Maps Considered As Adaptive Planning Resources*

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

Most research on robot map-learning formulates the task as learning a complete map of the environment during a 'mapping phase', and then using the constructed representation for goal achievement. We suggest that this view of a mapper, as a system that outputs an essentially static representation, is misguided. Rather, a mapping system should be considered an ongoing resource for planning, but one which adapts to the environment it finds itself in. We have developed a framework for designing such adaptive models, in which we have designed a system for mobile-robot map learning. The world is modelled as a set of places with relative positions and known actions that get the robot from place to place. Adaptive modelling requires that the modelling system not need to control the robot, so we achieve passive mapping by correcting mapping errors after they occur. Even in this paradigm, the mapper can also give the planner advice on certain actions that would help with mapping; if the planner decides to execute them, mapping will be improved, but if not, no harm is done. We are currently developing this system in simulation; some results are presented.

Why Learn Maps?

Imagine that you have been hired as an office courier for a large corporation, their main campus consisting of a number of different office buildings. Your job is to deliver packages between offices. To do this task you must know how to get from office to office, however the company cannot afford to wait several weeks for you to walk all around and familiarize yourself completely with all the buildings. Even with no knowledge, it is possible to use general principles to get around, and over time you expect to learn the structure of this environment better and better. And if anything should change (office rearrangements, new buildings, etc.) some difficulties in navigation will become evident, but even without paying specific attention to learning about the changes, you will adapt and learn to get about as well (or better) than before. Our goal is to understand this process of learning about an environment ('map learning'), and to apply this understanding to the design of robotic systems.

There are several features of the vignette above that point to 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 in 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 changability. We therefore propose the view of a mapping subsystem as an 'adaptive resource' for planning, which supports planning in a standard way, but whose performance improves over time, transparent to the planner. This approach may be contrasted with much previous work 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, [15, 4, 7, 22, 3]). 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 problems in the real world, 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.

Some previous research has a similar flavor, though not in this particular formulation. By explicitly formulating the problem in terms of adaptive modelling, we believe that the issues underlying mapping become clearer, and similarities between different systems are

*This work was partially supported by the Defense Advanced Research Projects Agency, contract number DAAA15-87-K-0001, administered by the Ballistic Research Laboratory.

†Supported by the Fannie and John Hertz Foundation.

more evident; however such analysis is beyond the scope of this paper. Slack's work [19] on Navigation Templates (NaTs) directly addresses the usability of the representation constructed; NaTs are constructed incrementally during robot execution. Mataric's distributed world representation [16] is also closely related to its use in robot action; landmarks are detected through analysis of the robot's behavior. This means that the landmarks that are found are usually going to be those relevant to the robot's actions. Her use of an 'active representation' also presages our notion of an adaptive resource.

Adaptive Modelling

Our view of robot maps as adaptive planning resources leads us naturally to examine, in the abstract, the notion of adaptive world models. In a sense, we construct homeostatic representations which need not ever be 'correct' (in your favorite sense), but are (almost) always useful. In particular, we work within the context of adaptive models of discrete state spaces. The remainder of this section describes the high-level architecture we propose for constructing adaptive state-space models.

Adaptive State-Space Models

There are two fundamental operations for which a state-space model is used: state estimation and state prediction. Planning is done using these operations in conjuction 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 this view of the proper function of mapping, the mapping system becomes, in effect, an *adaptive observer*. Narendra and Annaswamy, in the context of control theory [17, p. 141], define an adaptive observer as one "which simultaneously estimates the parameters and state variables of a dynamical system..." For robot mapping, the parameters to be determined are those constituting the structure of the environment, while the state is the location of the robot in that environment. Our problem is somewhat more difficult than that of the control theorists, since we cannot even rely on an underlying linear space. However, we think it is possible to design adaptive models of discrete state spaces, given a good understanding of the underlying domain physics (in our case, a behavior-based mobile robot). 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.

Discretizing a Continuous State-Space

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. We adopt a technique based on Kuipers' topological mapping work [13, 14], which uses control laws with good stability properties (actions) to pick out distinguished 'places' in a continuous space. This allows a distinguished set of points in the continuous space (actually, small regions) to be used to represent the entirety of the space, relative to the different actions provided. As noted by Kuipers, this approach allows a representation to directly support navigational planning. Generically speaking, the state space is represented as a finite state machine with non-deterministic transitions (which can often be assigned transition probabilities).

The Architecture

Based on the ideas discussed above, we present an architecture for building systems constituting adaptive discrete state-space models. We first divide the system at a high level into an *estimator* and an *adapter* (refer to Figure 1). 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 approximate closed form estimators). Hence the need for 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.

The adapter consists of an updating module and a restructuring module. This distinction is analogous to that between 'parameter' and 'structural' adaptation made in the adaptive control theory literature (see, eg, [17]). The updater adjusts the parameters of states in the representation, for example, the position of a known point. It also can monotonically change the topology of the represented state space by, for example, adding new state transitions. The exact type of updat-

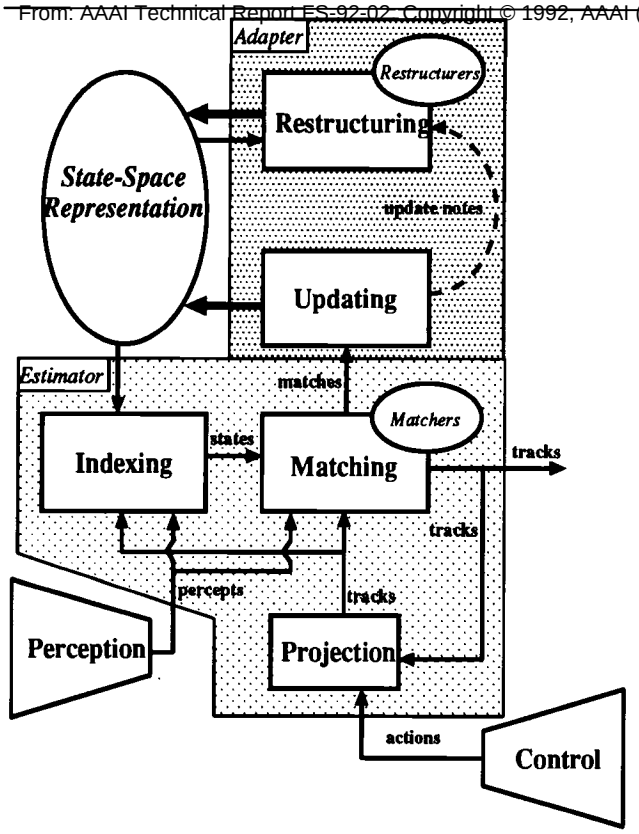

Figure 1: Adaptive state-space model architecture.

ing 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 be the trigger for a restructuring operation.

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 pioneered by Kuipers in [13]. The world is represented as a graph of places, labelled with local perceptual information, with transitions labelled with robot control routines, the actions taking the robot from one place to another. We extend this notion to directly take into account geometric knowledge as well. *Diktiometric*[1] representation explicitly represents geometric relations between places. A diktiometric representation (a *diktiometry*) consists of two graphs, a *path graph* and a *reference graph* (see Figure 2). Path graph nodes represent places, 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 places with respect to particular local reference frames.

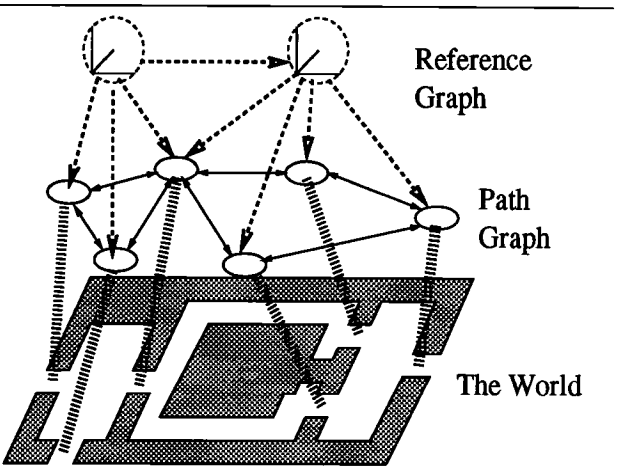

Figure 2: A schematic picture of a diktiometry, where doorways are taken as 'places'. Action links are solid, reference links are dashed.

Uncertain Geometry

We represent uncertain geometric relations between places 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 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

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

may incur extra costs). Rather than using such a statistical representation for odometric uncertainty, however, we advocate the use of intervals.

The primary reason that gaussian error 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 error model. However, when this model is not correct, the Kalman filter can be suboptimal and can even diverge [12]. 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 travelled[2]. This gives reasonably tight bounds on relative motion; use of sensory feedback (eg, visual motion analysis) may also help.

Putting It In Context

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.

Indexing places

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 places (at least). It is simply infeasible to examine the entire diktiometry for matching places 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 places. 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.

[2] Jonathan Connell, personal communication.

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 place, the expectational candidates are those which are predicted by the last action taken from that place. 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 place positions. The basic idea is to find places 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 places 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 places, the places 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. 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.

We propose correcting for this by maintaining a set of 'active' frames—those frames, anywhere in the map, whose relationship with current position estimates contains useful information. When the robot's uncertainty in its position estimate with respect to a frame is so high that no useful information is conveyed by the estimate, that frame should not be considered. The active frame set is maintained by dropping frames which are not informative, and adding neighbors of active frames which appear both informative and relevant. We heuristically define a frame F as being uninformative when (a) the robot's estimated position interval with respect to F contains the origin (ie, F), and (b) the center of the interval (the best point estimate we have) is not within F's envelope (the bounding interval of the position intervals of F's places). When these two conditions hold, uncertainty with respect to F is so great that the estimate implies that the robot both is and is not near F. This fact indicates that the estimate gives no useful information and so position-based indexing will not give useful information. A frame is made active when it is the neighbor of an active frame, is informative (ie, no uninformative) and relevant, i.e. the current position estimate intersects the frame's envelope. Now, by restricting our attention to only active frames, the search

can encompass more distant (in the reference graph) frames, and the quality of the places found will tend to be higher (since useless frames are elided).

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. The choice of indexing strategy thus strongly depends on the choice of perceptual representation. We have previously developed an image-based method for place 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 shown in [11], signatures can be matched to each other for fairly reliable recognition. The problem to address 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 place is suggested as a match candidate. This column indexing can easily be implemented as a k-d tree [18]; an iteratively growing hypercube search is used to search progressively further and further from the input column. In this way, good candidate places 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]).

Matching and updating

The state identification step for diktiometric mapping amounts to matching the robot's projected position and sensory inputs with candidate places (generated as above). If we were guaranteed 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 places in a candidate set, and an update method which is applied to places 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 small errors as possible). If no matchers apply, a new place is created.

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 place that was expected with consistent position estimate and view; the place's position estimate and view set are updated. LINK matches a place 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 place identification or odometric error. To deal with the case where a place's position interval does not contain its true position (due to update with an outlier), the system keeps track of the place's *nominal envelope*, the least bounding interval of the estimates used for updating the place's position. This is asymptotically guaranteed to contain the true position. Thus, α-MATCH matches a place such that the projected robot position is consistent with the place's nominal envelope, indicating a possible place position inconsistency. The place's position estimate is the grown to contain the nominal envelope, presumably consistent. To deal with a track drifting from true, β-MATCH allows a match with a place near the track's projected position; the track is then 'snapped' to the place. A fifth, 'pseudo-match' is used for place creation; when it is chosen to be used, a new place is added with a track's projected state.

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 α-MATCH or place creation to β-MATCH when a track has high confidence, and the converse when a track has low confidence.

In a similar fashion, such intuitive notions 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 way of expressing it.

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 place is thought to have been visited. If these are compared with the frequency that the link or place was expected, transients will be distinguished by a very low frequency of occurrence. The offending link/place is then removed from the diktiometry. This is also useful for dealing with slowly changing environments.

Place restructuring

A more fundamental type of error that cannot be discovered by using imprecise matching (hence requiring restructuring) is *structural inconsistency*. Incorrect place identification can lead to either multiple places in the world being represented by a single place node (*polytopy*) or the converse, a single real-world place 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 place 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: *geometric* constraints on the positions of places, *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 threshholds, giving the desired confidence in a diagnosis of mono/polytopy. If we give each two threshholds, 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.

Constraints	Splitting	Merging
Geometric	Multimodal distribution of position observations for a place node	Unimodal distribution of position observations for two different place 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 place restructuring.

Geometric constraints The essential insight here is that by making the reasonable assumption that all places 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 places. We do this incrementally by maintaining, for each place node, a set of *estims*, each a weighted point-estimate of the place's position (in its canonical reference frame). Each time an odometric interval is used to update the place's position estimate, the center of that interval is used as a point estimate to be added to the place's estim set. All estims within δ_{sep} of each other are merged, with new positions gotten by weighted averaging, and new weights the sum of the component's weights. Then, if the point estimates are drawn (with similar frequency) from two different distributions with modes δ_{sep} apart, that eventually the estim set will contain two estims with weight greater than a threshhold, and with weight ratio (max/min) less than a threshhold. So when this occurs, a split is indicated. The threshholds control the sensitivity of the process; the lower the first and higher the second, the more sensitive the process is, being quicker to split but increasing the chances of false positives. The two new places created by a split each take the sides of the original position estimate indicated by their respective estims. Action link sets are pruned eventually by the transient elision mechanism (see above). The dual use of estim sets for merging is similar—when two places have estim sets that can be merged such that the resultant set has one estim with weight greater than a threshhold and no other estim's weight is greater than a threshhold fraction of the first, a merge is indicated. A similar statistical argument supports this operation.

Local path constraints The next sort of restructuring we consider uses a functional kind of constraint. The idea is that each place has a consistent, if stochastic, input-output behavior (effects of performing ac-

tions). This being the case, polytopy is indicated by inconsistent effects at one place, and monotopy by two places 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 places have nearly identical incoming and outgoing action link sets (greater than a large fraction go to the same place via equivalent actions), then the places should be merged. The places 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 place node does not adequately represent a functional state of the robot. By assumption, each place corresponds to a single functional state (though the converse need not hold).

Non-local path constraints The 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 places 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 places it matched to. As well, each place P in a history is associated with other places 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 place, it deposits copies of its histories at the place. When two places 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 threshhold choices, it can work in concert with the two methods described above to provide effective restructuring.

Opportunistic Mapping

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 planner of actions that the mapper would find useful. These can be treated by the planner 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 planner, which has the responsibility of balancing the utility of the various goals it must achieve.

We propose to 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 [10]). These are suggested to the planner by an *opportunity checker*, which examines the current mapper state to determine if any scripts are applicable. Those which are, are sent to the planner where they may get executed. Even if they don't, there is no great loss, since the mapping system's correctness does not depend on the actions the robot takes. There are two sorts of opportunities that can be dealt with in this way—exploration and experimentation.

Exploration

Going where no robot has gone before The simplest form of exploration is done by simply going into uncharted territory. This can be done in our system by using the same meta-knowledge used to arbitrate between matchers to determine when the robot is near an area of the world it knows little about. The corresponding script simply sends the robot into the unknown area, saying in effect, "go west, young robot!"

Priming the pump The more interesting form of exploration comprises scripts which 'trick' the mapper into doing better, by maneuvering the robot in such a way that more useful information is gathered (through normal channels). For example, one script might say, "if you know your current position accurately and you are near a place with high positional uncertainty, go to that place," the reasoning being that by going there you improve your estimate of that place's position. The particular scripts that are used depend on the operation of the mapping system; we also think it may be possible to learn new scripts, since application of even a bad script doesn't hurt. We have developed a number of such scripts for our system, described in [10].

Experimentation

Delayed gratification 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 places. This information is associated with the place(s) involved, so that when the robot returns, the scripts are then suggested to the planner for execution. This sort of indexing of opportunistic goals is an instance of the general 'men-

tal notes' model of opportunistic behavior described by Birnbaum [5]. 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.

Results

We present here the results of some experiments we have performed in simulation on a preliminary version of the system described above. That system differs from the ideal system (currently being implemented) in the following particulars:

Indexing Perceptual indexing is not integrated into the system,

Matching A static priority system is used for matchers, rather than the full dynamic priority system,

Restructuring Only geometric constraints are used for splitting, and non-local path constraints for merging.

The simulator is described in [8]; space precludes a full discussion here. Briefly, places 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 3(a) shows a typical small environment used for evaluation. The world was designed to be confusing; every place 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 place (here, corners or doorways). Each run was 700 moves; a good maps were generally learned within 300. As Figure 3(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 3(c), which shows a significant improvement in mapper performance 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 planner's tradeoffs between goal-achievement and mapper script execution. This will form an important focus of our work in the near future.

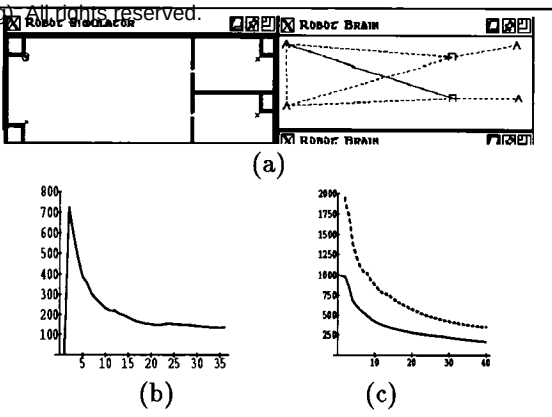

Figure 3: (a) A simple, but confusing, robot environment (left) and a typical map learned (right). (b) SSD position error (y-axis) vs. number of place visits (x-axis), averaged over 10 runs. (c) Improvement (in another, larger, world) by occasionally using exploration scripts (solid) vs. pure random walk (dashed).

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] Lawrence Birnbaum. *Integrated processing in planning and understanding.* PhD thesis, Yale University, 1986. Technical Report 489.

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

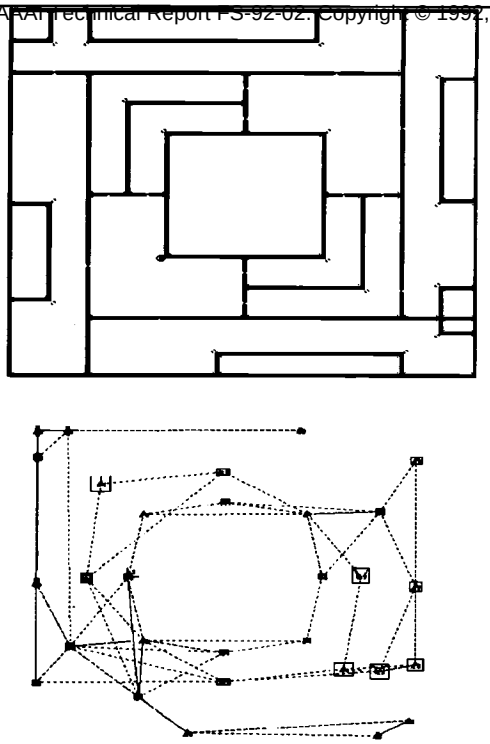

Figure 4: A complicated world and a learned map, learned after 3000 moves, with exploration scripts.

[7] Tom Dean, Dana Angluin, Kenneth Basye, Sean Engelson, Leslie Kaelbling, Evangelos Kokkevis, and Oded Maron. Inferring finite automata with stochastic output functions and an application to map learning. In *Proc. National Conference on Artificial Intelligence*, pages 208–214, 1992.

[8] Sean P. Engelson and Niklas Bertani. ARS MAGNA: The abstract robot simulator manual. Technical Report (forthcoming), Yale University Department of Computer Science, 1992.

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

[10] 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.

[11] Sean P. Engelson and Drew V. McDermott. Image signatures for place recognition and map construction. In *Proceedings of SPIE Symposium on Intelligent Robotic Systems, Sensor Fusion IV*, 1991.

[12] Robert Fitzgerald. Divergence of the Kalman filter. *IEEE Transactions on Automatic Control*, AC-16:736–747, December 1971.

[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] Kumpati S. Narendra and Anuradha M. Annaswamy. *Stable Adaptive Systems.* Information and System Sciences Series. Prentice Hall, Englewood Cliffs, NJ, 1989.

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

[19] Marc Slack. *Situationally Driven Local Navigation for Mobile Robots.* PhD thesis, Virginia Polytechnic Institute, 1990. NASA JPL Publication 90-17.

[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.