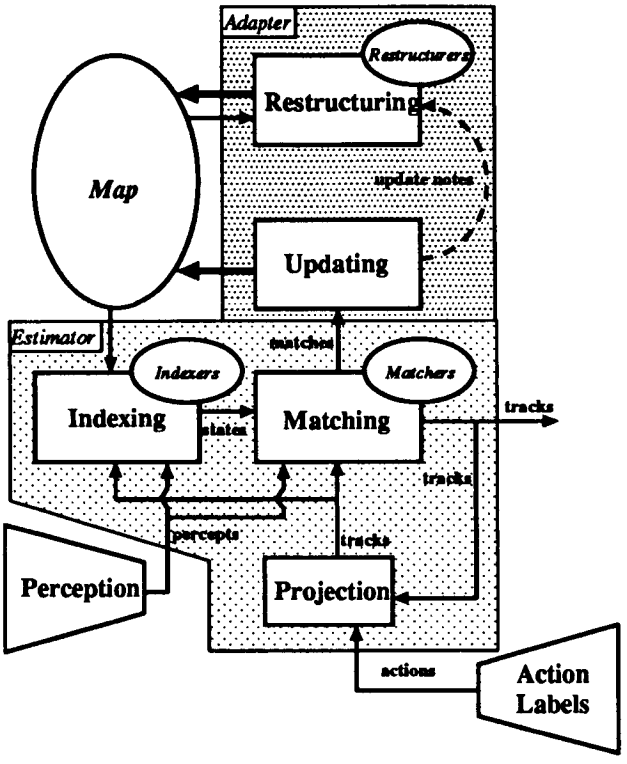

Figure 2: Adaptive state-space model architecture.

The adapter consists of an updating module and a restructuring module. The updater adjusts the parameters of states in the representation, for example, the position of a known waypoint. It also can monotonically change the topology of the represented state space by, for example, adding new state transitions. The exact type of updating that is performed depends on the matches found by the matcher; different matchers may (and usually will) have different associated update methods. A deeper sort of adaptation is restructuring, which uses information about the large-scale structure of the known state-space as well as integration of observations over longer terms to adjust the structure of the state-space. This may involve splitting or coalescing states, adjusting hierarchical relationships, and so on. Like matching, and for the same reason, we divide this module into a general restructuring engine and a task-dependent set of restructurers. To alleviate the problem of searching aimlessly in the state space for restructuring to do, it can be advantageous for the updater to inform the restructuring module when it performs an update, since a change made by the updater to the representation may trigger a restructuring operation.

3 Diktiometric Representation

The first step in specializing the architecture described above to robot mapping is the specification of a representation language for our maps. The straightforward method using the discrete state-space approach amounts to topological mapping, the method suggested by Kuipers in [13]. The world is represented as a graph of waypoint nodes, labeled with local perceptual information. Transitions are labeled with robot control routines which constitute the actions that move the robot between waypoints ([9] describes how these routines are derived). We extend this notion to directly take into account geometric knowledge as well. *Diktiometric*[1] representation explicitly represents geometric relations between waypoints. A diktiometric representation (a *diktiometry*) consists of two graphs, a *path graph* and a *reference graph* (see Figure 3). Path graph nodes represent waypoints, and arcs represent action transitions. Nodes in the reference graph represent local coordinate frames, with links giving known geometric relations. The two graphs are connected by *reference links*, giving the positions of waypoints with respect to particular local reference frames.

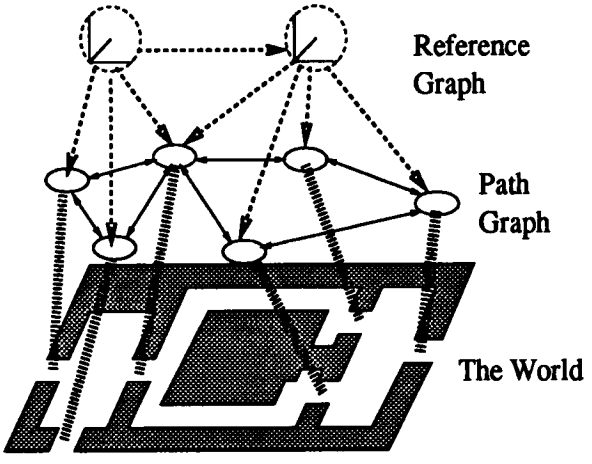

Figure 3: A schematic picture of a diktiometry, where doorways are taken as 'waypoints'. Action links are solid, reference links are dashed.

3.1 Uncertain Geometry

We represent uncertain geometric relations between waypoints relative to local reference frames with limited coverage. This ensures that, provided the robot knows which reference frame's domain it is in, relative uncertainly remains bounded. This is so regardless of the method used to represent uncertainty. In the multi-dimensional Gaussian distribution approach [20], the local reference frame

[1] From the Greek *δίκτυον* meaning 'network', and *μέτρον* meaning 'measurement'. Diktiometric representations represent the world as a network of waypoints with relative positions.

approach amounts to maintaining a set of smaller covariance matrices (with smallish eigenvalues) instead of a single large covariance matrix for the entire system (which will necessarily have some large eigenvalues, hence large uncertainty, even between nearby points). The local reference frames are related to each other by small covariance matrices; a position in any frame can be related to any other by appropriate composition. The local method is also a performance win when locality can be established (though doing so may incur extra costs). Rather than using such a statistical representation for odometric uncertainty, however, we advocate the use of bounding intervals.

The primary reason that Gaussian noise models are so widely used is the Kalman filter, and its non-linear cousin, the extended Kalman filter. These are the optimal linear recursive updating procedures, given a truly Gaussian noise model. However, when this model is not correct, the Kalman filter can be suboptimal and can even diverge. We therefore prefer a non-distributional approach. Rather than representing a probability distribution on the true value of a measurement, we give bounds on the possible true values. These bounds give us a set of possible values within which true must lie. Projection and updating can be done easily and efficiently using interval arithmetic [1]. Our contention is that odometry is subject to so many unmodellable sources of noise (slippage, bumps, voltage fluctuations, etc.) that the best that can be hoped for is reasonable bounds on relative position. In a typical office environment, with current equipment, odometric error over distances of several meters is about 10% of distance traveled[2]. This gives reasonably tight bounds on relative motion; use of sensory feedback (eg, visual motion analysis) may also help.

4 The Mapping System

The adaptive state-space architecture described above, combined with diktiometric representation, provides a framework for designing a robot mapping system which supports the flexible navigation planning tasks we wish to address. This section describes in more detail how this works.

4.1 Indexing waypoints

For some applications using the adaptive state-space model paradigm, indexing is trivial, due to there being a relatively small number of states. If, however, we wish to learn diktiometries of large-scale spaces in an open-ended fashion, we must deal with thousands of waypoints (at least). It is simply infeasible to examine the entire diktiometry for matching waypoints to extend a track. On the other hand, there is no fool-proof way of generating only the most 'similar' candidates that we know of. Hence, we developed heuristic indexing methods that should work well in practice. There are three categories of indexing we examine: *expectational*, *geometric*, and *perceptual*.

Each of the geometric and perceptual indexing modules produces a stream of sets of candidate waypoints. Each set in a stream contains better candidates than those following it, while members of a set are assumed to all be equivalently good. The indexing methods are integrated by running the modules in parallel and combining the candidate sets that come out.

Expectations The simplest form of indexing uses the path graph to predict the expected state of the robot after executing the current action. Given a particular current waypoint, the expectational candidates are those which are predicted by the last action taken from that waypoint. Naturally, there may be more than one, since actions can be non-deterministic. Expectational indexing produces one candidate set, used together with the first sets produced by other indexing.

Geometric Indexing The second sort of indexing is geometric indexing, based on waypoint positions. The basic idea is to find waypoints whose position relative to a track's is consistent with the last position change. We will discuss two indexing methods which use the reference graph to find waypoints likely to be near the current (projected) robot position.

The simplest method is to use a depth-limited search through the reference graph, starting from the current track's frame. If the new position estimate may be consistent with one of a frame's waypoints, the waypoints are checked individually for consistency and candidates suggested. This method assumes that the area has been reasonably well explored, so that the robot moves between frame regions known to be neighbors. This implies that each ply of the search is less plausible (as the search gets farther from the source), giving, as desired, a stream of candidate sets. On the other hand, the assumption of nearby frames giving correct candidates will not always be correct, particularly in the early stages of learning (however, more frames may be searched then, since there is less to search). On the other hand, the farther a frame is in the reference graph, the less useful information (in general) it gives for constraining the robot's position.

Perceptual Indexing The objective of perceptual indexing is to quickly find those stored percepts that are most similar to the current one. Furthermore, since the robot will never be in quite the same configuration twice, the indexing method should be robust with respect to small changes in position and orientation. We have developed an image-based method for waypoint recognition, using the notion of *image signatures*. An image signature is an array of values, each computed by a *measurement function* from a subset of the image (the image is tesselated). As demonstrated in [8], signatures can be matched to each other for fairly reliable recognition. The problem here is how to index this database so that similar signatures taken at different orientations will be found quickly. This can be done by indexing the signatures by their columns, since if two signatures are taken at different rotations, they will match at a horizontal offset. Hence, an input signature's columns are used to index the columns close to them, marking each signature found at the offset implied by the column match. When enough of a database signature's columns have been marked, the signature's waypoint is suggested as a match candidate.

Column indexing can easily be implemented as a k-d tree [18]; an iteratively growing hypercube search is used

[2] Jonathan Connell, personal communication.

to search progressively further and further from the input column. In this way, good candidate waypoints can be found based on perceptual cues, even if they are geometrically distant. A similar approach could also be taken using 3D estimates of the positions of visual features. Using the method of Atiya and Hager [2], feature triples can be represented by a set of 6 parameters describing a triangle. If triples of features going around the robot's viewpoint are stored thusly, a similar k-d tree approach can be used for indexing. Using this method, more sophisticated methods of 3D recognition and position registration can be done as well, given the required perceptual capabilities (eg, a system such as [21]).

4.2 Matching and updating

State identification in diktiometric mapping amounts to matching the robot's projected position and sensory inputs with candidate waypoints (generated as above). If we could guarantee that no mapping errors would ever occur, then the matching problem amounts to little more than filtering possibilities for consistency. However, as noted above, this is never the case, and so error diagnosis and correction must continually be performed. One way in which this is done is through the use of multiple matchers, each indicating a particular state of affairs, including mapping errors. Each matcher consists of a match test which compares the projected robot state with waypoints in a candidate set, and an update method which is applied to waypoints that are determined to match. Resolving between multiple applicable matchers is done by arranging them in a partial order; the maximal applicable matchers are used. This preference relation is constructed so that more reasonable decisions take precedence over less likely decisions (posit as few and as plausible errors as possible). If no matchers apply, a new waypoint node is created.

4.2.1 Matchers

There are four main matchers that we currently use. Two that correspond to normal extension of the map are CONTINUE and LINK. CONTINUE matches a waypoint that was expected with consistent position estimate and view; the waypoint's position estimate and view set are updated. LINK matches a waypoint with consistent position estimate and view which was unexpected, and so also adds a new action link to the path graph. Two other matchers correct for positional inconsistency caused by incorrect waypoint identification or odometric error. To deal with the case where a waypoint's position interval does not contain its true position (due to update with an outlier), the system keeps track of the waypoint's *nominal envelope*, the least bounding interval of the estimates used for updating the waypoint's position. This is asymptotically guaranteed to contain the true position. Thus, E-MATCH matches a waypoint such that the projected robot position is consistent with the waypoint's nominal envelope, indicating a possible waypoint position inconsistency. The waypoint's position estimate is the grown to contain the nominal envelope, presumably consistent. To deal with a track drifting from true, N-MATCH allows a match with a waypoint near the track's projected position; the track is then 'snapped' to the waypoint. A fifth, 'pseudo-match' is used for waypoint creation; when it is chosen to be used, a new waypoint is added with a track's projected state.

4.2.2 Dynamic priorities

A static priority scheme for matchers, while easy to implement, will seldom really be appropriate. For example, CONTINUE should only be preferred to LINK in well-explored areas, otherwise it is no better (since few links are known). We thus propose a dynamic scheme for prioritizing matchers, using estimates of the quality of the mapper's knowledge. This is done using local grid-based methods to maintain estimates of how well areas have been visited or traversed, as well as confidence in each individual track, based on its recent history. We can thus express preferences as the following:

- Prefer CONTINUE to LINK if the region just traversed has been well traversed (hence probably mapped) in the past.
- Prefer E-MATCH or waypoint creation to N-MATCH when a track has high confidence, and the converse when a track has low confidence.

In a similar fashion, other ideas of when particular types of matching are more appropriate can be expressed. This framework of a dynamic partial order controlled by meta-knowledge determined preferences seems both flexible enough to encode the required diagnostic knowledge, while providing a simple and modular mode of expression.

4.3 Transients

The first function of the restructurer in our system is to deal with map errors due to *transients*, non-existent states and transitions hallucinated due to nonreproducable conditions. Of course, if a hallucination is consistent, it may (usually) be considered real enough, as far as the robot is concerned. The only way to detect these transients, without taking over control of the robot, is to maintain statistics of when each link is thought to have been traversed and each waypoint is thought to have been visited. If these are compared with the frequency that the link or waypoint was expected, transients will be distinguished by a very low frequency of occurrence. The offending link/waypoint is then removed from the diktiometry. This also helps deal with (slowly) changing environments.

4.4 Waypoint restructuring

A deeper sort of error that cannot be discovered by using imprecise matching (hence requiring restructuring) is *structural inconsistency*. Incorrect waypoint identification can lead to either multiple waypoints in the world being represented by a single waypoint node (*polytopy*) or the converse, a single real-world waypoint represented by multiple nodes (*monotopy*). These sorts of errors can be dealt with by the system's restructurers. The concept behind these restructurers is that while one observation of a waypoint may not be sufficient to determine its identity, integration of many observations can yield statistically significant evidence of environmental structure. We deal below with two restructurers, one for *splitting* to deal with polytopy, and one for *merging* to deal with monotopy. To a great extent these two are inverses, and we apply similar methods for both, as summarized in Table 1. We divide the constraints applied into three categories:

Constraints	Splitting	Merging
Geometric	Multimodal distribution of position observations for a waypoint node	Unimodal distribution of position observations for two different waypoint nodes
Local	Separable correlated sets of input-output action link pairs	Nearly identical output link sets
Non-local	—	Multiple mergable track histories

Table 1: Constraints for waypoint restructuring.

geometric constraints on the positions of waypoints, *local path* constraints about local functional relationships in the path graph, and *non-local path* constraints applied to larger portions of the path graph.

Each of these restructurers can, in principle, operate independently. To integrate them, we propose to use a 'veto-based' strategy. Each restructuring method depends on certain thresholds, giving the desired confidence in a diagnosis of mono/polytopy. If we give each two thresholds, such that when the higher is satisfied we say that a diagnosis is *proposed*, and when the lower is not satisfied the diagnosis is *rejected*, the three strategies can be used in tandem as follows. If a diagnosis of either monotopy or polytopy is proposed by at least one restructurer, and is not rejected by any, we accept the diagnosis. This provides a sensible and modular integration of multiple constraints for diktiometry restructuring.

Geometric constraints The essential insight here is that by making the reasonable assumption that all waypoints are at least some minimum distance apart (call it δ_{sep}), we can discover when sets of position observations (from odometry) come from one or many waypoints. If then make the further weak statistical assumption that the distribution of position estimates for a waypoint is unimodal, we can use a multimodality test to test for structural inconsistency. If a single waypoint node represents two real waypoints, we would expect the distribution of position estimates matched to the node to be bimodal. Similarly, if two nodes correspond to one waypoint, their combined observation set should be unimodal. This will not always work, so we use other constraints as well.

Local path constraints The next sort of restructuring we consider uses a functional kind of constraint. The idea is that each waypoint has a consistent, if stochastic, input-output behavior (effects of performing actions). This being the case, polytopy is indicated by inconsistent effects at one waypoint, and monotopy by two waypoints with (nearly) identical effects. This is similar to Chrisman's approach to the closely related *perceptual aliasing problem* found in reinforcement learning [6]. Here, the merging method is simpler—if two waypoints have nearly identical incoming and outgoing action link sets (greater than a large fraction go to the same waypoint via equivalent actions), then the waypoints should be merged. The waypoints also should have been visited often enough to ensure that the known links are representative. Splitting requires examining how well incoming links predict outgoing links. If the incoming links can be partitioned into two sets such that the sets' 'images' (the sets of outgoing links taken after coming in on each link in a set) are (nearly) disjoint, then a split is indicated. This heuristic detects cases where a waypoint node does not adequately represent a functional state of the robot. By assumption, each waypoint corresponds to a single functional state (though the converse need not hold).

Non-local path constraints A third method uses non-local path constraints for determining restructuring. Currently, this only applies to merging. One problem with the local path approach to diagnosing monotopy is *merging deadlock*, where two different waypoints both need to be merged, but each inhibits the other being merged since local information does not suffice (the link sets are not identical). Hence what is needed is a sort of sub-graph isomorphism. Unfortunately, this is intractable, so we use a heuristic approximation. As the robot travels through the world, each track maintains a history of the waypoints it matched to. As well, each waypoint P in a history is associated with other waypoints that (a) were considered as matches but rejected, and (b) could be mergable with P (ie, their position estimates are consistent). An arbitrary length limit is placed upon histories to prevent them from growing without bound. When a track matches a waypoint, it deposits copies of its histories at the waypoint. When two waypoints have enough histories consistent with each other, the histories are used to 'sew up' a portion of the path graph. This may introduce spurious merges, but with conservative threshold choices, it can work in concert with the two methods described above to provide effective restructuring.

5 Opportunistic Exploration

Even though passive mapping, as described above, is important so that the robot can pursue its goals while learning, it can also be inefficient. This problem can be ameliorated somewhat, without sacrificing the benefits of passivity, by allowing the mapper to *advise* the deliberator of actions that the mapper would find useful. These can be treated by the deliberator as goals to satisfy when feasible; failure of a mapper goal does not invalidate a plan. So, if the robot decides it has an extra ten minutes before it has to deliver a package, it can take a short trip down a side corridor to see where it leads. The main point is that ultimate control lies inside the deliberator, which has the responsibility of balancing the utility of the various goals it must achieve.

We implement this idea by using *opportunity scripts*—short, stereotyped sequences of actions designed to help mapping in particular situations (an early version of this is described in [11]). These are suggested to the deliberator by an *opportunity checker*, which examines the current mapper state to determine if any scripts are applicable. Those which are, are sent to the deliberator where they may get executed. Even if they aren't, there is no great loss, since the mapping system's correctness does not depend on the actions the robot takes. There are two

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 waypoint visited; • If found, go there, otherwise exit; • Try to return, if possible.

RETRACE STEP

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

HEAD FOR UNCERTAINTY

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

HEAD FOR CERTAINTY

sorts of opportunities that can be dealt with in this way—exploration and experimentation.

There are two reasons to use opportunity scripts to aid 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 opportunity scripts to thus aid mapping; they are described below.

5.1 Exploration scripts

The essential idea of using opportunity 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.

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

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 waypoint it just came from; if it manages that, it tries to get back to where it started.

HEAD FOR UNCERTAINTY:

A more generally applicable heuristic for reducing map uncertainty is to simply head for nearby waypoints 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 waypoint is unambiguous.

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 waypoint whose position is known very precisely. If the robot reaches and recognizes the waypoint, the robot's position will then also be known more precisely, improving further mapping.

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 waypoints 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 waypoint has multiple action links coming from it labeled 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.

PROBE SPLITTAGE:

The kind of map ambiguity we consider for now is where two identical links from different waypoints end at the same waypoint and there is reason to believe that only one is real. This happens when a waypoint is split (see

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

DISAMBIGUATE TRACKS

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

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

above). Since both waypoints 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 waypoint 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 waypoints. This will accelerate elision of the invalid action links.

HEAD FOR UNEXPLORED AREA:

Most of the previous scripts have dealt with improving the system's knowledge of waypoints already in its map. Finding new, unknown waypoints 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 waypoint for a nearby area which looks unexplored, and if one is found, attempts to head in its direction.

HEAD FOR RARE WAYPOINTS:

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 waypoints which look likely to be transients. Since transients, by their very nature, are not encountered often, it is likely that a waypoint that has not been visited often is transient. If so, then repeatedly trying to reach the waypoint 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 waypoint.

5.2 Experimentation

The main type of experimentation that can be done in our framework delays diktiometry adaptation until more information has come in. Whenever a radical update or restructuring would normally be performed, a note is made of the operation to be performed, along with the information required before it can actually be performed and a set of exploration scripts to help gather that information. For example, before performing a merge, a script may be used to probe the distinctness of the two waypoints. This information is associated with the waypoint(s) involved, so that when the robot returns, the scripts are then suggested to the deliberator for execution. Again, as with exploration scripts, the particulars of the experimentation scripts used depend on the types of updating and restructuring done. We are currently working on developing a set of experimentation scripts for our system, but they have not yet been implemented.

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

6 Results

We present here the results of some experiments we have performed in simulation on the system described above. The simulator is described in [10]; space precludes a full discussion here. Briefly, waypoints are determined by configurations of walls and image signatures are simulated by noisy samples of wall 'color'. Wherever possible, worst-case assumptions were made with respect to sensor and effector noise; all such parameters are adjustable. The performance of the mapper was quantitatively evaluated by measuring *a posteriori* position error—the error inherent in allowing the robot to rely on the map to determine 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 useful performance metrics discussed in [9], but the different methods give qualitatively similar results—space does not permit inclusion here.

Figure 4(a) shows a typical small environment used for evaluation. The world was designed to be confusing; every waypoint looks the same as every other. For each run, the robot was controlled by an essentially random walk, while the mapper ran in the background. A move is defined as a sequence of actions ending with the robot in a distinguished waypoint (here, corners or doorways). Each run was 700 moves; a good maps were generally learned within 300. As Figure 4(b) shows, SSD position error starts out high, but quickly begins to converge to a low asymptote (non-zero due to inherent odometric error). This demonstrates the effectiveness of the system in a confusing environment, even with no mapper control over the robot.

The use of exploration scripts were tested by randomly executing them when applicable (generally 30% of the time). Comparative results are shown in Figure 4(c), which shows a significant improvement in mapper performance

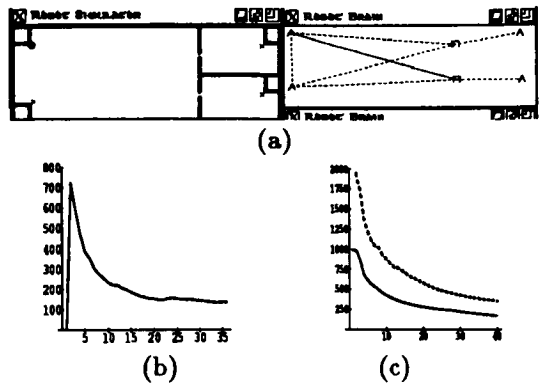

Figure 4: (a) A simple, but confusing, robot environment (left) and a typical map learned (right). (b) SSD position error (y-axis) vs. number of waypoint visits (x-axis), averaged over 10 runs. (c) Improvement (in another, larger, world) by occasionally using exploration scripts (solid) vs. pure random walk (dashed).

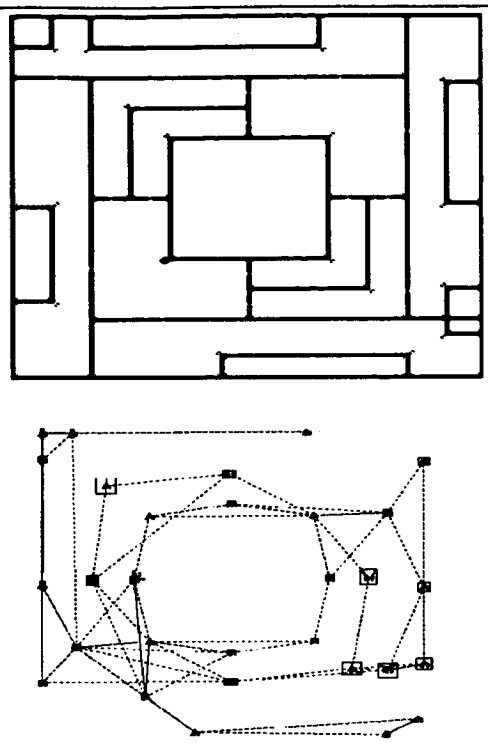

Figure 5: A complicated world and a learned map, learned after 3000 moves, with exploration scripts.

when exploration scripts are used. We expect a further improvement when we have taken into account the conflict resolution between different scripts; we are currently investigating how to compare scripts so that the most useful gets executed. This question is connected to the larger question of assigning utility to exploration and experimentation, which is important in terms of the deliberator's tradeoffs between goal-achievement and mapper script execution. This will form an important focus of our work in the near future.

7 Related Work

Much research on navigational mapping deals with problems of local metric representation (eg., [3; 7; 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 [14]. 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. Levitt *et al.* also utilize a topological map which avoids accumulation of navigation error, by using local reference frames based on landmarks [15]. However, their system appears to depend heavily on reliable landmark acquisition. Miller and Slack use information generated for reactive local navigation to build rough geometrical maps of rocky terrain [17]. 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]. 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. Hence they are forced to use strongly active strategies to learn maps reliably.

Our methods for dealing with mapping errors 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 [19] 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 [16] uses constraints derived from knowledge of the robot's underlying behavior to derive a topological map based on linear graph segments. Yeap [22] describes a hierarchical topological map, with place nodes described by 2D geometric models. Braunegg [5] develops a similar style of map, where rooms are characterised by the geometric arrangement of vertical edges, measured by stereo vision. Kriegman [12] 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.

8 Discussion

In this paper, we have shown how map-learning can be organized around the principle of passive mapping. Passive mapping involves two important components: error-tolerant representations and explicit error-correction strategies. In addition, exploration can be helpful, but should be optional. This can be implemented through the use of exploration scripts, as described above.

Our mapping system, as we have described, is a pas-

sive mapping system designed for indoor environments. In the future, we want to extend this work to other types of environments, such as city streets or forests. At present, though, this work is directly applicable to some real-world mapping tasks, such as those involved in inter-office delivery or some search-and-rescue tasks. Implementation of complete systems that could be deployed for such tasks is not yet feasible; it awaits other developments in effective perception and robust action.

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 *Proc. Int'l Conf. on Robotics and Automation*, 1991.

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

[6] Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proc. National Conference on Artificial Intelligence*, pages 183–188, 1992.

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

[8] Sean P. Engelson. Active place recognition using image signatures. In *Proceedings of SPIE Symposium on Intelligent Robotic Systems, Sensor Fusion V*, 1992.

[9] Sean P. Engelson. *Passive Map Learning for Mobile Robots.* PhD thesis, Department of Computer Science, Yale University, 1994. (forthcoming).

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

[11] Sean P. Engelson and Drew McDermott. Passive robot map building with exploration scripts. Technical Report YALEU/DCS/TR-898, Yale University Department of Computer Science, March 1992.

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

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

[14] Benjamin Kuipers and Yung-Tai Byun. A robust qualitative method for robot spatial reasoning. In *Proc. National Conference on Artificial Intelligence*, pages 774–779, 1988.

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

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

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

[18] Franco Preparata and Michael Ian Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 2nd edition, 1988.

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

[20] 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.

[21] C.J. Taylor and David Kriegman. Structure and motion from line segments in multiple images. In *Proc. Int'l Conf. on Robotics and Automation*, May 1992.

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