A Decision-Theoretic Approach to Planning, Perception and Control

Kenneth Basye Thomas Dean * Jak Kirman Moises Lejter
Department of Computer Science
Brown University, Box 1910, Providence, RI 02912
tld@cs.brown.edu (401) 863-7645

February 15, 1993

Abstract

We present an approach to building planning and control systems that integrates sensor fusion, prediction, and sequential decision making. The approach is based on Bayesian decision theory, and involves encoding the underlying planning and control problem in terms of a compact probabilistic model for which evaluation is well understood. We view planning in terms of enumerating a set of possible courses of action, evaluating the consequences of those courses of action, and selecting a course of action whose consequences maximize a particular performance (or *value*) function. In most planning problems in robotics, the results of many actions serve to increase knowledge, potentially improving the ability to make decisions. In our approach, the planning system explicitly takes into account the value of information gained through sensing and the value of movement that facilitates sensing.

*This work was supported in part by a National Science Foundation Presidential Young Investigator Award IRI-8957601, by the Advanced Research Projects Agency of the Department of Defense monitored by the Air Force under Contract No. F30602-91-C-0041, and by the National Science Foundation in conjunction with the Advanced Research Projects Agency of the Department of Defense under Contract No. IRI-8905436.

1 Introduction

A large class of problems in intelligent control can be characterized as sequential decision problems. These problems involve a sequence of operations of the form *sense, evaluate, act.* In the first stage, a robot or other control system[1] gathers data from sensors about the state of its world. Next, the robot must evaluate possible courses of action in light of its goals and the information it has available. Finally, the robot acts and the cycle begins again. The challenge for any intelligent control system arises in the first two stages. In the first, combining the readings of various different sensors into a coherent and accurate picture of the world is often difficult. This is the problem of *sensor fusion.* In the second stage, the difficulty is that evaluating various courses of action open to the robot typically requires the prediction of the consequences of actions in order to determine their suitability for the current situation.

The sort of prediction mentioned above is common in planning systems in Artificial Intelligence, and these systems are based on a well-developed approach to sequential decision problems. Many of these systems, however, assume that they are provided with an accurate, deterministic model of the world, including initial conditions. This has allowed these systems to ignore the

[1] In the discussion that follows, we will use the term "robot" exclusively; however, the techniques we describe are appropriate for other types of control systems as well.

sensing component of the sequence above and concentrate almost exclusively on improving the search strategies used for the evaluation component. Thus, these systems cope poorly (if at all) with uncertainty, and generally do not consider the value of information gained by sensing in deciding which actions to perform. Recently, *reactive* systems have emerged as an alternative to traditional planning approaches for robotic systems. These systems achieve their fast reaction time by compiling plans into rules that specify the action to take for any sensory input. This approach might involve compilation of the result of some planner, or simply an encoding of action-response rules chosen by the designer. This type of system can cope with situations in which the model of the world was inaccurate, or the result of executing actions was unexpected, but it can be very expensive in terms of the time spent compiling behaviors off-line, and in terms of space required to store the results of these computations.

In our work on planning, we have adopted a decision-theoretic approach to intelligent control systems. We encode a planning and control problem in a Bayesian network[?],[?], a probabilistic model of the aspects of the world relevant to the robot's objective. These aspects fall into three categories, related to the three parts of the sequence above. There are actions the robot can take, observations it can make about the world, and states of the world that have

some significance for prediction and evaluation. The building blocks of our model are random variables whose distributions range over possible actions, observations, and states, and probabilistic relations between these variables. To capture the temporal aspects of control problems, we use sequences of these variables to represent probabilities at successive time points. A full model can thus capture relations such as those between observations and states at a single time point, or those between an action at one time and states at later times. Our approach builds on standard techniques in optimal control, by employing a compact and easily specified model for representing the underlying stochastic process.

After the choice of relevant variables for the model is made, we must specify both the spaces for these variables and the probabilistic relationships between them. The connectivity of the network completely determines the dependency relations between random variables. Dependencies between variables are specified as conditional probability distributions in a process called *quantification*. We must also specify a method for evaluating the model to determine posterior probabilities for variables of interest, as well as a value function to be used to select the best action at each time point, based on some control objective. Operation of such a system is straightforward; evidence from sensing is incorporated into the network, the effect of different possible actions is estimated,

the value function is computed using the posterior probabilities given by the evaluation, and the action with the highest value is executed. In this paper we illustrate this approach and examine some of the issues that arise in the realization of such systems. We also report on an implementation of a controller for a task for mobile robots using this system.

We begin by explaining the decision-theoretic approach in more detail and pointing out its advantages and disadvantages. We describe a particular robot navigation problem and illustrate how our approach can be used to solve this problem in a simple manner. We then examine the kinds of abstraction required to make the resulting decision model computationally feasible. In Section ?? we consider the process of quantifying models and point out some design issues related to quantification. In Section ?? we report on the results of our initial experiments. Finally, in Section ??, we discuss related work and draw some conclusions from our experiences.

2 Stochastic Decision Models

We have adopted Bayesian decision theory [?] as a framework for designing high-level robotic control systems. This approach has a number of advantages. First, it provides a convenient basis for dealing with decision-making under uncertainty. Second, the approach naturally accounts for knowledge gained as

a result of actions. From a decision-theoretic perspective, there is no difference between actions that involve sensing, movement to facilitate sensing, and any other actions; a decision maker simply tries to choose actions that maximize expected value. Since additional information improves the system's ability to make decisions which increase its expected value, control systems based on decision theory have a natural tendency toward *active sensing*, the selection of actions which increase their knowledge. Having committed to a decision-theoretic approach, there are specific issues that we have to deal with: the most difficult are representing the problem, obtaining the necessary statistics to quantify the underlying decision model, and ensuring that the resulting model can be evaluated in a reasonable amount of time. (See Section **??** for a more detailed discussion of this.)

We encode our decision models as a *Bayesian network* [**?**]. A Bayesian network is a directed graph $G = (V, E)$. The vertices in V correspond to random variables and are often called *chance nodes*. The edges in E represent the causal and informational dependencies between the random variables. In the model described in this paper, chance nodes correspond to discrete-valued variables that encode states of knowledge about the world. Let Ω_C denote the set of possible values (the *state space*) of a chance node C. There is a probability distribution $\Pr(C = \omega, \omega \in \Omega_C)$ for each node. If the chance

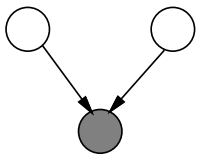

Figure 1: Simple Bayesian network

node has no predecessors then this is its marginal probability distribution; otherwise, it is a conditional probability distribution dependent on the states of the immediate predecessors of C in G.

The example in Figure **??** shows three nodes, representing the location of a robot (L_R), the location of a target (L_T), and the observations made about the target by the robot (O_T). The arcs indicate that the observations made are dependent both on the location of the target and on the location of the robot. In order to specify the network fully, we provide a conditional probability distribution for each node. In this case we provide prior distributions for the location of the robot and the target, and a conditional distribution for the node O_T: for each set of possible values the parent nodes might take, we specify the probability of each of the values O_T can take.

Our model involves a specialization of Bayesian networks called *temporal*

belief networks [?]. The temporal belief networks discussed in this paper are distinguished by the following (Markov) property: all the information necessary to predict any given world state is contained in the description of the world at the previous time; no information about earlier times is necessary. Topologically this corresponds to having no arcs that span more than one time-step. The methods we use do not require this property; however, the Markov property reduces the complexity of evaluating the networks. These temporal belief networks provide the basis for high-level robotic control by allowing us to encode the set of possible worlds determined by the robot's actions. We use the networks, coupled with a value function that provides a measure of the goodness of a projected world, to control our robots.

The idea is to represent all the information relevant to the control task at hand using nodes in the network — some of these will correspond to information available to the robot, such as its current state and sensory information; others will correspond to possible actions, or to random events outside the control of the system. For each node, a set of discrete elements representing all possible states for that node must be specified. Once we determine the set of nodes to be represented, we replicate it for the number of time points to be considered and add the appropriate arcs between nodes.

For practical applications, the system designer has to exercise critical judg-

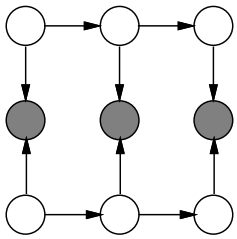

Figure 2: Temporal belief network

ment in deciding what to put into the decision model and what to leave out, since too little information will not allow the decision model to make good choices, while too much information will require more computation than is practicable. In some cases, the tradeoffs made by the designer in coping with computational complexity can themselves be quantified by decision-theoretic criteria [?].

Figure ?? shows a temporal network representing the three aspects of the world from Figure ?? replicated for three time steps. The arcs between time slices represent the fact that the location of the robot and target depend on their locations at the previous time step. Implicitly represented in this network are conditional independences, such as the fact that the location of the target does not depend on the location of the target two time-steps earlier except in so far as it depends on its location at the previous time step.

Arcs are added between nodes in the network to represent a causal or informational relation between the variables represented by the nodes. The

relationship is quantified using joint conditional probabilities. An arc may connect two nodes in the set of nodes corresponding to a single point in time (a time "slice"), or two nodes in consecutive time slices.

Once the model is built, using it to determine what action to take next is straightforward: given sense data, distributions over the states of the corresponding evidence nodes are fixed. Sequences of actions can then be considered by assigning values to the action nodes, evaluating the network to determine the resulting distribution over the nodes of interest, and then using a value function to compute the expected value of each alternative. The alternative with the best expected value is then executed. This approach corresponds approximately to the receding horizon open-loop control techniques used in stochastic control; the formulation of the model using Bayesian networks, however, allows us to exploit implicit conditional independence, thus simplifying the specification of the joint probability distribution and the task of computing a posterior distribution given some evidence.

In the next sections, we illustrate this idea by constructing the network and value function for a particular robot control problem. Some of the difficulties inherent in constructing and evaluating these networks will then be presented. The application that we have chosen to illustrate our approach involves a mobile robot navigating and tracking a moving target in a hallway

environment. The robot is provided with sonar and rudimentary vision. The moving target could be a person or another mobile robot. The robot's task is to track the target, reporting its location in the coordinate system of a global map. The environment consists of one floor of an office building. The robot is supplied with a coarse floor plan of the building showing the position of walls and doorways.

We assume that there is error in the robot's movement, requiring it to continually estimate its position with respect to the floor plan so as not to become lost; we refer to this as the *localization* problem. Localization and tracking are frequently at odds with one another. A particular localization strategy may reduce position errors while making tracking difficult, or improve tracking while losing registration with the global map; in particular, since it is often the case that different locations look alike to the robot's sensors, the robot has a better idea of its position if it is in a location with distinctive characteristics. It typically has a better idea of the target's location when it is near the target. If the target moves into an area in which the robot has difficulty differentiating locations, such as a long corridor, there is a tradeoff between staying in a distinctive area and following the target closely down the corridor. The trick is to balance the demands of localization against the demands of tracking. The mobile target localization (MTL) problem is particularly appropriate for plan-

ning research as it requires considerable complexity in terms of temporal and spatial representation, and involves time pressure and uncertainty in sensing and action.

2.1 A Model for Mobile Target Localization

For the MTL problem, we are interested in the positions of the tracking robot and its target, any possible sensor information available to the robot, and the actions available to the robot; these aspects will constitute our model of the world. Let L_R and L_T be variables ranging over the possible locations of the robot and the target, respectively. Let A_R be a variable ranging over the actions available to the robot. At any given point in time, the robot can make observations regarding its position with respect to nearby walls and corners and the target's position with respect to the robot. Let O_R and O_T be variables ranging over these observations. This example is typical of mobile robotic applications; the aspects of interest are actions, observations and states of the internal representation of the world. Therefore the graphs will contain action nodes, sensor nodes and state nodes. As a notational convenience, we will refer to nodes by the names of their associated variables.

To specify the set of elements (the state space) over which our spatial variables L_R and L_T will range, we quantize the physical space in which the

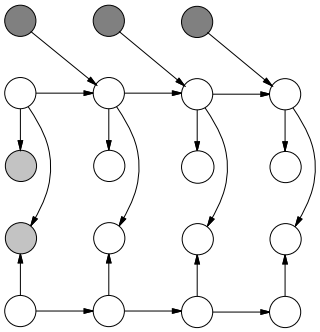

Figure 3: Probabilistic model for the MTL problem

robot and its target are embedded. We use a coarse floor plan and a model for the robot's sensors to generate the state spaces for L_R and L_T according to some representation scheme. (We will say more about this process and about representation in general in the next section.) A_R will be quantized over a set of primitive actions available to the robot, while O_R and O_T will be quantized over the set of possible sensor readings regarding the robot's own position and the set of possible readings regarding the position of the target relative to the robot.

Figure **??** shows a temporal belief network built with the variables mentioned above for planning up to four time steps into the future.[2] The nodes in

[2]To quantify the model shown in Figure **??**, we have to provide distributions for each of the variables conditioned on the values of its predecessors. We refer to a node X at time i as $X(i)$. For each i after the first, the arcs into $L_R(i)$ represent that the location of the robot depends on the action taken at the previous time step, $A_R(i-1)$, and the location of the robot at that time step, $L_R(i-1)$. The conditional probability table for $L_R(i)$ encodes our model of the effects of actions in this world. The arc into $O_R(i)$ specifies that the observations the robot will make about its whereabouts depend on its location at that time. The arcs into $O_T(i)$ specify that the observations the robot makes about the location of

the first time slice which have no parents are quantified by marginal probabilities; the others are quantified by conditional probability tables as described by the arcs.

One particular point is designated as the current time, *Now*[3] By convention, the nodes corresponding to observations indicate observations *completed* at the associated time point, and nodes corresponding to actions are meant to indicate actions *initiated* at the associated time point. The actions of the robot and the current observations of the robot serve as evidence to provide conditioning events for computing posterior distributions. The dark nodes in Figure ?? correspond to an action sequence whose effects we are estimating; the lighter shaded nodes correspond to evidence observed.

To update the model as time passes, all of the nodes are shifted into the past, discarding the oldest nodes in the process. As new actions are initiated and observations are made, the appropriate nodes are instantiated as conditioning (evidence) nodes. When the system is first instantiated, the prior probabilities for the nodes representing the states of the robot and its target can be set according to the information initially known about both robot and

the target depend on the location of the robot at that time and the location of the target (the observations are relative, e.g. near, far, not visible, etc). The conditional probability tables for $O_R(i)$ and $O_T(i)$ encode our models of the sensors. Finally the arc into $L_T(i)$ from $L_T(i-1)$ specifies that the location of the target depends on its previous location; the tables for all the nodes $L_T(i)$ encode the model of the target's movement

[3]To reduce our computational requirements, we summarize all information about previous states of the world in the single time slice corresponding to the present.

target. When information is to be shifted to the past, the new prior probabilities for the leftmost nodes in the network are taken from the posterior probabilities for the next leftmost nodes in the network; these represent the robot's belief about the next state for those nodes. The previous information in the leftmost nodes is discarded[4].

In order to choose the best action to perform, we need a measure of the value of a situation. Ideally, the way we would compute this value function is this: We postulate an action sequence for the robot, and we consider what the target might do. Given the actions of the robot and the target, we could construct and evaluate the resulting situation. We could then do this for all possible actions the target might take, and weight our estimates of the value of each situation by how likely it is to occur. However, since we cannot directly measure what the target does, we use our observations as a proxy for this measurement. That is, for each action of the robot, we consider the possible *observations* we might make about the target, evaluate the situation corresponding to that observation, and weight the computed value by the

[4]In the example in Figure ??, updating the model is done as follows: $A_R(1)$ is instantiated with the action selected. The action is performed by the robot, and evidence is gathered from the sensors. This evidence is used to instantiate the nodes $O_R(1)$ and $O_T(1)$. The beliefs in the network are then updated to take this information into account. This computes the new posteriors for $L_R(1)$ and $L_T(1)$, representing the current estimates of the robot's location and the target's location. Finally, the nodes in the network are shifted back, and the process repeats itself: the posteriors at $L_R(1)$ and $L_T(1)$ become the priors at $L_R(0)$ and $L_T(0)$, and the observations $O_R(1)$ and $O_T(1)$ become the observations $O_R(0)$ and $O_T(0)$.

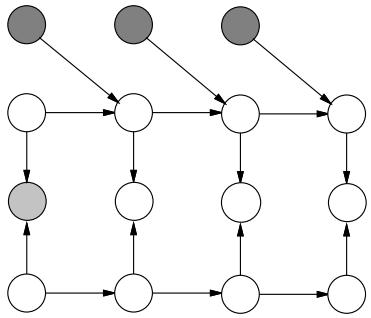

Figure 4: Simplified MTL network

likelihood of making that observation. This evaluation is reminiscent of the backing up of evaluations in dynamic programming approaches to stochastic decision problems.

To make the rest of this description simpler, we use an equivalent but topologically simpler network that combines all observations at a given time point into one node whose state space is just the cross product space of the two observation nodes, $\Omega_O = \Omega_{O_T} \times \Omega_{O_R}$ (see Figure **??**.)

To capture the notion of the value of the situation in which the robot has made some sequence of actions and made some observations, we use the following function:

$$EV_t(o|s) = - \min_{u \in \Omega_{L_T}} \sum_{v \in \Omega_{L_T}} \Pr(L_T(t) = v | O(t) = o, s) \mathrm{Dist}(u, v), \qquad (1)$$

where $o \in \Omega_O$ and s is an action sequence.[5] $\mathrm{Dist}(u, v)$ is the distance between locations u and v. One can think of u as ranging over guesses at where the target is; the sum gives us a measure of the likely error when making a particular guess — the higher the probability that the target is in some other location v and the further away v is from u, the worse the value. We are only interested in how good our *best* guess is, so we take the minimum of these sums. The value is negated since it is more intuitive to think of a high value as "good".

We then define the value function capturing the expected value of the information obtained as a result of executing s as

$$EV_t(s) = \sum_{o \in \Omega_O} \Pr(O(t) = o|s) EV_t(o|s). \tag{2}$$

To get the time-separable[6] value function EV, we use Equation **??** to measure the expected value of performing the sequence at each of the next n steps in time. If so desired, we may also discount the future consequences by using a weighted sum of the values at the different time points: $EV(s) = \sum_{t \in \mathcal{T}} \gamma(t) EV_t(s)$. The function γ specifies how much to weight future expected values.

[5] $\mathcal{X}(t) = x$ can be read as "node $\mathcal{X}$ has value x at time t".

[6] A value function for a temporal network is *time-separable* if it can be expressed as a weighted sum of the values of each of the time-slices, where the value of a time-slice is computed independently of the other time-slices.

The typical information processing sequence involves first obtaining information about the world from the robot's sensors and using this to instantiate the probability distributions of the sensor nodes in the network. (Some of the state nodes — those representing information about the current state of the robot — already have probability distributions.) Then we instantiate a sequence of action nodes (a possible plan), evaluate the network, and compute a value measure for that action sequence. We repeat this for different possible sequences of actions, and choose the sequence with the best expected value. Next the first action in this sequence is executed, new information is gathered from the sensors, and the process is repeated. The time required to compute the expected value of a given action sequence is dependent on the size of the state spaces for the random variables in the network and the length of the action sequence.

The approach described in this section allows us to integrate prediction, observation, and control in a single model. It also allows us to handle uncertainty in sensing, movement, and modeling. Behavioral properties emerge as a consequence of the probabilistic model and the value function provided, not as a consequence of explicitly programming specific behaviors, as discussed in Section ??.

2.2 Coping with Complexity

If a naïve approach is taken to designing systems using this control mechanism, the computation necessary to choose an action to perform becomes prohibitive in all but the most trivial of cases. To reduce the cost of evaluating the MTL decision model, one can use any of the following methods: (1) tailor the spatial representation to the robot's sensory capabilities, reducing the size of the state space for the spatial variables in the decision model, (2) enable the robot to dynamically narrow the range of the spatial variables using heuristics, thus further reducing the size of the state space for the spatial variables, and (3) consider only a few candidate action sequences, either selected from a fixed library of tracking strategies by taking into account the reduced state space of the spatial variables, or by dynamically pruning the space of possible action sequences[7].

We can restrict the range of particular spatial variables on the basis of evidence not explicitly accounted for in the decision model (e.g., odometry and compass information). For instance, if we know that the robot is in one of two locations at time 1 and the robot can move at most a single location during a given time step, then $L_R(1)$ ranges over the two locations, and, for $i > 1$, $L_R(i)$ need only range over the locations in or adjacent to those in

[7] A description of these techniques for reducing complexity can be found in [?]

$L_R(i-1)$. Similar restrictions can be obtained for L_T. For models with limited lookahead (i.e., small $|\mathcal{T}|$), these restrictions can result in significant computational savings.

A method for reducing the cost of decision making involves the use of *branch and bound* techniques on the space of action sequences; the fact that our value function is time-separable allows us to develop incremental algorithms to determine what the optimal action sequence for any particular situation would be. These algorithms can be designed to evaluate sequences of increasing length (up to the system's horizon) while pruning action sequences whose partial computed value can no longer beat alternative action sequences already considered. The time-separability of the value function also makes it possible to use dynamic programming techniques to compute an optimal policy (see [?]). It should be noted, however, that for many problems the computational complexity of these dynamic programming methods makes it impossible to compute the optimal policy off-line.

2.3 Emergent Behavior

Emergent behaviors, i.e., behaviors that are not explicitly programmed into the robot, but that emerge as a consequence of the model, are typical of what is to be expected from systems controlled by decision-theoretic criteria. In

decision-theoretic approaches to control, the designer provides the necessary expectations, actions, and utilities, and the emergent behavior of the resulting system is just that which maximizes utility given the evidence.

In our implementation of the MTL problem (see Section **??**) we have observed behaviors that we did not design, or indeed anticipate. For example, if the target is moving towards a fork, the robot will try to keep close behind it, as this will allow it to determine which branch the target takes. However, when the target moves toward a cul-de-sac, the robot keeps fairly far away from it, whereas we expected it to remain close behind. Analyzing the probability distributions and results of the value function, we discovered that the model allowed for the possibility that the target might slip behind the robot, leaving the robot unable to determine the location of the target without additional actions. If the robot stays some distance away, whatever action the target takes, the observations made by the robot are sufficient to determine the target's location. Figure **??** shows this behavior: in the top diagram, the robot is close to the target, and is considering the possible observations it might make at the next time step. The observation "target not visible" would not give the robot a good idea of the location of the target, since the two possible locations shown are equally likely. The position the robot is in is therefore not highly valued. The lower diagram shows the robot slightly

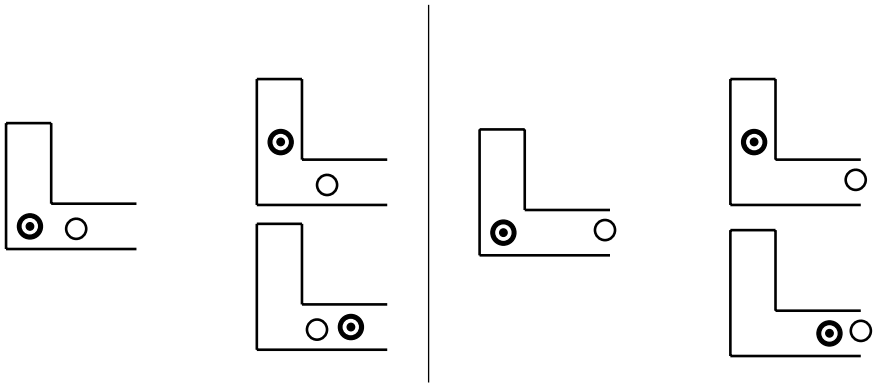

Figure 5: Being close to the target may not be optimal

further away from the target. Now, whichever observation it makes, it will be able to determine the location of the target.[8]

The MTL problem has a very simple model, and so we do not expect to see any complex behaviors exhibited by the robot; the desired behaviors could easily be crafted by hand. In more complex environments, with value functions and world models that are correspondingly more complex, designing the appropriate behaviors to solve a problem become very much more difficult, while decision-theoretic approaches require only that we represent the problem.

3 Abstractions

As we point out in the previous section, the sizes of the state spaces must be small, to allow the kind of decision model we have chosen to evaluate its

[8] the model of the target determines its maximum speed; in this case, the target could not move fast enough to get behind the robot in one time-step.

network in an amount of time consonant with real-time behavior. Since each state space represents an aspect of the world, reducing the size of a state space corresponds to forming an *abstraction*, in which different states of the world are mapped onto the same state in the state space. In this section, we discuss abstractions of three aspects of sequential decision problems: states, observations, and actions; we also present examples of the abstractions we use to solve the Mobile Target Localization problem.

3.1 State Abstractions

State abstractions are used to represent an aspect of the world not directly measurable using sensors. In the MTL problem, for example, the location of the robot is not measured directly, but is inferred from the current sensor inputs and an estimate of the previous location. The location of the target is even more indirectly inferred from the inputs.

It is often easy to decide which aspects of the world need to be monitored or inferred in order to solve the problem at hand. More difficult, however, is deciding how to quantize these aspects. The number of possible values the random variable represented by a state node can take will have a strong influence on the computation time required to evaluate the network. So if the granularity is too fine, the computation is likely to be too lengthy to

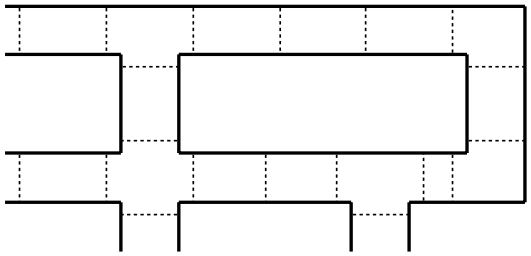

Figure 6: Tessellation of corridor layout

allow the robot to make a decision before the world changes, invalidating its observations. If the granularity is too coarse, however, the network will not contain enough information to solve the problem adequately.

In general, state nodes are dependent on sensor nodes, previous state nodes, and action nodes. Ultimately, information about the state comes from the sensors, so in most cases the granularity of the state node should be commensurate with the granularity of the sensor nodes to which it is related. Thus if our sensors can tell us that we are in a corridor, a T-junction, an L-junction, etc, it is unlikely that having a node whose values can represent the location of the robot in a 1 cm grid would be useful. On the other hand, having the sensors provide much more information than the state nodes need is wasteful.

Our choice of spatial representation for the MTL problem was influenced by the kinds of actions we expected to use, the sensors available to our robot, and the task at hand. The sensors are able to distinguish between corridors, T-junctions, corners, and so on, but they are not able to provide us with any

more precise information reliably. In terms of the goal to be solved (maintain an estimate of the location of the robot with respect to the map provided), a division of the world into corridors and junctions suffices. Figure ?? shows a portion of the world tessellated according to this scheme. We use these regions as the state space for the location of the target; for the state space for the location of the robot, we augment the regions with four quadrants to represent the directions the robot can face.

3.2 Sensor Abstractions

Since raw sensor readings typically cover a large space, and the state space of the sensor nodes must be small for the network to be evaluable in a reasonable amount of time, it is necessary to provide a mapping from the real sensor space to the observation space. Our robot can obtain raw sensor readings from 8 sonar transducers, configured in pairs pointing forward, backward, and to each side of the robot. Each sonar gives a reading in millimeters, between 30 and 6000 (where 6000 means 6m or more). Figure ?? shows one set of readings obtained from this configuration of sonars on entering a T-junction. Our choice of abstract sensor was induced directly by the spatial representation we chose; we wanted a sensor to tell us what kind of region we were in.[9] We

[9] We use two different types of sensors for the MTL problem: sonars for observations about the location of the robot, and simple vision for observations about the location of

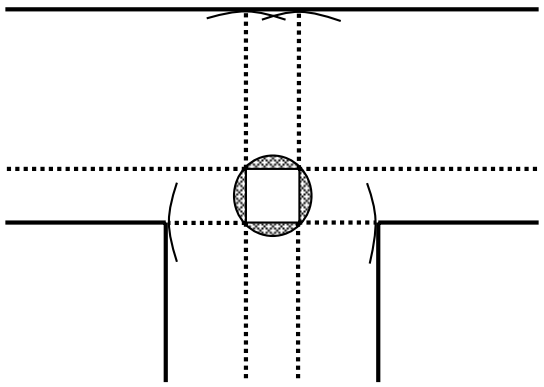

Figure 7: Sonar data entering a T junction

chose an abstract sensor space consisting of: corridor, T-junction, L-junction, dead-end, open space, crossing.

Having chosen a representation of the sensors for the belief network, we need to provide a mapping from the raw sensors on the robot to that abstract sensor space. In general, this mapping may be deterministic (for a given set of raw sensor readings we can supply the corresponding abstract sensor reading), or it may be probabilistic (for a given set of raw sensor readings, we supply a probability distribution over the various abstract sensor readings). There are two simple ways of determining this mapping from raw sensor readings to abstract sensor values: experimentally, by examining the raw sensor readings in a large number of situations and having a human specify the correct abstract sensor value, or heuristically, by using knowledge of the sensors and the world

the target. Our treatment of sensor abstractions is largely independent of the type of the physical sensor, however. Throughout, we will concentrate on the sensors involved in the location of the robot.

to specify the mapping directly.

Both of these techniques pose problems; [?] discusses these in some detail. Briefly, the problem with the experimental approach is that the sensor space is typically very large; a statistically significant sample translates to a large number of readings that must be tested individually. The heuristic approach is usually impractical because the domain knowledge necessary is not available, and may change with slight changes in the environment. These problems can often be alleviated by breaking the mapping from real to abstract sensor into several intermediate mappings: it is easier to provide a heuristic to reduce the large space of real sensor readings to some intermediate space small enough that we can perform experiments to find a mapping from it to the abstract sensor space.

3.3 Action Abstractions

The size of the action state space Ω_A is the number of possible actions that the robot can take. The cost of the decision procedure is proportional to $|\Omega_A|^h$, where h is the distance of the horizon in time-units, i.e. the length of the action sequences considered. Thus a reduction in the size of the action space is much more significant than a reduction in the size of the other spaces, in terms of its impact on the efficiency of the decision procedure. A simple way

to reduce the number of actions available is to define actions that operate at a higher level; instead of providing actions like "move forward 1cm" and "turn one degree left", we provide actions like "go down a corridor to the end", and "turn around the corner". One disadvantage of making actions complex, however, is that they are performed with no regard for the global directives; changes in the environment that might make this action useless will not be noticed unless that case has been explicitly represented in the action itself.

For the Mobile Target Localization problem, we provide the robot with a set of actions that is governed by our previous experience in trying to build simple, reliable controllers to guide the robot down corridors, around corners, etc. The tradeoffs are between the complexity of the actions and the complexity of the resulting decision model. Complex actions allow for a simpler decision model, since the model can operate at a higher level of abstraction. The choice of actions is also tightly connected to the choice of spatial representation discussed above: given that the purpose of the actions is to take the robot from one region to the next, a partition of the world into smaller, more detailed regions will require more detailed actions (and therefore a larger number of possible actions). Given our spatial representation, we create controllers to move down corridors, or to move into and out of junctions when told what type of junction they are in. They are designed to fail gracefully when possible

if the data does not fit in with the kind of junction expected, and also to keep the robot fairly well aligned along the axes of the corridors (to avoid inaccurate sonar readings due to specular reflections). The actions available to the robot are: travel down the corridor the length of one region, turn left around the corner, turn around, etc. Obviously some of these actions are not appropriate in certain circumstances; this observations can lead to ways of further reducing the number of action sequences that need to be considered at any point in time.

3.4 Quantification of the network

The one remaining problem in implementing a system based on belief networks is quantifying the network, that is to say, obtaining the conditional probabilities for each of the nodes in the network.

To determine the probabilities for a node, we iterate through all the possible combinations of values for the parents of the node. For each combination of values for the parents, we set up an experiment that corresponds to that combination, and then measure the value of the variable we are interested in. Typically we repeat the experiment a number of times to obtain a reasonable distribution for the value of the variable of interest. For example, to obtain the conditional probability of L_R in Figure **??** we place the robot in some location L, perform an action A, and then observe the new location of the

robot L'. After doing this a number of times for each combination of L and A, we have a probability distribution for the location of the robot given its previous location and the action performed.

4 Results

We have implemented a control system based on the model described here to drive a small mobile robot. The robot is equipped with 8 sonar transducers with a maximum range of 6 meters and a visual processing system capable of identifying moving targets in its visual field and reporting their motion relative to the robot.

Figure ?? shows the results of a sample run of our system in a small world consisting of two T-junctions and their connecting corridors. The robot begins with no information about the location of the target. After one step, it sees the target amble along another corridor, and then loses sight of it as the target moves on. Once the target is out of sight, the robot follows it around the corner. When it reaches the second T-junction, it cannot tell which of the two other branches the target followed, and in fact thinks the target might have turned around and slipped by the robot to go back down the corridor[10]. The

[10] In our current implementation, while the robot is executing an action, it does not make observations about the target which can be used by the control system.

robot first looks back down the corridor to see whether this was the case, and having established that it was not, checks the branches of the T-junction.

In our current implementation, to pick an action, the system considers all action sequences of lengths up to 3, selects the best sequence, and executes the first action of that sequence. The time required to evaluate a single action sequence of length 3 is 13 milliseconds — the complete evaluation of all action sequences and selection of the best action to take requires a total of 2 seconds.

5 Related Work and Conclusions

Probabilistic decision models of the sort explored in this paper are just beginning to see use in planning and control applications. Such models have been used to control machine tools [?], and account for the costs of inference in object recognition [?]. The emphasis in this paper is on using Bayesian networks to represent stochastic processes for sequential decision problems. These representational methods were designed to be used in conjunction with standard techniques from optimal control such as stochastic dynamic programming [?, ?, ?]. The methods described here have been extended to handle a wide range of stochastic processes including continuous-time, semi-Markov processes.

In this paper we have introduced probabilistic decision models as a means of

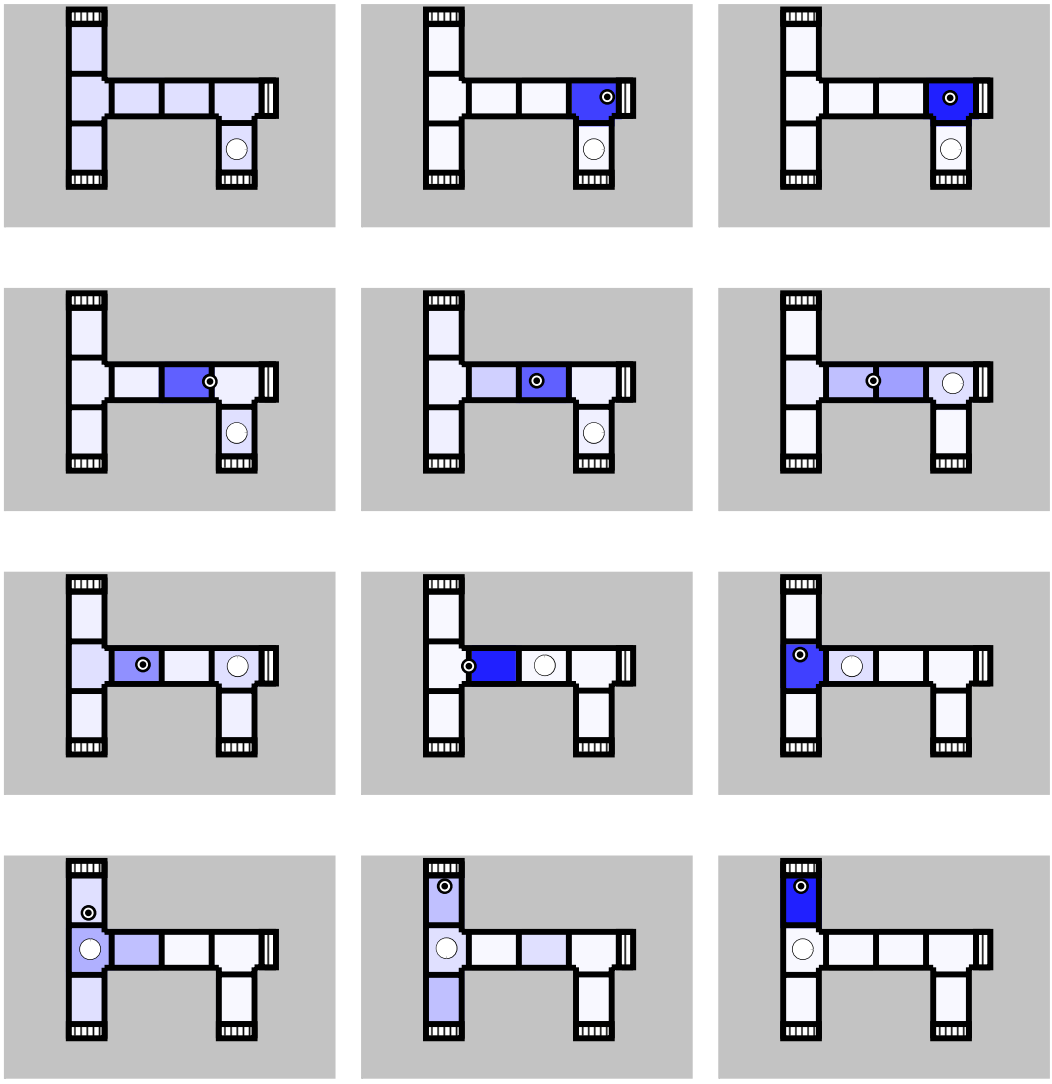

Figure 8: A sample run of our prototype

intelligent control. We have illustrated this approach with an example from our own work on the problem of tracking mobile targets in hallway environments.

Using probabilistic decision models presents some challenges. The careful development of abstractions of the important aspects of a problem is one such challenge; such abstractions keep the model sufficiently simple that the computation time required is practical. Using these models for control also provides a number of advantages. The benefits of information gathered by certain actions is included in the evaluations of those actions. These models also provide a unified approach to the problems of sensor fusion, prediction and evaluation.