Proceedings of the 2004 IEEE
International Conference on Robotics & Automation
New Orleans, LA • April 2004

Robust Motion Planning using Markov Decision Processes and Quadtree Decomposition

Julien Burlet, Olivier Aycard[1] and Thierry Fraichard[2]
Inria Rhône-Alpes & Gravir Lab., Grenoble (FR)
{julien.burlet & olivier.aycard & thierry.fraichard}@inrialpes.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. This paper describes a MDP-based planning method that uses a hierarchic representation of the robot's state space (based on a quadtree decomposition of the environment). Besides, the actions used better integrate the kinematic constraints of a wheeled mobile robot. These two features yield a motion planner more efficient and better suited to plan robust motion strategies.**

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, [1] 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 [2]), 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 [3]: they are based on probabilistic models of actions and perceptions and rely on Kalman filters [4], [5]. On the other hand, since the early nineties, approaches based on Markov Decision Processes [6] have been used to address both motion planning and motion execution problems [7]. 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. Unfortunately, their algorithm complexity is exponential in the number of edges of the graph which limits their application to complex problems. Research have been carried out in order to address this complexity issue by reducing the number of states through aggregation techniques [8]. This paper falls into this category, it describes a MDP-based planning method that uses a hierarchic representation of the robot's state space (based on a quadtree decomposition of the environment). Besides, the actions used better integrate the kinematic constraints of a wheeled mobile robot. These two features yield a motion planner more efficient and better suited to plan robust motion strategies.

The paper is organised as follows: the next section presents the MDP model and the quadtree decomposition. Section III describes in detail the approach proposed while section IV presents experimental results. Conclusions and future perspectives are given in section V.

II. Path Planning Methods

This section presents the two methods used in our approach: Markov Decision Processes and quadtree decomposition.

A. Markov Decision Processes

1) Definition: a Markov Decision Process (MDP) models an agent which interacts with its environment. It is defined as a 4-tuple $< S, A, T, R >$:

- S is a finite set of states characterising the environment of the robot in our case. S is usually obtained by a regular decomposition of the environment or thanks to a topological map;
- A is a finite set of actions which permits the transition between states. There is generally a discrete number of actions.
- $T : S \times A \times S \longrightarrow [0, 1]$ is the state transition function which encodes the probabilistic effects of actions; $T(s, a, s\prime)$ is the probability to go from state s to state $s\prime$, when action a is performed.

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

- $R : S \longrightarrow \mathbb{R}$ is the reward function used to specify the goal the agent has to reach and the dangerous parts of the environment. $R(s)$ gives the reward the agent gets for being in state s.

2) Optimal Policy: in MDP, the agent knows at each instant its current state. Actions must provide all the information for predicting the next state. Once the set of states S has been defined and the goal state chosen, an *optimal policy* π 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).

The two most important algorithms used to calculate the optimal policy are: *Value Iteration* [6] and *Policy Iteration* [9]. The Value Iteration algorithm proceeds by little improvement at each iteration and requires a lot of iterations. Policy Iteration however, yields greater improvement at each iteration and accordingly needs fewer iterations, but each iteration is very expensive.

Complexity results for this algorithms can be found in [10]. Each iteration is achieved in $|S|^3 + O(|A||S|^2)$ for Policy Iteration and $O(|A||S|^2)$ for Value Iteration. The number of iterations needed to converge is quite difficult to determine. both algorithms seems polynomials in $|S|$ and $|A|$ [10].

B. Quadtree Decomposition

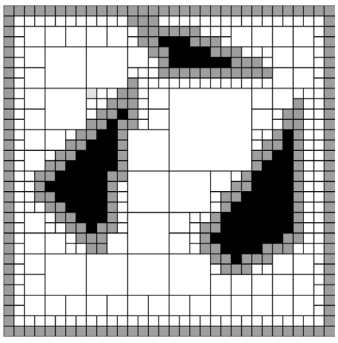


Fig. 1. Quadtree decomposition of a 2D environment: mixed cells are grey whereas full cells are black.

To decompose the environment, we use quadtree decomposition. It permits an approximate but fast and efficient modelling of the robot's 2D environment,. The principle of the quadtree decomposition is to recursively divide the environment in four identical square cells . Each cell is labelled as being " free" if there is no obstacle inside, "full" if it is filled with an obstacle and "mixed" otherwise. Mixed cells are divided again in four and the process goes on until a given resolution is reached. Fig. 1 depicts the result of the quadtree decomposition of a 2D environment: The number and size of the cells depends on the environment's characteristics.

III. Description of the Approach

Basically, our approach uses a quadtree decomposition to define the set of states of a MDP (so as to reduce the number of states). The decomposition is also used to define actions that better integrate the kinematic constraints of a wheeled mobile robot.

A. States Definition

As mentioned earlier, quadtree decomposition is used to determine the states of the robot. The quadtree decomposition of the robot's environment yield a finite set C of rectangular cells (Fig. 1). The size of the smallest cell corresponds to the robot size since it does not make sense to consider smaller cells. Moreover, the goal cell is chosen to have the minimum size (*ie* the robot size) to ensure that the robot will reach the goal with high accuracy. To define a state, the robot's orientation is taken into account: the $[-\pi, \pi]$ orientation range is discretized and a state 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 so as 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).

B. Actions Definition

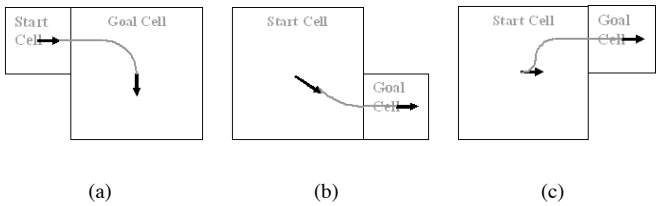


Fig. 2. Examples of Dubins actions.

"Classical" actions in MDP for mobile robots are of the type "initial rotation on the spot, straight motion and final rotation on the spot". To better account for the kinematic constraints affecting wheeled mobile robots and to limit the slippage effect stemming from on-the-spot rotations, 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 [11] that introduced them for car-like robots.

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. 2 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.

C. State Transition Function Definition

1) Introduction: in MDP, the transition function encodes in a probabilistic manner the non deterministic effects of actions. Due to the quadtree decomposition, number and diversity of

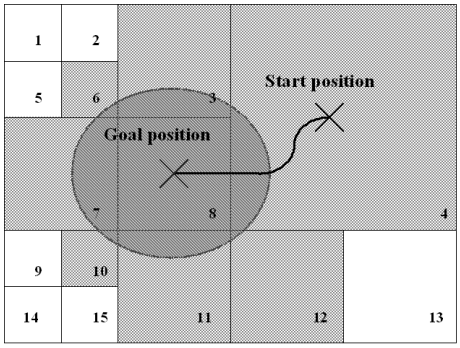
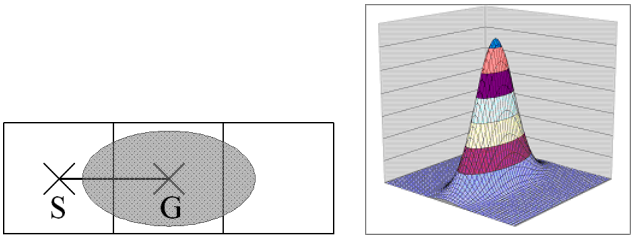


Fig. 3. How the transtion function is computed.



(a) (b)

Fig. 4. Uncertainty model for a translation.

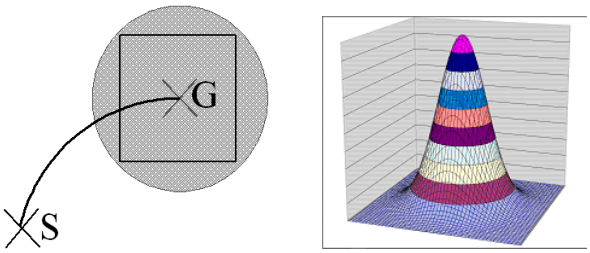
actions is finite. Furthermore, the number of possible cells' arrangements is considerable, that introduce problems to define the set of reachable states. So, we can not define state transition function like it's done in the other works [12], [13]. We need to abstract cells' arrangement and sequence composing the action. In next section, we present our method to compute the state transition function.

2) Principle: in the next part of the paper, we call *configuration* a triple <x,y,θ> where (x,y) is a geometric position in the environment and $\theta \in$ [-π,π] corresponds to the robot's orientation in the environment. We call $Conf$ the set of configurations.

Fig. 3 illustrates the principle we use to compute transition function. Let Cw be the wished configuration after the robot has done the action a from configuration Str. On the figure, a is in black, Cw correspond to left cross, and Str at the right cross. We define an intermediate function I depending on the action, the starting configuration and the wished configuration, modelling the incertitude on the wished configuration. We note $I : C \longrightarrow \mathbb{R}$ this function, and we call it *uncertainty function*. The elliptical dark grey disk on the figure 3 showing the set of configurations for which I is greater than ε with ε close to zero (to more visibility, we do not consider orientation on the figure). So the probability to reach a configuration in this dark grey area after the action a is performed is not negligible. A state s has a chance to be reach, if and only if, there exists a configuration c such as $c \in s$ and $I(c) \geq \varepsilon$. In practise, the set of cells having a probability of being reached is show in striped grey, and correspond to the intersection of C and the dark grey disk. After, we used I to compute the probability of reaching each states in striped grey, after action done starting from Str .

3) Uncertainty Function Definition: the purpose is to model by a function I the uncertainty on configuration after the action done. We define I as a Gaussian, since intuitively, this type of function represents well the type of uncertainty we have to face to. Indeed, the probability of reaching a position close to the wished configuration is high, even though the probability of reaching a position rather away from the wished configuration is nearly null.

So, defining the uncertainty function I consists of determining the parameters of the Gaussian. This parameters are defined by the action executed and the wished configuration. Actually, the probability of reaching the wished configuration is the highest, so the expectancy of I must correspond to the wished configuration. Also, more the action is complex and long, more the uncertainty is high, so the action will permit to define the covariance matrix (I has tree parameters :x,y,θ).



(a) (b)

Fig. 5. Uncertainty model for a circular arc motion.

We have seen that Dubins actions combine translation and circular motions. Before defining uncertainty function (precisely its covariance matrix) for any Dubins actions, we first define uncertainty function for elementary actions, *ie* translation and circular motions. The covariance matrix for this two types of actions is defined by learning or could be given a priori.

We now illustrate our uncertainty models in Fig. 4 and 5. In these figures, we do not consider orientations to clarify our presentation permitting to show a two dimensional representation of the set of states and configurations (we only consider cells and position, like in Fig. 3), and to consider I as a two parameters function.

Fig. 4 illustrates the uncertainty model for an elementary translation. Fig. 4(a) shows the set of possible positions (the elliptical grey disk) we obtain after the robot has executed the action in black from the left cross to reach the wished configuration (the right cross). The figure 4(b) shows the uncertainty function I which characterises the probability of reaching each position. Intuitively, it seems that the uncertainty on the position is most important in the translation axe than in the axe perpendicular to translation axe. Besides, the

covariance matrix depends on actions features, and so, the shape of I shows well this feature. Most the translation is long, most the uncertainty area is large, so most elements of covariance matrix are defined big. Also we see that even if the probability of reaching the wished configuration is weak, the probability of reaching the wished cell is sizable.

Fig. 5 illustrates the uncertainty model for a circular arc motion. Fig. 5(a) shows the set of possible positions (the grey disk) we obtain after the robot has done the action in black from the down left cross to reach the wish configuration (the up right cross). Fig. 5(b) shows the uncertainty function I which characterise the chance of reaching each position. It is most difficult to determine the uncertainty area for an arc of circle. Ref. [14] shows that this area could be approximated by a disk of centre corresponding to the wished position and depend on angle and radius characterising the arc. The symmetry of I shows this feature. Also, as in the translation case, we see that even if the probability of reaching the wished configuration is weak, the chance of reaching the wish cell is sizable. But there is more chance to reach an adjacent cell in the case of arc of circle than in the case of translation. Arc of circle displacement introduce more uncertainty than translation.

Then, the uncertainty function of actions composed by several translations and/or several arc of circle actions is simply obtained as follow : the expectancy stay the wished configuration, and covariance matrix M is defined by summing the covariance matrix of each element e_i of the sequence composing the action : $M = \sum M_{e_i}$. If the sequence of translation and arc of circle is important, the uncertainty on the final configuration is high. For example, if we look at the action of Fig. 3, it is composed by two arc of circle displacement and one translation. The expectancy corresponds to the gaol cross, and the covariance matrix is the sum of arcs covariance matrix and translation covariance matrix.

4) State Transition Function Computation: once I, the intersection of I with S, the set of states, are defined, we can assign a value at each state. This value is the probability for each state to be reached from a given state performing a given action. A state could be seen as a set of configurations, so it does not cause any problems to do intersection of I with S.

If we do not consider orientation, and take figure 3 as an example, we obtain the probability of reaching each cell (each state, but we abstract from orientation) after action done :

TABLE I

PROBABILITY OF REACHING A CELL

cell number	1	2	3	4	5	6	7	8
probability	0.0	0.0	0.03	0.14	0.0	0.01	0.13	0.64
cell number	9	10	11	12	13	14	15	
probability	0.0	0.01	0.03	0.01	0.0	0.0	0.0	

If we look at table I, we see that the the probability of reaching the wished cell (cell number 8) is 0.64, and the probability of staying in the same cell is 0.14 (cell 4) and so on for other cells.

So we obtain transition function because we compute the probability of reaching each state, using uncertainty function I which is defined by start state, action, and goal state.

5) On the Spot Rotations: we assume that uncertainty on the position is equal to zero in the case of a rotation on the spot. So only states corresponding to the same cell are take into account and the uncertainty function takes only orientation in parameter. Transition function for on-the-spot rotation could be defined directly and statically like it is done in [13].

D. Reward Function

the reward function is defined as follows:

$$R(s) = \begin{cases} 0 & \text{if } s \text{ is a goal} \\ -1 & \text{otherwise} \end{cases}$$

This function is used in [15] et [13]. This simple gain function is sufficient and permits to distinguish the goal state from other states.

IV. RESULTS

A. Number of States Reduction

The first main advantage of our method is a reduction of the number of cells due to the quadtree decomposition. In this section, we study this reduction in more detail and show the advantage of choosing a quadtree decomposition instead of a regular decomposition.

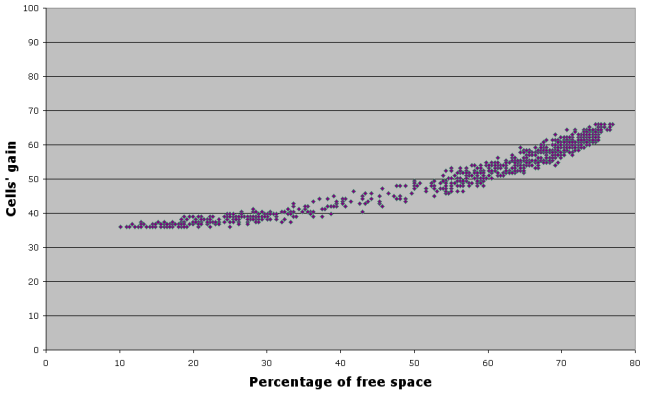


Fig. 6. Evolution of the cell number reduction when the proportion of the free space increases.

TABLE II

CELL NUMBER REDUCTION

Environment size	Average percentage of cells' gain
10 times the robot's size	40.9
20 times the robot's size	53.5
30 times the robot's size	78.7
60 times the robot's size	84.3

The chart of Fig. 6, illustrates the evolution of the cell number reduction when the proportion of the free space increases.

What is plotted is the ratio between the number of cells obtained by a regular decomposition and the number of cells obtained by a quadtree decomposition. They were computed on a set of on thousand randomly generated environments twenty times the size of the robot.. We can see that when the proportion of the free space increases, the cell number reduction increases too.

Table II shows how significant is the gain with respect to the size of environment: the bigger the environment, the higher the average cell number reduction. The cell number reduction is therefore maximum for large and quasi-empty environments.

Since the algorithmic complexities of both Policy Iteration and Value Iteration are a function of the number of states and number of actions for each iteration, the reduction of the number of cells, and accordingly the number of states, yields a gain in running time. Our approach permits to apply MDP to bigger environments.

B. Motion Plan Examples

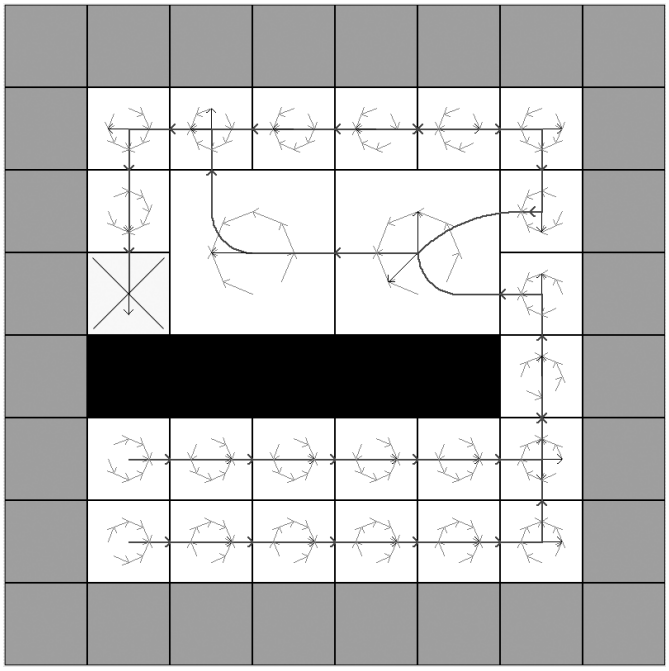


Fig. 7. Plan for 464 states (58 cells, 8 orientations) computed with Value Iteration in 7 s.

Figs. 7 and 8 show two plans generated using our method. On these plans, as described for Fig. 1, full cells are in black, free cells in white and mixed cells in grey. The goal corresponds to the cell with a cross. Each light grey arrows represents an on-the-spot rotation. Dubins actions are represented by black segments and circular arcs (with arrowheads attached to show the orientations).

When the plan 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. So, on the plan, we have eight states for one cell, thus there is eight actions per cell. Each actions correspond to the optimal action for one state.

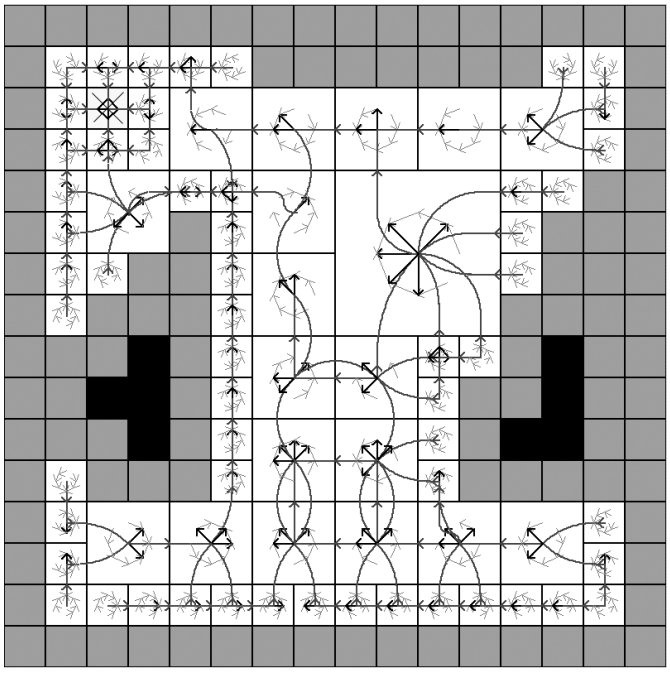


Fig. 8. Plan for 1496 states (187 cells, 8 orientations) computed with Value Iteration in 45 s.

On these figures, we can see that the main feature of MDP is kept: uncertainty on the action is integrated in the planning process. Indeed, safe actions are chosen: there are on-the-spot rotations and simple Dubins actions (like single translations or large circular arc motions). This phenomenon was foreseeable because a rotation on spot generates an uncertainty on the position close to zero, thus the collision risk is negligible. So the robot will prefer doing a rotation, to place itself in the position that will permit to do the safest displacement.

If we look at the right big case on Fig 7, we can see that the robot will prefer take the north and turn away from goal instead of reaching directly the goal. In fact, selecting the action that permits to reach the goal directly is dangerous since the corresponding Dubins action is complex (two circular arcs and one translation) and generates lot of uncertainty on the position at the end of the action, and furthermore the goal cell is close to some obstacle, so the collision risk is greater.

Also, obstacles remains repulsive: As we could see on the right side of Fig. 7 the robot attempts to reach the big cell in order to move away from the obstacle. Also, in Fig. 8, if the robot could go away from the obstacle by a safe action, it would do that. This is the cell, on the right side of the figure, for the actions planned in little cells.

Fig. 9 shows a path extracted from second plan (Fig. 8). To extract a path from plan, instead of displaying the optimal action for all states, we choose a state as the initial state and display the sequence of actions permitting reaching the goal from this initial state.

We obtain more smooth path than by using discrete actions.

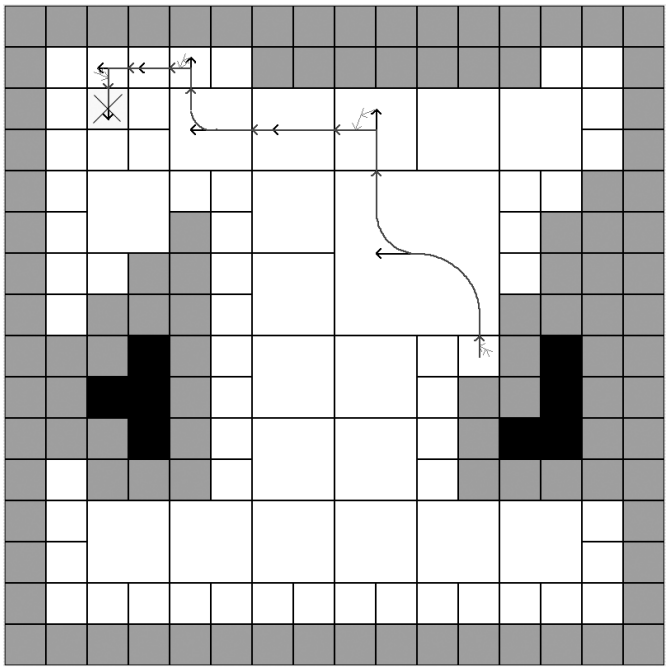


Fig. 9. Example of path

In particular, when the robot has to cross a big free space, because of quadtree decomposition and the actions we use, a smooth trajectory is obtain instead of having sequence of little translations and rotations on spot. In fact, this is the case on Fig. 9 of the first two actions of type segment-arc of circle which permit to cross a big free space, when it will need more than ten actions to obtain same displacement using discrete actions. Furthermore, executing a smooth trajectory, reduces the uncertainty on the final position, even though doing a sequence of translation and rotation, increases this uncertainty.

Also rotation on spot could be easily replaced by reverse move, if robot can not carry out rotation on spot (*eg* a car-like robot), or to obtain continuous smooth paths.

V. Conclusion

This paper has presented a MDP-based planning method for a wheeled mobile robot that uses a hierarchic representation of the robot's state space (based on a quadtree decomposition of the environment). Besides, the actions used better integrate the kinematic constraints of a wheeled mobile robot and limit the slippage effect stemming from on-the-spot rotations. Results show that the reduction in size of the set of states due to the quadtree decomposition permit to tackle more complex problems. Besides, the novel type of actions introduced reduces the non deterministic effects of the actions. These two features yield a motion planner more efficient and better suited to plan robust motion strategies.

The next step of this work is to evaluate our planning approach on a real robot using Markov localisation techniques as execution method [16]. An other interesting perspective is to study more complex and realistic methods to define the reward function. The reward function could be modified to make attractive certain classes of states. For instance, states where possible actions are less uncertain could be favored.

References

[1] N. J. Nilsson: *A Mobile Automaton: An Application of Artificial Intelligence Techniques*. IJCAI 1969

[2] J. C. Latombe. *Robot motion planning*. Kluwer Academic Press, 1991.

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

[4] R.E. Kalman. A new approach to linear filtering and prediction problems. *Trans. of the ASME, Journal of basic engineering, 82:35-45, March,* 1960.

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

[6] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[7] S. Koenig and R. Simmons. A robot navigation architecture based on partially observable markov decision process models. In *Kortenkamp et al. AI-ROBOTS,* 1998.

[8] P. Laroche, F. Charpillet, and R. Schott. Decompsition of markov decision processes using directed graphs. In *Poster session of the European Conference on Planning(EPC'99),* 1999.

[9] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, Massachussets, 1960.

[10] M. L. Littman, T. L. Dean, and L. Pack Kaelbling. On the complexity of markov decision processes. In *Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence (UAI-95), Montreal, Québec, Canada,* 1995.

[11] L. E. Dubins. On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents. American Journal of Mathematics, 79:497–516, 1957.

[12] A. R. Cassandra, L. P. Kaelbling et J. A. Kurien. Acting under uncertainty: Discrete bayesian models for mobile-robot navigation. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems,* 1996.

[13] P. Laroche, F. Charpillet et R. Schott. Mobile robotics planning using abstract markov decision processes. In *IEEE International Conference Tools with Artificial Intelligence,* 1999.

[14] T. Fraichard and R. Mermond. Path plannig with Kinematic and Uncertainty Constraints. In *Intelligent Autonomous Systems, page 30-37. International Scientific Issue, Ufa State Aviation Technical University (RU),* 1998.

[15] T. Dean, L. Kaelbling, J. Kirman et A. Nicholson. Planning with deadline in stochastic domains. In *Prooceedings of the 11th National Conference on Artificial Intelligence,* 1993.

[16] Dieter Fox, W. Burgard, and S. Thrun. *Markov localisation for mobile robots in dynamic environments*. Journal of Artificial Intelligence Research, 1999.