Robust Navigation using Markov Models

Julien Burlet, Olivier Aycard, Thierry Fraichard

Proceedings of IEEE/RSJ International Conference on
Intelligent Robots and Systems, August 2-6 2005. Edmonton,
Alberta, Canada.

Robust Navigation using Markov Models

Julien Burlet, Olivier Aycard and Thierry Fraichard
Inria[a] Rhône-Alpes & Gravir[b]
655 av. de l'Europe, Montbonnot, 38334 St Ismier Cedex, France
http://www.inrialpes.fr/sharp

13/06/2005

Abstract — To reach a given goal, a mobile robot first computes a motion plan (*ie* a sequence of actions that will take it to its goal), and then executes it. Markov Decision Processes (MDPs) have been successfully used to solve these two problems. Their main advantage is that they provide a theoretical framework to deal with the uncertainties related to the robot's motor and perceptive actions during both planning and execution stages.

While a previous paper addressed the motion planning stage [1], this paper deals with execution stage. It describes an approach based on Markov localization and focuses on experimental aspects, in particular the learning of the transition function (that encodes the uncertainties related to the robot actions) and the sensor model. Experimental results carry out with a real robot demonstrate the robustness of the whole navigation approach.

Autonomous Navigation, Mobile Robots, Markov Localization ...

[a]Institut National de Recherche en Informatique et en Automatique.
[b]Lab. Graphisme, Vision et Robotique.

Robust Navigation using Markov Models

Julien Burlet, Thierry Fraichard[1] and Olivier Aycard[2]
Inria Rhône-Alpes & Gravir Lab., Grenoble (FR)

***Abstract*— To reach a given goal, a mobile robot first computes a motion plan (*ie* a sequence of actions that will take it to its goal), and then executes it. Markov Decision Processes (MDPs) have been successfully used to solve these two problems. Their main advantage is that they provide a theoretical framework to deal with the uncertainties related to the robot's motor and perceptive actions during both planning and execution stages.**

While a previous paper addressed the motion planning stage [1], this paper deals with execution stage. It describes an approach based on Markov localization and focuses on experimental aspects, in particular the learning of the transition function (that encodes the uncertainties related to the robot actions) and the sensor model. Experimental results carry out with a real robot demonstrate the robustness of the whole navigation approach.

***Index Terms*— Autonomous Navigation, Mobile Robots, Markov Localization**

I. Introduction

By design, the purpose of a mobile robot is to move around in its environment. To reach a given goal, the typical mobile robot first computes a motion strategy, *ie* a sequence of action that will take it to its goal, and then executes it. Many researchers have studied these two problems since the late sixties-early seventies. In 1969, [2] introduced a planning approach based upon a graph representation of the environment whose nodes corresponds to particular parts of the environment, and whose edges are actions to move from a particular part of the environment to an other. A graph search would return the motion strategy to reach a given goal. Since then, different types of representations of the environment and different planning techniques have been proposed (for instance, motion planning computes a motion, *ie* a continuous sequence of positions, to move from one position to an other [3]), but the key principle remains the same.

The decoupling between the planning stage and the execution stage relies on the underlying assumption that the robot will be able to successfully execute the motion strategy computed by the planning stage. In most cases, this assumption is violated unfortunately, mostly because actions are *non deterministic*: for various reasons (*eg* wheel slippage), a motion action does not always take the robot where intended. To overcome this problem, mobile robots are equipped with different sensors in order to perceive their environment and monitor the execution of the planned motion. Then techniques known as localisation techniques are used to solve the problem at hand [4]: they are based on probabilistic models of actions and perceptions and rely on Kalman filters [5], [6]. On the other hand, since the early nineties, approaches based on Markov Decision Processes (MDP) [7] have been used to address both motion planning and motion execution problems [8]. Such approaches also use a graph representation of the robot's state space and their main advantage is that they provide a theoretical framework to deal with the uncertainties related to the robot's motor and perceptive actions during both planning and execution stages.

This paper follows upon a previous paper addressing the planning stage [1]. It introduced a MDP-based planning algorithm that features hierarchical decomposition of the state space and actions better taking into account the kinematics of wheeled robots : As a result, motion planning is faster and the plans produced more robust. This paper addresses the execution stage using a model derived from MDP called *Markov localization* which allows to estimate the position of a robot taking into account uncertainties on the robot's actions and perceptions [9].

This paper focuses on the experimental aspects related to the use of Markov Localization with a particular emphasis on how two key elements of Markov localization were obtain by learning, namely the transition function (that encodes the uncertainties related to the robot actions) and the sensor model. By modeling and integrating the robot uncertainties both in the planning and execution stage, we obtain a robust method to address the problem of robot navigation.

The paper is organized as follows: section II recalls the planning method while section III describes the execution stage. Section IV presents the learning of different parameters needed by Markov localization and an example of execution. Conclusions and future perspectives are given in section V.

II. Planning stage

In this section, we give a brief description of the planning stage defined precisely in the previous paper [1] in order to give the framework in which we place ourselves. In particular, we explain the choices made to model robot's environment and robot's actions. The first part quickly presents the Markov Decision Processes model. Then the second part explains how we have defined the parameters of a Markov Decision Process adapted to our planning problem. Next

[1]Research Associate at Inria.
[2]Associate Professor at Joseph Fourier University, Grenoble (FR).

part shows a result obtained with our planning method. Conclusion on planning stage is given in the last part.

A. Markov Decision Processes

In the planning stage, we use Markov Decision Processes (MDP) to model the robot which interacts with its environment. It is defined as a 4-tuple $< S, A, T, R >$ with: S a finite set of states characterizing the environment of the robot, A a finite set of actions which permits the transition between states, $T : S \times A \times S \longrightarrow [0, 1]$ the state transition function which encodes the probabilistic effects of actions (we note $T(s, a, s')$ the probability to go from state s to state s', when action a is performed), $R : S \longrightarrow \mathbb{R}$ the reward function used to specify the goal the agent has to reach and the dangerous parts of the environment.

In MDP, we suppose that the agent **knows at each instant its current state**. Actions must provide all the informations for predicting the next state. Once the set of states S has been defined and the goal state chosen, we compute an *optimal policy* π that gives the optimal action to execute in each state of S in order to reach the goal state(s) (according to a given optimality criterion).

B. Definition of MDP parameters

1) States Definition: A quadtree decomposition is used to determine the states of the robot. This is a geometric and hierarchical decomposition of robot's environment. The quadtree decomposition yield a finite set C of rectangular cells with different size. To define a state, the robot's orientation is taken into account: the $[-\pi, \pi]$ orientation range is discretized and a state s is defined as follows: $s =< c, o >$ with $c \in C$ and o is a subrange $[-\pi, \pi]$. In our case, we have eight orientation subranges in order to have a good compromise between complexity and realism. When the robot is in a state $s =< c, o >$, we consider that it is in the middle of c with the orientation o whatever its exact position in c and its exact orientation (which is in $[o - \frac{\pi}{8}, o + \frac{\pi}{8}]$ since we consider eight orientations).

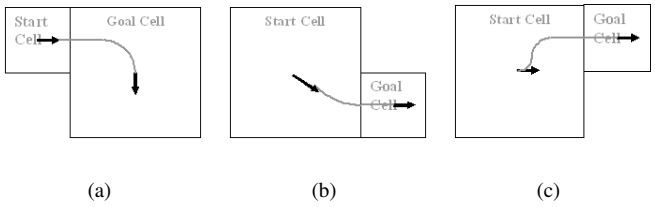


Fig. 1. Examples of Dubins actions.

2) Actions Definition: Actions of the MDP are classically defined as translations and rotations on the spot (in the next part of the text we call this actions: classical actions). Furthermore in this work we have introduced a novel type of actions defined by a sequence of motions along straight segments and circular arcs. Such actions are henceforth called Dubins actions as per [10] who introduced them for car-like robots. Actions composed by only one motion along straight segments or a circular arc are called atomics actions.

Given two adjacent states $s =< c, o >$ and $s' =< c', o' >$, the problem is to compute the Dubins action allowing the robot to reach c' with the orientation o', starting from c with the orientation o, without leaving c and c'. Since, such a Dubins' action does not always exist, we also consider the classical actions for the sake of completeness (a classical action between two adjacent cells always exists).

Fig. 1 depicts several examples of Dubins actions. Depending on the respective sizes and positions of the start and goal cells, a Dubins actions is made up of a finite number of straight segments and circular arcs.

3) Transition Function: A transition function was defined according to states and actions. This function encodes in a probabilistic manner the non-deterministic effects of each action defined.

Due to quadtree decomposition the number of cells arrangement is unpredictable. Also the diversity of actions is important. This is why we had to compute the transition function in two steps. In a first time, we model the uncertainty of actions independently from cells arrangements and for each *"atomic action" (i.e translations and circular motions)* composing the actions. This uncertainty is obtained by learning or could be fixed a-priori. In a second time, we combine uncertainty of each *"atomic action"* to obtain the uncertainty of entire action, and replace this uncertainty into the context of states defining robot's environment. Thus, we can obtain the transition function for every Dubins actions and for every cells' arrangement.

4) Gain function: A simple gain function permits to distinguish the goal state from other states and permits to compute a policy.

C. Example of policy computed

Figure 2 shows a computed policy. Black cells correspond to states including obstacles. And white cells correspond to states in the free space. The goal corresponds to the light gray cell. Each light grey arrows represents on-the-spot rotation. Dubins actions are represented by black segments and circular arcs (with arrowheads attached to show the orientation).

When the policy is computed, we assign to each state an action which is the optimal action in order to reach the goal. We have said that a state is defined as a couple <cell,orientation> and that we consider eight orientations' range. So, on the figure 2, we have eight states for one cell, thus there are eight actions per cell. Each action corresponds to the optimal action for one state.

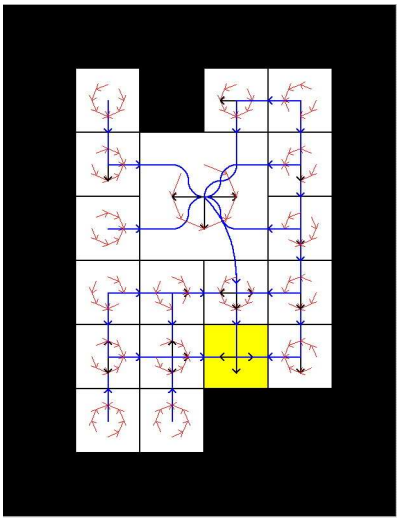


Fig. 2. Policy computed for an environment modeled by 216 (27×8) states

D. Conclusion

We have briefly defined in this section the approach we use to address planning stage. The optimal policies generated by our motion planner will be used during the execution stage. The next parts of the paper will explain the approach we used to address this second stage, and especially how the assumption that the agent knows at each instant its current state will be left.

III. Execution stage

In this section, we describe the method we have used to address the execution stage. Basically, our approach is based on a set of belief states which is updated during execution using Markov Localization. This set permits to determine the state in which the robot is most likely to be and then the policy computed in the planning stage gives the action the robot must execute.

A. Introduction

The planning stage produces an optimal policy π which gives the optimal action to execute in each state of S in order to reach a goal state. The purpose of the execution stage is to ensure that the robot reaches its goal using this optimal policy. At each step of this stage, the robot makes an observation, and its current state s_c is computed using this percept (Localization). Then once s_c is computed, the robot executes the action given by $\pi(s_c)$. This cycle is repeated until the robot reaches the goal. Let us see now, how the localization of the robot in its environment is made.

B. Localization

The purpose of localization is to determine the current state of the robot using its perceptive capacities. In order to determine this current state, we use Markov Localization to obtain at each time t the distribution probability over the set of states S which models the robot's belief on its current state knowing the observation until time t and actions done until time $t-1$. Let L_t be a random variable representing the state of the robot at time t, we note $Bel(L_t)$ this distribution probability. So, $Bel(L_t = s_i)$ denotes the probability of the robot being in state s_i at time t. Once we have computed this distribution, we choose the most probable state as the current state of the robot (if there are several states with maximum probability, one of them is selected randomly).

The initialization of $Bel(L_0)$ depends on the knowledge about the starting state :

- If the starting state is known with absolute certainty : $Bel(L_0)$ is a Dirac distribution centered at the starting state. In this case, we talk about *"tracking problem"*.
- If the starting state is unknown : $Bel(L_0)$ is a uniform distribution. We talk about *"global localization"*.

C. Markov Localization

Markov Localization [11] provides a general probabilistic framework well-suited for estimating the global position of a mobile robot based on actions and observations. It is a straight-forward application of state estimation within "Partially observable Markov decision processes" (POMDP) [12] in which the agent is a mobile robot and a state is a position of the robot within its environment. Markov localization permits to update $Bel(L_t)$, knowing that perception O_t was obtained at time t by the robot, action a_{t-1} has been performed at time $t-1$ and $Bel(L_{t-1})$. Markov localization gives for each state $s_i \in S$:

$$Bel([L_t = s_i]) = \frac{P(O_t|[L_t=s_i]) \sum_{s_j \in S} T([L_{t-1}=s_j], a_{t-1}, [L_t=s_i]) Bel([L_{t-1}=s_j])}{\sum_{s_k \in S} Bel([L_t=s_k])}$$

$Bel(L_t = s_i)$ is given by: 1) a prediction stage: $\sum_{s_j \in S} T([L_{t-1} = s_j], a_{t-1}, [L_t = s_i]) Bel([L_{t-1} = s_j])$, and 2) an estimation stage: $P(O_t|[L_t = s_i])$). The term $\sum_{s_k \in S} Bel([L_t = s_k])$ is the normalization term.

To use Markov localization, we need to determine $P(O_t|[L_t = s_i])$ called the sensor model and $T([L_{t-1} = s_j], a_{t-1}, [L_t = s_i])$ that corresponds to the transition function defined in the planning stage. Typically these two elements of the Markov localization permits to take into account the uncertainty on robot's perceptions and robot's displacements. They depend on the robot and are defined by learning. In the next section we will see how these elements are defined relatively to our experimental platform.

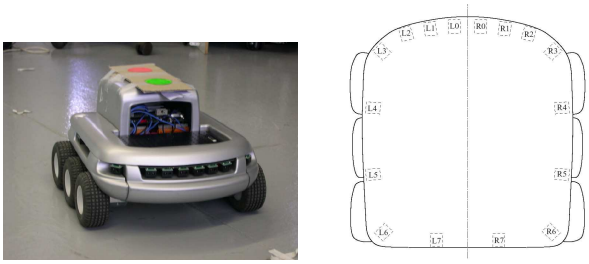
IV. Experimentations and Results

Evaluating our planning and execution approaches on a real robot is an essential step to prove their efficiency and robustness. To apply our method on a real robot we need to define a sensor model and an action model which both

depend on the robot's characteristics. In this section we will see how these two elements are defined and what results we have obtained.

A first part describes the experimental platform which permits us to validate our work. The second part is dedicated to the learning of transition function, while learning of sensor model is present in the third part. The fourth part presents an example of execution.

A. Experimental platform



(a) (b)

Fig. 3. The Koala robot and the infrared sensors layout

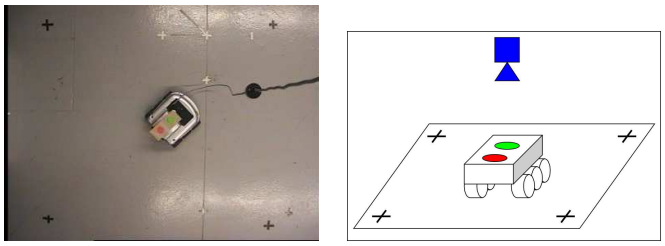
In our experimentations we use a robot called Koala[1] in a static known indoor environment cluttered with polygonal obstacles. The Koala robot is a mid-size robot (about 30 per 30 centimeters, Fig. 3(a)) with six wheels differentially driven (thus it is able to perform on-the-spot rotations). It is equipped with sixteen infrared proximity sensors (Fig. 3(b)). These sensors have a very limited range of perception: they detect obstacles in front of them at a distance of about fifteen centimeters, and within a field-of-view of ten degrees. Also the sensors' response is very noisy reducing their reliability.

B. Learning the transition function

We have seen that the transition function encodes in a probabilistic manner the non-determinist effects of the actions performed by the robot. In order to apply our method on the Koala, we need to learn this function. The purpose of this learning step is to measure and model the uncertainty of the actions executed by the Koala.

To do so, we let the robot perform several times a given set of atomic actions (i.e straight line and circular motion). For each run, we measure the difference between the theoretical and the actual configuration reached. To measure this difference with high accuracy, we have used a camera placed on the ceiling and two colored reference marks placed on the Koala to track its actual configuration (Fig. 4). Using those measures a function which models uncertainties is computed for each atomic action. We choose this function as a gaussian and so we obtain a covariance matrix C_j for each atomic action j. Figure 5 depicts an example of uncertainty function obtained for an circular motion. Finally we obtain the transition function for a Dubins action by combining the uncertainty functions of each atomic actions composing it.

[1] http://www.k-team.com/



(a) (b)

Fig. 4. Experimental set-up used to learn the transition function

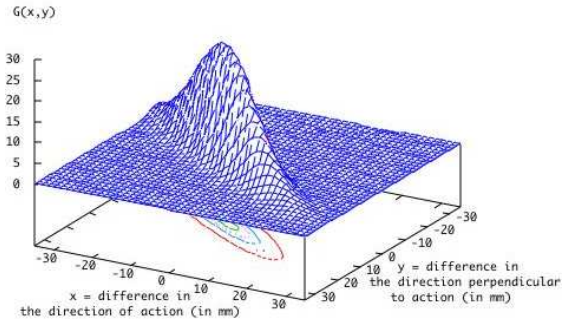


Fig. 5. Transition function of an arc-of-circle motion obtained after learning

C. Learning sensor model

To use Markov localization it is necessary to define a sensor model in order to obtain $P(O_t|[L_t = s_i])$ (the probability of doing the perception O_t knowing that the robot is in state s_i at time t). The Koala is equipped with sixteen infrared proximity sensors, so a Koala's perception corresponds to the perceptions of these sixteen sensors and O_t is a sixteen dimensional vector.

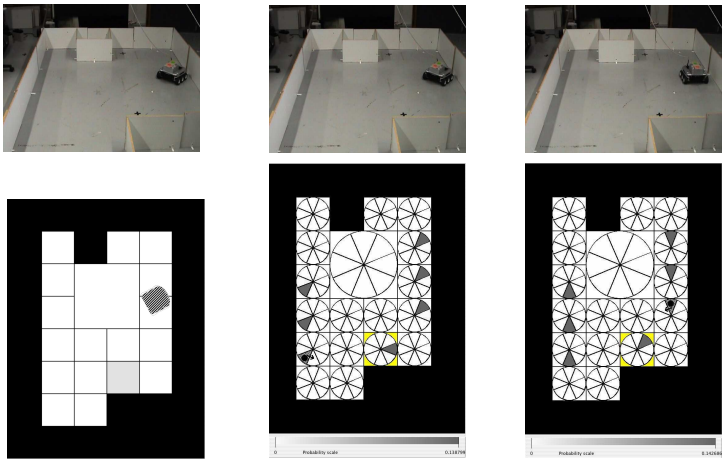
To learn the sensor model, a set of states with typical obstacle placement is defined first depending on the environment considered. For instance, the state where the robot has a single wall in front of it. Then learning is done for each state in this set. Learning is performed by randomly drawing a set of configuration corresponding to the typical state considered and recording the sensor measurements. A sixteen dimentional gaussian has been select to model the data obtain.

D. Example of execution

In this section, we show an example of execution using our approach. In this example, the Koala evolves in a static

environment depicted in figure 6(a), and its goal is to reach the cell in grey. Here, we are interested in a problem in which we suppose that the robot doesn't know its starting configuration, i.e a global localization problem.

During execution we show, after each time the robot has done a percept, the real position of the robot (given by a photo) and the distribution $Bel(L_t)$ over the set of state. So, the probability $Bel([L_t = s])$ which represents the robot's belief that it is a state $s =< c, ori >$ at time t is depict by a slices included in cell c and covering the range ori (hence we have eight slices in a cell because we consider eight subranges of possible orientations). The color of a slice depends on the value of $Bel([L_t = s])$: the highest the probability is, the darkest it is represented. For more details, a scale is given below each environment's representations (this scale depend on the best probability we have). Also, if there are no slices corresponding to a state, this mean that the probability of being in this state is null. The state choosen to be the current state of the robot (among the states with maximum probability) is showed with a black spot and the optimal action attached to it is represented by a black arrow, and was obtained using the optimal policy. The policy computed for the environment we choose is given by figure 2. Below, we describe in detail an execution in this environment and figures 6, 7, 8, 9 depict the different steps of this execution.

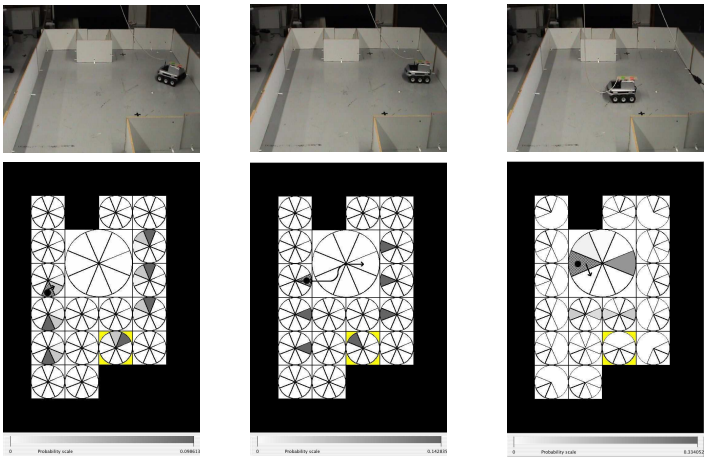


(a) (b) (c)

Fig. 6. Execution steps at times t = 0 (left), t = 1 (center) and t = 2 (right)

At time $t = 0$, before the first percept, the robot is put in an arbitrary configuration (Fig. 6(a)), and the distribution $Bel(L_0)$ is set to uniform. The goal cell is depict in light gray.

Then at time $t = 1$ a first perception is made and $Bel(L_0)$ is updated to obtain $Bel(L_1)$. The robot believes that there is a wall on its front right and noting else around (Fig. 6(b)). So, it doesn't know exactly where it is, but it believes that it must be in specific states. The current state among the high-probably states is choose to be the high-probably state at the bottom left. So, the robot will do the optimal action (a rotation on the spot to the left depicts by the dark arrow) attached to the current state, according to optimal policy (Fig. 2).



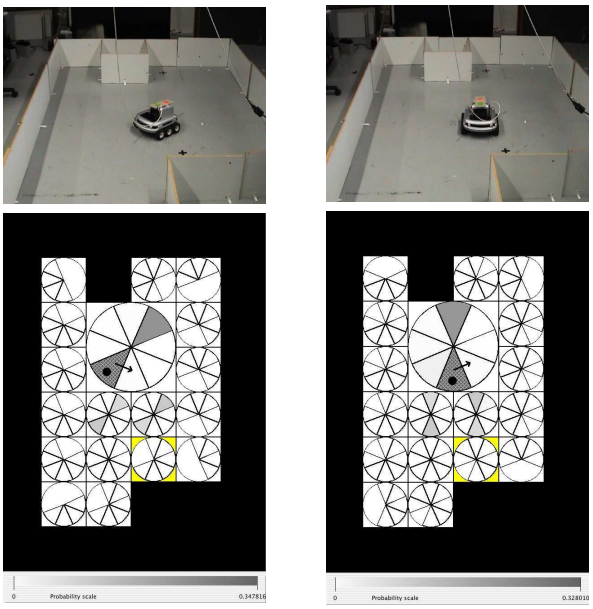
(a) (b) (c)

Fig. 7. Execution steps at times t = 3 (left), t = 4 (center) and t = 5 (right)

From time $t = 2$ to time $t = 4$, the robot has done rotations on the spot and has made new percepts. Thus its belief has been updated according to both action and sensor models. At time $t = 3$ the perception doesn't match very well the action model : it knows that it has done a rotation, but the response of the sensor model indicates a wall at its right. Because of its arbitrary starting orientation, its actual orientation is not perfect, so different states match the percept. But at time $t = 4$ the perception done and the model action applied confirm its real configuration : it believes that there is a wall on its back.

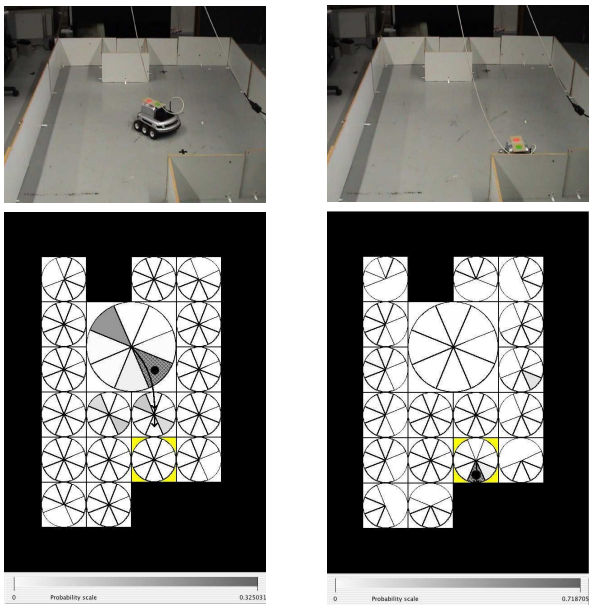
At time $t = 5$ the robot is in the whole place and its sensors can't sense any obstacle due to their short range of perception. It doesn't know if it comes from East or West. At this time, the real configuration of the robot corresponds to its believes, but it is not on the center of the big cell, it is on the South border of this cell oriented to the West.

From time $t = 6$ to time $t = 8$ it continues to execute actions and to update its belief knowing it is still in the center of the environment. And finally at time $t = 9$, it has performed the action and made a perception that it is front of a wall. Knowing it was somewhere in the whole space of environment facing of South-East, and it has now a wall on front of it, it's sure at 71 percent that it has reach the goal. The execution is a success since it has really reached the goal.



(a) (b)

Fig. 8. Execution steps at times t = 6 (left) and t = 7 (right)



(a) (b)

Fig. 9. Execution steps at times t = 8 (left) and t = 9 (right)

V. Conclusion and Perspectives

We have presented a complete method to address the robot navigation problem. Our method takes into account the uncertainties on robot's actions and perceptions that permit to raise the robustness and reliability. Learning the transition function and sensor model has allowed the implementation of our approach on a real robotic platform, allowing us to validate our work. Experimental results show than our method is well suited to address the robot navigation problem.

The next step of this work is to develop re-planing of actions. In the planing stage, we suppose that actions start in the middle of a cell and stop in the middle of another cell (this is the assumption made in MDP), in practice this is not the case. The purpose would be to modify this action during execution, in function of percept, so that an action starting anywhere in a state, stopping as close as possible in the middle of the stop cell.

Acknowledgments

This work was partially supported by CNRS (Centre National de la Recherche Scientifique) and DGA (Délégation Générale pour l'Armement).

References

[1] J. Burlet, O. Aycard, and T. Fraichard, "Robust motion planning using markov decision processes and quadtree decomposition," in *IEEE International Conference on Robotic and Automation*, 2004.

[2] N. J. Nilsson, "A mobile automaton: An application of artificial intelligence techniques," in *Proc. of the 1st IJCAI*, Washington, DC, 1969, pp. 509–520.

[3] J.-C. Latombe, *Robot Motion Planning*, ser. International Series in Engineering and Computer Science; Robotics: Vision, Manipulation and Sensors. Boston, MA, U.S.A.: Kluwer Academic Publishers, 1991, 651 pages.

[4] J. Borenstein, B. Everett, and L. Feng., *Navigating Mobile Robots: Systems and Techniques.* A. K. Peters, Ltd., Wellesley, MA: Kluwer Academic Publishers, 1996.

[5] R. Kalman, "A new approach to linear filtering and prediction problems," *Trans. of the ASME, Journal of basic engineering*, vol. 82, pp. 35–45, 1960.

[6] P. S. Maybeck., *Stochiastic Models, Estimation, and Control, Volume 1*. Academic Press, Inc, 1979.

[7] R. Bellman, J. Holland, and R. Kalaba, *On an Application of Dynamic Programming to the Synthesis of Logical Systems*. ACM Press, 1959, vol. 6, no. 4.

[8] S. Koenig and R. Simmons, *Xavier: A Robot Navigation Architecture Based on Partially Observable Markov Decision Process Models*, R. B. D. Kortenkamp and R. Murphy, Eds. MIT Press, 1998.

[9] W. Burgard, D. Fox, D. Hennig, and T. Schmidt, "Estimating the absolute position of a mobile robot using position probability grids," in *AAAI/IAAI, Vol. 2*, 1996, pp. 896–901.

[10] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *ajm*, vol. 79, pp. 497–517, 1957.

[11] D. Fox, W. Burgard, and S. Thrun, *Active markov localization for mobile robots*, 1998.

[12] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman, "Acting optimally in partially observable stochastic domains," in *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, vol. 2. Seattle, Washington, USA: AAAI Press/MIT Press, 1994, pp. 1023–1028. [Online]. Available: citeseer.ist.psu.edu/cassandra94acting.html