Learning the Behavior Model of a Robot

Guillaume Infantes
ONERA, France
guillaume.infantes@onera.fr

Malik Ghallab
INRIA, France
Malik.Ghallab@inria.fr

Félix Ingrand
LAAS-CNRS, France
University of Toulouse
felix@laas.fr

Abstract

Complex artifacts are designed today from well specified and well modeled components. But most often, the models of these components cannot be composed into a global functional model of the artifact. A significant observation, modeling and identification effort is required to get such a global model, which is needed in order to better understand, control and improve the designed artifact.

Robotics provides a good illustration of this need. Autonomous robots are able to achieve more and more complex tasks, relying on more advanced sensori-motor functions. To better understand their behavior and improve their performance, it becomes necessary but more difficult to characterize and to model, at the global level, how robots behave in a given environment. Low-level models of sensors, actuators and controllers cannot be easily combined into a behavior model. Sometimes high level models operators used for planning are also available, but generally they are too coarse to represent the actual robot behavior.

We propose here a general framework for learning from observation data the behavior model of a robot when performing a given task. The behavior is modeled as a *Dynamic Bayesian Network*, a convenient stochastic structured representations. We show how such a probabilistic model can be learned and how it can be used to improve, on line, the robot behavior with respect to a specific environment and user preferences. Framework and algorithms are detailed; they are substantiated by experimental results for autonomous navigation tasks.

1 Introduction

Nowadays, complex artifacts are designed from well specified and formally modeled components. But most often, the models of these components have been developed independently and often in different representations. They cannot be *composed* into a global functional model of the artifact. Consider for example the internet (how the topology, the routing mechanisms and the infrastructure of the network are designed and controlled for a better quality of service), or a complex multi-core processor (how its paging and caching mechanisms are efficiently exploited by a compiler): here and in other similar examples a significant observation, modeling and identification effort is required to get a global functional model in order to better understand, control, improve the use and eventually the design of the artifact.

Robotics provides an excellent illustration for this *composationnality problem* and for the need of global functional models. Autonomous robots are able to achieve more and more complex tasks, relying on more advanced sensori-motor functions. To better understand their behavior, to use them and improve their performance, it becomes necessary but more difficult to characterize and model at the global level how a robot behaves in a given environment. Low-level models of sensors, actuators and controllers cannot be easily composed into global models. Sometimes high level models such as *precondition–effects* operators used for planning are available, but generally they are too coarse to accurately represent the actual robot behavior.

Consider for example the autonomous navigation task, well mastered by most mobile robots. Models of low-level sensori-motor functions, such as error models of range sensors, are often available. Algorithms for map building, obstacle avoidance, path planning, or non-linear motor response to speed references, are modeled at design time. But these models are not easy to combine into a global description of what is going on during a navigation within a specific environment. It is hard to model globally how the navigation task is performed. But such a model is needed in order to predict how the robot behaves in different environments, how the actual task is progressing, what are its chance of success, its resource consumption and time to completion, etc. As a side effect (even if it's not its main purpose), an explicit model can also be used to improve the robot performance and to better adapt it to a specific environment.

Adequate approaches for addressing these motivations depend on the purpose of the model to be learned. For example, the purpose of synthesizing black-box controllers, fine tuned for some environment and task, has been widely studied. Reinforcement learning methods [62, 56] have, in particular, been successfully demonstrated for synthesizing controllers of complex motions and maneuvers for

humanoid robots [2, 49] and autonomous helicopters [9]. Our objective is to synthesize an explicit and predictive model of the already available controller of the robot. We first want to *understand*, through this model, what the robot is doing, and, as a follow-up goal, to eventually improve the performance of the robot by running, in a predictive way, the learned model. To put our purpose in a figurative way: *we are not in the position of the robot designer, but in the position of the owner of a new robot, who wants to understand what his robot can do, and, it if has some built-in adaptive capabilities, how to make the best use of it, for his specific environment and preferences*.

In classical reinforcement learning techniques where the goal is to synthesize a controller, one learns state utilities and best actions. Learning an explicit model of the system may or may not be needed, e.g. to take into account temporal difference utilities, and eventually to guide exploration (as discussed in section 6.4). In our approach, we first focus on learning the model of the behavior, as given by the implemented controller, without taking into account rewards or costs. In a second step we exploit the learned model, in order to optimize the parameters of the controller for the specifics of the environment and user preferences. These two steps are akin respectively to *(i)* learning probabilistic models, in our case learning the model of a dynamic system with partial observability, and *(ii)* reinforcement learning, understood in broad sense of the set of techniques for improving the behavior from environment feedback, in our case learning the controller parameters.

We chose to describe the behavior of the robot as a discrete dynamic system represented with a *stochastic transition graph*. Hidden Markov Models (HMM) and Dynamic Bayes Networks (DBN) are example of such a representation for which learning techniques have been developed.

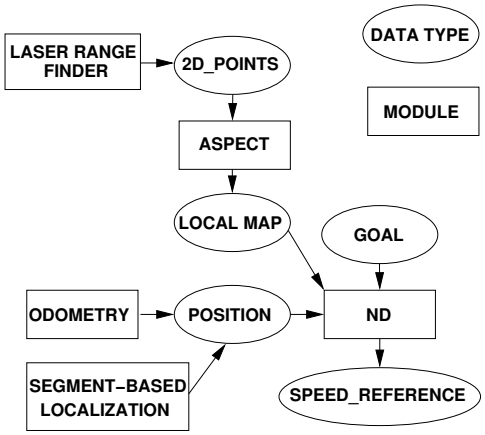
The Hidden Markov Models (HMM) have been explored in [22] for the precise motivations we are addressing here. In this article, the authors have shown that large HMM models of a robot behavior with dozen of hidden states can be learned with affordable computational cost from real navigation data. However, the HMM approach suffers from several limitations, computationally, since the HMM state space is flat and does not take into account available causal knowledge between state variables, and functionally since the learned model, a stochastic automata, cannot be easily used directly to improve the robot behavior.

This paper develops a novel approach relying on Dynamic Bayes Networks (DBN). The DBN representation significantly extends upon the HMM one since it introduces a state space structured into controllable parameters, environment variables, robot state variables and mission variables, with explicit causal links between these variables, within each state and from state to state. Furthermore, the DBN approach allows to improve the behavior of the robot using the learned model. The contributions proposed here are the following:

- a methodology for representing the behavior of a robot into a structured DBN model, divided into observable, controllable and hidden components, and the causal links between these components,
- original algorithms to learn such DBN models from real-world data with a particle filtering approach to handle the combinatorial explosion,
- algorithms to efficiently improve the robot behavior into using the learned DBN models along with a Dynamic Decision Network (DDN) approach,
- experiments with an autonomous navigation task that demonstrates significant improvements in the success rate of the modeled task.

We experimented with a robotics platform called Rackham, an iRobot B21R tour robot (figure 1(b)), using a complex navigation procedure called ND (Nearness Diagram navigation) [41]. A schematic view of this navigation procedure can be seen on figure 1(a). A laser range finder gives a set of points representing obstacles in front of the robot. These points are used by the *"aspect"* module to build a local map around the robot. ND uses the current position of the robot to compute the local goal in the robot's coordinate system. Then with this local goal and the map around the robot, depending on an estimate of the position of obstacles around the robot, it chooses a navigation strategy. Several strategies are available to ND, handling different situations, like open area, obstacle on the left, on the right, on both sides, very close obstacles. After choosing a strategy, ND repeatedly computes a speed reference depending on the precise map around the robot, in order to move towards the goal while avoiding obstacles. This navigation procedure has been developed at LAAS and in other laboratories, and by various people. As a result, it has become robust and efficient. Yet, it has reached such a complexity, that nobody is really able to explain and predict what's going on during a navigation. Furthermore, ND has a number of parameters which can be hand tuned to modify the navigation. Understanding how these parameters change the behavior, and how one can automatically modify them purposefully with respect to the environment is one of the problems addressed successfully here.

In the remaining of the article we first detail in section 2 the modeling problem addressed, explaining how the observation space and the state space can be obtained from sensor data and from navigation variables and how to structure the DBN model. The learning algorithms are presented and analyzed in section 3. We then present how the learned model can be used as DDNs to improve the robot navigation performance (section 4). Section 5 details the experimental results obtained. A discussion of these results conclude the paper (section 6).



(a) ND-based navigation procedure

(b) Platform

Figure 1: Rackham (RWI B21R) : a museum guide robot

2 Modeling a robot behavior

2.1 Formal problem statement

A global behavior model of a robot achieving a task in some partially known environment is an abstraction of the full rendition of the robot complex interaction with the environment into some useful description that easily allows to *explain* the past states, to *predict* future possible states knowing the past, and to *update* the prediction with respect to new observations. A simple representation would model the behavior as a stochastic transition graph $\lambda = (S, E, \tau, \delta, \Theta)$ where:

- S is a finite set of states, $s_t \in S$ is the state at time t; $s_{1:t}$ is the trajectory between time 1 and t;
- E is the finite set of possible observations, e_t is the observation at timestep t, $e_{1:t}$ the sequence of observations (the evidence);
- δ is the state transition function: $\delta[s_t|s_{t-1}, e_{t-1}]$ is the conditional probability of s_t giving the past state and observation in the first order Markov assumption;
- Θ is the observation function: $\Theta[e_t|s_t, e_{t-1}]$ is the conditional probability of e_t giving the current state and past observation;
- τ is the a-priori probability distribution of the initial state.

Acquiring a model λ allows to *explain* past behavior, i.e. to find the most probable trajectory $s_{1:t}$ knowing the observation sequence $e_{1:t}$ (the Viterbi algorithm),

to *predict* future possible states s_{t+k} of the robot knowing the past trajectory $s_{1:t}$, and to *update* the prediction s_{t+1} with respect to the observation e_t (i.e., filtering algorithm).

The approach used here is an extension of stochastic transition graphs to the more structured representation offered by *Dynamic Bayes Nets* (DBN). Instead of two flats sets (hidden states S and measurable observations E from which hidden states are inferred), DBNs rely on a collection of *state variables*, some hidden and some measurable, and an explicit causal structure between these variables. More formally, the model is $\lambda = (\mathbf{X}, G, \tau, \gamma)$ where:

- $\mathbf{X} = \{X^1, \ldots, X^n\}$ is a set of series of random variables, X_t^i being the state variable X^i at time t, $X_t^i = x_t^i$ is the value of that variable, $x_t^i \in D^i$, a finite range of X^i;
- $\mathbf{Y} \subset \mathbf{X}$ is the subset of state variables that are observable, $\mathbf{Y}_t$ are observable state variables at time t, and $\mathbf{Y}_{1:t}$ are observable state variables from 1 to t;
- $G = (\mathbf{X}, U)$ is a DAG whose nodes correspond to the state variables, edges in U are:
  - (i) restricted to either variables within the same timestep or between two consecutive steps, giving the first order Markov assumption, and
  - (ii) are identical over all timesteps, *i.e.*, either (X_t^i, X_t^j) is an edge $\forall t$ or is not an edge for any value of t, and similarly for (X_t^i, X_{t+1}^j);
- $\gamma(X_t^i) = P(X_t^i | \pi_t^i)$ is the conditional probability distribution of state variable X_t^i giving its parent variables;
- π_t^i is the set of parent variables of X_t^i in G, these variables are either in timesteps t or $t-1$;
- τ is the a-priori probability distribution of the state variables at timestep 1.

We are assuming that $\mathbf{X}$ is a DBN with respect to G, that is the joint probability density function is such that:

$$P(\mathbf{X}) = \prod_{i,t} \gamma(X_t^i)$$

In summary, the learning problem addressed here is defined as the following: from a collection of raw data acquired by the robot during a training phase

(a) define a meaningful finite set of state variables $\mathbf{X}$,

(b) structure the state variables into a causal graph G,

(c) find a "good" conditional probability distributions γ that best fit the training data.

We are proposing a methodology for addressing manually steps *(a)* and *(b)*, and algorithms for solving step *(c)* as an optimization problem. The usual optimization criterion in probabilistic modeling is to find a function γ that gives a high level to the probability of observing the training data. This is known as the likelihood of the observation: $L(\mathbf{Y}) = P(\mathbf{Y}|\lambda)$. The learning problem is formally defined as an optimization problem with respect to γ, the only free functional parameter when $\mathbf{X}$ and G are given :

$$\widehat{(\gamma)} = \underset{(\gamma)}{\operatorname{argmax}} P(\mathbf{Y}|\lambda = (\mathbf{X}, G, \tau, \gamma)) \tag{1}$$

2.2 Defining the set of observable state variable

In this section we briefly explain the methodology used for defining $\mathbf{Y}$, the set of observable state variables. We first have to refine sensor data into a set of features, then to further abstract away these features into a set of observation variables over finite ranges. A similar approach leads to the definition of the state variables for the robot configuration. In both cases, the state variables are defined and chosen manually. Although we will be using automated clustering techniques for these variables, this methodology relies mostly on engineering steps and requires a good knowledge of the robot.

Raw sensor data must be pre-processed by filtering, aggregation and fusion techniques. We relied on standard signal processing techniques, such as the sliding window method (used in [22]). Selection and filtering is done by defining a set of functions which refine the raw data into higher level interesting features. Defining these functions is highly dependent on the system to model, and on the priori knowledge of what information will be interesting in order to recognize the current state of the system. In most cases, state variables result from average values over the window. In some cases a function like the maximum over the window is more relevant, e.g., for the acceleration state variable since we are using a particular PID motion controller.

Clustering algorithms are needed in order to obtain a finite number of observation symbols and state symbols. In our case, we need unsupervised clustering, that is, we are not able to feed the algorithm with some well-clustered examples, because we do not know what might be good clusters for the whole model to behave correctly. We also need some control on the size of the ranges of state variables in order to limit the size of the model and make the learning algorithms tractable. Simple well-known algorithms like k-means [28] provide good results for clustering the range of a single variable. Other possible techniques include ϵ-means [14]

and affinity propagation [23]. We can also cite Kohonen self-organizing maps [33] that give good results (as in [22]) for clustering a multidimensional space.

Finally, a selection problem of the set of observable state variables is at hand. As illustrated below, we relied on the following general principles:

- Use all control parameters offered by the task itself, since control parameters are supposed to have a large influence (if not, they are generally assigned to some value and hidden from the user).
- Use a large amount of a-priori knowledge to build meaningful variables that summarize low-level sensor data. In our example, we need to refine the points and segments detected by the laser range finder in high level variables that reflect relative position to the closest obstacle, or global cluttering.
- Add few relevant variables until the model is too large for the complexity of learning algorithms.

Instantiation: observation and state variables for a navigation task

This approach has been implemented into our Rackham robot. The monitoring of raw data is done at a frequency of 2.5 Hertz. The scalar raw data are clustered using k-means clustering [28] in order to obtain a few clusters.

For our navigation task, the control parameters are the following:

- the two **size growing** parameters of the robot: path planning techniques grow obstacles by the size of the robot; a second parameter is a larger security area where there should be as few obstacles as possible;
- the **maximum linear and angular speeds** allowed;
- a **linearity factor** between the two speeds given by the motion generator controller: the linear and angular speeds are mutually constrained by a polynomial function that influences the type of trajectory produced.

The state variables describing the environment are the following:

- the **estimated cluttering** of the environment, defined as a weighted sum of distances to nearest obstacles around the robot (the weight is heavier in front of the robot, lighter on its sides);
- the **angle of the nearest obstacle**, which is the difference between the current heading of the robot and the heading leading to the nearest obstacle;
- the **distance to the nearest obstacle**;
- the total **number of segments** in the perceived scene;
- the **number of valleys** or possibles ways found for reaching the goal.

The state variables describing the current robot configuration are

- the current **linear and angular speeds** of the robot (v and w);
- the robot current **linear and angular** accelerations (Δv and Δw);

Finally, we need a few state variables to describe the mission achievement status:

- the **variation of the distance to the goal** (estimated as an euclidean distance);
- the ratio of the **mission achieved** so far in the navigation;
- **mission status** in *begin*, *end*, *fail* and *normal*;
- the current **navigation strategy** being used (see [41] for a description of the different navigation strategies available and how a adequate one is repeatedly selected for the current navigation situation);
- the perceived **human adequacy** of the behavior: how *aggressive* or how slow and boring the robot is perceived by the human user.

We introduced this last variable because we empirically noticed that some navigation settings led the robot to move too fast or too close to humans in some situations, while other settings led to a too slow and cautious motion. None of theses settings is superior in all situations and for all kind of interactions with respect to the success of the mission. Furthermore, these settings lead to quite subjective appreciations. Hence we introduced this variable that is not issued from measures but from the user's appreciation: its values are given by the user during the training phase.[1]

normal;

The ranges of theses variables are discretized into clusters of 3 to 6 values. The number of different possible observations and states is more than 15×10^9. This indicates that even with a very large number of runs, all possible observations will never be seen, leading to many non-informative transition probabilities into the DBN. We will show later how to deal with such a sparse DBN for decision making.

We now show how we can structure this flat set of variables into a causal graph.

2.3 Structuring state and observation variables into a causal graph

The structure of the model λ whose parameters we will learn later is shown on figure 4, where vertexes are state variables, shaded vertexes are hidden state variables, and edges are causal links corresponding to conditional probability dependency.

[1] On may imagine that a new robot is delivered with standard settings to the house or work environment of its user who trains it to its liking and the specifics of the environment.

We explain in this section how we obtained this particular structure from the set of variables described in the previous section.

Grouping variables into functional sets

First, we structure the state variables into 4 sets depending on their high-level "nature":

1. The first set contains all *control parameters*. They have a direct influence on the behavior, but are not influenced in any way in the model. Their values will be changed and fine tuned as a result of the learned model and optimization procedure. These parameters are: the size growing parameters, the maximum speeds and the linearity factor.

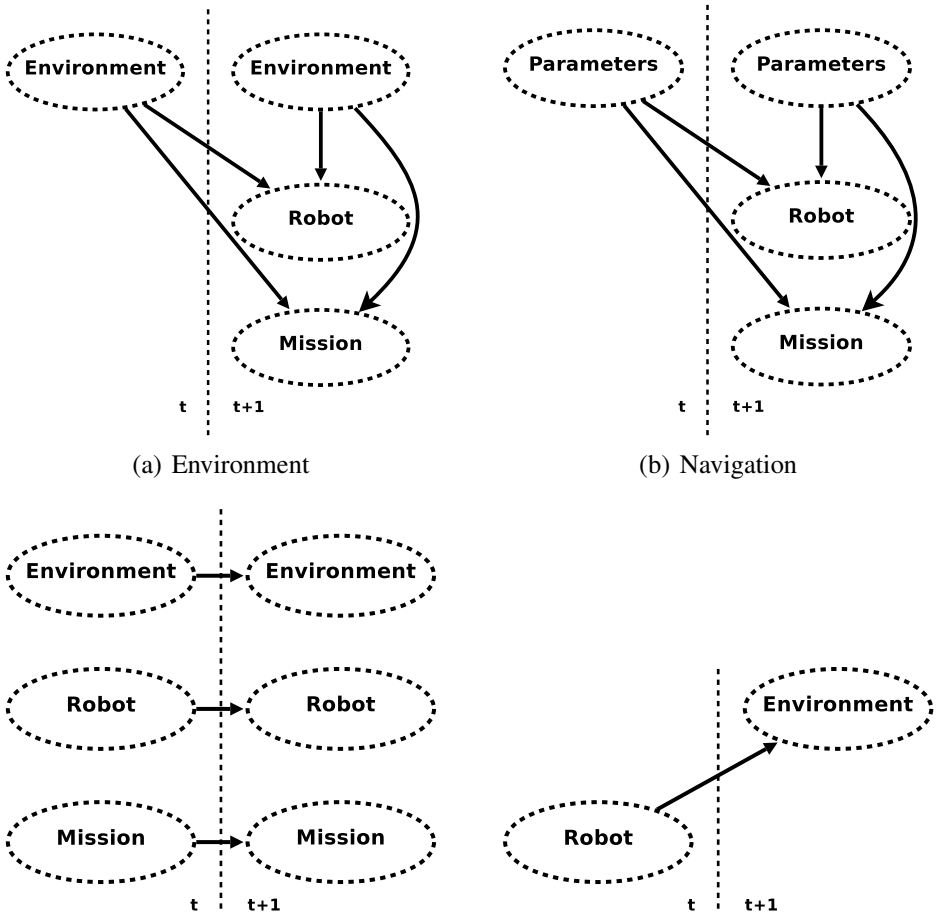
2. The second set contains *environment variables*. They have a direct influence on the behavior of the robot; they are subject to some changes that we need to model. More precisely, we need to model the effects of the surrounding environment on the behavior of the robot, as well as the effects of the behavior of the robot on its environment (the environment itself does not change, but the perception of the environment by the robot does; we could talk about *subjective environment* here). These environment variables are the cluttering, the angle of nearest obstacle, the distance to nearest obstacle, the number of segments and the number of valleys.

3. The third set contains information about the observed *robot configuration* itself: its linear and angular accelerations, and its linear and angular speeds.

4. The fourth set contains information about the *mission* that the robot is doing: the variation of the distance to the goal, the achieved ratio of the navigation, the current strategy, the mission status, and the human adequacy.

This structuring of state variables is very useful for the definition of causal links. All variables within a set have the same functional role. Hence all variables within a set will have the same set of causal links to the other sets of variables. Let us now explain what links are needed between these sets. On figure 2 are shown four "functional" causal links between the different sets.

- The first one, fig. 2(a) shows the influence of the *environment* variables on the other sets: the environment causes changes in both the *robot state* variables and in the *mission* variables. Both timesteps t and $t + 1$ have an influence because we do not know what influences the most: the value of the environment variable, or the change of value.
- The *parameters* have a similar influence on both the *mission* and *robot state variables*, as shown on figure 2(b). Here again, both timesteps have an influence, meaning that both value and value change can be taken into account into the



(a) Environment (b) Navigation

(c) State variables (d) Dynamics

Figure 2: Global causal structure

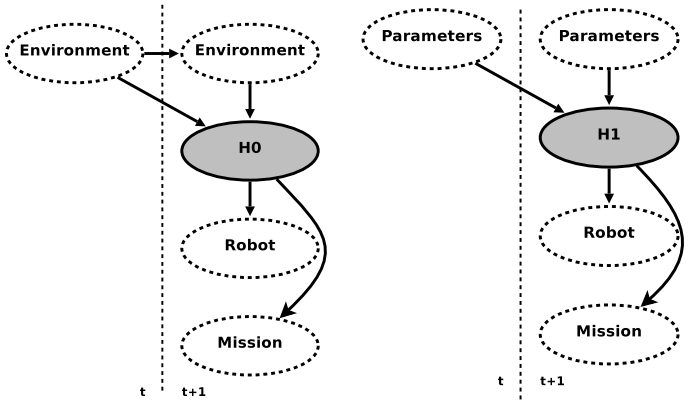
model.

- We want to model the dynamic evolution of every set but the *parameters*; this gives the causal links shown on figure 2(c). In this case, we do not connect every variable from the outgoing set to the incoming set, but only one variable to itself at the next timestep.
- Finally, since the *environment* variables describe the environment as perceived by the robot, the *robot state* variables have an influence on the environment variables, as shown on figure 2(d).

Introducing hidden state variables

Whereas our causal links between sets of variables seem reasonable, we face one major problem trying to deal with this model as-is: it is too large to be stored into main memory, thus cannot be used on-line (and of course may not be learned on-line either). So we need to reduce the complexity of the model, that is to reduce the number of incoming links for the variables. One solution to tackle down the complexity is to introduce hidden variables and let the learning algorithm deduce the values, and thus implicitly the correlation by itself. In [3], the author explain how hidden variables can give a more compact representation, and in [17], we can see how this point is further refined.

In our case, we do not try to add hidden variables automatically (as shown in [6]), but instead we rely a priori knowledge about where such variables can be useful. As variables in a given set have the same high-level functionality, there are likely to be correlated. We introduce an unobservable variable in order to compress the information of the environment, which is very likely to be highly correlated. This is shown on figure 3(a). In the same idea, we introduce another variable that represents an abstraction of the state of the robot and its mission, as shown on figure 3(b). Altogether with figures shown on fig. 2, this gives the model of figure 4.



(a) H_0: Environment (b) H_1: Robot and Mission

Figure 3: Introducing hidden state variables to reduce the complexity

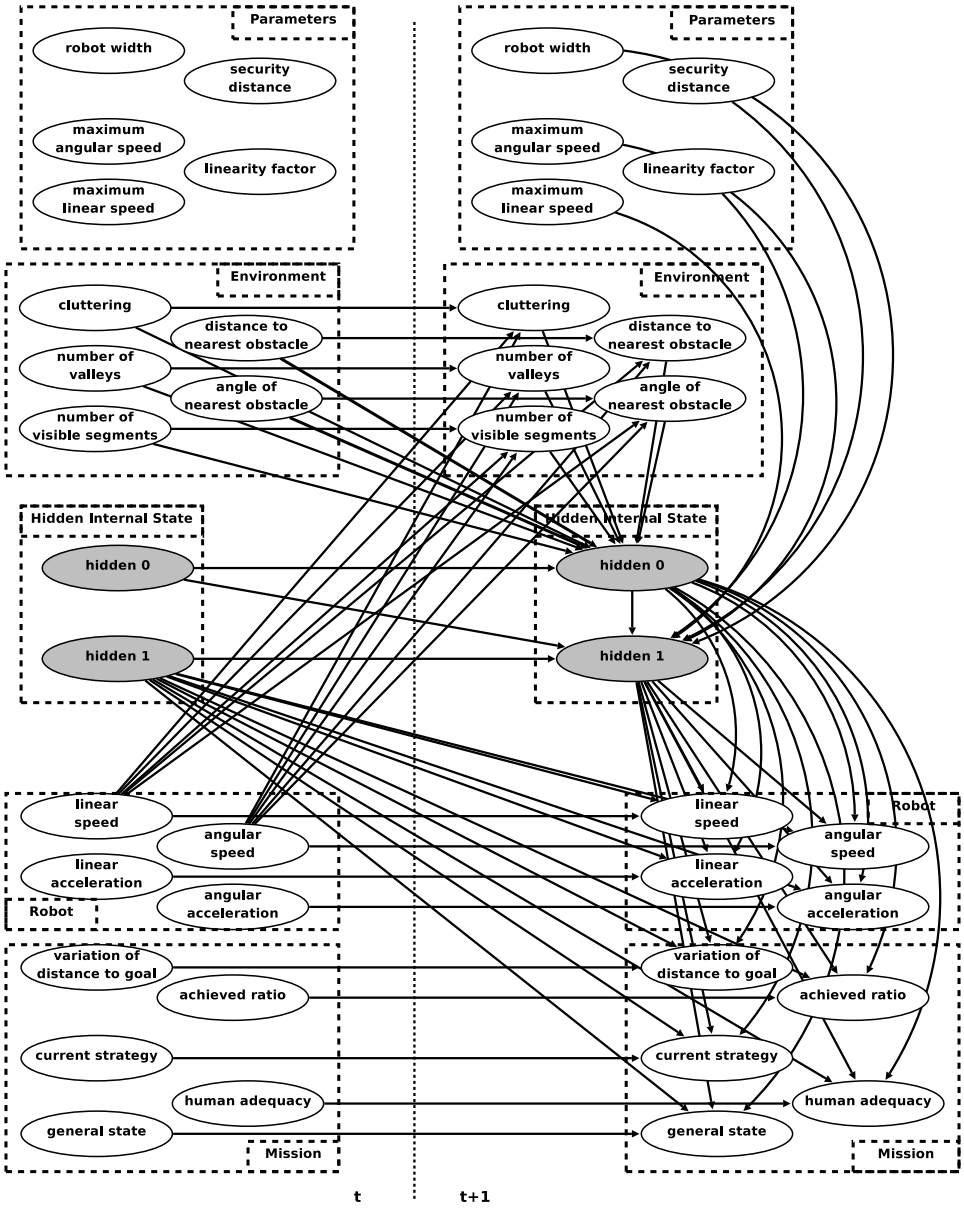


Figure 4: General view of the model to learn

3 Learning transitions probabilities

We focus here on the quantization part of the learning problem, i.e. giving X, G and τ find γ such that:

$$\widehat{(\gamma)} = \underset{(\gamma)}{\operatorname{argmax}} P(\mathbf{Y}|\lambda = (\mathbf{X}, G, \tau, \gamma))$$

There are several original contributions in this section. The main one is a formalization of the Expectation-Maximization algorithm for dynamic bayesian networks. While this algorithm has been previously used for DBNs [7, 71], we could not find any complete formalization nor pseudo-code. Furthermore, the approximation scheme and the scaling procedure needed for any practical use of EM were not really explained in the available literature. We have introduced a particle filtering approach in order to scale up the procedure. We believe that our approximation scheme is different from previous particle-filter approaches (for instance in [34]).

We first present a particle filtering technique we developed to reduce the complexity of the probabilistic inference. We then explain the basic expectation and maximization steps and detail the learning algorithm.

3.1 The correlated inference problem

In order to deduce a variable value, information about its parents is needed. Precisely, the joint probability over all instances of the parents π_t^i of variable X_t^i is needed. Consider the simplified example in figure 5. In this case, we want to deduce the value of the speed of a robot at timestep t depending on two variables, **obstacle** $\in \{yes; no\}$ and **acceleration** $\in \{+; -\}$.

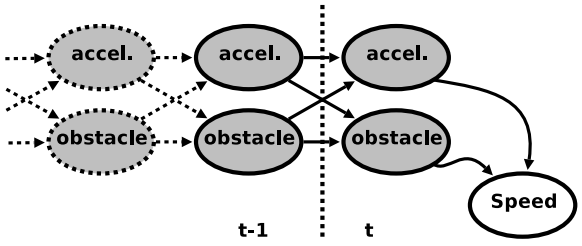


Figure 5: Example of intractable DBN

In order to compute a probability distribution over values of speed variable, joint probability over variables $acceleration_t$ and $obstacle_t$ is needed. But in order to compute this probability, joint probability over $acceleration_{t-1}$ and $obstacle_{t-1}$

is also needed, and so on. In the worst case, memorizing or re-computing the joint probability over all variables instances for all timestep is needed for exact inference. To reduce the complexity, we have to use an approximate inference.

A widely-used method for approximate inference is particle filtering [27]. It is based on discrete sampling (monte-carlo) of probability distributions, and on re-sampling this samples taking new observations into account in order to keep a good estimate. Each sample is called a particle and can be seen as a hypothesis on the possible values of a variable.

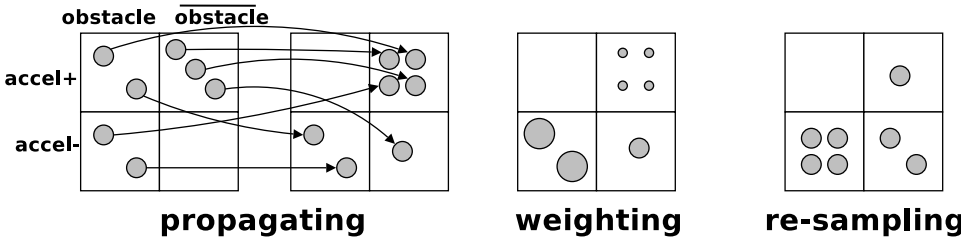


Figure 6: Particle filtering principle

The principle is illustrated on figure 6, where the system state is described with the two previous variables. Thus the four possible states are $(accel.^+, obstacle)$, $(accel.^+, \overline{obstacle})$, $(accel.^-, obstacle)$, $(accel.^-, \overline{obstacle})$. For example, the initial distribution shown on the left tells us that there are 3 chances out of 7 to be in state $(accel.^+, \overline{obstacle})$. In the first phase, named propagation, each particle is updated through the stochastic transition model of the system independently of the others. In the second phase, every particle p_i is given a weight which is the probability of having the observation knowing that the state is the one described by p_i (in the figure, having the observation being in $(accel^-, obstacle)$ is much more probable than being in $(accel^-, \overline{obstacle})$ than in $(accel.^+, \overline{obstacle})$). Finally the re-sampling phase does probabilistic sampling over weighted particles in order to maintain a high number of particles on the most probable hypothesis. This permits a more accurate estimate. Formally we denote:

- r_t^i: the i^{th} particle, a vector of possible values for each state variable at timestep t (every r_t^i is one possible instance of $\mathbf{X}_t$);
- w_t^i: the weight of particle i at time t;
- $R(w_1, ..., w_N)$: a random function that gives values from 1 to N proportionally to $w_1, ..., w_N$.

γ is the conditional probability function of the model previously defined. The three steps done at every timestep t for every particle i are:

- propagating: $r_t^i \leftarrow \gamma(r_{t-1}^i)$
- weighting: $w_t^i = P(\mathbf{Y}_t|r_t^i)$
- re-sampling: $r_t^i \leftarrow r_t^{R(w_t^1,\ldots,w_t^M)}$

3.2 Probabilistic dependence quantization

The main difficulty with statistical learning of such models is that some variables are hidden. If all variables were observable, all the work would be to count every transition instance in the training set, then normalize these counts in order to obtain probabilities. But as some state variables are hidden, we have to estimate the hidden part of the needed information. The basis for this estimate is a model having the same structure. Thus, the algorithm iterates as a local search with two different phases as shown on algorithm 1. While the maximizing part of the algorithm is straightforward, the expectation part needs more attention.

Algorithm 1: General Expectation-Maximization principle

```
while P(Y_{1:T}|λ) increases do
    using λ, compute expected probability distributions upon hidden
    variables;
    update λ accordingly to expected statistics in order to maximize
    P(Y_{1:T}|λ);
end
```

What is needed is to compute every transition probability to a variable from its parents at every timestep. In order to do that, let us define, in addition to the notations introduced in the definition of $\lambda = (\mathbf{X}, G, \tau, \gamma))$ the following notations

- $\varphi^i(\mathbf{Y}_t) = P(\mathbf{Y}_t|X_t^i, \pi_t^i)$ the joint probability distribution of the observable state variables $\mathbf{Y}_t$ given the values of another state variable X_t^i in the same time period t and of the parents variables π_t^i of X_t^i;
- According to the context, π_t^i denotes either the set of parent variables of X_t^i or the tuple of values of these variables, i.e., $\pi_t^i = (X_t^j = x_t^j, \ldots, X_{t-1}^k = x_{t-1}^k)$ for all parents of X^i, some of them being in the previous time period.
- Whenever needed, will make explicit the value of a state variable for which the conditional probability is computed, i.e., $\gamma(x_t^i)$ for the value $X_t^i = x_t^i$.

In order to learn the model, that is to compute the conditional distributions γ, using an Expectation- Maximization algorithm, we apply a forward-backward message passing scheme as used for hidden Markov models learning [52].

The forward message is defined for a state variable as: $\alpha(X_t^i) = P(\mathbf{Y}, X_t^i|\lambda)$; for its parents as: $\alpha(\pi_t^i) = P(\mathbf{Y}, \pi_t^i|\lambda)$. The forward message for one variable can be deduced from the one of its parents using:

$$\alpha(X_t^i) = \sum_{\pi_t^i} \alpha(\pi_t^i)\gamma(X_t^i)\varphi^i(\mathbf{Y}_t) \tag{2}$$

The sum is over all π_t^i tuples.

We now have to deduce the forward message for an instance of the parents variables knowing the individual forward messages. If the parents were perfectly independent, we could use :

$$\alpha(\pi_t^i) = \prod_{X_t^k \in \pi_t^i} \alpha(X_t^k) \tag{3}$$

But as we are modeling dependencies between variables, we know that every instances of values of parents do not have the same probability, and we denote this belief:

$$B(\pi_t^i) = P(\pi_t^i|\mathbf{Y}_{1:t}, \lambda) \tag{4}$$

Thus we have:

$$\alpha(\pi_t^i) = [\prod_{X_t^k \in \pi_t^i} \alpha(X_t^k)]B(\pi_t^i) \tag{5}$$

To compute the α values for all variables, we simply apply these two equations recursively from timestep 1 to T, in the right order of causality (i.e. using the direction of causal links from parents to children). But as we have seen on section 3.1, variables may be strongly correlated depending on the structure of the model, and assuming complete independence between them would lead to a very poor approximation. Luckily, the forward message is related to the classic inference problem so we can approximate the value of $B(\pi_t^i)$ using a particle filtering technique. While computing α from time 1 to T, we maintain a particle filter and we deduce an approximate of the corresponding probabilities of seeing each π_t^i, simply by marginalizing and normalizing where needed.

The backward message is defined for a state variable as:

$$\beta(X_t^i) = P(\mathbf{Y}_{t+1:T}|X_t^i)$$

This probability is computed recursively backward from time T to 1. For a given timestep t, we need to consider every value of every children variable X^k of variable X^i (noted as $X^k \in Ch(X^i)$), and all tuples of values of the parents of this

variable X^k that contain a value $X^i = x^i$. We then take the average of the probability to see the observation sequence on every possible trajectory, times the corresponding transition probability and the current observation probability:

$$\beta(X_t^i) = \sum_{X_t^k \in Ch(X^i)} \sum_{\tilde{\pi}_t^k} \sum_{x^k \in D^k} \frac{\beta(X_t^k)\gamma(x_t^k)\varphi^i(\mathbf{Y}_t)}{|D^k| C_{\pi^k}} \tag{6}$$

where the second sum is over $\tilde{\pi}_t^k$ all tuples of π_t^k containing the value $X_t^i = x_t^i$; D^k is the range of X^k; $C_{\pi_t^k} = |\{\pi_t^k : (X_t^i = x_t^i) \in \pi^k\}|$ the number of tuples of the parents X_t^k that contain the value $X_t^i = x_t^i$.

When all αs and βs are computed, we obtain simply:

$$\gamma(X_t^i) = \frac{\alpha(\pi_t^i)\gamma(X^i)\varphi^i(\mathbf{Y}_t)\beta(X_t^i)}{\varphi(\mathbf{Y}_{1:T}|\lambda)} \tag{7}$$

from which we can update the model with (K is a normalizing factor):

$$\gamma(X^i) \leftarrow \frac{\sum_t \gamma(X_t^i)}{K} \tag{8}$$

The entire algorithm is shown on algorithm 2.

Every update of the model has a complexity of $O(R \times N^{2M+1} \times T^2)$, where R is the number of variables, N is the maximum number of value per variable, M is the maximum number of parents for a variable and T is the number of observations. Because the probabilities of seeing the observation decreases exponentially to zero, a scaling factor must be applied in order to implement the algorithm, as for the classic algorithm for HMMs. The scaling factor is dependent on every variable and every timestep. The scaled mechanism insert steps as shown on algorithm 3.

4 Using the learned model to improve the behavior

One of the motivations for learning an explicit model of a particular robot behavior using DBN is to use it to better control the robot. Indeed, the DBN is a predictive model which can be used to compute the various outcomes according to various hypothesis and parameters values, and then to choose the one which best fits the current goal. The main idea is to use the DBN as a DDN (dynamic decision network), where some variables are control variables (i.e. in our example, the one which parametrizes the controller: robot width, security distance, maximum angular and linear speeds, linearity factor) while the general state and human adequacy variables are reward/cost variables.

Algorithm 2: Expectation-Maximization for general models

while $\varphi(\boldsymbol{Y}_{1:T}|\lambda)$ *increases* **do**
  `// expectation phase`
  for *t = 0 to T* **do**
    forall π^i **do**
      compute $B(\pi_t^i)$ knowing $B(\pi_{t-1}^j), \forall j$ and $\mathbf{Y}_t$, using particle filtering;
    end
    forall X^i*, respecting causality direction* **do**
      compute every $\alpha(\pi_t^i)$ using eq. 5;
      compute $\alpha(X_t^i)$ knowing $\alpha(\pi_t^i)$ using eq. 2;
    end
  end
  for *t = T to t = 0* **do**
    forall X^i*, in inverse causality order* **do**
      compute $\beta(X_t^i)$ using eq. 6;
    end
  end
  `// maximization phase`
  forall X^i, π^i **do**
    forall *t* **do**
      compute $\gamma(X_t^i)$ using eq.7;
    end
    $\gamma(X^i) \leftarrow \sum_t \gamma(X_t^i)$ (8);
  end
  compute $\varphi(\mathbf{Y}_{1:T}|\lambda)$;
end

Algorithm 3: Scaling mechanism patch

`// add the following steps between lines 2 and 2`
1 $scale_{i,t} = \sum_{X^i} \alpha(X_t^i)$;
2 **forall** X^i **do** $\alpha(X_t^i) \leftarrow \frac{\alpha(X_t^i)}{scale_{i,t}}$;
`// add the following step between lines 2 and 2`
3 **forall** X^i **do** $\beta(X_t^i) \leftarrow \frac{\beta(X_t^i)}{scale_{i,t}}$;

The learned model is formally a dynamic Bayesian network. In order to use this DBN for decision making, we can use the DDN formalism [70], as it is very close. There are two main differences with respect to DBN:

1. in a DDN some transitions are labeled with cost or rewards;
2. in a DDN some variables are said controllable and algorithms for decision making give values to this controllable variable in order to maximize utility.

The conditional transition probabilities can be used "as-is" to define transition probabilities in a dynamic decision network. So we can simply add utilities on some transitions, or on some values of some variables[2] for the first point. For the second point, we want to underline the fact that there is no difference between a Bayesian variable and a controllable one. Giving a set of controllable variables only informs the decision making algorithm which variable values it is allowed to change.

A DDN defines a "factored partially observable Markov decision process" (f-POMDP [54]) and many algorithms have been developed to compute near-optimal policies for such problems [64, 50, 57, 51]; but these approaches remains computationally expensive and are not always useful depending on the optimality criterion tied to the application.

4.1 Costs, rewards, optimality and horizon issues

Optimizing the behavior of a robot is far from a generic decision-making problem for several reasons:

- There is no simple "best-behavior" criterion. We first would like to reduce the number of failures cases, while reducing the energy or time needed to complete a task and make the robot behavior appear safe and friendly to users. While giving costs and rewards to *failure* and *success* is straightforward, we can also try to make difference between successful behaviors by giving smaller rewards to quick and safe behaviors, and smaller costs to slow and dangerous ones. This secondary criterion is clearly application-dependent[3].
- Another point is that because the size of the observation and state spaces is too large, the learned conditional probabilities are not very precise. Hence the behavior optimization process should be in accordance with the quality of the model.

[2] so that every transition that lead to this value gets the cost/reward

[3] Do we prefer the robot to be quick or to be friendly? That's a matter of taste...

- A third difference between classical decision making and our optimization problem is that there are very few dead-ends, meaning that in most cases there will be a decision that leads to success. The only case of a dead end would be if the robot is too close to an obstacle, and any decision would lead to a collision.

The corresponding dynamic decision network is shown in figure 7, where squares represent controllable variables, and diamonds variables whose some values give rewards or costs.

In order to optimize the behavior, we want to ensure it is a good behavior at any moment, in a pro-active way, relaxing the need for exact optimality. We claim that:

- A (non-optimal) good behavior can be achieved efficiently as sliding-horizon decision-making, where the decisions do not result from a complete policy, but are recomputed over a short horizon.
- This approach is robust even if the model is approximate.
- The structure of DBNs is well-adapted to do this.

4.2 Proposed algorithm

We propose here an efficient algorithm that exploits the structure of the model and the approximate inference mechanism described in section 3.1.

The key idea is to maintain a belief state over all variables of the network, and to project this belief state in the future in order to estimate the expected utility for every decision. The algorithm is shown on algorithm 4.

Algorithm 4: General scheme for quick decision making

```
1 sample current belief state;
2 forall possible decision do
3 |   infer particles until some time criterion knowing decision;
4 |   collect and memorize expected utility;
5 end
6 choose best utility;
```

The belief state is continuously approximated as a particle filter, and so re-sampling it, in order to decrease computational time, comes very easily. Then for every decision (which are in our models instances of controllable variables) this new particle filter is inferred without weighting and re-sampling, because future observations are not available. The particles can be propagated until some time limit (that should be greater than the date of the next decision). What seems to be

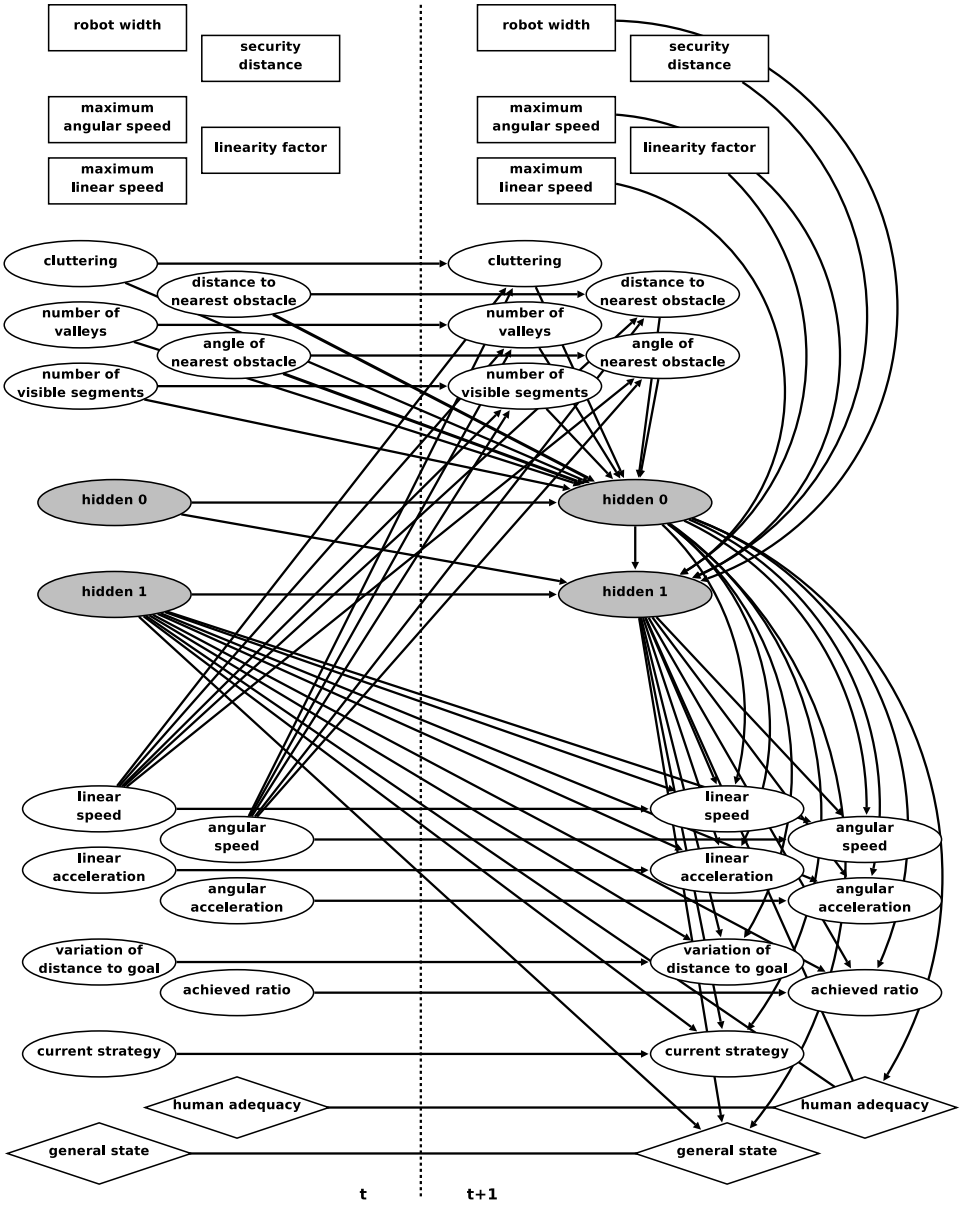


Figure 7: Dynamic Decision Network for optimization of robot behavior

a good time limit is the "death" of the particle, meaning that the particle reaches a final state of the whole system. This should avoid traps in the model where good utilities may lead to complete failure. The utility are simply collected by looking at transitions taken by the particle, and adding corresponding utilities. In our case, we call this algorithm fairly often, around every second. More often would be a waste of resources, because due to the dynamics of the whole system, the global configuration of the robot into its environment does not change more often than that.

4.3 Exploration versus exploitation in a structured space

In our implementation, we further qualified transition probabilities with a *confidence factor* in order to improve the decision algorithm. Indeed, when the data training set is too sparse and does not cover enough the possible observation set, many transitions in the DBN model are quantified by the learning algorithm but do not correspond to informative probabilities. Consequently, we qualified the transition probabilities with the number of time each transition is updated during the learning phase. The decision-making algorithm collects both utilities and confidence factors and uses these confidence factors to focus the search on decisions whose particles went through well learned transition and not through non informative transitions.

This leads to a decision-making procedure along two criteria: $Confidence$ and $Utility$. Relying on arbitrary weights for these criteria or even seeking a Pareto-like dominant decision did not seem appropriate in our setting. Instead, we divided possible decisions into several classes ordered by confidence level:

- if we have a class of high confidence, we discard all other classes and proceed as described earlier, i.e., choose decision that maximizes utility;
- when we do not have a high enough confidence class, several strategies are possible. For our experiments, we choose to avoid as much as possible failure states, choosing confidence as primary criterion, and utility as second, but long-term strategies may prefer to choose low confidence classes in order to be able to complete training set and improve future decisions.

Another way to understand why this mechanism is needed is through the notion of *generalization*. If we had a flat representation of the state space, using explicit states, this confidence measure would not be needed, as the evaluation of the utility would rely (during the Monte Carlo simulation) "real" states. But in our representation, when we infer particles to estimate expected rewards, we also take into account states that have not been necessarily encountered in training data, and eventually may not be possible at all.

While this may seem a drawback of the factored representation, it is in fact a small price for the significant benefit of the DBN representation that allows a direct generalization of value of states to neighbors. In real-world settings, since it is not possible to meet in training data every state a large number of times, one has to generalize observed rewards to neighboring states. In [39], the authors show two possible algorithms, known as the *structural credit assignment problem* aiming at explicitly generalizing the knowledge about rewards and costs. One is based on feature distance, the other is based on automated clustering of states. In our setting, the structural assignment is done implicitly through the structured representation, and every state transition implying the right variable (state component) transition will lead to the reward associated to this variable transition. But the generalization may be too strong, or over optimistic, and the confidence mechanism allows to weight this generalization, by giving explicitly the number of times the transition was really encountered. If it is zero (or low), then we know that the estimated reward value given by the algorithm is based mostly on generalized values, not on real values. On the contrary, if the confidence is high, then the estimation does not come from generalization of knowledge.

5 Empirical Evaluation

For the learning phase, we run over 120 navigation experiments of the robot into different environments, for a total of several hundred meters of motion into cluttered environments, with people around the robot. Navigations can take up to several minutes, and thus the robot may encounter very different types of environment within the same run, like narrow passages (corridor) then open area (empty room) or cluttered area (crowded for instance). Thus what we are looking for are not a parameter set for a given run but reactive change of parameters depending on the current environment in order to achieve good behavior. During the learning phase, we randomly choose the control parameters at random frequencies.

In order to characterize the run in a more fine-grained way than *failure* or *success* at the end, the operator was asked to give his perceived value of the **human adequacy** variable (ranging into *good*, *aggressive*, and *shy*). He also has a *fail* button to stop navigations before dangerous failure (e.g. collisions). Furthermore, a navigation timeout was automatically detected when the robot did not move during a given amount of time, and the *fail* value was given to the general state variable.

Typically, spending too much time in a *shy* state lead to a timeout failure, while too much time in the aggressive state lead to an emergency stop (and also a *fail* value to the general state variable). The idea behind this experimental protocol was to make a difference between the *aggressive* and *shy* states, that may be wanted in

some cases, and the *failure* states, which are always bad. For instance, in order to achieve a very efficient behavior in terms of average speed, it may be good to allow very aggressive behavior for a long time, changing parameters only if a collision is expected.

The total number of values collected for each variable is about 7,000. Still, this covers a very small part of the observation space (about 15×10^9). The learning of the DBN takes several minutes per iteration of EM. The learned DBN stabilizes after less than 10 iterations. All algorithms are implemented on-board.

5.1 The learned model

To evaluate the quality of the learned DBN, we repeatedly divided our experimental data into $(n-1)$ sets of training data and one set of test data. This was done n times, each set being used once as test data. We repeatedly trained the DBN on the $(n-1)$ training data, and compared the predictions using the learned model with the test data. For each test data, we used the learned DBN to predict at each timestep the next state, and we compared the predicted state to the actual one.

We compared our results to a simple "same-as-before" predictor, which predicts that the values of variables do not change. For a low dynamics, this apparently coarse predictor gives apparently acceptable average prediction values. This is mainly due to the fact that most often variables change one at a time (because they are loosely correlated), so all other variables are well predicted with this "same-as-before" assumption. The evaluation results are shown in table 1, where the first column is the variable name, the second the score of the "same-as-before" predictor, the third the percentage of correct predictions by the DBN and the last one the difference between the DBN prediction score and the "same-as-before" predictor. The upper part gives the predictions for the five environment variables, while the lower part lists the four robot configuration variables and the five mission state variables.

Considering these last variables, which are consequence of the navigation algorithm, we can see large improvement for the five robot configuration variables, and a smaller one for the four mission variables. This is mainly due to the fact that the "same-as-before" predictor is already very good for the four last variables. This is easily explainable because of the dynamic of the system: if the environment and speed chosen change often, this is not the case for the **percentage of mission accomplished**: it changes only 3 times during a whole run. The same phenomenon occurs for the **general behavior**: it has almost all the time the *normal* value. The adequacy to human also has most of the time the *okay* value, and finally the strategy chosen also does not change often, because it is almost independent of the distance to obstacles.

variable	same-as-before	dbn	improvement
cluttering	66.6	75.9	9.3
angle of nearest obstacle	58.7	72.8	14.1
distance to nearest obstacle	67.8	76.3	8.5
number of segments	58.0	66.7	8.7
number of valleys	59.7	66.2	6.5
v	55.4	80.3	24.9
ω	61.9	81.5	19.6
Δv	55.6	81.8	26.2
$\Delta\omega$	49.7	75.5	25.8
Δ distance to goal	64.4	85.4	21.0
% mission achieved	81.3	89.9	8.6
mission status	84.3	91.6	7.3
navigation strategy	70.2	83.9	10.7
human adequacy	74.4	85.3	10.9

Table 1: Table of well-guessed one-step predictions (%)

For the purpose of improving the robot behavior on the basis of the learned model, we are mainly interested in having good predictions for the mission state variables (as opposed to environment variables). The prediction results for all robot and mission state variables, but of the angular acceleration, are higher than 80 percent, with a quite good average result. This confirms that the behavior of the navigation task is reasonably well modeled and that the learned model can be effective in improving the robot behavior. We notice that even for the environment variables the learned DBN performs better than "same-as-before" predictor. This is due to the use of the previous speed references that lead the robot away from the nearest obstacle.

5.2 Defining different behaviors

Two behavior modes are available for this robot. The first one relies on static parameters that have been finely *hand-tuned* over weeks of experimentation in order to have a safe enough navigation to use the robot into a museum [1], with human beings around. The second mode is the one we used to collect the training set, i.e. setting parameters *randomly*, as briefly described at beginning of section 5: sets of parameters are chosen randomly (using uniform distribution) upon all possible sets at a random frequency. More precisely, parameters are changed every n seconds, n is a uniform random variable taking variables between 2 and 10 sec-

behavior	random	hand-tuned	efficient	friendly	mixed
success (%)	40	98	98	90	98
av. time (s)	74.5	32.3	25.4	30.0	30.0
stdd. time (s)	23.0	1.1	2.6	6.5	5.4

Table 2: Behavior results

onds; n is changed every time a parameter set is chosen (thus is not the same over a given run). The static *hand-tuned* parameter set behaves very well in dynamic environment (with many humans) and seldom fails in some static over-constrained environments. The *randomly* chosen parameter set fails quite often, as shown in the results below (See Table 2).

As explained in section 4 we consider the learned model as a dynamic decision network. For most applications, we will seek an *efficient* behavior, that minimizes the average navigation time, to finish the task as quickly as possible. In order to do this, we need to map rewards into the dynamic decision network accordingly. One solution would be to put a reward on high levels of instantaneous speed, hoping that it will lead to lowering average time taken. But in order to ensure not to fail, we would have to also put a reward on the *success* value of the **general state** variable. The precise value given to both these rewards would certainly have an influence on the behavior that would be hard to predict in advance. Furthermore, it not sure that giving rewards to instantaneous speed would lead to lowering the total navigation time, because it may lead to "almost dead-ends" situations (very close to obstacles) in which a very low speed is needed for a safe avoidance trajectory, whereas going slower may give it more time to change its sight and navigate further from obstacles[4]. In order to deal with this problem, we decided to give a reward to the *success* value. We further introduced a discount factor that gives higher weights to immediate rewards. This mechanism is similar to the one used for computing infinite-horizon policies in Markov decision processes. Thus, we use the learned model, augmented with reward on the *success* variable as defining a dynamic decision network; instead of extracting a static exhaustive policy from it, we apply algorithm described in section 4.2 in order to optimize the controller parameters. The final controller is not learned in the proper sense, but is improved with the learned model.

While the first behavior is *efficient*, we noticed that it is not really *human-friendly*: the robot takes high accelerations and passes close to obstacles and persons, which can be quite unpleasant. The solution is then to use the **human ade-**

[4]in our robotic setup, sight change is quite slow, and real speed reference given to the motors is a function of the maximum speed (the controllable variable) and distance to obstacles

quacy variable, that captures subjective feeling of the behavior of the robot, giving utility to *good* value and costs to *aggressive* and *shy* values (this last value is used when the robot has a too slow, boring and over cautious behavior). We could also give costs to low values of **distance to nearest obstacle** and high values of **linear acceleration**, but the adequacy variable, while subjective, also covers these cases. In the same vein, rewards could be given to high speeds in order to ensure efficiency, and cost to high accelerations in order to ensure smoothness of the trajectory[5]. In the experiments, we again used the algorithm from section 4.2 to tune the controller using the DDN defined with our learned DBN; we gave the same rewards to *success* values and to *good* values of human friendliness variable (and the opposite reward for *aggressive* value of friendliness).

Based on the observed behavior, we found that these two behaviors (*efficient* and *human-friendly*) may be mixed, but if their respective rewards are on the same scale, we have a multi-criterion decision problem, with several equilibria depending on the precise values given to the rewards. in the bad cases, the algorithm may for instance choose a parameter set that collects rewards having good human adequacy during several decision cycles then fails. To prevent this, reward for success has to be incommensurately higher than reward for correct human adequacy. This leads to implicit priority classes: the controller first chooses successful behaviors, and among these, the most human friendly ones. We set the reward associated to *success* to 10^6 (counted only once at the end of the trajectory) and the rewards associated with friendliness to 1 for *good* and -1 to *aggressive* and *shy* values that can be encountered only hundreds to a few thousands times during a navigation. In this way, we simulate a two-level decision process using only simple summation of rewards.

The precise values given to rewards have not been tuned at all, they just ensure the conditions explained above. Table 2 shows success rates and average time for a given navigation task for the 5 different behaviors. As expected, the two last ones show a more subjective, yet easily perceivable, friendly behavior.

One can notice that the hand-tuned behavior achieves very high success rate, and a good average time, compared to a random parameter affectation. But the *efficient* behavior achieves the same ratio, with a much better average time. This is mainly due to the fact that the hand-tuned parameters are static, given at the beginning of the mission, while our system adapt these parameters depending on the environment. While the hand tuned parameters ensure safety of the robot on every situation (cases of failure are when the robot gets stuck, not into collisions), our system allows a much more flexible adaptivity.

[5] this is reflected by the way the variables are built: speeds are measured as average speeds over a time window, whereas accelerations are a maximum value over the same time window

The *friendly* behavior shows worse performance, but had a much more pleasant behavior for the surrounding people. We were not able to quantify this precisely because our evaluation is subjective: measures would have requires statistical polls over a significant set of human observers. However, it was clear that most unpleasant motions of the robots were totally avoided. Indeed the *efficient* behavior showed many high accelerations and decelerations, specially close to obstacle, which make people uncomfortable for themselves and for the robot (more subjective observations are detailed on next section). The lower success rate is again not due to collisions, but to too cautious behaviors, in which the robot gets stuck because the set of parameters does not allow it to pass close to people. The "mixed" behavior showed a very good trade-off between the two previous behaviors.

5.3 Analysis

We would like to emphasize that without using the DBN structure presented in section 2.3, particularly without combining the environment variables into one hidden variable, the model would not even fit in the computer memory. Using what may appear as a complex causality structure allows us to easily handle several variables. That is because our model captures the influence of all environment and control variables (current and previous timesteps) over every consequence variable while automatically finding correlations through hidden variables. Without hidden variables, we would have too large probabilistic tables for the consequence variables, since the size of a table of a variable is exponential in the number of its parents.

The results of the controller were obtained on about 30 navigations. Clearly the subjective values of the variable **human adequacy** has an influence on the obtained behavior. In our training set, the operator introduced a bias by giving a *good* value for **human adequacy** very early when the robot accelerated out of a cluttered zone. This is mainly due to the fact that the environment was highly cluttered in our experimental setup, leading to very slow behavior of the robot when avoiding obstacles; the operator wanted the robot to accelerate as soon as possible. In our evaluation experiments, this behavior was clearly noticeable, while hard to measure precisely. The behavior of the robot somewhat reflected the impatience of the operator.

In summary, our experimental assessment clearly demonstrates that the explicit probabilistic models learned have good predictive capabilities. They can be used to fine-tune the robot behavior and adapt it to the subjective view of its users. These learned models allow to understand the robot behavior and better interact with it. They also endow it with adaptation capabilities that increase its robustness and simplify its development. Without such adaptation capabilities, good parameters typically require months of manual fine-tuning in various environments. Our mod-

els can do this automatically in a better way while increasing robustness to the variability of the environment and accommodation to users.

6 Discussion

As stated earlier, our objective here is twofold: *(i)* learn an explicit and predictive model of the robot behavior observed as a dynamic system, using its already available controller in a particular environment, and *(ii)* optimize the controller parameters for the specifics of the environment and user preferences. These objectives fits broadly into the following areas of machine learning techniques:

- learning probabilistic models, in our case of a dynamic system with partial observability, and
- reinforcement learning, understood in the broad sense of the set of techniques for learning actions, in our case the controller parameters, from environment feedback.

In the remaining of this section we will discuss successively our positioning with respect to the state of the art for these two objectives, then we will cover the issue of learning the structure of the probabilistic model, relevant for the first point, and the issue of active exploration while learning, which is relevant for the two objectives

6.1 Learning probabilistic models of a dynamic system

Learning the model of a dynamic system as a stochastic automata is a widely used approach. Many applications such as speech recognition are based on the HMM framework [52] that uses a *flat* set of states. This is a good solution when the state space can be given in advance. In some cases, using the same set for the observation space and the state space is also possible; this allows to use the state transitions to filter noisy sensor data.

The definition of a meaningful state space for modeling the behavior of an autonomous robot is however a tricky issue. An interesting approach has been developed in [22]. In this article, the authors define an a-priori a set of behaviors, such as “task in progress”, or “task stuck”. The set of states is obtained automatically using the training data that are qualitatively hand-labeled by an external observer with these a-priory defined behaviors. The process is further refined by clique extraction technique. This is clearly a way to define the state space and to introduce structure into the HMM framework. Typically, the HMM learning algorithms do not allow taking this structure into account.

In the general case, the state space does not need to be a flat set. It can be

structured, for example through a set of state variables. The influences between state variables are quantified from part of the state to others. This approach is know as factored models. It has been used in factored Markov decision processes [4, 5], and in dynamic Bayesian networks [11, 44]. While factored representations allow for expressing dependency relations between parts of the state space, the learning algorithms are not always able to easily take advantage of and learn these dependency relations. Several works have been done on automatic learning of state space structure, as we will see in section 6.3.

Other approaches address the problem of building stochastic models from experimental data. In [36], the authors learn human activities models and in [59] the emphasis is first, on building a stochastic model, then on using classic decision-making algorithm. Our approach tries to be more generic and to closely integrate the decision making and the learning algorithm itself.

A number of authors have been exploring a slightly different kind of relationships than conditional probabilities for building stochastic automata. These models are part of graphical models, but are not Bayesian models. They are Conditional Random Fields (CRF), introduced in [35, 66], and extended to dynamic systems in [60]. These families of models lead to simpler algorithms for inference and learning because they need less normalizations (both for learning and inference). That is, the correlations between linked variables are expressed with a set of potential functions that cannot be considered as joint probabilities until normalization. The main advantage is that these models allow to express more complex relationships (like higher order Markov dependency) while remaining tractable. The drawback is that one does not have direct access to conditional probabilities. Recent successful work using this kind of model includes [37].

More recently, a new non-parametric framework has shown promising results: the Gaussian Processes framework [53]. Within this framework, it is possible to have efficient filtering using a gaussian distribution over a space of gaussian probability laws, allowing to handle directly continuous variables. In [31], the authors show how to do efficient state estimation with a batch learned gaussian process. In [32], they extend this framework to learn gaussian processes with hidden variables that allow to filter sensor noise very efficiently. Gaussian processes representation of the state is a very promising approach. However, it can be used only for state estimation, it does not handle decision making. That is, some planning mechanism has to be put on top of it somewhat artificially, eventually in the form of a POMDP with external state estimation. An example of using a POMDP with external filtering mechanism (not a Gaussian process) is described in [55].

6.2 Optimizing the behavior through reinforcement learning

Reinforcement learning [62, 56] is one of the most popular and successful technique for synthesizing and improving a robot controller. It has been developed in a wide spectrum of situations, from simple subsumption architectures [38] to more complex robots [2, 49, 9] and tasks [68]. In a typical setting, the controller is expressed as an MDP. Reinforcement learning algorithms, such as TD(λ) [61] or Q-Learning [67], are used to estimate, explicitly or implicitly, the MDP parameters and utility function as well as a fixed optimal control policy [62]. In more changing environments and tasks, the policy can be re-assessed at each decision step, e.g. with dynamic programming techniques within a sliding horizon approach [42].

In order to further scale up MDP-based techniques, several authors have explored structured state spaces. For example, [19] considers a model of the world structured into sequences of states for each observation; this allows to infer non observable states in video recognition applications by assuming persistent and reliable observations. Relational Markov Decision Processes offer a more general representation where the set of possible states of the world is described in first order predicate calculus, with a closed world assumption; the set of feasible actions is also described in a relational form. Informally speaking, the representation combines classical planning for the structure of the state space, with MDPs for the non-deterministic action part. Relational reinforcement learning [15, 63] combines induction logic techniques and reinforcement learning techniques. It still raises a large number of open challenges [47].

Reinforcement learning however is not limited to learning MDP policies, in flat or structured spaces. It can be useful in a variety of contexts, e.g., [20] argues that reinforcement learning can be used to speed up inference in bayesian networks. Moreover, reinforcement learning is often stated in the very wide framework where an agent learns to improve its behavior by acting in the word and getting feedback and rewards. In this general sense, the part of this paper described in section 4 is a form a reinforcement learning, where the actions to be learned are the parameters of the controller. As stated earlier, our DDN representation is a factored partially observable MDP, to which reinforcement learning algorithms inferring near-optimal policies can be applied. The simpler and efficient algorithm we have developed maintains for all network variables a belief state approximated as a particle filter, and proceeds to a limited forward propagation to choose the actions (*i.e.*, parameter values) with maximum utilities. At the principal level and in a completely different context, this technique can be compared to Monte Carlo localization algorithms with particle filtering [21].

6.3 Learning the structure of the state and observation spaces

Structure learning is an open issue and a very active field of research. The machine learning community has recently come with very promising approaches (using recent developments in information theory) like in [17]. The algorithms generalize the expectation-maximization scheme in order to find an optimal structure while escaping local maxima. However they involve a very high computational cost.

A promising approach is to use mixed-initiative structure discovery, relying on human knowledge and automatic exploration. Purely automated techniques are tied to some optimization criteria, while graphical models owe their popularity to their more intuitive interpretation.

In approaches that rely on structured spaces, as in dynamic Bayesian networks, finding a minimal structure that encodes variable independence is a problem in itself. Work about learning Bayesian structure has long history. For only observable variables, a first exact search family gives very expensive algorithms, even for small structures [48, 58]. The K2 algorithm [10] acts as a local search in the structure space using a scoring criterion taking into account the explanation power of the network and its complexity. But this local search falls into multiple local optima. A third family of algorithms uses a local search into the equivalence space of networks [43, 45] in order to avoid falling in local optimum, but they are still very complex.

When some variables are not observed, the problem is even more complex because an algorithm can also add or delete a variable, and not only causal links. The algorithm has also to choose the arity of the hidden variable automatically. Automatic methods use mathematical criterions that show a trade-off between complexity of representation of the model itself and the accuracy of the prediction. Many metrics have been widely used, [69] compares some of them.

A generic local search technique has been proposed in [24], and various heuristics have been proposed to choose where to add hidden variables [18] and to choose the right arity of hidden variables [16]. The information-theoretic measure of mutual information is also very useful [25] to select where to add or delete causal links. A very promising method, described in [17], combines a local search of the structure, based on information-theoretic compression of the hidden variables allowing to predict accurately the observable variables, with the quantitative update of the transition model itself. This method allows to obtain a very efficient structure while escaping local optima of the quantitative learning.

Recently, the notion of mixed-observability MDP has been introduced in [46]. In this framework, the internal state is decoupled between observable parts and hidden ones; the structure proposed is close to ours. In this article, the authors use the observable part to represent the belief space by pieces (this can be seen

as discretizing the belief space using the discrete observable part), which allows them to solve the POMDP more efficiently than with only one large belief space. We preferred to develop a simple reactive approach for decision making due to the low dynamics of our system; but this clearly seems to be an interesting alternate direction.

6.4 Active exploration while learning

An important issue one has to consider when dealing with real-world applications is that all models, and in particular learned models, are approximations that need to be qualified. In our case, the size of the observation space is huge, while compared to the size of training data. Thus in the general case, we permanently need to improve our learning.

In section 4 we proposed a simple scheme to deal with unknown parts of the model. Such a procedure may be further explored along with other approaches, such as for example:

- the interesting techniques of [12] for assessing the uncertainty of the parameters of a model, and [13] for evaluating the value of learning, and computing future expected utilities;
- the algorithm E^3 [30] that builds another model on top of the (PO)MDP to solve while defining explicitly a set of known states, for which learning is not needed anymore, and an unknown state that covers all states of the (PO)MDP where learning still should be considered (this approach has been extended to factored models in [29]); this algorithm, called R_{max} [8] has been proposed to compute an asymptotically optimal policy using the model build by E^3;
- the approach of [26] to take the decision-making mechanism itself into account in order to guess were learning is needed for the decision making to behave better.

Interestingly, the EM algorithm family along with Monte Carlo sampling has also been used in [65] for model-free reinforcement learning, in order to obtain directly the policy without learning the model first. In [40], the authors use this kind of technique to optimize motion of a robot in order to minimize uncertainty of localization. They predict the information gain and the cost using the prior and so dynamically solve the exploration/exploitation dilemma in the context of path planning.

7 Conclusion

Complex systems, such as autonomous robots, are built from components and subsystems whose interactions and inter-operations are seldom fully understood. To better model and control the resulting systems, one need to learn their global behavior from observation data.

We proposed an approach based on probabilistic modeling using dynamic Bayesian and decision networks. The model relies on state variables that are either observable, hidden or controllable. As a result, we obtain a model which explains the considered robot behavior, but which also allows us to use the learned model to optimize the control parameters in order to improve its performance.

The proposed approach is generic and can be applied to a number of complex robotic behaviors. We illustrated it with an indoor navigation task in an open environment. We showed that the learned model explains the robot behavior with good predictive capabilities and allows to fine tune the control parameter with better performance than hand tuned parameters in terms of failure rate and elapsed time as well as user acceptance.

The proposed approach is generic enough to be applied to other tasks than autonomous navigation. The main difficulty remains to come up with the right DBN structure, and to find the characteristics of the environment that have an influence on the task. We now plan to use this approach to learn other robotic behaviors (e.g. grasping, human interactions, etc) and study how different models can be combined and jointly used.

References

[1] R. Alami, R. Chatila, S. Fleury, M. Herrb, F. Ingrand, M. Khatib, B. Morisset, P. Moutarlier, and T. Siméon. Around the lab in 40 days. In *IEEE ICRA*, 2000.

[2] K. Berns and T. Luksch, editors. *Autonome Mobile Systeme 2007, 20. Fachgespräch, Kaiserslautern, 18./19. Oktober 2007*, Informatik Aktuell. Springer, 2008.

[3] J. Binder, D. Koller, S.J. Russell, and K. Kanazawa. Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29(2–3):213–244, 1997.

[4] C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999.

[5] C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1-2):49–107, 2000.

[6] X. Boyen, N. Friedman, and D. Koller. Discovering the hidden structure of complex dynamic systems. In *Proceedings of the 15th Annual Conference on Uncertainty in AI (UAI)*, pages 206–215, 1999.

[7] X. Boyen and D. Koller. Approximate learning of dynamic models. In *Advances in Neural Information Processing Sytems (NIPS-11)*, 1998.

[8] R. I. Brafman and M. Tennenholtz. R-MAX — a general polynomial time algorithm for near-optimal reinforcement learning. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 953–958, 2001.

[9] A. Coates, P. Abbeel, and A. Y. Ng. Apprenticeship learning for helicopter control. *Communications of the ACM*, 52(7):97–105, 2009.

[10] G. Cooper and E. Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4), 1992.

[11] T. Dean and K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1990.

[12] R. Dearden, N. Friedman, and D. Andre. Model based bayesian exploration. In *UAI*, 1999.

[13] R. Dearden, N. Friedman, and S. Russell. Bayesian Q-Learning. In *Proceedings of the National Conference on AI (AAAI)*, 1998.

[14] R.O. Duda and P.E. Hart. *Pattern classification and scene analysis*. John Wiley and sons, Menlo Park, CA, 1973.

[15] S. Dzeroski, L. De Raedt, and K. Driessens. Relational reinforcement learning. *Machine Learning*, 43:7–52, 2001.

[16] G. Elidan and N. Friedman. Learning the dimensionality of hidden variables. In *Proceedings of Conference on Uncertainty in Artificial Intelligence (UAI)*, 2001.

[17] G. Elidan and N. Friedman. The information bottleneck EM algorithm. In *Proceedings of Conference on Uncertainty in Artificial Intelligence (UAI)*, 2003.

[18] G. Elidan, N. Lotner, N. Friedman, and D. Koller. Discovering hidden variables: A structure-based approach. In *Conference on Advances in Neural Information Processing Systems (NIPS)*, 2000.

[19] A. Fern and R. Givan. Relational sequential inference with reliable observations. In *ICML*, 2004.

[20] C. Fox, N. Girdhar, and K. Gurney. A causal bayesian network view of reinforcement learning. In *FLAIRS*, 2008.

[21] D. Fox, W. Burgard, F. Dellaert, and S. Thrun. Monte carlo localization: efficient position estimation for mobile robots. In *AAAI*, 1999.

[22] M. Fox, M. Ghallab, G. Infantes, and D. Long. Robot introspection through learned hidden markov models. *Artificial Intelligence*, 170(2):59–113, february 2006.

[23] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315:972–976, 2007.

[24] N. Friedman. The bayesian structural EM algorithm. In *Proceedings of Conference on Uncertainty in Artificial Intelligence (UAI)*, 1998.

[25] N. Friedman, K. I. Nachman, and D. Peïtr. Learning bayesian networks structure from massive datasets: The " sparse candidate" algorithm. In *Proceedings of Conference on Uncertainty in Artificial Intelligence (UAI)*, 1999.

[26] C. Guestrin, R. Patrascu, and D. Schuurmans. Algorithm-Directed Exploration for Model-Based Reinforcement Learning in Factored MDPs. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 235–242, 2002.

[27] K. Kanazawa, D. Koller, and S. J. Russel. Stochastic simulation algorithms for dynamic probabilistic networks. In *Proceedings of Conference on Uncertainty in Artificial Intelligence (UAI)*, 1995.

[28] T. Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, R. Silverman, and A.Y. Wu. An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, July 2002.

[29] M. Kearns and D. Koller. Efficient Reinforcement Learning in Factored MDPs. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 1999.

[30] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. In *Proceedings of International Conference on Machine Learning (ICML)*, 1998.

[31] J. Ko and D. Fox. Gp-bayesfilters: Bayesian filtering using gaussian process prediction and observation models. *Autonomous Robots*, 27(1):75–90, 2009.

[32] J. Ko and D. Fox. Learning gp-bayesfilters via gaussian process latent variable models. In *Proceedings of Robotics Science and Systems*, 2009.

[33] T. Kohonen. *Self-Organisation and Associative Memory*. S. Verlag, 1984.

[34] D. Koller and R. Fratkina. Using learning for approximation in stochastic processes. In *Proceedings of International Conference on Machine Learning (ICML)*, 1998.

[35] J. Laffery, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labelling sequence data. In *Proceedings of International Conference on Machine Learning (ICML)*, 2001.

[36] L. Liao, D. Fox, and H. Kautz. Learning and Inferring Transportation Routines. In *Proceedings of National Conference on Artificial Intelligence (AAAI)*, 2004.

[37] L. Liao, D. Fox, and H. Kautz. Localtion-based activity recognition using relational markov networks. In *Proceeding of International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

[38] S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. In *AAAI*, pages 768–773, 1991.

[39] S. Mahadevan and J. Connell. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence*, 55(2-3):311–365, 1992.

[40] R. Martinez-Cantin, N. de Freitas, E. Brochu, J. Castellanos, and A. Doucet. A bayesian exploration-exploitation approach for optimal online sensing and planning with visually guided mobile robot. *Autonomous Robots*, 27(2):93–103, 2009.

[41] J. Minguez, J. Osuna, and L. Montano. A "Divide and Conquer" Strategy based on Situations to achieve Reactive Collision Avoidance in Troublesome Scenarios. In *Proceedings of International Conference on Robotics and Automation (ICRA)*, 2004.

[42] B. Morisset and M. Ghallab. Learning how to combine sensory-motor functions into a robust behavior. *Artificial Intelligence*, 172(4-5):392–412, 2008.

[43] P. Munteanu and M.Bendou. The EQ framework for learning equivalence classes of bayesian networks. In *Proceedings of the IEEE International Conference on Data Mining*, pages 417–424, 2001.

[44] K. Murphy. *Dynamic Bayesian Networks; Representation, Inference and Learning*. PhD thesis, UCB, Berkeley, 2002.

[45] J. D. Nielsen, T. Kocka, and J. M. Pena. On local optima in learning bayesian networks. In *Proceedings of UAI*, 2003.

[46] S. C. W. Ong, S. W. Png, D. Hsu, and W. S. Lee. Pomdps for robotic tasks with mixed observability. In *Proceedings of Robotics Science and Systems*, 2009.

[47] M. Van Otterlo and K. Kersting. Challenges for relational reinforcement learning. In *ICML, Workshop on Relational Reinforcement Learning*, 2004.

[48] J. Pearl. *Causality: models reasoning and inference.* Cambridge University Press, 2000.

[49] J. Peters, S. Vijayakumar, and S. Schaal. Reinforcement learning for humanoid robotics. In *Humanoids2003, Third IEEE-RAS International Conference on Humanoid Robots*, 2003.

[50] J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence*, volume 18, pages 1025–1032, 2003.

[51] P. Poupart. *Exploiting structure to efficiently solve large scale partially observable Markov decision processes*. PhD thesis, University of Toronto, 2005.

[52] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.

[53] C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. The MIT Press, 2006.

[54] S. Russell and P. Norvig. *Artificial Intelligence: a modern approach*, pages 625–628. Prentice Hall, 2003.

[55] S. R. Schmidt-Rohr, S. Knoop, M. Lösch, and R. Dillman. Bridging the gap of abstraction for probabilistic decision making on a multi-modal service robot. In *Proceedings of Robotics Science and Systems*, 2008.

[56] W. D. Smart and L. P. Kaelbling. Effective reinforcement learning for mobile robots. In *ICRA*, pages 3404–3410. IEEE, 2002.

[57] M.T.J. Spaan and N. Vlassis. Perseus: Randomized point-based value iteration for POMDPs. *Journal of Artificial Intelligence Research*, 24:195–220, 2005.

[58] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction and Search*. Springer Verlag, 1993.

[59] F. Stulp and Michael Beetz. Optimized execution of action chains using learned performance models of abstract actions. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

[60] C. Sutton, A. McCallum, and K. Rohanimanesh. Dynamic conditional random fields: Factorized probabilistic models for labelling and segmenting sequence data. *Journal of Machine Learning Research*, 8:693–723, 2007.

[61] R. S. Sutton. Learning to predict by methods of temporal differences. *Artificial Intelligence*, 3:8–44, 1988.

[62] R.S. Sutton and A.G. Barto. *Reinforcement learning: an introduction*. MIT Press, 1998.

[63] P. Tadepalli, R. Givan, and K. Driessens. Relational reinforcement learning: An overview. In *ICML, Workshop on Relational Reinforcement Learning*, 2004.

[64] J.A. Tatman and R.D. Shachter. Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, 20(2):365–379, 1990.

[65] N. Vlassis, M. Toussaint, G. Kontes, and S. Piperidis. Learning model-free robot control by a monte carlo em algorithm. *Autonomous Robots*, 27(2):123–130, 2009.

[66] H. M. Wallach. Conditional random fields: An introduction. Technical Report MS-CIS-04-21, Dept. of Computer and Information Science, University of Pennsylvania, 2004.

[67] C.J. Watkins. *Models of delayed reinforcement learning*. Ph.D Thesis, Cambridge University, 1989.

[68] A. Wilson, A. Fern, S. Ray, and P. Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *ICML*, pages 1015–1022, 2007.

[69] S. Yang and K.-C. Chang. Comparison of Score Metrics for Bayesian Network Learning. *IEEE Transactions on Systems, Man and Cybernetics*, 32(3):419–428, may 2002.

[70] N.L. Zhang, R. Qi, and D. Poole. A computational theory of decision networks. *International Journal of Approximate Reasoning*, 11(2), 1994.

[71] G. Zweig and S. J. Russel. Speech recognition with dynamic bayesian networks. In *Proceedings of AAAI*, 1998.