

Learning Behaviors Models for Robot Execution Control

Guillaume Infantes and Félix Ingrand and Malik Ghallab

LAAS-CNRS 7, Avenue du Colonel Roche, 31077 Cedex 4, Toulouse, France

email: [infantes,felix,malik]@laas.fr

Abstract

Robust execution of robotic tasks is a difficult problem. In many situations, these tasks involve complex behaviors combining different functionalities (e.g. perception, localization, motion planning and motion execution). These behaviors are often programmed with a strong focus on the robustness of the behavior itself, not on the definition of a “high level” model to be used by a task planner and an execution controller. We propose to learn behaviors models as Dynamic Bayesian Networks. Indeed, the DBN formalism allows us to learn and control behaviors with controllable parameters. We experimented our approach on a real robot, where we learned over a large number of runs the model of a complex navigation task using a modified version of Expectation Maximization for DBN. The resulting DBN is then used to control the robot navigation behavior and we show that for some given objectives (e.g. avoid failure), the learned DBN driven controller performs much better (we have one order of magnitude less failure) than the programmed controller. We believe that the proposed approach remains generic and can be used to learn complex behaviors other than navigation and for other autonomous systems.

Introduction and Motivations

Tasks execution on autonomous robots is a very complex process. The building blocks of these plans, i.e. the tasks and actions, can be quite complex (we refer to them as behaviors). Moreover, often no explicit model exist of how it performs in various environments. Last, even if a deterministic model of these behavior exists, it may not be appropriate to handle the intrinsic non determinism of the environment, and of the behavior execution outcomes. As a result, one has to cope with a planning problem which has to plan actions execution with poor model or with no model at all.

Systems may be modeled as a Hidden Markov Model (HMM) [Fox *et al.* 2006]. In these models, the internal state is *hidden* to the observer, who can only see the *observation*, that represent effects of the system on the environment. Yet, this representation does not allow the finer control of the action execution. An aspect of complex actions we want to address is that some of these activities may be controllable. We propose to learn action models as highly structured stochastic processes: Dynamic Bayesian Network (DBN).

We detail how such a model can be obtained from a number of real-world runs. We then show how it can be used to

then control the action itself while executing. We also sketch how this could be embedded in a more general controller able to supervise the action execution (to avoid failure) and to decide when the model has to be refined for new situations arising. The paper is organized as follow. The next section presents a complex robot behavior to model and to adapt. We then present how we can learn such a structured stochastic model. The following section presents how we can use the learned model, followed by a section on results obtained on a real robot (i.e. the learning phase as well as controlling the robot using the learned model). We then conclude the paper with a discussion on the current work as well as some perspectives we are currently pursuing.

Modeling and Adapting Robot Behavior

To conduct our experiments, we modeled a robotic navigation task, based on ND reactive obstacle avoidance [Minguez, Osuna, & Montano 2004]. The laser range finder gives a set of points representing obstacles in front of the robot. This points are used 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 these data, it computes a speed reference that tends to move the robot towards the goal position while avoiding obstacles.

Structure of a Navigation Task

A navigation controller can be seen as a “black box” taking a relative goal position and some data about the environment as inputs and giving as output a speed reference for the robot to execute. We can model this by having an internal hidden state of the robot, with hidden variables. The control parameters of the navigation are observable and have an influence on the internal state. It changes the environment, because it is measured through the sensors of the robot: if the robot goes into a dead-end, the environment will seem very cluttered. The general structure can be seen on figure 1.

Choosing the variables is highly dependent on the process to model. The control parameters are given by the navigation controller, but we must choose wisely environment variables that represent important parameters for decisions made by the control process. In our reactive navigation task, the cluttering of the environment is to be taken into account, so is the closest obstacle, which has much influence on the

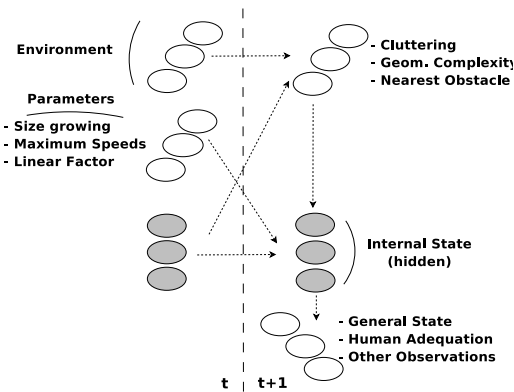


Figure 1: Abstract DBN structure for a navigation task

future sight of the robot. On the other hand, we avoid including the position of the robot, so that the model can be used in places different than the one where the learning was performed. The effects of the navigation task we need to recognize are the *fail* and *success* states, but we also aim at controlling more precisely the behavior in a qualitative way: some behaviors are successful, but not optimal. We also include as many variables as we can that could give us a hint on the internal state, for the learning process to be more effective. Finally, we could also model resources usage.

Instantiation

For our navigation task, the control parameters are: the two **size growing** parameters of the robot: one surface represents the robot (where obstacles are totally forbidden); and 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 environment variables chosen are: the **cluttering** of the environment, defined as a weighted sum of distances to nearest obstacles around the robot; the **geometrical complexity** of the environment; the **angle of the nearest obstacle**; the **distance to the nearest obstacle** and the **global number of segments** in the scene.

The output observable variables are: the current **linear and angular speeds** of the robot; its current **linear and angular accelerations**; the **variation of the distance to the goal** (estimated as an euclidean distance); the number of possible ways to go (**valleys**) built by ND; the **achieved ratio** of the mission; the **current strategy** chosen by ND; the **general state** in *begin, end, fail, normal*; the **human adequation** of the behavior¹.

The number of different potentially possible observations 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 probabili-

¹Some behaviors leads to success, but may be considered as too aggressive for humans around the robot, or on the contrary too shy, and thus to slow. Its value has to be given by the operator during the learning phase.

ties into the DBN. We will show later how to deal with such a sparse DBN for decision making.

DBN Learning

To learn a DBN, we use an adaptation of the classical Expectation-Maximization (EM) algorithm defined for Hidden Markov Models [Fox *et al.* 2006]. The adaptations to make this algorithm tractable for DBN are not straightforward: we need to maintain an approximated “belief state”, i.e. a probabilistic hypothesis over the possible assignment of the values of the variables because they are strongly correlated. We choose to approximate this belief state as a particle filter, with a variable number of particles.

Parameters Learning

A DBN λ is defined by a set of variables (hidden or observable), a set of possible values for each variable, and a set of probabilistic causal links between the variables [Dean & Kanazawa 1990]. An evidence O is as a sequence of observations (i.e. instantiations of observable variables of λ).

A good model λ with respect to O gives a high $P(O|\lambda)$ (likelihood of the evidence). In the DBN framework, λ may be adapted to the evidence either by modifying its structure or the probabilities attached to the causal links in a given structure. The EM algorithm can deal with hidden variables, but not with structure of the model.

In the HMM case, EM first compute probabilities of being in every hidden state, then every transition probability and every probability of seeing each observation from each hidden state, this for every time step. This is often referred as the Expectation step. For a DBN, the hidden state becomes an instantiation of the corresponding variables, and so does an observation. The state has to be approximated as a *belief state*, i.e. a set of weighted hypotheses over all possible instances. Then we can update the transition probabilities to maximize $P(O|\lambda)$. Then we go again into Expectation and Maximization steps. This algorithm is proved to reach a local maximum of $P(O|\lambda)$.

Choosing a good representation for the belief state is critical for the algorithm to behave correctly. A flat representation over the variables would loose all the correlations among them, being a very bad approximation. On the other side, maintaining an exact belief state imply to keep all correlations over time, and thus is exponentially complex over the length of the evidence. Approximating the belief state with a particle filter seems to be a good trade-off. But while the size of the hypothesis state is very large, we choose to have a variable number of particles, because the probability of seeing the observations knowing the belief state might become very low. We maintain this probability by creating more particles until this probability reaches its initial value.

Structure Learning

The EM algorithm updates only the transitions probabilities, without changing the structure of the DBN. An algorithm has been proposed [Friedman 1998] to add inside the loop some structural changes. The main issue is to evaluate the new model efficiently knowing the current one. Furthermore, this algorithm acts only as a local search into the

structure space, so the optimal structure may not be reached. The GES algorithm [Chickering 2002] has been proposed to find an optimal structure of a Bayesian network. However, this technique relies upon sufficient statistical knowledge of the process, which is not our case due to the very large size of the observation space. In our case, the global structure is known. In order to decrease the size of the DBN and speed up the learning process, we plan to use these techniques only on the hidden variables of the process (and on the links toward the observable variables).

Adaptation of the Behavior

One use of the model presented in section is to find optimal values for the ND motion generator controller, depending on the environment. ND is usually used with a fixed set of parameters, that are not optimal in every situation. We aim at having a fine-grained adaptation of these parameters, modeled as a DBN, to optimize the navigation itself.

We need to introduce utilities into some of our variables with respect to an optimality criterion. We need to give high rewards to variable values that will lead to a desired behavior and penalties to values that will lead to a bad one. Typically, we need to avoid the *fail* value of the **general state** variable, while we want to reach the *end* value.

We also introduce a secondary criterion on the **human adequation** of the navigation behavior. Between two successful behaviors, we prefer a behavior where the *normal* value of the **human adequation** appears often, and try to avoid the *aggressive* and *shy* values of this variable. All these utilities will be given as scalar values, so we need to give bigger values to the **global state** variable, and smaller ones to the secondary criterion, to avoid the case where the robot could collect reward by having a proper behavior with respect to humans, yet fail, thinking it is better to be gentle than achieving its task.

Decision

In this application, the temporal aspect is primordial. The command process works at a frequency of 2.5 Hertz. The behavior adapter has a frequency of 1.66 Hz to collect the observations, but we do not need it to change the parameters at such a high frequency. Typically the parameters changes should occur at most at 0.5 Hertz, or less. Otherwise, the system could demonstrate a very unstable behavior.

So the problem we face is quite different from a classical Dynamic Decision Network [Zhang, Qi, & Poole 1994] resolution, where a decision is taken after each observation, and the result is a decision tree which associates to each observation sequence an optimal decision sequence. Furthermore, the branching factor for a decision tree would be too high. We need a radically different strategy: we consider that when we take a decision, it will remain the same for a given amount of time. So we independently evaluate every decision towards a given horizon, and choose the best one.

The DBN we build includes a model of the evolution of the environment, thus we can accurately predict what will happen in the future. Starting from a belief state for current time step (i.e. a probabilistic hypothesis on the current state

of the system, including hidden variables), we can infer from the DBN the future belief states for each set of parameters and deduce the corresponding utilities.

The belief state is represented as a particle filter, each particle represents a weighted hypothesis on the values of the variables. When inferring the particles over time, we can compute the expected utility by simply summing utilities encountered by particles. This can be done for a few steps forward (which leads to imprecise expectations), or until all particles reach a final state (defined by the *end* and *fail* values of **general state**). The size of the estimation of the belief state (i.e. the number of particles used) is therefore a critical bottleneck for real-time decision making; to solve this issue, we sample the belief state to focus only on the few most probable hypothesis.

Meta Decision

We introduce a confidence factor in every transition probability as the number of updates of a transition probability during the learning phase. This helps us to differentiate an equiprobability due to a never seen transition into the data set from an informative one. When inferring particles to predict expected utilities of the decisions, we also collect the confidence factors of the transitions the particles used. If a particle comes with a high confidence factor, this means that this particle played again a learned set, whereas if the confidence factor is low, the particle went through never learned transitions. Thus the decision has to be taken in a two dimensional space: *utility* \times *confidence*. If we prefer to take a decision with high confidence, this will lead to play again a previously learned observation, and the robot will preferably exploit its model. On the contrary, if we choose a decision with a low confidence factor, this will lead go through never learned transitions of our model, making the robot more “exploratory” of its capabilities, with a higher risk of failure.

Results

Navigation Behavior Learning

The model management is implemented into a real-world robotic architecture, on a B21R 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.

The a priori adjacency structure of the graph is: in the same time step, every environment variable is linked to every hidden variable, every hidden variable is connected to every other hidden one, and finally every hidden variable is connected to every post-control observable variable; from step t to step $t + 1$, every environment variable is linked to itself, every control variable is linked to every hidden variable, and every hidden variable is connected to itself.

We did about 120 navigations of the robot into different environments, choosing randomly the control parameters at random frequencies; for a totalizing a few hundred meters of motion into cluttered environments, with people around the robot. The operator was asked to give the value of the **human adequation** (into *good*, *aggressive*, and *shy*). He also has a *fail* button to stop navigations before dangerous

failure like collisions. A failure case was automatically detected when the robot did not move during a given amount of time. The number of observations collected is about 7.000. Yet, it does not cover the total observation space at all. The learning of the DBN takes several minutes per iteration of Expectation-Maximization. The learned DBN stabilizes after less than 10 iterations.

To evaluate the quality of the learned DBN, we learn the DBN on all observations but one, and try to predict this observation. This is done by inferring a particle filter on the DBN, choosing the most probable observation in this particle filter, and comparing this predicted observation to the actual one. The global observation is seldom exactly the same, but is generally very close, that means that in most cases the particle filter predicts well most of the observable variables. An overview of recognition results is shown on table 2.

variable	random guess	dbn
cluttering	33.3	74.3
geom. complexity	33.3	48.4
angle of obstacle	25.0	70.1
dist. to obstacle	25.0	73.1
# segments	33.3	65.4
v	33.3	59.4
w	25.0	35.9
dv	33.3	89.0
dw	33.3	90.2
Δ dist. to goal	33.3	59.7
# valleys	33.3	49.8
% mission	33.3	44.4
general	25.0	95.6
adequation	33.3	49.3
average	30.9	64.6

Figure 2: Table of well-guessed one-step predictions (%)

We can notice that the recognition results are significantly better than a random prediction. Furthermore, if some aspects of the internal mechanisms of ND have not been well modeled (as the strategy choice, or number of valleys build) because of the lack of the environment variables that uses ND “for real”, the general behavior and the human adequation are well predicted. And these are the variables we use in particular for control.

Control

The decision level of the behavior adapter was implemented as follows: the current belief state is maintained along the navigation, and when a choice is to be made, it is sampled in order to limit the size of the particle filter for prediction.

In our control experiments, the choices were made at a fixed period of 2 seconds, changing reactive obstacle avoidance parameters dynamically during navigation, without stopping the robot motion. Therefore, the navigation behaves better than with default static hand-tuned parameters usually used on this robot, and much better than with random parameters. Typically, during the learning phase with random parameters, 1 run out of 3 ends on a failure state

due to collisions or to blocked situations, while with the behavior adapter, the failures happen only once out of 10 experiments. This happens while using a parameter set never tried during the learning phase (due to a confidence factor allowing learning more than using the model). Also, in the learning phase, we label as *good* the parameters producing very high accelerations when the robots enters an open area. We give rewards to this value for decision making, and this behavior is clearly noticeable in these experiments, while it is not with the default hand-tuned parameters. This gives us quicker navigation for open areas than with default parameters. The average speedup of the navigation will be quantified in a future work.

Conclusion

We have presented an innovative approach to learn DBN models of complex robot behaviors. In our context, where we want to learn a model where controllable variables remain observable, DBN are preferable to HMM. We developed a modified version of EM for DBN. We presented our behavior controller, and how it can be used to learn the model and then use it online. The approach has been implemented and successfully tested on a robotic platform. We showed that for a fairly complex navigation task, we were able to learn a model which when used to dynamically set the controllable parameters (according to given objectives) performs much better than the default set, or a random set.

On the long run, we keep in mind that these learned behaviors should not only be used to control the robot execution (to avoid failure and to optimize the execution), but can also be taken into account by a high level planner able to choose among a set of such behaviors.

Although our application involves a navigation task for a museum guide robot, we believe that the approach is applicable to other behaviors and could be used for other autonomous systems.

References

- Chickering, D. 2002. Optimal structure identification with greedy search. *Journal of Machine Learning Research* 3:507–554.
- Dean, T., and Kanazawa, K. 1990. A model for reasoning about persistence and causation. *Computational Intelligence* 5(3):142–150.
- Fox, M.; Ghallab, M.; Infantes, G.; and Long, D. 2006. Robot introspection through learned hidden markov models. *Artificial Intelligence* 170(2):59–113.
- Friedman, N. 1998. The bayesian structural em algorithm. In *Proceedings of UAI*.
- Minguez, J.; Osuna, J.; and Montano, L. 2004. A “Divide and Conquer” Strategy based on Situations to achieve Reactive Collision Avoidance in Troublesome Scenarios. In *Proceedings of ICRA*.
- Zhang, N.; Qi, R.; and Poole, D. 1994. A computational theory of decision networks. *International Journal of Approximate Reasoning* 11(2):83–158.