The Mixed Approach for Motion Planning: Learning Global Strategies from a Local Planner

B. Faverjon * and P. Tournassoud **

(*) INRIA Sophia-Antipolis, Av. Emile Hugues, 06565 Valbonne, France
(**) INRIA, Domaine de Voluceau, BP 105,78153 Le Chesnay Cedex, France

Abstract

In this paper, we propose a mixed approach for motion planning that decomposes the problem into two levels. At the global level, we build a graph whose nodes represent relatively large cells of the Configuration Space of the robotic system. Adjacent cells are connected by edges weighted by the probability for the local planner to succeed in computing a trajectory from a point in one cell to a goal in the other. These probabilities are used by a minimum cost path finding algorithm to generate subgoals for the local planner. They are updated using a Bayesian rule from the results of the execution of planned trajectories at the local level. At the global level, no geometric information is stored, thus eliminating the expensive transformation of obstacles into the Configuration Space needed by usual global methods. We take advantage of the ability of our local planner to move close to obstacles so that only a crude discretization of the Configuration Space is needed. This makes it possible to apply this technique to robotic systems with a large number of degrees of freedom. In mobile robot applications, sensors being used by the local planner, this method achieves the learning of planning strategies in an unknown environment without building a complete geometric model of the world.

1. Introduction

1.1. Position of the Problem

The general problem of motion planning can be stated as follows : given an algebraic description of the boundary of the moving objects and of the obstacles in Cartesian Space, find a collision free path for the moving objects from a set of initial positions to a set of goals. This general problem can be decomposed in a variety of sub-problems, according to whether the description of the world is planar or three-dimensional, whether there is only one moving object or many of them, whether there are connected by revolute or prismatic joints to form articulated chains or not. Theoretical issues posed by this problem have been thoroughly examined by Schwartz and Sharir *[Sc].* They propose an algorithm answering the general motion planning problem, based on Tarski's algorithm for deciding statements in the quantified elementary theory of real numbers. Their algorithm, which is not of direct practical application, is polynomial in the number of constraints describing the obstacles, but exponential in the number of degrees of freedom. Other works suggest the inherent exponential complexity of planning. An early result by Reif *[Re]* proves PSPACE hardness of a special instance of the planning problem. Other special cases where in turn examined by Hopcroft, Joseph and Whitesides *[Ho],* Spirakis and Yap *[Sp]* among others.

Few practical algorithms have nevertheless been proposed for the case of highest interest, that of an articulated chain of solids in a three dimensional environment. Algorithms developed by Lozano-Perez *[Lo]* Faverjon *[Fa84, Fa86],* are based on a representation of obstacles in the Configuration Space of the manipulator. The Configuration Space of a system is any set of independent parameters that enables to describe the position of all points bound to the moving bodies. As a straightforward translation of the description of obstacles from Cartesian Space to Configuration Space is not possible for manipulators, these algorithms rely on a subdivision of configuration parameters in small ranges. This provides a grid whose cells are either labelled as free, or intersecting Configuration Obstacles, the transform of obstacles in Configuration Space. A path is searched as a sequence of nodes in the graph describing the connectivity between free regions of Configuration Space.

Such a description requires a memory space exponential in the number of configuration parameters, and again, planning of a path is exponential in the number of configuration parameters. This imposes a practical limitation on the number of degrees of freedom involved: known algorithms limit themselves to planning a global path for the first three joints of a manipulator, while a heuristic approach is used for motions of the hand.

On the other hand, *local* methods have proved to be very powerful for computing motion of a manipulator. Local information on the environment only is used to compute displacements at any time, without keeping track of any landmarks. The so-called Potential Field Method relies on a minimization including a term attracting the manipulator towards the goal, and repulsive terms that push bodies of the manipulator away from the obstacles. In *[FT87]* we propose an alternative to the Potential Field Method for locally computing trajectories. A task is expressed by the minimization of the relevant measures of the problem written as a function of configuration parameters. Moving objects have a simplified local view of the environment as planes separating them from the obstacles, that are transformed into linear constraints in Configuration Space.

1.2. Overview of the Approach

In this paper we propose to uncouple the general problem of path planning into a low complexity local planner and a higher complexity global planner working on a graph of cells representing relatively large sets of configuration parameters. The main idea underlying this approach

is that we want to turn the power of local methods to profit so as to deal with the high complexity of global planning only at the relevant level of description. At the higher level no geometric description of the obstacles is used but only weigths indicating the probability for a trajectory computed locally not to lead to any

Given initial and goal configurations, a classical minimum cost path finding algorithm in the graph yields a list of cells giving the global shape of the path joining the node containing the initial configuration to the node containing the goal. Then the robot, starting from its initial position, describes a trajectory computed by the local planner, taking as subgoal a point located inside the next cell on the path. During the execution of the path, the weights are updated using a model of learning from results of motions generated by the local planner.

The robot eventually reaches the final goal, or there is failure, meaning the robot is *blocked* while aiming at some cell. In this case, we put a higher weight to the corresponding transition and compute again a global path from current configuration, based on the updated weights. This produces a new path avoiding the problematic arc.

Let us underline that this approach is relevant only in the case it is performed using a loose grid in Configuration Space, or we again deal with a space of small cells either occupied or free, which is not better than the global approach in terms of computational cost Dealing with a loose graph is only made possible because of the intrinsic power of the local planner that produces long pieces of collision free trajectory.

Our approach presents other interesting features. First local computation can be based on a local model of the visible obstacles acquired through proximity sensors such as ultrasonic sensors, or a stereo vision system, as a substitute to a complete geometric model of the environment. Transitory mobile obstacles will in particular be taken into account by the local planner.

Figure (Ia) shows the trajectory of a mobile robot in an unkown environment using this approach. Figure (Ib) shows the trajectory obtained with the same initial and goal positions when we take into account the knowledge obtained from the first execution.

The following sections describe in more details the different parts of this approach, namely the Sute Graph, global planning, local planning, and the learning process.

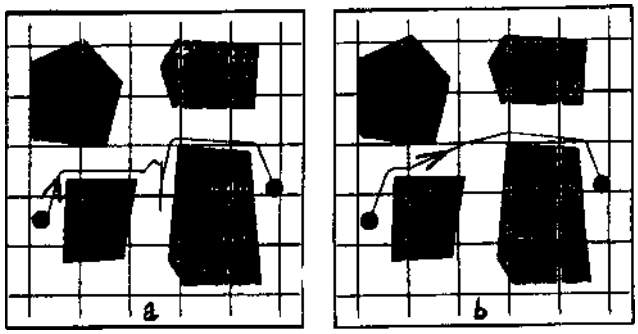


Figure 1. a) Before learning, b) After learning.

2. The State Graph

In the sequel the term *robot* stands for any robotic system regardless of its nature (multi-linked manipulator or mobile robot) and of the number of moving objects involved. We denote $(q^1,...,q^1)$ a set of *n* dimensional configuration parameters describing the state of the system. We assume that an algorithm to locally compute pieces of trajectories is available. Let us first partition Configuration Space into cells of the type $C_i = \Pi_{k=1...n}[q_i^k - \delta^k/2, q_i^k + \delta^k/2]$, where $(q_i^1,...,q_i^n)$ are the configuration parameters of the center of cell C,, and & is the width of a cell in the *k-th* direction. Starting with this description we build a graph whose nodes stand for the cells themselves. It is called *State Graph* in the sequel. When the robot configuration lies inside a cell C,- it is said to be in state C_r Each node has *In* neighbors, namely the nodes which stand for cells of centers $(q_i^1,...,q_i^k - \delta^k,...,q_i^n)$ or $(q_i^1,...,q_i^k + \delta^k,...,q_i^n)$„ These represent rectangular transitions in Configuration Space.

As opposed to the standard Configuration Space approach, we will store at this higher level no description of the obstacles as seen in Configuration Space. For each oriented transition between two neighboring cells C, and C; in the State Graph we only memorize a weight that estimates the difficulty the robot has to enter cell C. when coming from cell C,-. More precisely we define pij,-. as the probability for the local planner to succeed in making the robot enter cell C, from neigboring cell Cj, when aiming at some point located inside C,j

We call *path* any connected sequence of nodes of the State Graph. The probability for a path to provide a successful trajectory is defined as the product of probabilities *pij* along the path. This implies we make an hypothesis of independence, namely that the probability of realizing a transition is independent of the sequence of nodes we followed so far. In practice we attribute to each arc of the graph a weight equal to - log(pij) and we minimize the cost $g = -\Sigma \log(p_{ij})$ for traversed transitions along the path from initial cell *CQ* towards goal cell C_g.

Initially, in absence of any infomation on its environment, the robot initializes the graph with given a priori probabilities ofpij=p. Hence the path we compute first is a path of minimum length in terms of the number of cells traversed from the initial configuration to the goal. Later as probabilities vary to reflect the knowledge the robot has of its environment, the path we compute realizes a compromise between minimum distance and the assurance that we will reach the goal.

3. On Global Planning

3.1. Searching for a path

For searching a path from an initial cell to a goal one, we make use of an A* algorithm maximizing the product of probabilities *pij* along the path. This translates into minimizing the cost $g = -\Sigma \log(p_{ij})$ for traversed transitions along the path from initial cell CQ to goal cell C_g. The A* algorithm makes it possible to use a heuristic function to guide the search. This function gives at each node an estimate of the cost of the optimal path from this node to the goal. This algorithm yields the optimal path if the heuristic is *admissible,* this is if the heuristic cost hi for any node i is lower than the optimal cost from that node to the goal node.

The heuristic cost we use for a node C j is the number of cells N,- that are traversed by a straight line to the goal, weighted by the average probability £ for all transitions of the State Graph.

$$h_i^* = -N_i \log p$$

This heuristic function is generally not admissible, but gives good

results in our experiment*. It can be seen as an estimate of the mininum cost we expect if we suppose that the difficulty to move around is about the same everywhere.

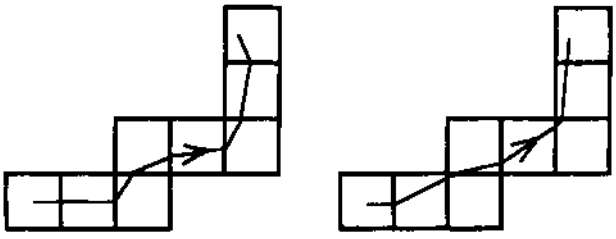
It must be also noticed that since all weights are finite in our model and the graph is connected, a path is always found by the global planner. Thus, the feasibility of a trajectory has to be evaluated from the actual cost of the global path. We will see in Section 6 how the heuristic cost function can be used for this purpose.

3.2. Choosing Subgoals on the Path

As we run the local algorithm, a simple choice of a subgoal is the center of the next cell on the path. However, this choice yields a trajectory generally close to the drunkards walk because of the discretization. In order to avoid this reprehensible behavior, we propose another choice that gives a smoother trajectory. The idea is to locally optimize the length of the trajectory. So, the subgoal is chosen as the point on the common face with next cell that minimizes the sum of:

- the distance from the current position to this point,
- and the distance from this point to the center of one of farther cells on the path.

The farther this cell is chosen, the more we anticipate future displacements. In the case the robot is blocked, nearer cells are used in order to provide new subgoals. An example of a trajectory obtained with this method, in absence of any obstacle, is shown in figure 2.



trajectory with no anticipation trajectory with a one-cell anticipation

Figure 2 : Compulation of subgoals

4. On Local Planning

The local planner takes as input a subgoal in the Configuration Space and tries to move the robot to this goal taking into account information on the local environment. This information can be obtained by sensing, using proximity or vision sensors, or from CAD models of the robot and the environment. Note that in our approach, no hypotheses are done concerning the local method. The interaction between the global and local levels is limited to the sending of subgoals from the global to the local planner, and the observation of the successive configurations of the robot by the global supervisor to achieve the learning.

We have described in *[FT87]* the approach we use for local computation of the trajectory of a general manipulator system. It is posed as a minimization of a functional of the configuration parameters under linear anti-collision constraints. For the standard instance of the path planning problem (go from one point to another in Configuration Space) we simply minimize the norm of the difference between actual joint increments and desired ones, computed as the vector supported by the line towards the goal in Configuration Space respecting bounds on maximum velocities. A local model of obstacles in Configuration Space, namely an intersection of free half-spaces of Configuration Space, is derived from tangent planes separating the mobile object from each obstacle in Cartesian Space. For each moving solid that lies at a distance *d* less than an influence distance *di* from an obstacle at current time /, we impose the following constraint on the variation of the distance:

$$\delta d \geq -\xi \frac{d - d_s}{d_i - d_s} \delta t \qquad (1)$$

where d_5 is the security distance at which the robot must stop, £ a positive weighting coefficient for adjusting convergence speed, and & the time increment. This constraint thus realizes a velocity damper forcing the moving solid to stay on its side of the separating plane. The variation of the distance can be written $\delta d = \mathbf{n} \cdot \mathbf{J}\, \delta \mathbf{q} = \mathbf{J}^t \mathbf{n}$. 5q where 5q is the *n* dimensional vector of the variations of configuration parameters, and J the jacobian matrix for the robot at point x of the moving object nearest from the obstacle (see Figure 3). From inequation (1) we then derive a simple linear constraint on joint increments Sq between time *t* and f+&, which writes witii \-}'.*n*:

$$\mathbf{v} \cdot \delta \mathbf{q} \geq -\xi \frac{d - d_s}{d_i - d_s} \delta t \qquad (2)$$

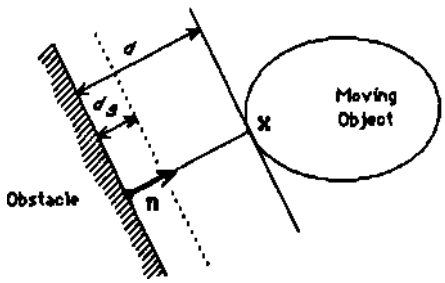


Figure 3 : The Velocity Damper constraint

In *[FT86]* we describe this approach with more details and present the extension to the case of a robotic system composed of more than one manipulator. We also make use of a similar kind of constraint that presents more anticipation as far as coordination of movements is concerned : moving objects are bound to stay on their side of a separating plane, possibly moving that is tangent to both moving bodies, or the moving body and a fixed obstacle. Figure 4 illustrates a trajectory obtained using this method.

We use this kind of constraints in the examples that illustrate this paper, namely a system composed of one or more mobile robots described by discs, in a planar world of polygonal obstacles. Details of the local method in this case are given in section 7.

Note that all local methods may fail at some point because of local minima generated by concave arrangements of obstacles. In this case, the robot stops without reaching its goal.

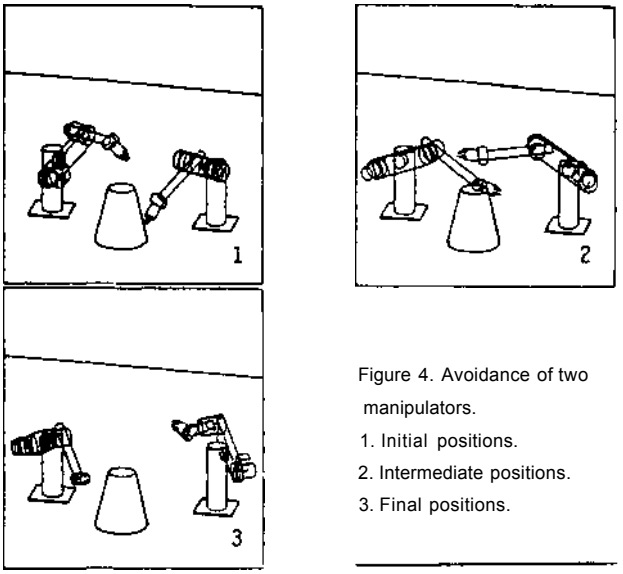


Figure 4. Avoidance of two manipulators.
1. Initial positions.
2. Intermediate positions.
3. Final positions.

5. Learning Global Knowledge from Path Execution

5.1. The Bayesian Approach

We call *py* the probability for the robot to enter cell C, coming from neigboring cell C,, if it is aiming at some point located inside cell C..

When the local planner is realizing such a scheme, two types of events may happen:

- $(x_{ij})^+$ or success in entering cell Cj, event of probability pij,
- $(x_{ij})^-$ or failure in entering the cell, event of probability $1-p_{ij}$. A failure occurs either when the robot is blocked in its current cell, or when it is forced by the obstacles to enter another neighbor of C, before C.. In the following, we ommit the subscripts *ij* for clarity.

We want to estimate the probalility *p* associated to a given transition from the events that happen during executions of trajectories. We use the Bayesian approach as follows.

Let us suppose that *n* events have hapenned so far that concern the given transition. x^k, $k=1,\ldots,n$. The value of x^k is 1 in case of success and 0 in case of failure. We denote X^n the vector (x^k), $k=1,\ldots,n$ and/the probability distribution function. The Bayesian learning model states :

$$f(p \mid X^{n+1}) = \frac{f(x^{n+1} \mid p)\, f(p \mid X^n)}{\int f(x^{n+1} \mid p)\, f(p \mid X^n)\, dp} \qquad (3)$$

In our case, $f(x^{n+1} \mid p)$ is p for $x^{n+1} = 1$ and $(1-p)$ for $x^{n+1} = 0$, which can be written :

$$f(x^{n+1} \mid p) = p^{x^{n+1}} (1-p)^{1-x^{n+1}}$$

Thus formula (3) rewrites :

$$f(p \mid X^{n+1}) = \frac{p}{p_n}^{x^{n+1}} \frac{1-p}{1-p_n}^{(1-x^{n+1})} f(p \mid X^n) \quad (4)$$

where p_n denotes the mean value of p after n events, that is :

$$p_n = \int p f(p \mid X^n)\, dp \qquad (5)$$

In this recursive form, formulae (4) and (5) can be used to compute the probability distribution law *ofp* after *n* events, and its mean value that we will use as an estimate of p. When we have no a priori knowledge on the environment, we initialize the probability distribution with the uniform distribution law, that is $f(p)=1$ for p in $[0,1]$ and $p_0=0.5$.. It can be shown that in this case, the mean value of *p* after *n* events, *s* of them being successful, simply writes:

$$p_n = \frac{s+1}{n+2} \qquad (6)$$

The proof is given in appendix. Let us make the following remarks.

- The absence of knowledge that we have modelized by a uniform probability distribution law can be interpreted simply from this formula: the a priori knowledge is the same as if we have already made two trials, one of them being a success, and the other a failure. This interpretation is helpful if we want to incorporate a priori knowledge: *P0* can be initiated to s_0/n_0 where *S0* is the number of successes for n0 hypothetic trials.
- As the events xk are supposed to be independent, the probability distribution law does not depend on the order in which the events hapenned, and thus all the memory of a transition is expressed by the pair *(n, s)* where *n* is the number of events on this transition and *s* the number of successful events.

5.2. Updating Probabilities

Updating of the transition probabilities is first performed whenever the motion supervisor detects a transition in the State Graph. If the transition is the one ordered by the global planner, the corresponding probability is increased by incrementing the number *n* of events and the number *s* of successes. Else the robot has been deviated from its way by an obstacle, and we decrease the probability of the desired transition. This is performed by incrementing only the number of events. In this case, we must also modify the path so that it remains connected. This is done by adding to the path the current node, and the common neighbor of the current node and the next cell on the path (hatched cells in figure 5).

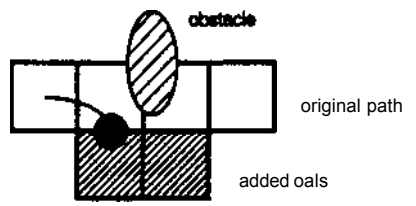


Figure 5 : Modification of the path when the actual transition is not the desired one.

Updating the transition probabilities is also performed when the robot is blocked before reaching its goal. If it is not inside the goal node, we decrease the probability of the desired transition and invoke the global planner using the updated State Graph. As the probability of the problematic transition has been decreased, it will eventually be avoided by the newly computed path. If a failure occurs inside the goal node, the goal is declared not reachable. This event may have several causes:

- the goal lies inside an obstacle (or another connected component of free space).
- the local method failed to reach the goal because of obstacles inside the goal node.

In the latter case, a solution would be to move the robot outside the goal node by setting a subgoal in a neighbor chosen from local information, before aiming again at the goal. This strategy might succeed in making the robot pass round the obstacle. But the main problem in fact is to distinguish between the two cases above. Elements of answer are given in section 6.

5.3. The Motion Planning Algorithm

The motion planning algorithm is written below in a C-like language. Procedures in bold characters correspond to message exchanges with the local planner or the robot controller.

```
go_to(goal) /* goal is the configuration we want to reach */
{
    node = find_node(get_current_configuration());
    goal_node = find_node(goal);
    do {
        path = compute_global_path(node,goal_node);
        send_local_planner(subgoal(path)); /* initializes motion */
        do {
            new_node=find_node(get_current_configuration());
            if (path ≠ Ø and new_node ≠ node) {
                if (new_node = car(path)) {
                    increase_transition_probability(node,new_node);
                    path = cdr(path);
                }
                else {
                /* the robot escaped from the sequence of nodes on the path */
                    decrease_transition_probability(node,car(path));
                    path = cons(new_node+car(path)-node,path);
                    /* updates the path */
                }
                node = new_node;
                send_local_planner(subgoal(path));
                /* changes local subgoal */
            }
        }
        while (robot_is_running());
        if (robot_is_at_goal()) return(success);  /* else the robot is blocked */
        if (node = goal_node) return(failure in last cell);
        decrease_transition_probability(node,car(path));
        /* a new global path must be computed */
    }
    while ();
}
```

6. Reasonning on the Global Knowledge

6.1. Deciding on the Feasibility of a Global Path

As mentionned in the previous section, the case when die goal cannot be reached must be treated with some care. This happens when the robot intersects the obstacles in this configuration, or lies in another connected component of free space. In both cases, we do not want the robot to roam around indefinitely while trying to reach the goal. In practice we impose a threshold on the ratio of the optimal cost and the heuristic cost when we perform the planning starting from some cell C_i:

$$g_i^* / h_i^* < k.$$

The heuristic function can be seen as the expected cost in case of uniform distribution of difficulty in the environment Thus, if the actual optimal cost is much higher than this estimate, it means that the optimal path is much longer than the expected one, or that there exists on the optimal path one or more transitions with a low probalility of success. So, the constant k measures the relative length of the detour we tolerate in order to have a reasonnable chance to succeed.

6.2. Teaching by Showing Mode

A simple way to give information to the system is to teach the robot some safe trajectories. While executing these trajectories, the system increases the probabilities associated with the traversed transitions. If these trajectories are repeated, the corresponding cost will become lower and lower. Further, the system will use pieces of such trajectories when other trajectories will be computed automatically. However, if modifications of the environment are detected by the local planner, corrections will be made to the specified trajectories.

6.3. Exploration Mode

In this mode, the mobile has no a priori knowledge of its environment. We want it to behave properly after a number of executions of trajectories. A straightforward way to proceed is to generate goals inside one of the nodes with smallest *history.* The history of a node C,* is simply defined as the sum onj of numbers n.- of events that have so far happened for transitions from neighboring nodes C. to C,i The robot will thus try to explore first those nodes for which it has least information.

When the path towards the goal is executed, the history of all traversed nodes is updated at the same time as transition probabilities. A problem arises when the path is declared not feasible. This does not give any relevant information on the difficulty to enter the goal node from a neighbor, but we increase its history although the corresponding transitions are not updated, so that the robot does not try again and again to enter a node that it cannot reach. This conveys an information such as "this goal configuration probably lies inside an obstacle or another connected component of free space".

The exploration process ends when all nodes have received an history higher than a specified threshold.

6.4. Detecting Abnormal Situations

Once the environment has been thoroughly explored, we can make use of our knowledge to detect abnormal situations by comparing the events that happen to a priori probabilities. As an example, if a new obstacle appears in the environment, it can be detected from the failures it will imply for transitions with a high probability of success.

65. Adaptation of the Size of the Grid

In order to decrease the number of nodes in the graph it may be useful to adapt the discretization of the Configuration Space to the planning difficulty in the various regions. This can be done by analysing the variation of the estimated transition probabilities. Indeed, if we have too large cells, the transition probabilities will not converge towards 0 or 1 because of alternate success and failure. Such transitions are characterized more precisely by a great variance for the value of events x*, equal to $s(n-s)/n^2$. To solve this problem, it is possible to split the corresponding

node into several new cells, thus obtaining a finer description of this part of the environment at the global level.

In our work on global planning *[Fa84]* we proposed to used an octree in the Configuration Space to represent Configuration Obstacles and free space of dimension 3. The octree is a tree based on a recursive subdivision of the 3D space into rectangular cells. The root represents the whole space. A cell is decomposed into eight cells whose side is half that of their father. This structure can be generalized to higher dimensions, the number of sons of a node being 2d in a d-dimensional space. It is called a 2d-tree.

Using such a structure proceeds as follows. Exploration starts with a regular grid corresponding to a relatively high level in the 2d-tree. When an event happens for a given transition, we first look at the variance of the value of such events and compare it to a threshold. If the variance is smaller than the threshold, the usual updating is performed. Else, the aim node is split into its sons and new transitions are initialized as follows.

- Transitions between sons and neigbors of the father are initialized assuming that the events that happened to the former transition are uniformly distributed on the 2d-1 new transitions. If the variance of the former transition is greater than the threshold, they are initialized with the a priori uniform law.

- Transitions between two sons are initialized with the uniform law.

This process can similarily be used for solving the problem of dead-locks inside the goal node.

Splitting is performed until the size of a node is smaller than a threshold dependent on the ability of the local method to reach a subgoal at a distance corresponding to that threshold.

7- Validating Tool

7.1. Overwiew

We have implemented a simulation tool for validating our approach. It deals with motion planning for one or two discs in the plane cluttered with convex polygonal obstacles. A straightforward application is motion planning for mobile robots.

The Configuration Space for a disc is the 2D space of parameters x and y corresponding to the position of the center of the disc. The Configuration Space is divided into square cells standing for the nodes of the State Graph. The size of these cells is set to about the width of the corridors in which the robot navigates.

In the case of two discs, the State Graph is obtained as the cross product of the discretized Configuration Spaces of both discs.

Before giving some results, we describe in detail the local method we use in this particular case.

7.2. The Local Method

We suppose that the mobile robot is able to build a model of its local environment at any time using proximity sensors or vision. In our simulation, the local environment is obtained by clipping the polygonal obstacles by a disc of visibility.

The principle of the method for avoiding an obstacle is to slide on one of the two common separating tangent lines to the mobile and the obstacle. This gives a better behavior of the mobile than the so called Potential Field method because of the anticipation it provides.

More precisely, a convex obstacle gives rise to two separating tangent lines. A line represents a candidate *displacement* from the current position to the point of contact with the obstacle. We add the displacement in straight line towards the goal to the list of candidate displacements. We then prune that list by only considering displacements for which the disc does not intersect other obstacles during the motion (see figure 6).

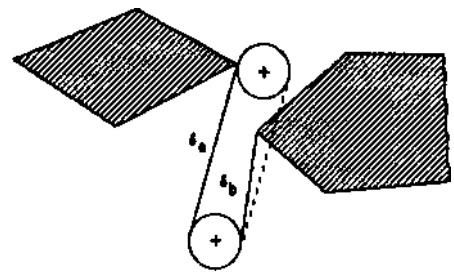


Figure 6. Displacement S_a is not admissible, δ_h is admissible.

If no admissible displacement exists in the half plane towards the goal, the robot is blocked and another global planning is performed after updating of the State Graph. Else the displacement we realize is the one that minimizes the angular difference with the goal. The robot is moved along that direction of a distance dependent on time increment and the all computation is performed again.

7.3. Results

Figures 7 and 8 illustrate results for one disc. Objects that are represented are : obstacles in grey, cells of the grid, the robot in its initial (hatched) and final positions. Figure (7a) shows a trajectory executed without any knowledge of the environment Small circles are positions at which the global planner was called. Figure (7b) shows the second attempt whith same initial and goal positions. No failure occurs this second time.

Figure 8 shows, for a more complex environment, the path computed by the global planner after learning (8a), and the resulting trajectory (8b). The graph that is used is the result of 50 learning trajectories. The threshold for the ratio of the optimal cost and the heuristic cost was 3.

Figure 9 illustates the method in the case of two discs. As rectangular transitions only are allowed, probabilities are initiated with the corresponding value for the given disc alone. If the width of a cell is about the size of a disc, we set a priori a low probability for transitions that would make both robots occupy the same square of the grid. If robots are much smaller, coordination of the motions of the discs is first left to the local planner, making use again of tangent separating lines to both discs. In that case cooperation of the two robots in constrained parts of the environment, for example crossing in a corridor, is performed by using the general learning scheme described in this paper.

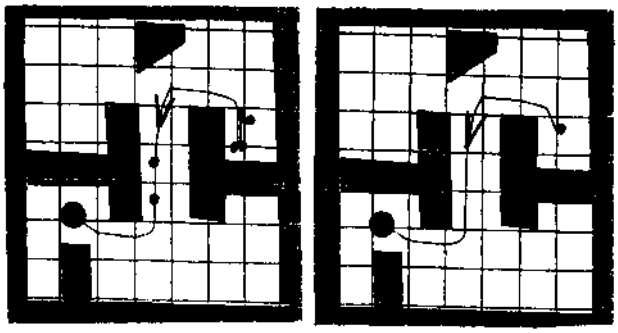


Figure 7. (a) (b)

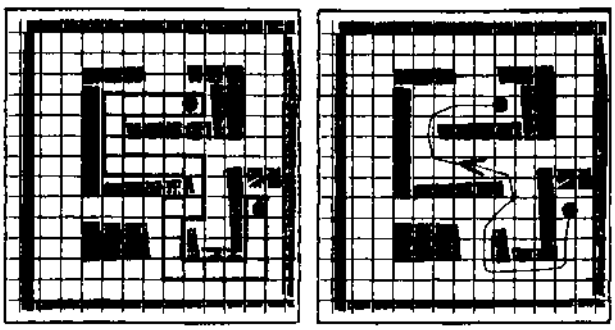
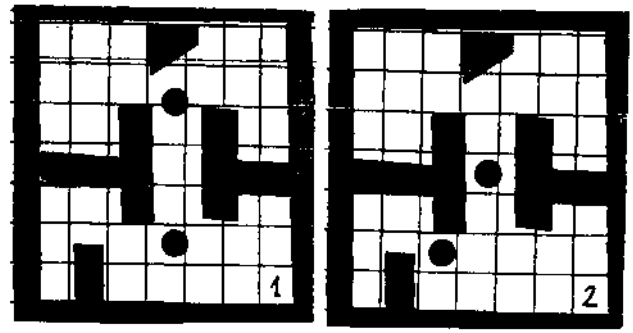


Figure 8. (a) (b)



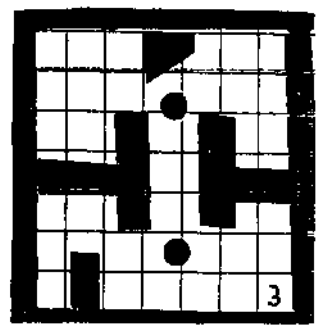


Figure 9.
1. Initial positions.
2. Intermediate positions.
3. Final positions.

8. Conclusion

The method we have presented in this paper is general and can be applied to various motion planning problems in Robotics. Learning of global strategies is performed very simply from the execution of trajectories. The method can be used to explore an unknown environment in mobile robot applications. Knowledge of the environment can also be given to the system by showing it some safe trajectories. This approach in which the search for the global shape of the trajectory is deconnected from the fine local motion computations realizes the fusion between the two types of methods we have experimented earlier, and we think that some other interesting results will probably appear in the future, based on similar ideas.

Appendix : Computation of a Transition Probability

The general formula for the probability distribution function is :

$$f(p \mid X^{n+1}) = \frac{p}{p_n}^{x^{n+1}} \frac{1-p}{1-p_n}^{(1-x^{n+1})} f(p \mid X^n)$$

where p_n denotes the mean value of p after n events, that is :

$$p_n = \int p f(p \mid X^n)\, dp$$

Solving the recursion, we can write :

$$f(p \mid X^n) = p^s (1-p)^{n-s} f_0(p) / D_n$$

where $s = \sum x^k$ is the number of sucesses and :

$$D_n = \prod_{i=1}^{n-1} p_i^{xi} (1-p_i)^{1-xi}$$

If the a priori distribution law is uniform, i.e. $f_0(p) = 1$ for any p in [0,1], we obtain :

$$p_n = 1/D_n \int p^{s+1}(1-p)^{n-s}\, dp = 1/D_n\ I_n^s$$

It can be shown easily that integral I_n^s verifies the recursive formula :

$$I_n^s = \frac{s+1}{n+2}\ I_{n-1}^{s-1}$$

Thus, if the event is a success, we derive :

$$p_n = 1/D_n\ I_n^s = 1/D_{n-1}\ 1/p_{n-1}\ \frac{s+1}{n+2}\ I_{n-1}^{s-1}$$

and then $$p_n = \frac{s+1}{n+2}$$

Similar equations can be derived in the case of a failure, which proves the announced result.

References

[Fa84] Faverjon B. Obstacle Avoidance Using an Octree in the Configuration Space of a Manipulator. IEEE Int. Conf. on Robotics and Automation, Atlanta, March 1984.

[Fa86] Faverjon B. Object Level Programming of Industrial Robots. IEEE Int. Conf. on Robotics and Automation, San Franscisco, April 1986.

[FT86] Faverjon B., Tournassoud P. A Hierarchical CAD System for Multi-Robot Coordination, NATO Advanced Research Workshop on Languages for Sensor-Based Control in Robotics, Italy, Sept 1986.

[FT87] Faverjon B., Tournassoud P. A Local Based Method for Path Planning of Manipulators with a High Number of Decrees of Freedom. EEEE Int. Conf. on Robotics and Automation, Raleigh, USA, April 1987.

[Ho] Hopcroft, J., Joseph, D. and Whitesides, S. On the Movement of Robot Arms in Two-dimensionnal Bounded Regions. 23rd IEEE Symp. FOCS (1982c) 280-289.

[Lo] Lozano-P6rez, T. Spatial Planning: A Configuration Space Approach. IEEE Transactions on Computers. C-32(2), 108-120,1983.

[Re] Reif, J. Complexity of the Mover's Problem. Proc. 20th An. IEEE Symp. FOCS (1979), 421-427.

[Sc] Schwartz, J.T. and Sharir, M. On the Piano Movers' Problem II: General Techniques for Computing Topological Properties of Algebraic Manifolds. Technical Report N"41. New York University Computer Science Department, Courant Insutute of Mathematical Sciences, 1982.

[Sp] Spirakis, P. and Yap, C.K. On the combinatorial Complexity of Motion Coordination. Technical Report N#76. New York University Computer Science Department, Courant Institute of Mathematical Sciences, 1983.