Learning Behaviors Models for Robot Execution Control

Infantes Guillaume and **Ingrand Félix** and **Ghallab Malik**[1]

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 structured stochastic processes: Dynamic Bayesian Network. 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, optimize speed), the learned DBN driven controller performs much better than the programmed controller. We also show a way to achieve efficient incremental learning of the DBN. We believe that the proposed approach remains generic and can be used to learn complex behaviors other than navigation and for other autonomous systems.

1 Introduction and Motivations

Tasks execution on autonomous robots is a very complex process: exploration rovers are expected to plan and properly execute their science observation missions; museum guide robots have to robustly execute their tour plans for visitors interested in a given subject; service robots must plan and execute daily activities for the elderly people they assist etc. The building blocks of these plans, i.e. the tasks and actions, can be quite complex. We refer to them as behaviors. These behaviors usually have an intrinsic complexity (e.g. navigation for a museum guide robot involve localization, perception, motion planning, etc.). Moreover, often no explicit model exists 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 as to cope with a planning problem where one must plan actions execution with poor model, or even, in some situation, with no model at all.

Some approaches [2] try to model actions deterministically, and apply a probabilistic bias afterward. This kind of approach may not be relevant for intrinsically stochastic systems, which are more common in real-world domain. Bayesian Networks [19] give to the authors of [9] the ability to model the actions and predict the effects of a manipulation task of a robot. But while this kind of model captures the stochastic nature of the system, it does not take into account its long-term dynamics.

Systems may be modeled as a Hidden Markov Model (HMM) [20]. 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. These stochastic models have proved quite adapted in domains such as speech recognition [20], modeling human activity [16], probabilistic plan recognition [5] and learning topological and metric maps [14]. In [10], the authors present an approach to models robot actions using HMM. Indeed, the resulting HMM actions can be used to recognize or to plan the modeled actions.

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. Finding the proper parameter values with respect to the current environment and goal can be quite a challenge if again no model of the underlying action is available. Hence we would like to learn action models rich enough to allow us to use them to perform any of recognition, planning, control or supervision of the action. We propose to learn action models as highly structured stochastic processes: Dynamic Bayesian Network (DBN) [8].

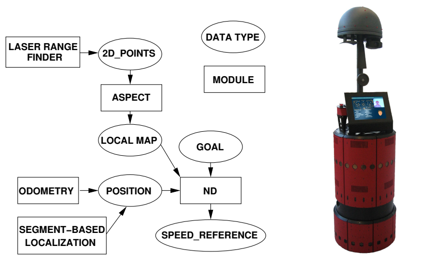
We detail how such a model can be obtained from a number of real-world runs. We then show how it can be used to control the action itself while executing. We also sketch how this could be embedded in a more general controller able to supervise the action execution (to avoid failure) and to decide when 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.

2 Modeling and Adapting Robot Behavior

To conduct our experiments, we modeled a robotic navigation task, based on ND[2] reactive obstacle avoidance [17]. A schematic view of this navigation modality can be seen on figure 1(a). 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 by the *aspect* module. 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, it computes a speed reference that tends to move the robot towards the goal position while avoiding obstacles. This modality is used on our B21R robot (figure 1(b)).

[1] LAAS-CNRS, 7, Avenue du Colonel Roche, 31077 Cedex 4, Toulouse, France, email: {surname.name}@laas.fr

[2] Note that despite the fact that ND was partially developed in our lab (over a number of year), it is so intrinsically complex, and so "difficult" to tune, that no model exists which could help us supervise, control and plan its execution.



(a) ND-based navigation modality (b) Platform

Figure 1. Rackham: An RWI B21R modified to be a museum guide robot [7]

2.1 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 the environment 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 2.

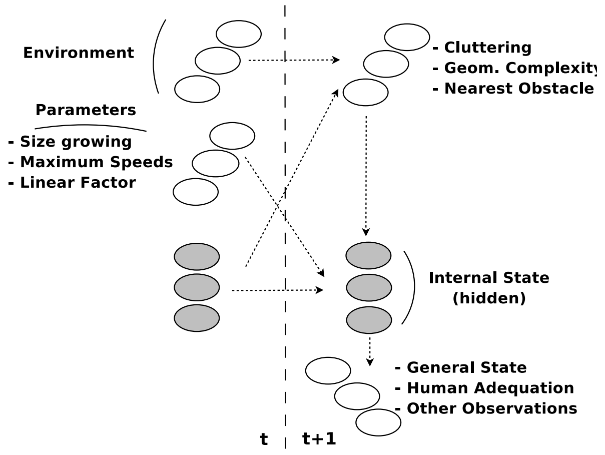


Figure 2. Abstract DBN structure for a navigation task

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 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 from 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 resource usage.

2.2 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.[3]

The environment variables chosen are: the **cluttering** of the environment, defined as a weighted sum of distances to nearest obstacles around the robot; the **angle of the nearest obstacle**; the **distance to the nearest obstacle**; the global **number of segments** in the scene and the number of possibles ways to go (**valleys**) built by ND.

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 **achieved ratio** of the mission; the **current strategy** chosen by ND; the **general state** in *begin*, *end*, *fail*, *normal*; the **human adequacy** of the behavior[4].

All the variables are discretized (into clusters from 3 to 6 values). 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 probabilities into the DBN. We will show later how to deal with such a sparse DBN for decision making.

3 DBN Learning

To learn a DBN, we use an adaptation of the classical Expectation-Maximization (EM) algorithm defined for Hidden Markov Models [20]. 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.

3.1 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 [8]. An evidence O is as a sequence of observations (i.e. instantiations of all 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

[3] The linear and angular speeds are mutually constrained by a polynomial function that influences the type of trajectory produced.
[4] Some 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.

as a *belief state*, i.e. a set of weighted hypotheses over all possible instances. Furthermore, the observable variables are not decoupled from the hidden ones, so we compute an observation probability not only from hidden variables, but from the whole belief state. 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 lose all the correlations among them, being a very bad approximation. On the other side, maintaining an exact belief state implies keeping all correlations over time, and thus is exponentially complex over the length of the evidence [4]. 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 changing the number of particles in order to have always the same probability of seeing the observation [15].

3.2 Structure Learning

The EM algorithm updates only the transitions probabilities, without changing the structure of the DBN. An algorithm has been proposed [11] 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 [6] 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),. For now, we use the structure described in section 5.1.

3.3 Incremental Learning

An open issue for learning stochastic models is the possibility to add knowledge into a previously learned model. This is difficult because the learning phase is a local search, and the algorithm must take into account all observations with the same weight.

We propose the following algorithm: we first learn λ_1 upon a training sequence $(O_1..O_i)$. Then we learn λ_2 over $(O_j..O_n)$, *taking* λ_1 *as a start point for the local search of* λ_2 [5]. When its done, we simply sum λ_1 and λ_2. For this merging to work, it must be applied after *each iteration* of EM, i.e. the merged DBN must be used for Maximization and not only for Expectation. We think this is because the role of the hidden variables may be changed otherwise, loosing the "meaning" of them. If the merging is done only between the different phases, the newly learned DBN optimizes itself only in the space of the current trunk of observations and goes somewhat "too far" from the previous one. In this case, the sum of the two DBNs does not reflect the reality of the space of the sum of the observations. This will be investigated in future work.

[5] Note that λ_1 was learned from a random starting point. Thus the next phases of the learning should be better than the first, because the start point is not random, but somewhat represents a part of the dynamic of the system.

4 Adaptation of the Behavior

One possible use of the model presented in section 2.1 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 adaption of the parameters of the controller, 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 adequacy** of the navigation behavior. Between two successful behaviors, we prefer a behavior where the *normal* value of the **human adequacy** 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 greater values to the **general 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 to achieve its task.

4.1 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 [23] 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 the 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.

4.2 Meta Decision

We introduce a confidence factor in every transition probability. This confidence factor is 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 re-played in 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 re-play 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. We encounter here the classic learning agent "exploration versus exploitation" trade-off, in an original fashion.

5 Results

5.1 Navigation Behavior Learning

The model management is implemented into a real-world robotic architecture [1], 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 [13] in order to obtain a few clusters.

The a priori adjacency structure of the graph is the following:

- 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 connect to every post-control observable variable;
- from step t to step $t+1$, every observable 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 total of 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 adequacy** (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. As EM is a local search, we use a "conservative" DBN as a starting point, meaning that with this DBN the probability that the variables change their value is low (about 10%).

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 3. 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 adequacy are well predicted. And these are the variables we use in particular for control.

variable	random guess	dbn
cluttering	33.3	87.3
angle of obstacle	25.0	86.4
dist. to obstacle	25.0	88.5
# segments	33.3	81.6
# valleys	33.3	79.9
v	33.3	90.8
w	25.0	79.0
dv	33.3	91.9
dw	33.3	92.2
Δ dist. to goal	33.3	77.8
% mission	33.3	97.1
strategy	20.0	84.8
general	25.0	96.4
adequacy	33.3	92.5
average	29.0	87.6

Figure 3. Table of well-guessed one-step predictions (%)

Using more hidden variables gives better results, meaning that 2 hidden variables with an arity of 2 is not enough. But using more hidden variables makes the learning process much longer because of the "complete" structure we use. Here we consider 49 causal links, while with 3 variables it comes up to 73. We intend to decrease this amount of links using structure learning in future work.

5.2 Control

The decision level of the behavior adapter was implemented using OpenPRS [12]. 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[6]. 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.

Collecting observations can be made in the same time than active behavior adapting; the next step is to decide when the robot should use computed behavior or behaviors that extend the variety of the set of collected observations.

[6] This is due to a confidence factor allowing learning more than using the model.

5.3 Incremental Learning

To evaluate the incremental learning algorithm, we learned a DBN in five phases as explained in section 3.3. Each phase uses 20% of the total observation. On figure 4, the phases are numbered on the X-axis. The Y-axis represents the log-likelihood obtained. The upper line shows the log-likelihood obtained with a single phase of learning (upon all observations). While the summed DBN does not behaves as well as the one learned in a single phase, the difference is very small.

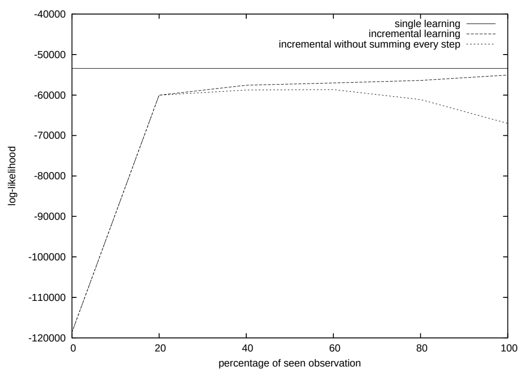


Figure 4. Incremental learning likelihoods

6 Conclusion and Discussion

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.

This work is closely related to [16], where the authors tries to model human activities with hierarchical DBN. Our approach tries to deal with more variables, and with sparse data sets. Furthermore, we propose a direct way to use our model not only for diagnosis, but to control the process itself. In [18], the authors learn to recognize robotic behaviors that fits with human judgment, but the emphasis is more on “how to built variables that make a intuitive sense” than on use of this kind of model. The proposed approache is a more classical planning one [21]. We propose an efficient way for mixing learning and decisions making. Our approach is in the same spirit than [3], where the authors model the system as neural nets or with tree induction, then take decision based on a MDP, for robotic autonomous navigation. We propose a more integrated approach, where the decision-taking mechanism is much closer from the model itself. Furthermore, it allows taking into account hidden variables, i.e. a more complex structure of the model itself. We propose a way to build a model using more features than the one proposed. In [22] the authors present an original way of optimizing tasks execution. High-level tasks are built with a number of parameters that must be optimized for smooth execution. Performance models are learned for the parameter sets as tree rules, but much of the work is hand-made. Our approach is automated, and the models we build are more expressive (while harshly human-readable), due to the presence of hidden variables. Furthermore, the models we built are fully probabilistic, while this is not the case in this work. Both approaches show very high computational cost when different complex tasks have to be mixed.

In 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.

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

REFERENCES

[1] R. Alami, R. Chatila, S. Fleury, M. Herrb, F. Ingrand, M. Khatib, B. Morisset, P. Moutarlier, and T. Simon, ‘Around the lab in 40 days...’, in *Proceedings of ICRA*, (2000).
[2] E. Amir, ‘Learning partially observable deterministic action models’, in *Proceedings of IJCAI*, (2005).
[3] T. Belker, M. Beetz, and A. Cremers, ‘Learning action models for the improved execution of navigation plans’, *Robotics and Autonomous Systems*, **38**(3-4), 137–148, (2002).
[4] X. Boyen and D. Koller, ‘Tractable inference for complex stochastic processes’, in *Proceedings of UAI*, (1998).
[5] H. Bui, ‘A General Model for Online Probabilistic Plan Recognition’, in *Proceedings of IJCAI*, (2003).
[6] D. Chickering, ‘Optimal structure identification with greedy search’, *Journal of Machine Learning Research*, **3**, 507–554, (2002).
[7] A. Clodic, S. Fleury, R. Alami, M. Herrb, and R. Chatila, ‘Supervision and interaction’, in *Proceedings ICAR*, pp. 725–732, (2005).
[8] T. Dean and K. Kanazawa, ‘A model for reasoning about persistence and causation’, *Computational Intelligence*, **5**(3), 142–150, (1990).
[9] A. Dearden and Y. Demiris, ‘Learning forward models for robots’, in *Proceedings of IJCAI*, (2005).
[10] M. Fox, M. Ghallab, G. Infantes, and D. Long, ‘Robot introspection through learned hidden markov models’, *Artificial Intelligence*, **170**(2), 59–113, (february 2006).
[11] N. Friedman, ‘The bayesian structural em algorithm’, in *Proceedings of UAI*, (1998).
[12] F. Ingrand, R. Chatila, R. Alami, and F. Robert, ‘PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots’, in *Proceedings of ICRA*, (1996).
[13] 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).
[14] S. Koenig and R. G. Simmons, ‘Unsupervised Learning of Probabilistic Models for Robot Navigation’, in *Proceedings of ICRA*, (1996).
[15] D. Koller and R. Fratkina, ‘Using learning for approximation in stochastic processes’, in *Proceedings of ICML*, (1998).
[16] L. Liao, D. Fox, and H. Kautz, ‘Learning and Inferring Transportation Routines’, in *Proceedings of AAAI*, (2004).
[17] 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 ICRA*, (2004).
[18] T. Oates, M. D. Schmill, and P. R. Cohen, ‘A method for clustering the experiences of a mobile robot that accords with human judgements’, in *Proceedings of IJCAI*, (2000).
[19] J. Pearl, *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*, Morgan Kaufmann, 1988.
[20] 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).
[21] Matthew D. Schmill, Tim Oates, and Paul R. Cohen, ‘Learning planning operators in real-world, partially observable environments’, in *Proceedings of ICAPS*, (2000).
[22] F. Stulp and Michael Beetz, ‘Optimized execution of action chains using learned performance models of abstract actions’, in *Proceedings of IJCAI*, (2005).
[23] N.L. Zhang, R. Qi, and D. Poole, ‘A computational theory of decision networks’, *International Journal of Approximate Reasoning*, **11**(2), 83–158, (1994).