12

Collision Free Path Planning for Multi-DoF Manipulators

Samir Lahouar, Said Zeghloul and Lotfi Romdhane

1. Introduction

Path planning is a very important issue in robotics. It has been widely studied for the last decades. This subject has gathered three interesting fields that were quite different in the past. These fields are robotics, artificial intelligence and control. The general problem of path planning consists of searching a collision free trajectory that drives a robot from an initial location (position and orientation of the end effector) to a goal location. This problem is very wide and it has many variants such as planning for mobile robots, planning for multiple robots, planning for closed kinematic chains and planning under differential constraints. It includes also time varying problems and molecular modeling, see (LaValle, 2006) for a complete review. In this study we focus on the case of multi-Degrees of Freedom (DoF) serial manipulators.

The first works on serial manipulators path planning began in the seventies with Udupa (Udupa, 1977), then with Lozano-Pérez and Wesley (Lozano-Pérez & Wesley, 1979) who proposed solving the problem using the robot's configuration space (CSpace). Since then, most of path planning important works have been carried out in the CSpace. There are two kinds of path planning methods: Global methods and Local methods. Global methods (Paden et al., 1989; Lengyel et al., 1990; Kondo, 1991) generally act in two stages. The first stage, which is usually done off-line, consists of making a representation of the free configuration space (CSFree). There are many ways proposed for that: the octree, the Voronoï diagram, the grid discretization and probabilistic roadmaps. For each chosen representation, an adapted method is used in order to construct the CSFree, see (Tournassoud, 1992; LaValle, 2006). The representation built in the first stage is used in the second one to find the path. This is not very complicated since the CSFree is known in advance. Global methods give a good result when the number of degrees of freedom (DoF) is low, but difficulties appear when the number of DoF increases. Moreover, these methods are not suitable for dynamic environments, since the CSFree must be recomputed as the environment changes. Local methods are suitable for robots with a high number of DoF and thus they are used in real-time applications. The

Source: Industrial-Robotics-Theory-Modelling-Control, ISBN 3-86611-285-8, pp. 964, ARS/plV, Germany, December 2006, Edited by: Sam Cubero

potential field method proposed by Khatib (Khatib, 1986) is the most popular local method. It assumes that the robot evolves in a potential field attracting the robot to the desired position and pushing its parts away from obstacles. Because of its local behavior these methods do not know the whole robot's environment, and can easily fall in local minima where the robot is stuck into a position and cannot evolve towards its goal. Constructing a potential field with a single minimum located in the goal position, is very hard and seems to be impossible, especially if there are many obstacles in the environment.

Faverjon and Tournassoud proposed the constraint method (Faverjon & Touranssoud, 1987), which is a local method acting like the potential field method in order to attract the end effector to its goal and dealing with the obstacles as constraints. Although it yields remarkable results with high DoF robots, this method suffers from the local minima problem.

Probabilistic methods were introduced by Kavraki et al. (Kavraki et al., 1996) in order to reduce the configuration free space complexity. These methods generate nodes in the CSFree and connect them by feasible paths in order to create a graph. Initial and goal positions are added to the graph, and a path is found between them. This method is not adapted for dynamic environments since a change in the environment causes the reconstruction of the whole graph. Several variants of these methods were proposed: Visibility based PRM (Siméon et al., 2000), Medial axis PRM (Wilmarth et al., 1999) and Lazy PRM (Bohlin & Kavraki, 2000).

Mediavilla et al. (Mediavilla et al., 2002) proposed a path planning method for many robots cooperating together in a dynamic environment. This method acts in two stages. The first stage chooses off-line, a motion strategy among many strategies generated randomly, where a strategy is a way of moving a robot. The second stage is the on-line path planning process, which makes each robot evolve towards its goal using the strategy chosen off-line to avoid obstacles that might block its way.

Helguera et al. (Helguera & Zeghloul, 2000) used a local method to plan paths for manipulator robots and solved the local minima problem by making a search in a graph describing the local environment using an A* algorithm until the local minima is avoided.

Yang (Yang 2003) used a neural network method based on biology principles. The dynamic environment is represented by a neural activity landscape of a topologically organized neural network, where each neuron is characterized by a shunting equation. This method is practical in the case of a 2-DoF robot evolving in a dynamic environment. It yields the shortest path. However, the number of neurons increases exponentially with the number of DoF of the robot, which makes this method not feasible for realistic robots.

Here, we propose two methods to solve the path planning problem. The first method (Lahouar et al., 2005a ; Lahouar et al., 2005b) can be qualified as a

global method. It is suitable for serial robot manipulators in cluttered static environments. It is based on lazy grid sampling. Grid cells are built while searching for the path to the goal configuration. The proposed planner acts in two modes. A depth mode while the robot is far from obstacles makes it evolve towards its goal. Then a width search mode becomes active when the robot gets close to an obstacle. This mode ensures the shortest path to go around an obstacle. This method reduces the gap between pre-computed grid methods and lazy grid methods. No heuristic function is needed to guide the search process. An example dealing with a robot in a cluttered environment is presented to show the efficiency of the method.
The second method (Lahouar et al., 2006) is a real-time local one, which is used to solve the path planning problem for many manipulator robots evolving in a dynamic environment. This approach is based on the constraints method coupled with a procedure to avoid local minima by bypassing obstacles using a boundary following strategy. The local planner is replaced by the boundary following method whenever the robot gets stuck in a local minimum. This method was limited to 2-DoF mobile robots and in this work we show how it can be applicable to a robot with n degrees of freedom in a dynamic environment. The path planning task is performed in the configuration space and we used a hyperplane in the n dimensional space to find the way out of the deadlock situation when it occurs. This method is, therefore, able to find a path, when it exists and it avoids deadlocking inherent to the use of the local method. Moreover, this method is fast, which makes it suitable for on-line path planning in dynamic environments.

2. Sampling and construction of the CSpace

Many planning algorithms need samples of CSpace in order to compute a trajectory. There are many ways of sampling; the easiest way is to use a grid with a given resolution. The number of the grid cells grows exponentially according to the number of DoF of the robot. In the same way, the time and the memory space required to compute and store the grid increase. Random sampling was introduced in order to reduce the number of samples needed to represent the CSpace. It consists of choosing random configurations and constructing a graph representing feasible paths between them. This method needs a long time of computation.
We give an example of sampling using a grid with a low resolution and we define constraints used to detect if there is a free path between two neighboring cells. On one hand, these constraints make the path between two neighboring cells in the CSfree safe even if the step is quite large, and on the other hand they speed up the collision checking process as the constraints computed in a cell are useful to check all the neighboring cells. There is no need to check for collision in all cells of the grid before starting to search for a path. The con-

straints calculated in a cell allow us to judge whether a path exists to a neighboring cell or not.

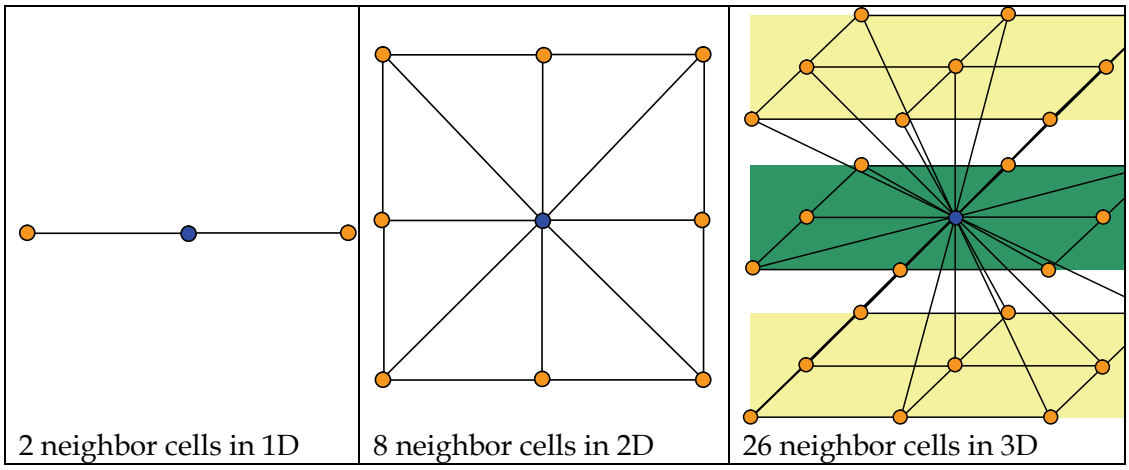

Figure 1. Each cell has 3^N-1 neighbors

Therefore, the constraints-calculating process is equivalent to 3^N-1 times the collision checking process, as a cell has 3^N -1 neighbors (Fig. 1). The number N represents the number of DoF of the robot.

3. Non-collision constraints

Here, we define non-collision constraints necessary to accelerate the global method (see paragraph 4) and useful for the local planner of the second method (see paragraph 5). Non-collision constraints as proposed by Faverjon and Tournassoud are written as follows:

$$\dot{d} \geq -\xi \frac{d - d_s}{d_i - d_s} \quad \text{if} \quad d \leq d_i \tag{1}$$

With d is the minimal distance between the robot and the object and $\dot{d}$ is the variation of d with respect to time. d_i is the influence distance from where the objects are considered in the optimization process, d_s is the security distance and ξ is a positive value used to adjust the convergence rate.

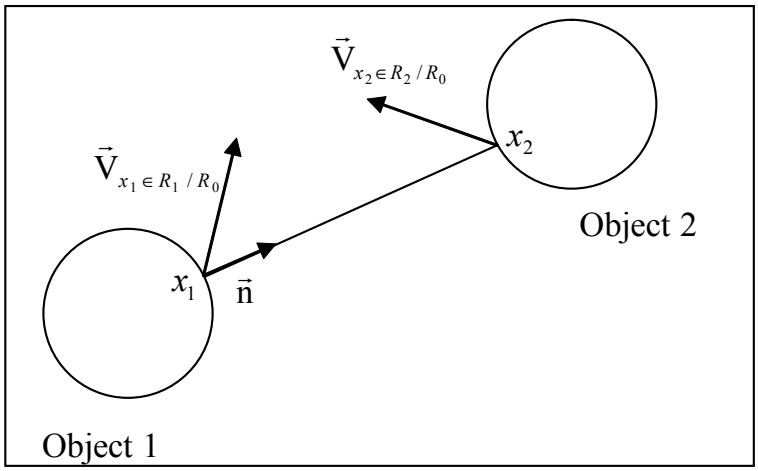

Figure 2. Two objects evolving together

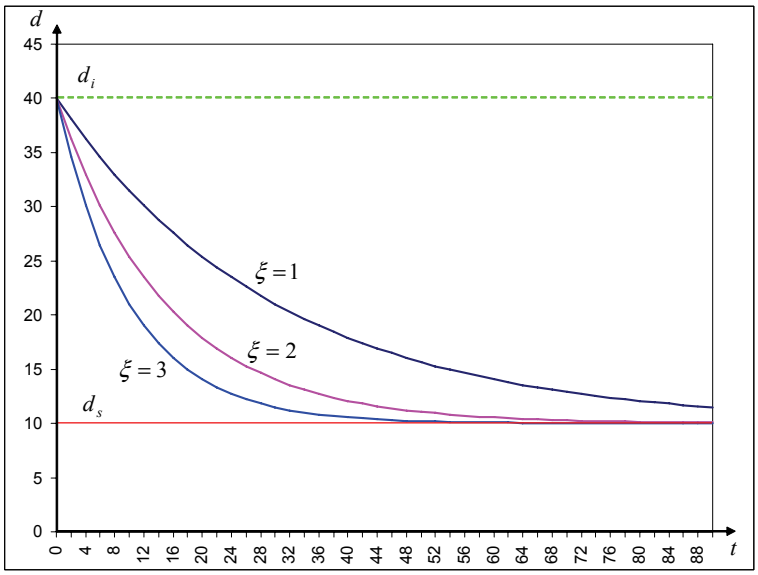

Figure 3. Evolution of the distance according to the convergence rate

If we consider two mobile objects in the same environment as shown in Fig. 2, $\dot{d}$ can be written as follows:

$$\dot{d} = \mathrm{V}^T_{(x_2 \in R_2 / R_0)} . \mathbf{n} - \mathrm{V}^T_{(x_1 \in R_1 / R_0)} . \mathbf{n} \tag{2}$$

Where $\mathrm{V}_{(x_i \in R_i / R_0)}$ is the velocity vector evaluated at the point x_i of object i having the minimal distance with the second object and $\mathbf{n}$ is the unit vector on the line of the minimal distance.
The non-collision constraints, taking into account the velocities of objects, are written as:

$$V^{T}_{(x_1 \in R_1 / R_0)} . \mathbf{n} - V^{T}_{(x_2 \in R_2 / R_0)} . \mathbf{n} \leq \xi \frac{d - d_s}{d_i - d_s} \tag{3}$$

A robot evolving towards an obstacle, if it respects constraints given by equation (1), it will evolve exponentially to the security distance without going closer than this distance (see Fig. 3).
Fig. 4 shows a PUMA robot placed next to a static obstacle. The constraint corresponding to that obstacle is written as:

$$V^{T}_{(x_1 \in R_1 / R_0)} . \mathbf{n} \leq \xi \frac{d - d_s}{d_i - d_s} \tag{4}$$

By introducing $\mathbf{J}_{x_1}(q)$, the Jacobian matrix of the robot in configuration q defined in point x_1, we get:

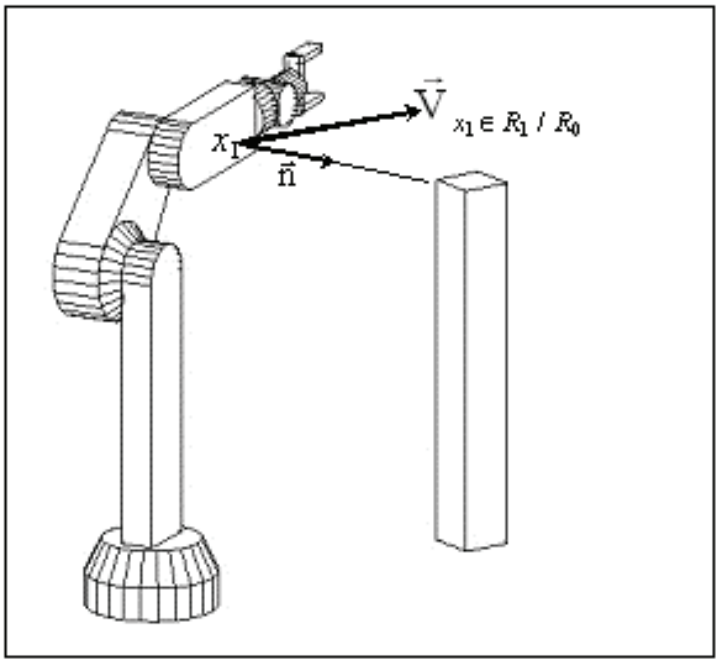

Figure 4.The distance between a robot and an obstacle

$$\mathbf{n}^{T} \mathbf{J}_{x_1}(q)\ \Delta q \leq \xi \frac{d - d_s}{d_i - d_s} \tag{5}$$

Condition (5) will be written in the following manner:

$$[a_1 \quad \cdots \quad a_N][\Delta q_1 \quad \cdots \quad \Delta q_N]^{T} \leq b \tag{6}$$

with $[a_1 \quad \cdots \quad a_N] = \mathbf{n}^T \mathbf{J}$, $\Delta q = [\Delta q_1 \quad \cdots \quad \Delta q_N]^T$ and $b = \xi \frac{d - d_i}{d_s - d_i}$

Figure 5. shows two PUMA robots evolving together. We consider that each robot is controlled separately.
In that manner, each robot is considered as a moving obstacle by the other one.

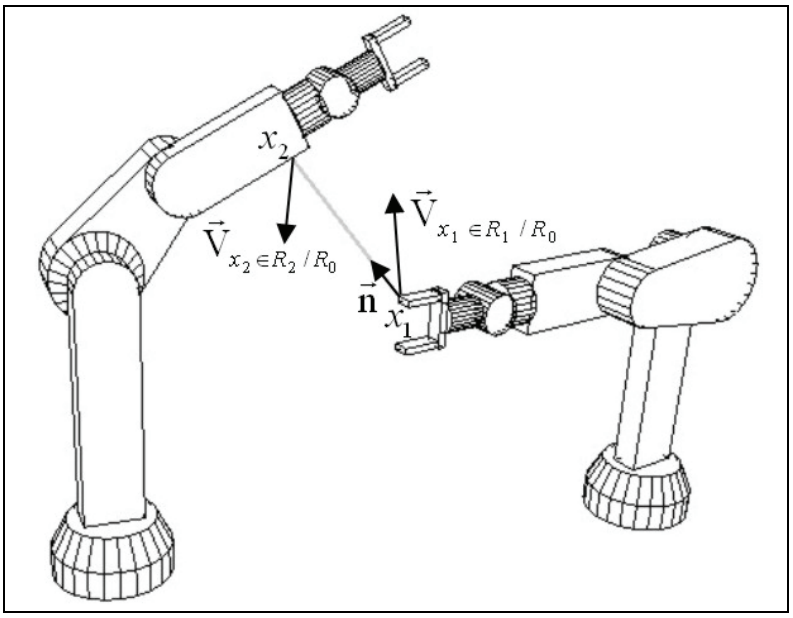

Figure 5. Two PUMA robots working in the same environment

The motion of the two robots must satisfy the following conditions:

$$V^T_{(x_1 \in R_1 / R_0)} . \mathbf{n} \leq \xi' \frac{d - d_i}{d_s - d_i} \text{ and } -V^T_{(x_2 \in R_2 / R_0)} . \mathbf{n} \leq \xi' \frac{d - d_i}{d_s - d_i} \tag{7}$$

Where $\xi' = \frac{1}{2} \xi$. While adding the two conditions of equation (7), we notice that the non-collision constraint defined by (3) is satisfied. So with a suitable choice of the parameters ξ , d_i and d_s , it is possible to use only condition (5) to avoid collisions with all objects in the environment.
In the next paragraph, we propose an approach that does not construct the whole grid, representing the CSpace. Only cells necessary to find the path to the goal position are checked for collision.

4. Path planning in static cluttered environments

The planner we propose uses two modes. The first one makes the robot evolve towards its goal position if there is no obstacle obstructing its way and the second mode is active near the obstacles and enables the robot to find the best way to avoid them. This latter mode is the most important as it needs to generate all the cells near the obstacle until it is avoided. For this reason, we do not have to store all the cells but just the ones near the obstacles which are sufficient to describe the CSfree.

4.1 Definitions

In order to explain the algorithm of this method, we need to define some terms.

A Cell
The algorithm we propose is based on a "Cell" class in terms of object oriented programming. A cell c_i is made of:

***A pointer to the parent cell (c_i.parent)*:**
the path from the initial configuration to the goal is made of cells. Each one of these cells has a pointer to the parent cell, that generated it previously. Starting from a cell, the next one in the path is the one that is closest to the goal and respecting the non-collision constraints. When the goal cell is reached the algorithm stops and the path is identified by all the selected cells.

***A configuration defining a posture of the robot*:**
each cell corresponds to a point in the CSpace. If a cell configuration is written as $q_1 = \begin{bmatrix} q_1^1 & \cdots & q_N^1 \end{bmatrix}^T$ where N is the number of DoF of the robot, and let Δq be the step of the grid, the neighboring cells are then defined as the configurations belonging to the following set:

$$Vic(q_1) = \left\{ q = \begin{bmatrix} q_1^1 + s_1 \Delta q & \cdots & q_N^1 + s_N \Delta q \end{bmatrix}^T ; (s_1, \ldots, s_N) \in \{-1,0,1\}^N / (0, \ldots, 0) \right\} \quad (8)$$

***A distance to the goal(c_i.distance_to_goal)*:**
it represents the distance in configuration space between the goal configuration and the cell configuration. This distance allows the planner's first mode to choose the closest cell to the goal configuration. While the robot is far from obstacles, the shortest path to the goal configuration is a straight line in CSpace.

***A boolean "collision" variable (c_i.collision)*:**
it takes false if the cell verifies the constraints and true if it does not.

***A boolean "computed" variable(c_i.computed)*:**
used by the planner in order to know whether the cell has already been used to search for the path or not.
***A boolean "near an obstacle" variable (c_i.near_an_obstacle)*:**
used by the second mode of the planner allowing it to stay stuck to the obstacle while performing its width search in order to find the best direction to go around the obstacle.

Queue
Another important item in our approach is the Queue, Q, which is defined as an ordered set of cells. The first cell in the Queue is named head and denoted h(Q). While the last cell is the tail of the Queue and denoted t(Q). If the Queue is empty we write $h(Q)=t(Q)=\emptyset$.

In order to handle the Queue Q, we use some operators that we define here.
$h^+(Q,c_1)$ adds the cell c_1 to the head of Q.
$t^+(Q,c_1)$ adds c_1 to the tail of Q.
$h^-(Q)$ removes the head cell from Q.
$t^-(Q)$ removes the tail cell from Q.

Stop Condition
We define the stop condition as the condition for which we judge that the goal position has been found. We write this condition as follows:

$$\left\| q_{goal} - q \right\| < \Delta q \tag{9}$$

where q_{goal} is the goal configuration, q is the configuration of the cell verifying the stop condition and Δq is the step of the grid.
If the algorithm can no longer evolve and the stop condition is not satisfied, it means that there is no possible solution for the given density of the grid.

4.2 Algorithm

The algorithm outlined in Fig. 6, starts by constructing the initial cell in step 1. It sets the parent pointer to NULL and evaluates the distance to the goal. The algorithm uses a variable c representing the cell searched by the algorithm.
$א$ is the set of explored cells and $א_1$ is the set of unexplored cells in the vicinity of cell c.
Step 6 computes non-collision constraints using distances between obstacles and robot parts evaluated in the posture defined by cell c. Steps 8 to 13 construct unexplored cells in the vicinity of cell c. For each cell the parent pointer

is set to c, the distance to goal is evaluated and the non-collision constraints are checked. A cell is considered a collision if it does not verify constraints given by equation (3).

Step 15 determines the nearest cell to the goal in the vicinity of c, using the distance to goal already evaluated. If that cell is not an obstacle, it is placed in the head of the queue Q at step 17. This makes the planner perform a depth search since there is no obstacle bothering the robot.

However, if the cell computed by step 15 is a collision, all non-collision cells in the vicinity of c that are close to collision cells are placed in the tail of the queue Q by step 22. This makes the planner perform a depth search until the obstacle is bypassed.

1. *Construct initial cell* c_1
2. *Set* $c = c_1$
3. *Let* $\aleph = \{c_1\}$
4. *While* $c \neq \emptyset$ *and c does not satisfy the stop condition do*
5. *c.computed=true*
6. *Compute non-collision constraints for the configuration represented by the cell c*
7. $\aleph_1 = Vic(c) \setminus \aleph$
8. *For each cell* $c_2 \in \aleph_1$ *do*
9. *Set c2.parent = c*
10. *Evaluate c2.distance_to_goal*
11. *Verify the non-collision constraints and determine c2.collision*
12. *Set c2.computed to false*
13. *End for*
14. $\aleph = \aleph \cup \aleph_1$
15. *Choose* c_3 *in* $\aleph_1$ *with the minimal distance to goal*
16. *If c3.collision=false then*
17. $h^+(Q, c_3)$
18. *Else (c3.collision=true)*
19. *For each* $c_2 \in Vic(c)$ *such as c2.collision=true do*
20. *For each* $c_3 \in Vic(c_2) \cap \aleph$ *set* *c3.near_an_obstacle=true*
21. *End for*
22. *For each* $c_2 = Vic(c) \setminus Q$ *such as c2.Near_an_obstacle = true and c2.collision=false and c2.computed=false do* $t^+(Q, c_2)$
23. *For each* $c_2 \in Q$ *such as* $Vic(c_2) \subset \aleph$ *remove c2 from the Queue Q and set c2.computed=true*
24. *End if*
25. $c = h(Q)$
26. $h^-(Q)$
27. *End while*

Figure 6. Pseudo-code of the method

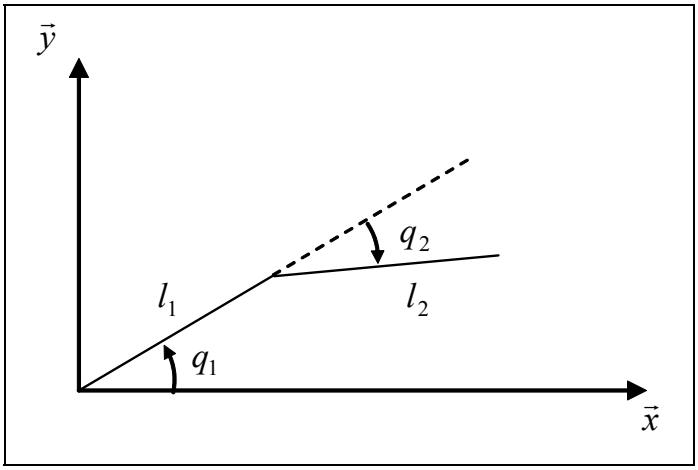

Figure 7. A 2 DoF robot

Steps 19 to 21 evaluate the "near an obstacle" property. This property is set to false when the cell is constructed. Then for each cell in the vicinity of a collision cell, itself in the vicinity of the cell c, this property is set to true.
Step 23 removes from the queue Q all cells for which their vicinity has been already explored and sets their computed property to true, so they do not return to the queue when the algorithm evolves. The search procedure is stopped when a cell verifying the stop condition is found and the path is done by joining this cell to the initial cell by going back through the parent cells using the pointer of each cell. The procedure can also be stopped if the Queue Q is empty, in that case there is no possible path for the chosen resolution of the grid.

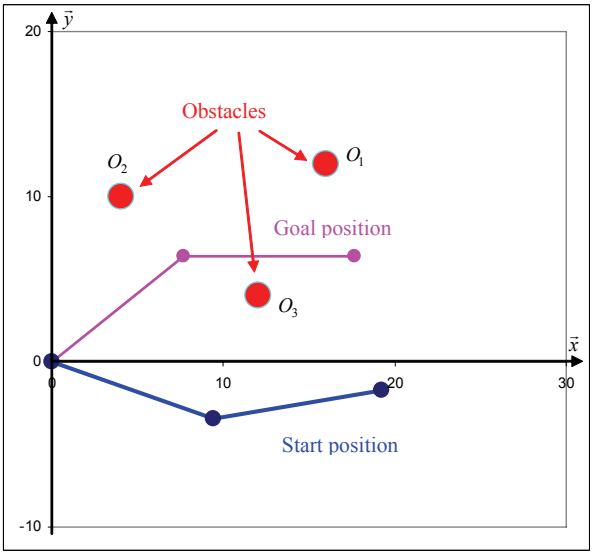

Figure 8. Path planning consists of moving the robot from the start position to the goal position while avoiding obstacles

Obstacle	O_1	O_2	O_3
x	16	4	10
y	12	10	4

Table 1. Position of obstacles

4.3 A planar example

In order to illustrate the proposed algorithm we consider a 2D example, of a 2R robot (Fig. 7) evolving among point obstacles. The simulations are made using three point obstacles defined by table 1.

The start configuration is $q_s = [-20° \quad 30°]^T$ and the goal configuration is $q_g = [50° \quad -45°]^T$. Fig. 8 shows the robot in its starting and goal positions, respectively, and the three point obstacles. We set the lengths of the arms of the robot $l_1 = l_2 = 10$.

Fig. 9 shows the CSpace of the robot, the dark regions correspond to CSpace obstacles.

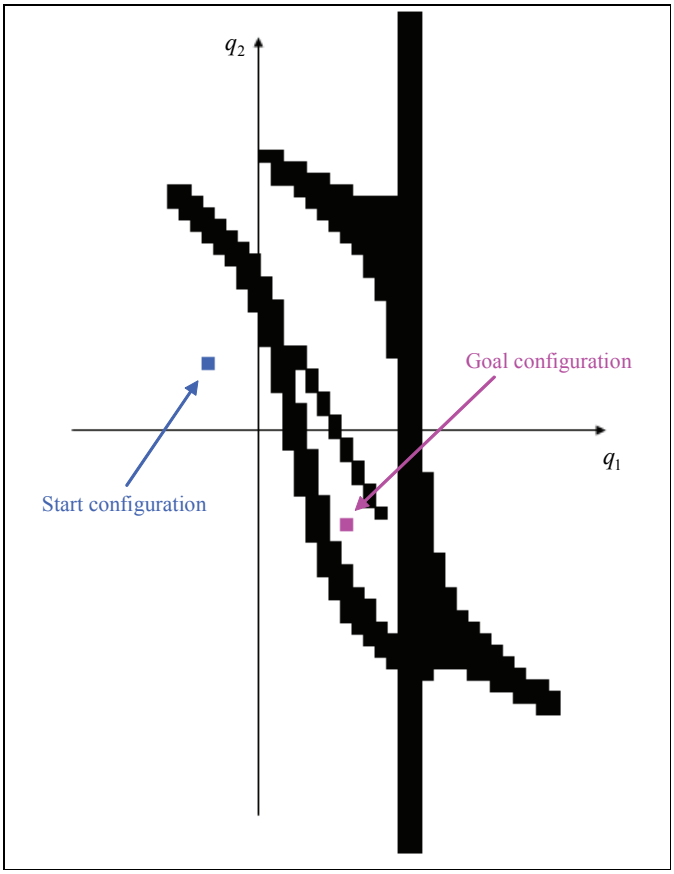

Figure 9. The configuration space

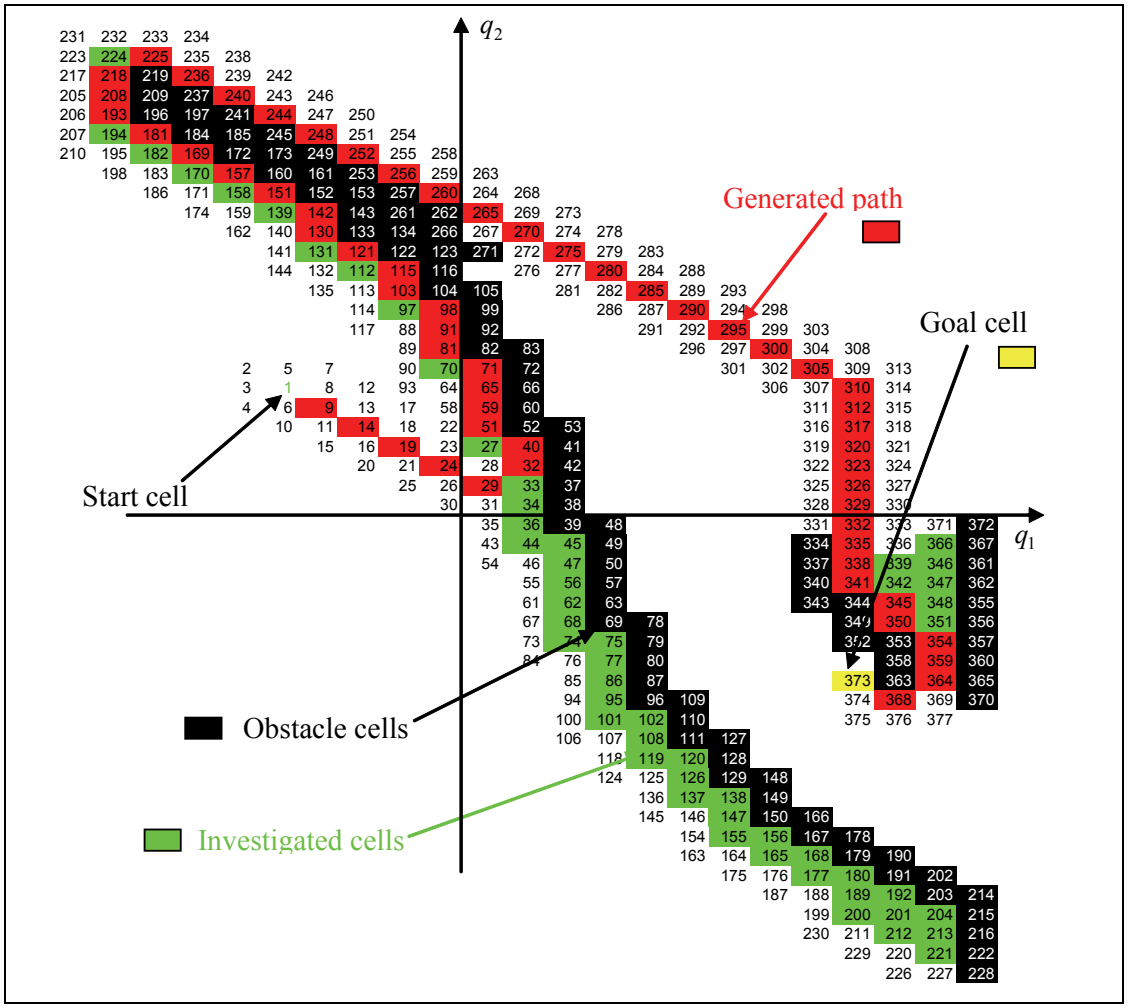

Figure 10. Cell generation order

The construction order of cells is shown in Fig. 10. The algorithm evolves towards its goal using the depth-search mode while there is no obstacle bothering it. When an obstacle is detected the algorithm uses the width-search mode. The algorithm overlaps the obstacle in order to find the best direction to bypass it. When the obstacle is avoided the depth search mode is resumed. The algorithm gives the best way to go around the C obstacle (which is the portion of CSpace corresponding to a collision with one obstacle).

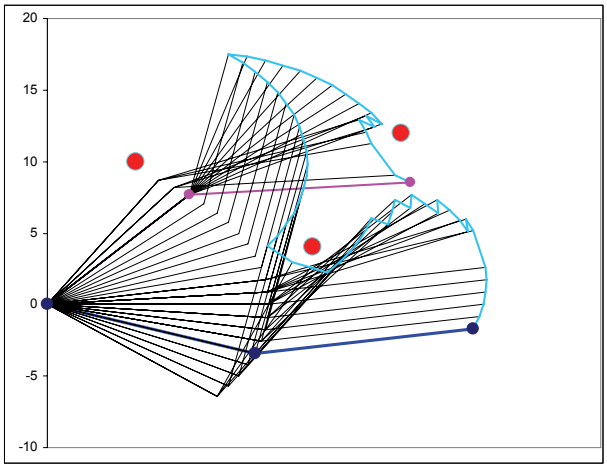

Figure 11. Simulation results for the planar robot

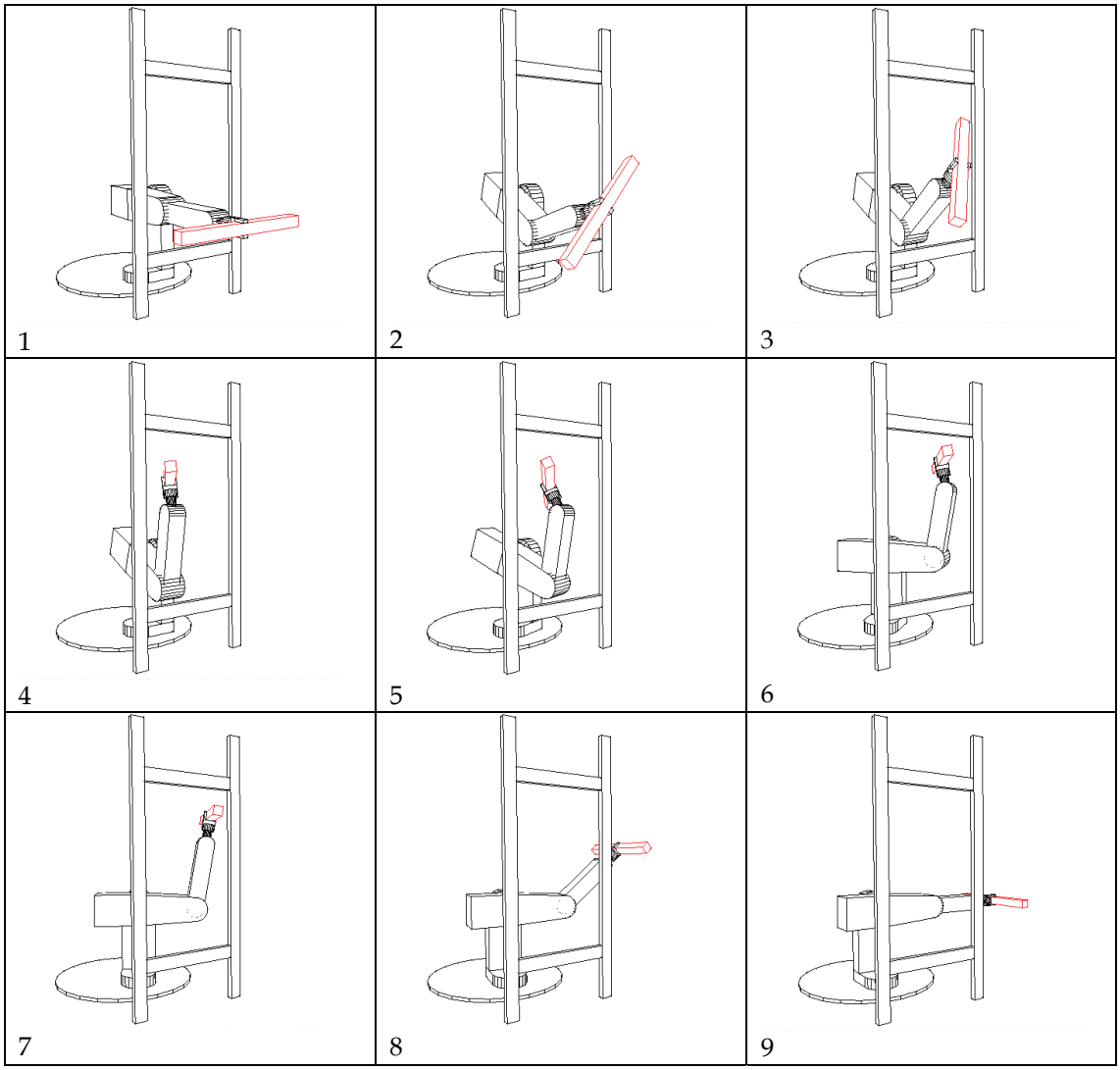

Figure 12. Simulation results for the PUMA robot

The result of the simulation is shown in Fig. 11. Moreover, out of 5329 cells, which corresponds to 73 points on each axis, only 375 cells were computed. This represents less than 10% of the whole workspace.

4.4 Simulation and results

The simulation has been performed on a robotic-oriented-Software named SMAR (Zeghloul et al., 1997). This software is made of two modules: a modeling module and a simulation one. The modeling module is used to generate a model of the robot in its environment. The simulation module is used to simulate the motion of the robot in its environment. It contains a minimal distance feature we used to implement our algorithm.
Fig. 12 shows the simulation results of a 5 DoF ERICC robot carrying a large object and standing in an environment containing ladder-shaped obstacles. The planner determines the path in 20 steps. The robot is carrying a beam whose length is greater than the width of the ladder-shaped obstacle. Regular local path planners would be stuck in the initial position. The proposed method explores all possible configurations capable of going around the obstacle and chooses the one that yields the minimum distance to the goal. The sequence of frames shown in Fig. 12, shows the solution found by the proposed planner. In this case the total number of cells is 12252303 while the number of computed cells is only 220980, which represents less than 2% of the whole workspace.

5. Real-time path planning in dynamic environments

The method described above is useful in the case of cluttered static environments. It can be used offline to generate repetitive tasks. In many cases robots evolve in dynamic environments, which are unknown in advance. That is why we propose to solve the path planning problem for many manipulator robots evolving in a dynamic environment using a real-time local method. This approach is based on the constraints method coupled with a procedure to avoid local minima by bypassing obstacles using a boundary following strategy.

5.1 Local Method

In this method, we use a local planner based on an optimization under constraints process (Faverjon & Touranssoud, 1987). It is an iterative process that minimizes, at each step, the difference between the current configuration of the robot and the goal configuration. When there are no obstacles in the way of the robot, we consider that it evolves towards its goal following a straight line in the CSpace. The displacement of the robot is written as follows:

$$\Delta q_{goal} = \frac{q_{goal} - q}{\left\| q_{goal} - q \right\|} \Delta q_{\max} \quad \text{if} \quad \left\| q_{goal} - q \right\| > \Delta q_{\max} \tag{10}$$

$$\Delta q_{goal} = q_{goal} - q \qquad \text{if} \quad \left\| q_{goal} - q \right\| \le \Delta q_{\max} \tag{11}$$

where q_{goal} is the goal configuration of the robot, q is the current configuration of the robot and $\Delta q_{\max}$ is the maximum variation of each articulation of the robot. If there are obstacles in the environment, we add constraints (defined in paragraph 3) to the motion of the robot in order to avoid collisions. Path planning becomes a minimization under constraints problem formulated as:

$$\text{Minimize} \left\| \Delta q - \Delta q_{goal} \right\| \text{ Under non - collision constraints} \tag{12}$$

where Δq is the change of the robot joints at each step. We can formulate then the planning problem as follows:

$$\text{Minimize} \left\| \Delta q - \Delta q_{goal} \right\| \text{ Under linear constraints } \mathbf{n}^T \mathbf{J}\, \Delta q \le \xi \frac{d - d_i}{d_s - d_i} \tag{13}$$

The local planner can be represented by an optimization problem of a nonlinear function of several parameters, subject to a system of linear constraint equations. In order to solve this problem, we use Rosen's gradient projection method described in (Rao, 1984). When the solution of the optimization problem Δq corresponds to the null vector, the robot cannot continue to move using the local method. This situation corresponds to a deadlock. In this case, the boundary following method is applied for the robot to escape the deadlock situation.
In the next section, we define the direction and the subspace used by the boundary following method.

5.2 Boundary following method

Before explaining the method in the general case of an n-DoF robot, we present it for the 2D case. The proposed approach to escape from the deadlock situation is based on an obstacle boundary following strategy.

The 2D case
This method was first used in path planning of mobile robots (Skewis & Lumelsky, 1992; Ramirez & Zeghloul, 2000).
When the local planner gets trapped in a local minimum (see Fig. 13), it becomes unable to drive the robot farther. At this point the boundary following

method takes over and the robot is driven along the boundary of the obstacle until it gets around it. The robot in this case has the choice between two directions on the line tangent to the obstacle boundary or on the line orthogonal to the vector to the goal (Fig. 13). It can go right or left of the obstacle. Since the environment is dynamic and unknown in advance, we have no idea whether going left or going right is better. The choice of the direction is made randomly. Once the obstacle is avoided the robot resumes the local method and goes ahead towards the goal configuration.

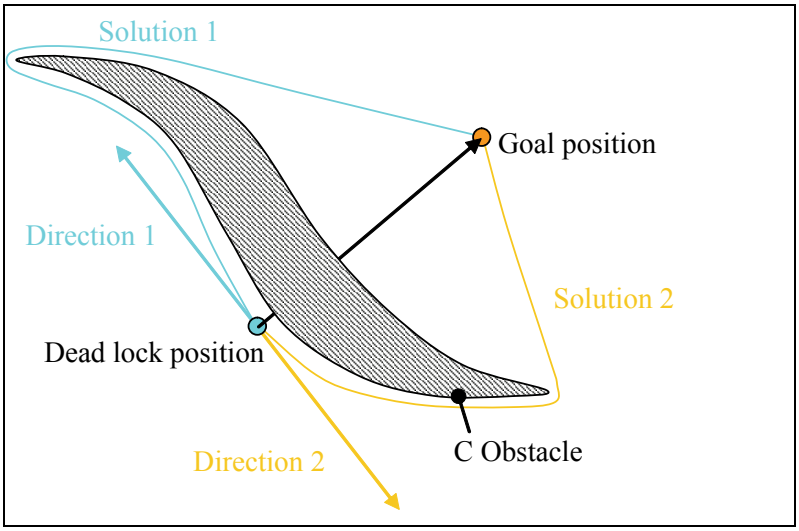

Figure 13. Two possible directions to bypass the obstacle in the case of a 2DoF robot

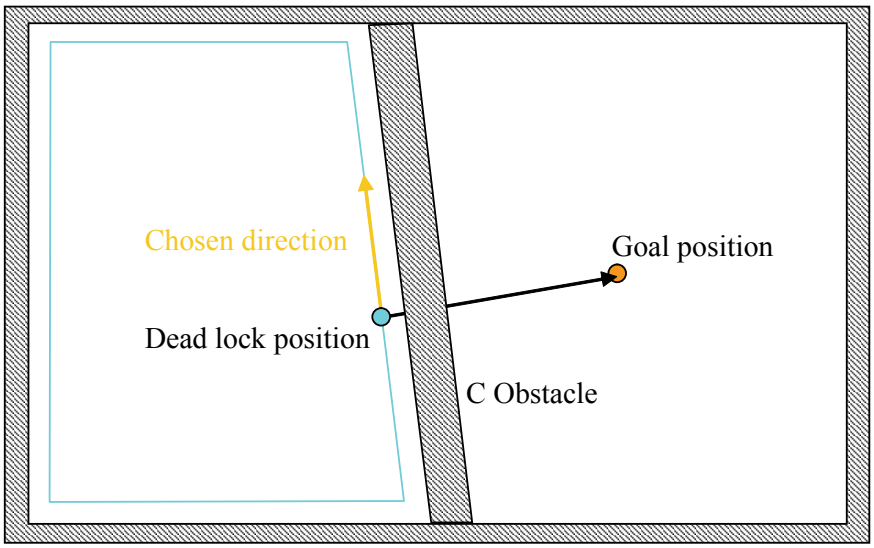

Figure 14. The case where there is no feasible path to the goal