Passive Distance Learning for Robot Navigation

Sven Koenig **Reid G. Simmons**
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890
skoenig@cs.cmu.edu reids@cs.cmu.edu

Abstract

Autonomous mobile robots need good models of their environment, sensors and actuators to navigate reliably and efficiently. While this information can be supplied by humans, or learned from scratch through active exploration, such approaches are tedious and time-consuming. Our approach is to provide the robot with the topological and geometrical constraints that are easily obtainable by humans, and have the robot learn the rest while in the course of performing its tasks. We present GROW-BW, an unsupervised and passive distance learning algorithm that overcomes the problem that the robot can never be sure about its location if it is not allowed to reduce its uncertainty by asking a teacher or executing localization actions. Advantages of GROW-BW include that the robot can be used immediately to perform navigation tasks and improves its performance over time, focusing its attention to routes that are more relevant for its tasks. We demonstrate that GROW-BW can learn good distance, sensor, and actuator models with only a small amount of experience.

1 Introduction

We are interested in providing the technology for office or hospital delivery robots that are autonomous. Assume that you have just purchased such a delivery robot. Before it can be used, it must gain some knowledge of its new environment. This can be achieved by either providing the robot with the necessary information or letting it explore its environment autonomously. Both methods have disadvantages. Providing the robot with the necessary information suffers from the problem that some information is difficult or impossible to provide by humans. The sensor and actuator models of the robot, for example, depend not only on its environment, but also on characteristics of the robot itself, and one cannot expect consumers to be familiar with details of their newly purchased delivery robots. Other data could be provided by the consumers, but might be cumbersome to obtain. If they do not know the exact lengths of their corridors, for example, they have to measure them – a task that the robot could do itself. Letting the robot explore its environment autonomously, a method that many researchers have investigated [Kuipers and Byun, 1988] [Basye *et al.*, 1989] [Mataric, 1990] [Dean *et al.*, 1992], suffers from the problem that the robot cannot be used immediately and, during exploration, is likely to get into situations of confusion or danger that require human intervention, since it has no initial knowledge of its environment. We therefore suggest combining both methods: the robot is provided with some information that is easily available to humans, and it then autonomously learns the rest of the information needed for reliable navigation while in the process of performing its delivery tasks.

We start by supplying the robot with a topological map of its environment. A topological map specifies landmarks (such as corridor junctions) and how they connect. Such a map can easily be obtained from a sketch drawn by people familiar with the environment. Figure 2 (center and right), for example, shows a sketch of a corridor environment and the corresponding topological map. Once equipped with a topological map, the robot could use landmark-based navigation to perform delivery tasks. However, landmark-based navigation techniques suffer from the problem that imperfect sensors occasionally miss landmarks and even

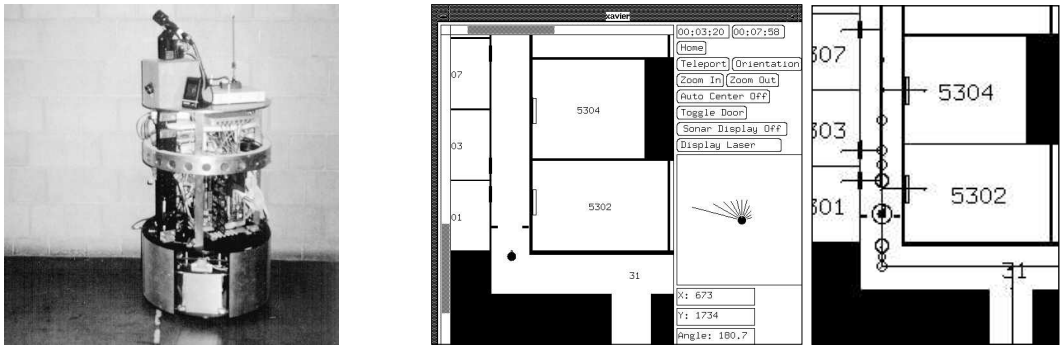

Figure 1: Xavier and two screen shots of its user interface

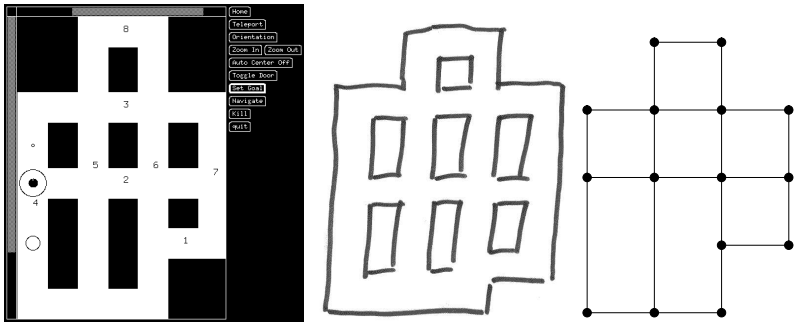

Figure 2: Corridor environment, sketch, and corresponding topological map

perfect sensors are not able to distinguish between all landmarks, such as corridor junctions of the same type (perceptual aliasing problem).

The reliability and efficiency of the robot can be improved by adapting its sensor and actuator models to its environment and, a simpler task for people, by providing it with distance information. However, people often err even with respect to distances – unless they measure them. Although the sketch of Figure 2 (center), for example, correctly specifies the topology, some of the arc lengths are incorrect. It is therefore much more reliable and convenient to let the robot learn the distance, sensor, and actuator models itself. We want the learning to be unsupervised (not to require a teacher during learning, after it has been supplied with the topological map) and passive (not to explicitly control the robot's actions). Unsupervised, passive distance learning is not a trivial task, because the robot can never be sure about its location: it has no distance information available initially, its sensors and actuators are noisy, and it cannot reduce the uncertainty about its location by asking a teacher or executing localization actions. In fact, its positional uncertainty may be quite significant. For example, Figures 1 (right) and 2 (left) show that after traveling some distance, the robot is unsure about its location (the sizes of the circles are proportional to the probability mass at each location). On the other hand, unsupervised, passive learning has the advantages that the robot can be used immediately to perform delivery tasks (since it has a topological map available) and it does not require a separate training phase or (ideally) any external help. In addition, the robot never stops learning: whenever it moves, it gains more and more experience with its environment which it continually uses to improve its distance, sensor, and actuator models and, as a consequence, also its navigation performance. Since it gains more information about routes that the robot traverses more often, learning focuses its attention to routes that are more relevant for the delivery tasks.

In the next several sections, we describe our algorithm for learning distances, sensor models, and actuator models in an indoor office environment. We conclude by presenting experimental results showing that the algorithm can learn good models with only a small amount of experience.

Our research is carried out on Xavier and its simulator (Figure 1). Xavier is built on an RWI B24 base and includes bump sensors, sonars, a laser range sensor,

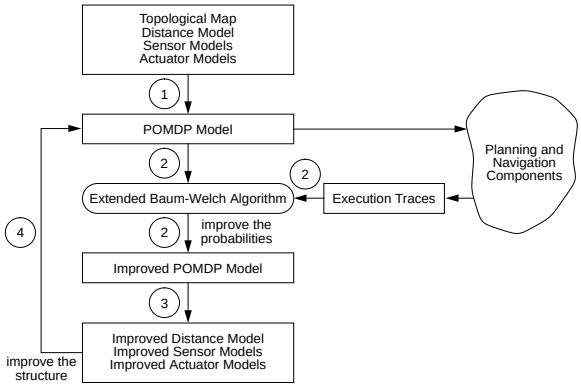

Figure 3: Overview of the GROW-BW algorithm

and a color camera on a pan-tilt head. Control, perception, and planning are all carried out on two onboard, multi-processing 486-based machines. Xavier roams the corridors of our building and can be controlled by users worldwide via its experimental World Wide Web interface, that allows them to specify goal locations and tasks that Xavier has to perform there. The interface can be reached via Xavier's homepage at http://www.cs.cmu.edu/∼Xavier. Eventually, Xavier will be used to deliver memos, letters, and printouts between the offices in our building.

2 Our Distance Learning Approach

We have developed GROW-BW, an unsupervised, passive distance learning algorithm that uses an extension of the Baum-Welch (BW) algorithm [Rabiner, 1986]. GROW-BW is an efficient algorithm that does not affect the other components of the robot system (except by making them operate more reliably) and can tune the initial ("factory programmed") sensor and actuator models to better match the environment of the robot while it learns the distances (despite the fact that GROW-BW never knows the ground truth about what the sensors were actually observing). Furthermore, it can take additional knowledge into account, if available, such as equality constraints between the lengths of corridors, bounds on the possible corridor lengths, or subjective probability distributions over them.

Instead of learning an exact corridor length, GROW-BW learns a probability distribution over the possible lengths, which is more robust to sensor and actuator noise. Formally, for each corridor segment c, GROW-BW learns a probability distribution p_c over the possible lengths of the corridor $l \in [l_{min}(c), l_{max}(c)]$, where $l_{min}(c)$ and $l_{max}(c)$ are the minimal and maximal bounds on the length of the corridor segment and where "length" refers to the perceived length of the corridor, which includes the dead-reckoning error of the robot. Then, $p_c(l)$ is the probability with which GROW-BW believes that the perceived length of corridor c is l.

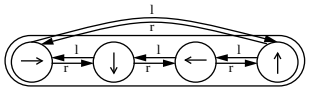

Figure 4: Four states representing one location

Figure 3 illustrates the GROW-BW algorithm. First, a topological map, augmented with sensor and actuator models and an initial distance model (e.g., a uniform distribution over the possible corridor lengths), is automatically compiled into a Partially Observable Markov Decision Process (POMDP) model (①). This model is used directly by our probabilistic planning [Koenig *et al.*, 1995] and navigation methods [Simmons and Koenig, 1995] to direct the robot to a given goal location. Better distance, sensor, and actuator models improve the navigation performance of the robot. The robot therefore improves its models from experience using an extension of the Baum-Welch algorithm (②). The experience is given in form of sequences of action and sensor reports (execution traces) that are generated automatically whenever the robot moves. The resulting POMDP has less distance uncertainty and improved sensor and actuator models (③). Finally, it may be the case that $l_{max}(c) \gg l_{min}(c)$. To avoid having to consider all possible lengths initially (or in cases where the given bounds do not actually include the real length), we use a hill-climbing technique that iteratively changes the structure of the POMDP based on the results of the extended Baum-Welch algorithm (④). It starts with a small bound $l_{max}(c)$ and grows it if necessary until there is a high probability that the real corridor length is contained within the bounds.

3 The POMDP Model

POMDPs are popular models for optimal decision making in uncertain conditions [Cassandra *et al.*, 1994] [Parr and Russell, 1995]. Our POMDP incorporates the distance uncertainty and the sensor and actuator models of the robot. It is specified as a finite set of states S, a set of actions $VA(s) \subseteq VA$, for

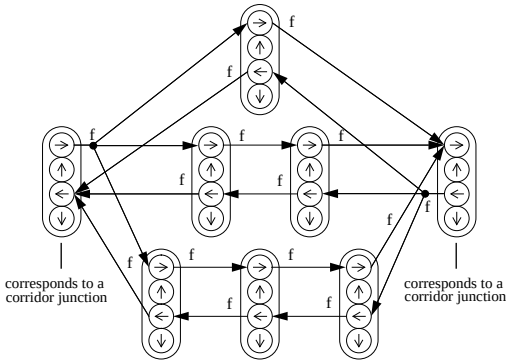

Figure 5: Corridor of length 2 to 4 meters

each state $s \in S$, that can be executed in that state, transition probabilities $p(s'|s, va)$ for all $s, s' \in S$ and $va \in VA(s)$ (the probability that the successor state is s' if the robot executes action va in state s), and sensor probabilities $p_{vs}(f|s)$ for all $vs \in VS$, $f \in F(vs)$, and $s \in S$ (the probability that sensor vs reports feature f when the robot is in state s). Each state encodes both the location and orientation of the robot. We discretize locations with a resolution of one meter and orientations into the four compass directions (this assumes that corridors are straight and perpendicular to each other). Right and left turn actions are defined for every state (Figure 4). Forward actions transition from location to location, but are not defined for states that face walls. All actions are nearly deterministic, but there is a small chance that the robot ends up in any of the three unintended orientations (not shown in the figures).

The POMDP is compiled automatically from a topological map. The corridor part between two adjacent junctions in the topological map is modeled as sets of parallel chains that share their first and last states (Figure 5). Each chain corresponds to one of the possible lengths $l \in [l_{min}(c), l_{max}(c)]$ for that stretch of corridor c. From each junction, forward actions have probabilistic outcomes according to the probabilities $p_c(l)$. Each forward transition after that is (nearly) deterministic. Thus, our POMDP model explicitly models distance uncertainty and differs in this respect from a similar model by [Nourbakhsh *et al.*, 1995], that does not model distances at all. It can therefore be quite large; the sizes of our POMDPs are typically on the order of thousands of states. It is possible, however, to reduce the number of states required to model a corridor c from being quadratic in $l_{max}(c) - l_{min}(c)$ to being linear in $l_{max}(c)$, at the cost of a loss in model accuracy [Simmons and Koenig, 1995].

4 The Baum-Welch Algorithm

The Baum-Welch algorithm [Rabiner, 1986] is a simple expectation maximization (EM) algorithm for learning POMDPs from observations. It is best known for its application to speech recognition and handwriting recognition, but it has also been applied in robotics, for example to interpret tele-operation commands [Hannaford and Lee, 1991; Yang *et al.*, 1993]. In the following, we describe how we use the Baum-Welch algorithm to improve the initial POMDP.

Whenever the robot moves, a sensor interpretation module converts its continuous motion into discrete action reports and produces reports of high-level features from the raw sensor data. In the case of Xavier, for example, the sensor interpretation module integrates data from the wheel encoders over time to produce a stream of discrete action reports (going forward one meter, turning left ninety degrees, and turning right ninety degrees). Similarly, sonar readings are bundled into three "virtual sensors" that report observations of walls and openings of various sizes (small, medium, and large) in front of Xavier and to its immediate left and right. An execution trace contains these action and sensor reports in chronological order.

We use the Baum-Welch algorithm to estimate a POMDP that better fits the given execution traces, in the sense that the probability with which the POMDP explains the sensor reports (given the action reports) is increased. The Baum-Welch algorithm operates as follows: It first uses the given POMDP and all information contained in the execution traces to calculate, for every point in time, a probability distribution over all states that represents the belief that the robot was in a certain state at a certain point in time. It then estimates an improved POMDP from these probability distributions, using a maximum likelihood approach. This estimation process is then repeated with the same execution traces and the improved POMDP until some termination criterion is satisfied. The run time of each iteration of the Baum-Welch algorithm is linear in the product of the total length of the given execution trace and the size of the POMDP, typically being on the order of seconds to minutes for our application.

We have extended the Baum-Welch algorithm to address memory constraints and the problem that collecting training data is time consuming:

- The Baum-Welch algorithm has to run on-board

the robot and shares its memory with many other processes that run concurrently. To decrease the amount of memory that it requires, we use a sliding "time window" on the execution trace. Time windows add a small overhead to the run time and cause a small loss in precision of the improved POMDP, but allow the memory requirements to be dynamically scaled to the available memory.

- Given the relatively slow speed with which mobile robots can move, we also want the Baum-Welch algorithm to learn good models with as few corridor traversals as possible. To reduce the amount of training data that it needs to estimate good models, we extended the learning algorithm to take advantage of available prior knowledge, such as geometrical constraints that can be deduced from the topological map. One might know, for example, that two corridors are the same length, because both are intersected orthogonally by the same pair of corridors. This decreases the number of parameters that have to be learned and therefore the amount of training data needed to prevent overfitting.

The original Baum-Welch algorithm uses frequency-based estimates, but these are not very reliable when the execution traces are short. To understand why, consider the following analogy: If a fair coin is flipped once and comes up head, the frequency-based estimate is that it always comes up head. If this model were used to predict future coin flips, one would be very surprised if the coin came up tails next time – this would be inconsistent with the learned model. Our extended Baum-Welch algorithm solves this problem by using Bayes' rule (Dirichlet distributions) instead of frequencies. For more details and an empirical evaluation of the extended Baum-Welch algorithm, see [Koenig and Simmons, 1996].

5 The GROW-BW Algorithm

The Baum-Welch algorithm improves the probabilities of a POMDP, but never changes its structure (the number of states and their connectivity). This poses a problem, because the distance model is partly encoded in the structure of the POMDP: the possible lengths of a corridor are determined by the structure, while the probability distribution over the possible lengths is determined by the probabilities. Consequently, the Baum-Welch algorithm cannot assign a positive probability $p_c(l)$ to corridor lengths $l \notin [l_{min}(c), l_{max}(c)]$ nor can it change the bounds. Thus, it cannot learn the real corridor length if the bounds are off – but they might not be known. Guessing $l_{min}(c)$ is easy: we can use the smallest positive length according to our discretization granularity. Guessing $l_{max}(c)$ is harder: we could, of course, guess a ridiculously large value, but this has the drawback that the POMDPs become very large – and the memory requirements of distance learning algorithms determine their tractability. Instead, we investigate learning algorithms that are able to change the structure of the POMDP.

Alternatives to the Baum-Welch algorithm for learning POMDPs are described by [Chrisman, 1992], [Stolcke and Omohundro, 1993], and [McCallum, 1995], among others. These algorithms are able to change the structure of a POMDP, but have the disadvantage that they either require a large amount of training data, learn task-specific representations only, or cannot utilize prior knowledge. Consequently, we have designed a novel POMDP learning algorithm that we call GROW-BW. GROW-BW achieves its power by utilizing the regularities in the structure of our POMDP models of the corridors. It takes advantage of the fact that the Baum-Welch algorithm learns a good POMDP for the given structure, even if the structure is incorrect. This allows it to start with a small POMDP, learn the best model for that structure, see if the model is "good enough," and grow the model if not.

Initially, GROW-BW guesses a small upper bound $l_{max}(c)$ on the real corridor length (Figure 6). It then compiles a POMDP and uses the extended Baum-Welch algorithm to improve it. If the Baum-Welch algorithm indicates that it is likely that the real corridor length is close to the upper bound, GROW-BW increases the upper bound, adds a new parallel chain to the corridor segment (Figure 5), and repeats the procedure. In this way, if the initially chosen upper bound was too small, it can be increased to fit the real length of the corridor.

GROW-BW is a hill-climbing algorithm and, thus, can suffer from myopic effects. Consider the most myopic version of GROW-BW, that uses the parameter values $X = Y = Z = 0$ (X is related to the initial difference between $l_{min}(c)$ and $l_{max}(c)$, Y is related to the ranges of lengths to consider when determining whether the model is "good enough," and Z is related to how much to grow the model at each step). To simplify our argument, assume that a robot with (almost) perfect sensors and actuators moves back and forth in the environment shown in Figure 7(A). If $l_{max}(c) = 4$ for all

The GROW-BW algorithm uses the following parameters: $X = 0, 1, 2, \ldots$; $Y = 0, 1, 2, \ldots, X$; $Z = 0, 1, 2, \ldots$, and $P \in (0, 1)$. In its simplest form, it uses $X = Y = Z = 0$ and a small positive value for P.

1. For each corridor c: set $l_{max}(c) := l_{min}(c) + X + 1$. (If a lower bound $l_{min}(c)$ on the real corridor length is not known, use $l_{min}(c) = 1$.)
2. Compile a POMDP (see Section 3).
3. Use the extended Baum-Welch algorithm on the POMDP and the given execution traces to determine improved $p_c(l)$ for all corridors c and corridor lengths l with $l_{min}(c) \leq l \leq l_{max}(c)$ (see Section 4).
4. For each corridor c: if $\sum_{l \geq l_{max}(c)-Y} p_c(l) \geq P$, then set $l_{max}(c) := l_{max}(c) + Z + 1$.
5. If any $l_{max}(c)$ was changed in Step 4, then go to Step 2, else stop.

Figure 6: The GROW-BW algorithm

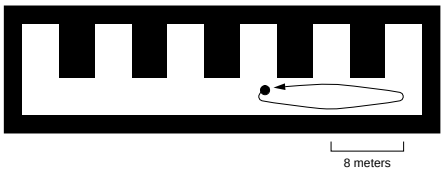

Figure 7: Example of myopic effects

corridor pieces, then the best fitting model is the one where all traversed corridor pieces are four meters long. (The robot expects to see a corridor opening every four meters, but sees them only every eight meters. Thus, it cannot explain four observations on each round-trip, and no distance model whose corridors are at most four meters long can do better.) This leads GROW-BW to increase $l_{max}(c)$ to five for all traversed corridor segments. However, at this point the model where all corridor segments are four meters long is still among the models that, of all models considered, explain the observations best (another such model is the one where adjacent corridor pieces of the main corridor alternate between lengths three and five). If the Baum-Welch algorithm learns this model, then GROW-BW stops without having learned the real corridor lengths.

Note that we have constructed this example artificially – the problem does not show up if the robot encounters both ends of the main corridor while it moves forward

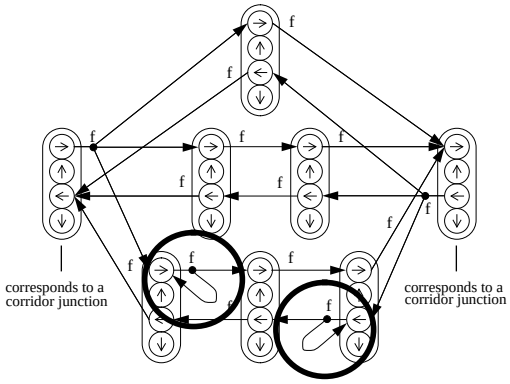

Figure 8: Corridor with self-transitions

and backward. Despite the theoretical limitations of hill-climbing, our experience with GROW-BW shows that it appears to work well in practice. We attribute this to architectural features of buildings – they are usually constructed in a way that prevents people from getting lost, which appears to dampen myopic effects. However, it is possible that a problem similar to the one described could show up in conjunction with office doors along a corridor. We therefore recommend using a less myopic version of GROW-BW by setting the parameters X, Y, and possibly Z to values that are larger than the typical distance between adjacent office doors. Similarly, P (the probability threshold that triggers growing the model) has to be chosen small enough to prevent GROW-BW from terminating prematurely.

Other problems arise when the real corridor length is greater than $l_{max}(c)$, since then the execution traces can be inconsistent with the POMDP, in the sense that the model cannot explain the experience. One problem this might lead to is that the position estimation component of the navigation system may rule out all possible locations, leading the robot to become totally uncertain as to where it is. The robot then has to explicitly relocalize itself, which may take a fair amount of time. Another problem is that learning can no longer take place. As an example, again consider the environment shown in Figure 7 and assume that the robot traverses the main corridor from beginning to end for a total distance of 40 meters. This, however, is impossible according to a model that assumes $l_{max}(c) = 4$ for all corridor pieces. We avoid both these problems by having the POMDP compiler add self-transitions (with a small probability Q) in both directions of the longest chain in the POMDP representation of each corridor segment (Figure 8). In this way, all corridor lengths l

with $l_{min}(c) \leq l$ have positive probability. This does not mean, of course, that the GROW-BW algorithm is no longer needed. Using such a POMDP directly with the Baum-Welch algorithm would not work very well if $l_{real}(c) > l_{max}(c)$, because only the probabilities $p_c(l)$ for $l_{min}(c) \leq l < l_{max}(c)$ can be specified individually. The probabilities for $l_{max}(c) \leq l$ are exponentially decreasing according to the following formula:

$$p_c(l) = \left(1 - \sum_{l_{min}(c) \leq l' < l_{max}(c)} p_c(l')\right) (1 - Q)\ Q^{l - l_{max}(c)}.$$

6 Experiments

We use the prototypical corridor environment shown in Figure 9(A) to illustrate the power of our learning algorithms. Remember that they discretize the possible corridor lengths with a precision of one meter. To match this assumption, all corridor lengths in this environment are multiples of one meter. In many ways, the environment is more complicated than what we have available in our building. It has many parallel corridors and indistinguishable junctions, which amplifies the perceptual aliasing problem. The experiment uses the real-time Xavier simulator, a highly realistic simulation of Xavier including noisy sensors and actuators, that has the exact same interface as Xavier itself, but allows us to make the experiments repeatable. It is *not* based on the POMDP model used for navigation and consequently violates the independence assumptions made by POMDP models (just like reality). The learning algorithms can be used unchanged on Xavier itself. In this case, the execution traces are provided by Xavier instead of the simulator.

We do not inform the robot about its start location or orientation, its route, or its destination. Instead, we let it gain experience with the environment by guiding it through every corridor once, using two execution traces with different start locations. The only information that it has available is the topological map, the data from its sensors, and the following obvious equality constraints between corridor lengths: (These constraints are not necessary for the learning algorithms, but they increase the quality of the learned models if the number of corridor traversals is small [Koenig and Simmons, 1996].)

$$l_{real}(c_1) = l_{real}(c_3) = l_{real}(c_6) = l_{real}(c_{10})$$
$$l_{real}(c_2) = l_{real}(c_5) = l_{real}(c_9)$$
$$l_{real}(c_4) = l_{real}(c_7) = l_{real}(c_8)$$
$$l_{real}(c_{11}) = l_{real}(c_{13})$$
$$l_{real}(c_{12}) = l_{real}(c_{14}) = l_{real}(c_{18}) = l_{real}(c_{21})$$
$$l_{real}(c_{15}) = l_{real}(c_{19})$$
$$l_{real}(c_{17}) = l_{real}(c_{20})$$

Given this information, the task of the robot is to annotate the topological map with distance information and to adapt its initial sensor and actuator models to its environment. This learning task is particularly hard, since we assume that the robot does not even know its approximate start location or orientation. As a consequence, several different routes can be consistent with the sensor data, especially since the robot has noisy sensors and actuators and has no initial estimates of the corridor lengths available. For example, the probability that the left and right virtual sensors overlook a corridor junction is about fifty percent. This relatively high probability is due to the sensors being quite conservative: they don't report features until they have collected sufficient evidence. Also, since the virtual sensors are implemented as asynchronous processes, they sometimes do not report features in time.

Our first experiment uses the extended Baum-Welch algorithm directly. To make sure that it is able to learn the real corridor lengths, we estimate the minimal and maximal corridor lengths cautiously to guarantee that $l_{real}(c) \in [l_{min}(c), l_{max}(c)]$: we use $l_{min}(c) = 2$ meters and $l_{max}(c) = 14$ meters for every corridor piece c. The resulting POMDP has 6672 states and 80346 state transitions. Figure 9(B) depicts the corridor lengths with the largest probability $p_c(l)$ in the learned model: all 21 predicted corridor lengths correspond to the real corridor lengths.

Our second experiment uses GROW-BW with the parameters $X = 0$, $Y = 0$, $Z = 1$, $P = Q = 0.05$, and $l_{min}(c) = 2$ meters for all corridor segments. That is, the initial estimate for every corridor segment is $l_{max}(c) = 3$ meters and, if GROW-BW extends a corridor length, it increases it by two meters. GROW-BW assumes a uniform probability distribution over the possible corridor lengths. Given this information, GROW-BW needs only four iterations to converge. Figure 9(C) shows $l_{min}(c)$ and $l_{max}(c)$ for the final model. The corresponding POMDP has only 1176 states and 16260 state transitions, and is thus much smaller than the POMDP from our first experiment (the POMDPs used in the first three iterations of GROW-BW are, of course, even smaller). This is the case, because GROW-BW stops to expand the upper

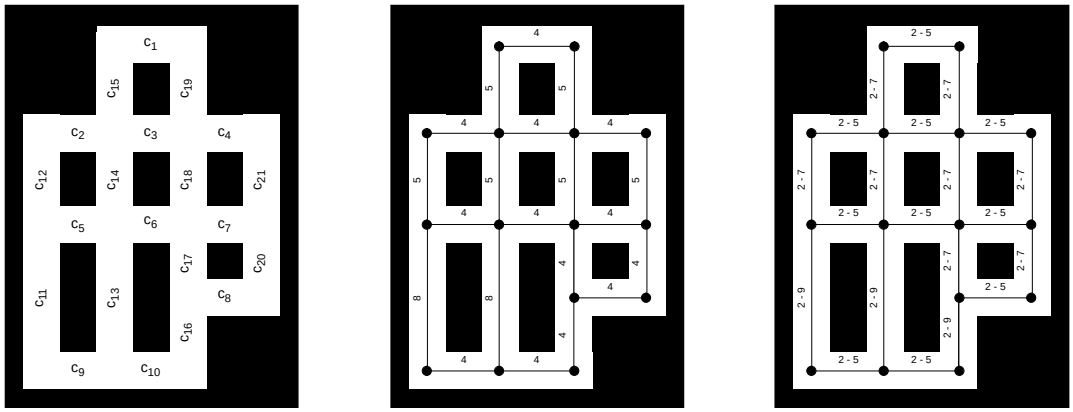

Figure 9: Experimental results

bound of a corridor length when it is sufficiently sure that it is larger than the real corridor length (where the required amount of certainty is determined by the parameters Y and P). Because of the small sizes of the POMDPs, GROW-BW is 1.84 times faster than the extended Baum-Welch algorithm, although it has to call the extended Baum-Welch algorithm repeatedly. The probabilities $p_c(l)$ that GROW-BW learns are similar to those learned in the first experiment, and the corridor lengths with the largest probability $p_c(l)$ are even identical: again, all corridor lengths are learned correctly.

We repeated both experiments eight more times with different robot routes. The results are summarized in Table 1. Each corridor length that, after learning, does not have the largest probability among all possible lengths counts as one mistake in the column "corridors." If several corridor lengths were constrained to be identical, we count only one mistake per corridor group in the column "groups."

The POMDPs learned by GROW-BW were of the same quality as the ones of the extended Baum-Welch algorithm: The learned sensors and actuator models were similar when we evaluated them according to a) how much they reduced the positional uncertainty of the robot and b) how much they increased the probability with which the POMDP could generate (or, synonymously, explain) long simulator execution traces. Furthermore, both algorithms learned good (although not perfect) distance models with only one traversal of each corridor: in all cases, they erred by only one meter when they made a mistake. In general, they can learn good distance models with one to three corridor traversals, depending on how confusing the corridor environment is. Note that, although the dead-reckoning error of our robot is not overly large, we cannot expect the learning algorithms to learn all corridor lengths perfectly, because – for example – the robot sometimes takes sharp and sometimes wide turns around corners which affects the distances traveled along the corridors.

The experiments show that the sizes of the POMDPs produced by GROW-BW are roughly between four and six times smaller than the size of the POMDP that we used in conjunction with the extended Baum-Welch algorithm. As a result, GROW-BW is almost two times faster than the extended Baum-Welch algorithm.[1] Thus, GROW-BW produces results similar to those of the extended Baum-Welch algorithm, but works on much smaller POMDPs and therefore needs less memory and often less run time. The effect is even more pronounced when the models of our building are used, since they are much larger than the model used here. We could augment GROW-BW with a post-processing step that prunes the final POMDP, thus making it even smaller.

The corridor environment used in this example was extremely small and thus one could have used distance learning methods with a runtime that is exponential in the total length of the execution traces, such as methods that match the routes probabilistically against the topological map (possibly combined with branch-and-

[1]The seventh experiment in Table 1 is an exception. It contained a highly ambiguous execution trace and GROW-BW expanded the upper bound of one corridor up to a length of 21(!) meters, which required 10 iterations. We could not replicate this phenomenon when we used execution traces that traversed each corridor more than once.

Table 1: Comparison of GROW-BW with the extended Baum-Welch algorithm

	mistakes of ext. Baum-Welch		mistakes of GROW-BW		improvement in the number of states	improvement in run time
	corridors (out of 21)	groups (out of 8)	corridors (out of 21)	groups (out of 8)		
1	0	0	0	0	4.79×	1.80×
2	0	0	0	0	4.46×	1.76×
3	0	0	0	0	5.67×	1.67×
4	5	2	5	2	5.67×	2.08×
5	5	2	5	2	5.20×	1.99×
6	0	0	0	0	5.20×	1.97×
7	3	2	3	2	3.66×	0.53×
8	0	0	0	0	5.20×	1.78×

bound methods to prune the search space). GROW-BW has two advantages over such methods: First, the model that it learns (a POMDP) can directly be used by our probabilistic planning and navigation methods. Thus, there is no need for a model transformation that might degrade the quality of the model. Second (and more importantly), the run-time of GROW-BW is only linear in the length of the execution trace. We have also used GROW-BW to learn environments in which the successful execution of actions does not provide any information about the position of the robot, namely for learning the distances between adjacent office doors and corridors in a long hallway that is traversed by a robot that does not know its starting position.

7 Extensions

We have assumed that GROW-BW can be provided with a correct topological map. Although this is a realistic assumption for many robot learning scenarios, weakening it broadens the application area of our algorithm. Consequently, we are working on extending GROW-BW to be able to correct slightly inaccurate topological maps. We are also investigating whether it can be combined with the passive topological map learning approach by [Engelson and McDermott, 1992] to extend its applicability to scenarios where a qualitative map is not available at all.

8 Conclusion

In this paper, we have described GROW-BW, a distance learning algorithm that annotates a given topological map with distance information. GROW-BW uses an extension of the Baum-Welch algorithm as a subroutine. It is an unsupervised (does not require a teacher during learning) and passive (does not need to control the robot at any time) learning method. GROW-BW overcomes the problem that the robot can never be sure about its location if it is not allowed to reduce its uncertainty by asking a teacher or executing localization actions. It has the advantage that the robot can be used immediately to perform navigation tasks, and autonomously improves its performance over time as it gains more experience with its environment, focusing its attention to routes that are more relevant for its tasks. It works transparently with the other components of the robot system, can adapt the factory programmed sensor and actuator models to the environment of the robot while it learns the distances, and is efficient. It uses sliding "time windows" to minimize the amount of memory required, and as much or as little additional knowledge as is available to minimize the amount of experience required to learn good models. It can utilize, for example, equality constraints on the lengths of two corridors, bounds on the possible corridor lengths, or subjective probability distributions over them. We demonstrated that GROW-BW can learn good distance models with only a small amount of experience, often with considerably less space and time than can the extended Baum-Welch algorithm, by itself.

In conclusion, GROW-BW learns quantitative information that is difficult to obtain from humans (distances as well as sensor and actuator models), but is able to utilize a large variety of qualitative (and quantitative) information that humans can easily provide. In contrast, many other map learning approaches in the literature attempt to learn maps from scratch, not utilizing prior knowledge that is easily available.

Acknowledgements

Thanks to Lonnie Chrisman, Richard Goodwin, Joseph O'Sullivan, and the rest of the Xavier group for

helpful discussions on a variety of topics. This research was sponsored by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations or the U.S. government.

References

(Basye *et al.*, 1989) Basye, K.; Dean, T.; and Vitter, J.S. 1989. Coping with uncertainty in map learning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 663–668.

(Cassandra *et al.*, 1994) Cassandra, A.R.; Kaelbling, L.P.; and Littman, M.L. 1994. Acting optimally in partially observable stochastic domains. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. 1023–1028.

(Chrisman, 1992) Chrisman, L. 1992. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. 183–188.

(Dean *et al.*, 1992) Dean, T.; Angluin, D.; Basye, K.; Engelson, S.; Kaelbling, L.; Kokkevis, E.; and Maron, O. 1992. Inferring finite automata with stochastic output functions and an application to map learning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. 208–214.

(Engelson and McDermott, 1992) Engelson, S.P. and McDermott, D.V. 1992. Error correction in mobile robot map learning. In *Proceedings of the IEEE International Conference on Robotics and Automation*. 2555 – 2560.

(Hannaford and Lee, 1991) Hannaford, B. and Lee, P. 1991. Hidden Markov model analysis of force/torque information in telemanipulation. *The International Journal of Robotics Research* 10(5):528–539.

(Koenig and Simmons, 1996) Koenig, S. and Simmons, R.G. 1996. Unsupervised learning of probabilistic models for robot navigation. In *Proceedings of the International Conference on Robotics and Automation*.

(Koenig *et al.*, 1995) Koenig, S.; Goodwin, R.; and Simmons, R.G. 1995. Robot navigation with Markov models: A framework for path planning and learning with limited computational resources. In *International Workshop on Reasoning with Uncertainty in Robotics*.

(Kuipers and Byun, 1988) Kuipers, B.J. and Byun, Y.-T. 1988. A robust, qualitative method for robot spatial learning. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. 774–779.

(Mataric, 1990) Mataric, M.J. 1990. Environment learning using a distributed representation. In *Proceedings of the IEEE International Conference on Robotics and Automation*. 402–406.

(McCallum, 1995) McCallum, R.A. 1995. Instance-based state identification for reinforcement learning. In *Advances in Neural Information Processing Systems 7*.

(Nourbakhsh *et al.*, 1995) Nourbakhsh, I.; Powers, R.; and Birchfield, S. 1995. Dervish: An office-navigating robot. *AI Magazine* 16(2):53–60.

(Parr and Russell, 1995) Parr, R. and Russell, S. 1995. Approximating optimal policies for partially observable stochastic domains. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 1088–1094.

(Rabiner, 1986) Rabiner, L.R. 1986. An introduction to hidden Markov models. *IEEE ASSP Magazine* 4–16.

(Simmons and Koenig, 1995) Simmons, R. and Koenig, S. 1995. Probabilistic robot navigation in partially observable environments. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 1080–1087.

(Stolcke and Omohundro, 1993) Stolcke, A. and Omohundro, S. 1993. Hidden Markov model induction by Bayesian model merging. In *Advances in Neural Information Processing Systems 5*. 11–18.

(Yang *et al.*, 1993) Yang, J.; Xu, Y.; and Chen, C.S. 1993. Hidden Markov model approach to skill learning and its application to telerobotics. In *Proceedings of the IEEE International Conference on Robotics and Automation*. 396–402.